UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Evidenčné číslo: dfbb9e25-1713-4d16-af87-c23ee246dab0

Pravdepodobnostná robotika v lokalizácii, mapovaní a plánovaní

2011 **Bc. Lukáš Riško**

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

Pravdepodobnostná robotika v lokalizácii, mapovaní a plánovaní

Diplomová práca

Študijný program : aplikovaná informatika
Študijný odbor: 9.2.9. aplikovaná informatika
Školiace pracovisko: Katedra aplikovanej informatiky
Školiteľ: Mgr. Pavel, Petrovič PhD.

Bratislava 2011 **Bc. Lukáš Riško**

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Lukáš Riško
Študijný program: aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.2.9. aplikovaná informatika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Pravdepodobnostná robotika v lokalizácii, mapovaní a plánovaní

Cieľ: Preštudovať spôsoby lokalizácie, mapovania a navigácie pomocou pravdepodobnostnej robotiky. Implementovať zvolenú metódu, analyzovať, alebo porovnať s inou metódou, navrhnúť vlastné zlepšenie.

Vedúci: Mgr. Pavel Petrovič, PhD.

Dátum zadania: 11.11.2009

Dátum schválenia: 03.05.2011

za

prof. RNDr. Ivan Kalaš, CSc.
garant študijného programu

.....
študent

.....
vedúci

Týmto prehlasujem, že som diplomovú prácu vypracoval samostatne, a všetku použitú literatúru uvádzam v zozname.

Bratislava, Máj 2011 Lukáš Riško

Ďakujem svojmu školiteľovi Mgr. Pavlovi Petrovičovi PhD. za jeho pomoc pri písaní tejto práce.

ABSTRAKT

Riško, Lukáš: Pravdepodobnostná robotika v lokalizácii, mapovaní a plánovaní
[Diplomová práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky
a informatiky.Katedra aplikovanej informatiky.
Vedúci diplomovej práce: Mgr. Pavel Petrovič PhD.
Bratislava. Fakulta matematiky, fyziky a informatiky UK. 2011. 55 strán.

Cieľom tejto práce je popísať pravdepodobnostné riešenia robotických problémov ako sú lokalizácia a mapovanie prostredia robotom vybaveným odometriou a ultrazvukovými senzormi. Tieto senzory sa vyznačujú vlastnosťami, ktoré robia loklizáciu a mapovanie netrivialnými úlohami. Zameráme sa na implemntácie algoritmu Particle filter, pri ktorom distribúciu belief reprezentuje množina váhovaných vzoriek. Pre naše experimenty bol použitý reálny mobilný robot.

ABSTRACT

Riško, Lukáš: Pravdepodobnostná robotika v lokalizácii, mapovaní a plánovaní
[Diplomová práca]. Univerzita Komenského v Bratislave. Fakulta matematiky, fyziky
a informatiky.Katedra aplikovanej informatiky.
Vedúci diplomovej práce: Mgr. Pavel Petrovič PhD.
Bratislava. Fakulta matematiky, fyziky a informatiky UK. 2011. 55 pages.

The aim of this work is to decribe probabilistic aproaches for localization of mobile robot and aproaches for building a map of an environment with robot, using a combination of odometry and sonar range sensors. These sensors have a number of properties that make localization and map building a non-trivial process. We focus on implementation of Particle filter algorithm, where the set of particles approximates the belief distribution. For our experiments, we have been using real mobile robot.

Obsah

Kapitola 1

Úvod

Loklizácia a tvorba mapy prostredia sú dve z hlavných požiadaviek na autonómne roboty. Pre správne vykonávanie akcii, na ktoré je robot nasadený, musí často vedieť, kde v prostredí sa nachádza a ako dané prostredie vyzerá. V práci sa budeme venovať pravdepodobnostným metódam používaným v robotike. Ukážeme, že ak je robot vybavený senzormi na meranie vzdialenosti a odometriou, je možné algoritmicky dostatočne presne určiť jeho polohu vzhľadom na mapu prostredia, ktorá je apriórne daná. Po tomto probléme prejdeme na jeho zovšeobecnenie, ktorým je simultánna lokalizácia a mapovanie prostredia (SLAM). Ten ostával od svojho vzniku v roku 1986 dlho nevyriešeným. V súčastnosti existuje množstvo rôznych metód pre jeho riešenie. Väčšina z nich je však špecifická pre konkrétnu robotickú platformu a prostredie. Vysporiadať sa s niektorým z dvoch spomínaných problémov, znamená predovšetkým vysporiadať sa s podproblémami v nich obsiahnutými. Ide hlavne o pravdepodobnostné modelovanie pohybu a snímania systému, reprezentáciu a generovanie mapy prostredia.

Práca začína popisom základného pojmového aparátu používaného v provdepodobnostnej robotike. Po zadefinovaní pojmov nevyhnutných pre pochopenie náročnejších kapitol, uvedieme schému Bayesovho filtra ako základného algoritmu pre rekurzívny odhad neznámej pravdepodobnostnej distribúcie z prichádzajúcich dát. Zameráme sa na jednu z jeho možných implemntácii, ktorou je Particle filter. Ten je jadrom lokalizačného algoritmu Monte Carlo, ktorý podrobne popíšeme. Pre riešenie problému SLAMu sa zameriame na skupinu algoritmov využivajúcich Rao-Blackwellized Particle filter. Reprezentantom prostredia budú pre nás celú dobu gridmapy. Spomenieme ich zovšeobecnenie (res-

ponse gridmapy), ktoré sa omnoho lepšie hodí na mapovanie prostredia ultrazvukovými senzormi a odvodíme si techniku pre ich aktualizáciu zakomponovaním prijatých senzorových dát. Druhá časť práce popisuje naše implementácie algoritmov do Javy a experimenty prevedené na reálnom mobilnom robotovi vybavenom ultrazvukovými senzormi. Podrobne popíšeme ako bol modelovaný jeho pohyb a snímanie.

V závere sa vyjadríme k dosiahnutým výsledkom a spomenieme možné námety na vylepšenie vytvorených aplikácii.

Kapitola 2

Pravdepodobnostná robotika

2.1 Základné pojmy a pravidlá

2.1.1 Mapy

Aby bolo možné popísať procesy v prostredí ako je pohyb či snímanie, je potrebné najprv popísať prostredie samotné. Reprezentantom prostredia bude *mapa*, čo je zoznam objektov prostredia spolu s ich vlastnosťami, resp. zoznam možných pozícii v prostredí spolu s ich vlastnosťami [20]. Formálne:

$$m = (m_1, m_2, ..., m_M) \tag{2.1}$$

Indexovanie prvkov toho zoznamu je možné chápať dvoma spôsobmi, ktoré zároveň separujú mapy do dvoch základných skupín:

- *feature-based maps*

  Každý prvok m_i zoznamu m reprezentuje jeden objekt prostredia, pričom i je jeho identifikátor, m popisuje jeho vlastnoť - pozíciu. Mapa je teda zoznam pozícii kľúčových objektov prostredia nazývaných *landmarky*. Aby bolo možné ich identifikovať a navzájom rozlíšiť sú potrebné apriórne znalosti o prostredí, čo je nevýhodou tejto reprezentácie. [18]

- *location-based maps*

  Každý prvok m_i zoznamu m reprezentuje jednu pozíciu v prostredí, pričom i je jej identifikátorom, m je jej vlastnosť - obsadenosť. V planárnom pro-

stredí býva index i vyjadrený formou dvojice (x, y), čo sú Karteziánske súradnice pozície. Ich nevýhodou oproti mapám prvého typu je to, že sú volumetrické. Obsahujú nielen informácie o objektoch ale aj o absencii objektov. Location-based mapy, ktoré cele prostredie diskretizujú na sieť rovnakých štvorcových buniek a o každej bunke uchovávajú jedinu vlastnoť - obsadenosť, sa označujú ako *gridmapy (occupancy grid map)*. Nakoľko reálne prostredie nie je často tvorené kolmými predmetmi, gridmapy nie su absolútne presné. Voľbou vhodnej veľkosti buniek je možne dosiahnuť, že gridmapa bude obsahovať všetky potrebné informácie [10]. Čím ale menšie rozmery bunky, tým väčší počet všetkých buniek, čo znamená väčšie priestorové nároky.

Ak by sme uvažovali napríklad prostredie s dvoma rovnakými štvorcovými prekážkami s dĺžkou hrany 10 milimetrov, umiestenými tak, že ich stredy sú na súradniciach $(15, 15)$ a $(25, 25)$ príslušná faetures-based mapa by mohla byť $m_{fb} = ([15, 15, 10], [25, 25, 10])$. Gridmapa s bunkami o dĺžke hrany 10 milimetrov toho istého prostredia by bola $m_g = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

2.1.2 Veličiny a ich náhodné premenné

V pravdepodobnostnej robotike sú veličiny modelované ako *náhodné premenné*. Každá náhodná premenná nadobúda hodnoty zo svojej domény podľa istých pravidiel. Veličiny, ktoré sa budú v naších problémoch vyskytovať sú:

- *pozícia robota*

  Náhodnú premennú vyjadrujúcu pozíciu robota vzhľadom na mapu prostredia budeme označovať x, niekedy ako trojicu (x, y, h), kde x a y sú súradnice robota, h je jeho natočenie. Z kontextu bude vždy zrejmé, či x vyjadruje pozíciu kompletne alebo iba jej prvú zložku.

$$x = (x, y, h) \tag{2.2}$$

- *vzdialeností namerané senzormi*

  Vzdialenosti namerané senzormi (*measurement data*) bude reprezentovať náhodná premenná z. Ak robot disponuje K senzormi, potom

$$z = (z^1, z^2, ..., z^K) \tag{2.3}$$

- *zmena natočenia a vzdialenoť prejdená v poslednom kroku*

  Zmenu súradníc pozície a natočenia robota (*control data*), ktorá nastala jeho poslednou vykonanou akciou, reprezentuje náhodná premenná

$$u = (\Delta x, \Delta y, \Delta h) \tag{2.4}$$

- *mapa*

  Mapa prostredia, tak ako bola sformalizovaná v 2.1 nadobúda rôzne hodnoty, takže je to v podstate náhodna premenná modelujúca prostredie.

2.1.3 Belief

V úlohach, ktorým sa budeme venovať, bude pozícia robota vždy veličina, ktorú priamo nepozoruje. Robot ju musí v každom časovom okamihu odvodiť z pozorovaných veličín, ktorými sú vzdialenosti namerané senzormi či zmeny pozície. Pri pravdepodobnostnej paradigme sa nejedná o odvodenie jednej hodnoty nepozorovataľnej veličiny, ale o odvodenie pravdepdobnostnej distribúcie nepozorovanej náhodnej premennej podmienenej pozorovanými náhodnými premennými nad množinou jej všetkých možných hodnôt. Táto pravdepdobnostná distribúcia sa označuje ako *belief (domnienka)*. Formálne:

$$bel(x_t) = p(x_t \mid u_{1:t}, z_{1:t}) \tag{2.5}$$

Je to pravdepodobnosť toho, že v čase t je pozícia robota x_t, ak boli doteraz vykonané pohyby $u_{1:t} = (u_1, u_2, ..., u_t)$ a senzormi namerané vzdialenosti $z_{1:t} = (z_1, z_2, ..., z_t)$. Cieľom je, aby táto distribúcia bola čo najviac podobná skutočnej distribúcii skúmanej premennej. [14].

Je potrebné rozlišovať *apriórny* a *posteriórny* belief. Belief (2.5) je posteriórny, nakoľko je podminený aj posledným meraním z_t. Apriórny belief v nejakom okamihu t je ten, ktorý robot má pred spracovaním posledného merania z_t

$$bel^{-}(x_t) = p(x_t|u_{1:t}, z_{1:t-1}) \tag{2.6}$$

Reprezentácie belief a pravdepodobnostných distribúcii vôbec sa delia na:

- *spojité*

  Distribúcia náhodnej premennej je spravovaná nejakou funkciou hustoty pravdepodobnosti (*probabilistic density function, pdf*). Všetko potrebné pre vypočet pravdepodobnosti konkrétnej hodnoty odhadovanej náhodnej premennej je predpis funkcie a jej parametre [14]. Nájsť pre distribúciu vhodnú funkciu hustoty a jej parametre je často netriviálna úloha.

- *diskrétne*

  Tieto reprezentácie diskretizujú spojitý priestor hodnôt náhodnej premennej na konečný počet diskrétnych hodnôt. Pravdepodobnosť konkrétnej hodnoty môže byť tym pádom uchovávaná explicitne.

2.1.3.1 Váhovaná množina vzoriek

Jednou z diskrétnych aproximácii pravdepodobnostných distribúcii, ktorú budeme používať, je konečná množina váhovaných vzoriek (*weighted sample set*) distribuovaných v priestore podľa belief.

$$S = \{(x^i, w^i) | i = 1..N\} \tag{2.7}$$

Každá vzorka je dvojica, ktorej prvá zložka je konkrétna možná hodnota náhodnej premennej - *hypotéza*. Druhá zložka dvojice je váhou vzorky - *pravdepodobnoť splnenia hypotézy*. Predpokladáme, že

$$\sum_{i=1}^{N} w^i = 1 \tag{2.8}$$

V ideálnom prípade je vierohodnosť toho, že vzorka $s = (x, w)$ sa vo výbere S nachádza, úmerná posteriórnemu belief $bel(x)$.

Ak je potrebné z množiny vzoriek vybrať, či vytvoriť jedného reprezentanta v_{est} hodnoty veličiny zastupenej náhodonou premennou x je možné použiť niektorú z nasledujúcich metód, z ktorých posledná je často najvhodnejšia no zároveň výpočtovo nazložitejšia [17].

- *váhovaný priemer* $v_{est} = \sum_{i=1}^{N} w^i * x^i$
- *najlepšiu vzorku* $v_{est} = x^j \mid w^j = max(w^i) : i = 1..N$
- *váhovaný priemer vzoriek sústredených okolo najlepšej vzorky (robust mean)*

2.2 Bayesov filter

Algorithm 1 Bayes Filter

Input: $bel(x_{t-1}), u_t, z_t$

Output: $bel(x_t)$

1: **for all** x_t **do**

2: $\quad bel^-(x_t) = \int p(x_t|u_t, x_{t-1})\ bel(x_{t-1})\ \mathrm{d}x$

3: $\quad bel(x_t) = \eta\ p(z_t|x_t)\ bel^-(x_t)$

4: **end for**

Všeobecná schéma algoritmov pre výpočet belief nesie názov *Bayesov filter* 1. Algoritmy implementujúce Baysov filter sú rekurzívne. Na výpočet $bel(x_t)$ potrebujú poznať $bel(x_{t-1})$, ktorý updatujú spracovaním vstupných dát u_t a z_t. Použitie dát u_t je na 3.riadku algoritmu nazývanom *predikcia*. Jej výsledkom je apriórny belief v čase t, získaný s posteriorného belief predchádzajúceho časového okamihu $bel(x_{t-1})$ a vstupných dát u_t. V ďalšom kroku algoritmu nazývanom *korekcia* je apriórny belief použitý spolu s vstupnými dátami z_t na výpočet posteriórneho belief.

Ako pri každom rekurzívnom algoritme, tak aj pri Bayesovom filtri má zmysel klásť si otázku, aká ja počiatočná hodnota rekurzívnej premennej. Odpoveď závisí od informácii, ktoré robot má k dispozícii na začiatku riešenia úlohy. Pre rôzne typy úloh, sú tieto začiatočné informácie rôzne.

2.2.1 Kalmanov filter

Najznámejšou technikou implementujúcou Bayesov filter je *Kalmanov filter* [20]. Ten prevádza výpočet belief so spojitou reprezentáciou. Aby ho bolo možné použiť, musia byť splnené nasledujúce predpoklady:

- inicializačný belief musí byť vyjadrený Gaussovou funkciou
- model pohybu musí byť lineárna funkcia s pridaným Gaussovským šumom
- model snímania musí byť lineárna funkcia s pridaným Gaussovským šumom

Existujú techniky, ktoré umožňujú použiť Kalmanov filter aj na nelineárne systémy. Jednou z nich *Extended Klaman filter*. Napriek veľkej popularite Kalmanovho filtra a jeho dirivácii, obrátime v tejto práci pozornosť na inú skupinu implemntácii Baysovho filtra. Viac o Kalmanových filtroch je možné nájsť v [19], [20], [21].

2.2.2 Particle filter

Algorithm 2 Particle Filter

Input: $S_{t\text{-}1}$, z_t, u_t

Output: S_t

1: $S_t^- = \emptyset$
2: **for** i = 1 **to** N **do**
3: sample $x_t^i \sim p(x_t \mid u_t, x_{t-1}^i)$
4: $w_t^i = p(z_t \mid x_t^i)$
5: $S_t^- = S_t^- \cup \langle x_t^i, w_t^i \rangle$
6: **end for**
7: $S_t = \emptyset$
8: **for** i = 1 **to** N **do**
9: draw k with probability $\propto w_t^k$
10: $S_t = S_t \cup \langle x_t^k, w_t^k \rangle$
11: **end for**

Particle filter 2 nazývaný tiež *Sequential Monte Carlo, (SMC)* je implentácia Bayesovho filtra, pre výpočet belief s diskrétnou reprezentáciou vo forme množinou váhovaných vzoriek 2.7. Jeho myšlienka je pomerne jednoduchá. Na vtupe dostane pohybové dáta u_t, senzorové dáta z_t a N-porvkovú množinu váhovaných vzoriek distribuovaných v stavovom priestore tak, že aproximujú $bel(x_{t-1})$. Najprv sa zmení stav každej vzorky na základe modelu pohybu, ktorý stochasticky generuje nový stav vzorky na základe distribúcie $p(x_t \mid x_{t-1}, u_t)$. Takýto model pohybu bližšie popíšeme v nasledujúcej kapitole. Po zmene stavu vzorky nastáva zmena jej váhy, čiže zmena vierohodnoti hypotézy, ktorá vraví že robot sa nachádza v stave rovnakom ako je stav danej vzorky. Nová váha vzorky je taká, akú dá model snímania vyjadrujúci pravdepodobnosť zosnímania dát z_t na pozícii vzorky. Na rozdiel od modelu pohybu, ktorý je pre particle filter špecifický, v roli modelu snímania je možné použiť ľubovoľný model. Posledná fáza algoritmu je resamplovanie. V podstate je to pravdepodobnostná implementácia Darwinovej myšlienky *"Survival of the fittest"*. Jeho úlohou je výberom s opakovaním vytvoriť novú množinu vzoriek. Pravdepodobnosť toho, že vzorka sa dostane do nového výberu má závisieť od jej váhy. Resamplovanie nie je nutné vykonávať pri každom aplikovaní filtra. Jednou z možnosti ako určiť, či je resamplovanie potrebné, je zitiť, či hodnota

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (w_t^i)^2} \tag{2.9}$$

nazývaná *effective number of particles* je pod nejakou zvolenou konštantou.

Vhodný počet vzoriek závisi od úlohy, žiadanej presnosti výsledku a hardwarových možností. Platí, že čím je počet vyšší, tým je metóda stabilnejšia, no tým sú aj vyššie jej vypočtové nároky. Počet vzoriek nemusí byť počas celej doby konštantný.

2.2.2.1 Importance resampling

Podstata resamplovania je z množiny vzoriek vytvoriť novú výberom s opakovaním z pôvodnej množiny vzoriek s prihliadnutím na ich váhu. Vzorky s malou váhou, majú malú pravdepodobnoť, že sa dostanú do nového výberu a tak zanikajú. Efekt resamplovania je nahradiť váhu vzorky na nejakej pozícii skupinou vzoriek na tej istej pozícii. [10]

Najjednoduchšia metóda výberu s opakovaním je algoritmus *Select with replacement* 3. Jeho časová zložitosť je závislá od zložitosti utriedenia poľa náhodných čísel z uniformného rozdelenia, čo je $O(Nlog(N))$

Algorithm 3 Select with replacement

Input: $W = \{w_i\}, \sum_{i=1}^{N} w_i = 1$

Output: $SelectedIndices = \{j_i\}$

1: $Q = \{q_k \mid k = 1...N,\ q_k = \sum_{i=1}^{k} w_i\}$
2: $R = \{r_k \mid k = 1...N+1,\ r_k = rand(0,1)\}$
3: $S = sort(R)$
4: $S[N+1] = 1$
5: $i = 1$
6: $j = 1$
7: **while** i ≤ N **do**
8: **if** S[i] ≤ Q[j] **then**
9: SelectedIndices += $\{j\}$
10: i++
11: **else**
12: j++
13: **end if**
14: **end while**

V práci [2] bol navrhnutý resamplovací algoritmus bežiaci v linárnom čase. Kým predchádajúci algoritmus 3 využíval na vytvorenie usporiadanej množiny náhodných čísel z uniformného rozdelenia triedenie náhodných čísel z intervalu $(0, 1)$, algoritmus *Select with replacement in linear time* 4 ju vytvára pomocou kumulatívnych súm záporných logaritmov náhodných čisel z intervalu $(0, 1)$.

Algorithm 4 Select with replacement in linear time

Input: $W = \{w_i\}, \sum_{i=1}^{N} w_i = 1$

Output: $SelectedIndices = \{j_i\}$

1: $Q = \{q_k \mid k = 1...N,\ q_k = \sum_{i=1}^{k} w_i\}$
2: $R = \{-log(r_k) \mid k = 1...(N+1),\ r_k = rand(0, 1)\}$
3: $T = \{r_k \mid k = 1...(N+1),\ t_k = \sum_{i=1}^{k} r_i\}$
4: $T = normalize(T)$
5: $i = 1$
6: $j = 1$
7: **while** i ≤ N **do**
8: **if** S[i] ≤ Q[j] **then**
9: SelectedIndices += $\{j\}$
10: i++
11: **else**
12: j++
13: **end if**
14: **end while**

Ďalšie metódy resamplovania je možné nájť v [17], [9]

2.3 Model pohybu

Implementovať fázu predikcie Bayesovho filtra 1 znamená reprezentovať pravdepodobnostnú distribúciu

$$p(x_t \mid x_{t-1}, u_t) \tag{2.10}$$

nazývanú *model pohybu (motion model)*. Ten popisuje pravdepodobnosť prechodu pozície zo stavu x_{t-1} do stavu x_t vplyvom vykonaných pohybov u_t. Pohyb robota je činnosť, ktorá jeho domnienku o stave pozície znejasňuje, zahmlieva. Ak má robot pred pohybom nejakú hypotézu o svojej pozícii, ktorej vierohodnoť je akákoľvek, po vykonaní pohybu vierohodnoť tejto a každej inej hypotézy klesne.

2.3.1 Sample motion model

Sample motion model 5 je implementácia modelu pohybu pre particle filter. Je schopný generovať nové vzorky na základe distribúcie $p(x_t \mid x_{t-1}, u_t)$, čím túto distribúciu istým spôsobom reprezentuje. Tak ako bolo spomínané v kapitole 2.1.2 predpokladáme, že v čase t sú pozícia robota a pohybové dáta vyjadrené náhodnými premennými

$$x_t = (x_t, y_t, h_t) \tag{2.11}$$

$$u_t = (\Delta x_t, \Delta y_t, \Delta h_t) \tag{2.12}$$

Algorithm 5 Sample motion model

Input: $x_{t\text{-}1} = (x_{t\text{-}1},\ y_{t\text{-}1},\ h_{t\text{-}1})$, $u_t = (\Delta x, \Delta y, \Delta h)$

Output: $x_t = (x_t,\ y_t,\ h_t)$

1: $\Delta t = \sqrt{\Delta x + \Delta y}$
2: $\Delta h^* = \Delta h + sample(\alpha_1 \Delta h + \alpha_2 \Delta t)$
3: $\Delta t^* = \Delta t + sample(\alpha_3 \Delta t + \alpha_4 \Delta h)$
4: $x_t = x_{t-1} + \Delta t^* \cos(h_{t-1} + \Delta h^*)$
5: $y_t = y_{t-1} + \Delta t^* \sin(h_{t-1} + \Delta h^*)$
6: $h_t = h_{t-1} + \Delta h^*$

Algoritmus najprv vyráta prejdenú vzdialnosť Δt z na vstupe získaných prejdených vzdialeností v jednotlivých súradnicových smeroch $\Delta x, \Delta y$. Hodnote tejto vzdialenosti a hodnote zmeny natočenia Δh zo vstupu pridá gaussovký šum. Tento šum reprezentujú hodnoty vybrané z gaussovského rozdelenia so stredom v nule a varianciou závislou od parametrov α_1, α_2, α_3, α_4 a hodnôt Δt a Δh. Význam parametrov modelu je nasledovný:

- α_1 hovorí ako silno vplyvá zmena natočenia na chybu natočenia,
- α_2 hovorí ako silno vplyvá zmena polohy na chybu natočenia,
- α_3 hovorí ako silno vplyvá zmena polohy na chybu polohy,
- α_4 hovorí ako silno vplyvá zmena natočenia na chybu polohy.

Jednduchý spôsob ako vyberať hodnoty z gaussovkej distribúcie je algoritmus *Sample from normal distribution* 6 .

Algorithm 6 Sample from normal distribution

Input: *variance*

Output: x

1: $x = \frac{variance}{6} \sum_{i=1}^{12} rand(-1, 1)$

2.4 Model snímania

Zmena vierohodnosti hypotézy nastáva pri korekcii. Implementovať ju znamená vyjadriť pravdepodobnostnú distribúciu

$$p(z_t \mid x_t) \tag{2.13}$$

nazývanu model snímania (*measurement model*). Ten popisuje pravdepodobnoť zosnímania dát u_t, ak stav pozície je x_t.

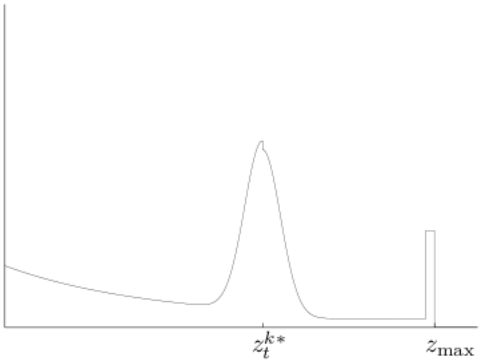
Ak zosnímané dáta z_t obsahujú viacero hodnôt $(z_t^0, z_t^1, ..., z_t^K)$, získané či už viacnásobným použitím jedného senzora alebo použitím viacerých senzorov, potom predpokladáme, že výsledok merania jedného senzora neovplyvňuje výsledok merania ďalších a pravdepodobnosť namerania celej K-tice hodnôt je daná súčinom jednotliých pravdepodobností

$$p(z_t \mid x_t) = \prod_{i=0}^{k} p(z_t^i \mid x_t) \tag{2.14}$$

Predpoklad nezávislosti jednotlivých meraní je iba teoretickým zjednodušením. V praxi, napríklad pri použití ultrazvukových senzorov, nie vždy platí. Šírka priestoru, v ktorom sonar dokáže zachytiť prekážku, je tak veľká, že snímateľné priestory susedných senzorov sa na niektorých miestach prekrývajú. Nájsť však presne hranice priestoru snímateľného jedným senzorom a pravidlá popisujúce závislosti meraní senzorov, by bolo náročné. Zjednodušený, nezávislosť predpokaldjúci model je často postačujúci.

2.4.1 Beam measurement model

V algortime Particle filter slúži model snímania na ohodnotenie vzoriek. Každej vzorke nastaví hodnotu váhy v závislosti od toho, aká je pravdepodobnosť zosnímania senzorových dát na jej pozícii, teda na základe ditribúcie $p(z_t \mid x_t)$. Algoritmus *Range finder measurement model* 7 je na rozdiel od *Sample motion model* 5 možné použiť aj v iných implementáciach Bayesovho filtra ako je Particle filter.



Obr. 2.1: Beam range finder model

Model 2.1 v základnej verzii podrobne opísanej v [20] reprezentuje distribúciu $p(z_t \mid x_t)$ spojito pomocou špecifickej funkcie hustoty pravdepodobnosti zloženej zo štyroch zložiek. Každá z týchto zložiek je nejaká štandardní funkcia hutoty pravdepodobnosti a modeluje niektorú z chýb, ktorých príčiny vzniku sú

- *šum merania*

  Presnosť a rozlíšenie senzorov sú obmedzené. Zosnímaná hodnota býva zaokrúhlená a často mierne odlišná od skutočnej. Tato zložka býva často modelovaná *gaussovou distribúciou* so stredom v ideálnej hodnote merania a varianciou, ktorá je parametrom modelu.

- *neočakavané objekty v okolí*

  V dynamickom prostredí je možné zosnímať objekty, ktoré mapa prostredia neobsahuje. Pravdepodobnoť takéhoto zosnímania klesá s rastúcou vzdialenosťou. Býva modelovaná *exponeniálnou distribúciou.* V statickom prostredí ju možno vynechať.

- *zlyhanie sensora*

Senzor niekedy nezachytí prekážku a vráti maximálnu hodnotu. Tento prípad je modelovaný *point-mass distribúciou.*

- *náhodné chyby*

  Hoci je príčiny náhodných chýb komplikované popísať, je nutné ich v modeli zohľadníť. Každej možnej hodnote je priradená rovnaká nenulová pravdepodobnoť, že ju senzor vráti. Zložka je modelovaná *uniformnou distribúciou.*

Každá zo spomínaných distribúcii prispieva do výslednej distribúcie 2.1 svojou váhou. Váhy zložiek sú popri parametroch jednotlivých distribúcii parametrami modelu snímania. Tie je možné nastaviť "od oka", alebo vydolovať nejakým *expectation maximization* algoritmom z dát obsahujúcich páry nameraných a skutočných vzdialeností $(z_i, *z_i)$.

Algorithm 7 Beam measurement model

Input: $z_t,\ x_t$

Output: $prob(z_t \mid x_t)$

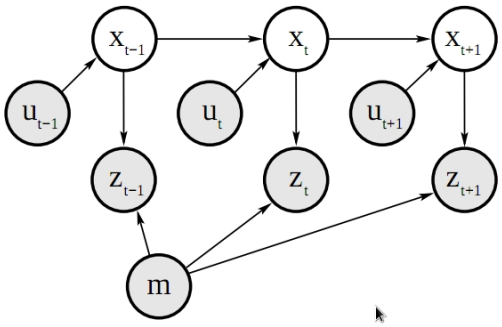
1: *compute* z_t^* *according to* x_t
2: $prob(z_t \mid x_t) = w_{norm} * p_{norm}(z_t, z_t^*)$
3: $prob(z_t \mid x_t) += w_{exp} * p_{exp}(z_t, z_t^*)$
4: $prob(z_t \mid x_t) += w_{pointmass} * p_{pointmass}(z_t^*)$
5: $prob(z_t \mid x_t) += w_{uni} * p_{uni}(z_t)$

2.5 Lokalizácia

Lokalizácia mobilného robota (*mobile robot localization*) je problém, pri ktorom je cieľom určiť pozíciu robota x_t vzhľadom na danú mapu statického prostredia m, ktorú ma robot apriórne k dispozícii. Formálny popis problému lokalizácie je v každom časovom okamihu t čo najpresnejčie určiť distribúciu

$$bel(x_t) = p(x_t |\ u_{1:t}, z_{1:t}, m) \tag{2.15}$$

Grafický model problému loklizácie je znázornený na Algoritmy lokalizácie, ktoré reprezentujú belief provdepodobnostnou distribúciou nad množinou možných pozícii a pre update belief po vykonaní pohybu a merania využívajú schému Bayesovho filtra 1, sa súhrne označujú ako *Markov Localization.*



Obr. 2.2: Grafický model problému loklizácie

Lokalizačné problémy je možné kategorizovať podľa množstva informácii, ktoré sú robotovi dané, do týchto skupín:

- *Position tracking*

  Robot pozná svojú počiatočnú polohu. Jeho úlohou je iba vysporiadať sa s chybami odometrie. Nepresnosť je iba lokálna a po celý čas sústredená okolo skutočnej polohy. Može byť vyjadrená unimodálnou Gaussovou distibúciou.

- *Global localization*

  Robot nepozná svojú počiatočnú polohu. Belief robota o svojej pozícii nie na začiatku možne vyjadriť unimodálnou distribúciou, čo znamená že tento porblém nie je možne riešiť Kalmanovými filtrami, vyžadujucími unimodálny inicializačný belief. Particle filtre a ich vzorková reprezentácia belief schopná aproximovať ľubovoľnú distribúciu je populárne metóda na riešenie tohto porblému.

- *Kidnapped robot problem*

  Zahŕňa globálnu lokalizáciu a naviac robot môže byž kedykoľvek nesvojvoľne premiestnený na iné mieto. Jemne modifikovaný particle filter je shopný riešiť túto úlohu. Stačí nepatrnú časť zo všetkých vzoriek po každom resamplovaní roztrúsiť na náhodných pozíciach. Ak bol robot loklizovaný a následne premiesntený, zhluk vzoriek v okolí lokalizovného miesta postupne zanikne a lokalizácia beží akoby od začiatku.

Algorithm 8 MCL

Input: S_{t-1}, z_t, u_t, m

Output: S_t

1: $S_t^- = \emptyset$
2: **for** i = 1 **to** N **do**
3: sample $x_t^i \sim p(x_t \mid u_t, x_{t-1}^i, m)$
4: $w_t^i = p(z_t \mid x_t^i, m)$
5: $S_t^- = S_t^- \cup \langle x_t^i, w_t^i \rangle$
6: **end for**
7: $S_t = \emptyset$
8: **for** i = 1 **to** N **do**
9: draw k with probability $\propto w_t^k$
10: $S_t = S_t \cup \langle x_t^k, w_t^k \rangle$
11: **end for**

2.5.1 Monte Carlo Localization

Lokalizácia Monte Carlo *(MCL)* je algoritmus zo skupiny *Markov localization* algoritmov. Pravdepodobnostnú distribúciu náhodnej premennej vyjadrujúcej pozíciu robota reprezentuje množiny váhovaných vzoriek. Po vykonaní pohybu a merania sa vzorky updatujú použitím algoritmu *particle filter* 2 s tou zmenou, že v príslušnom modeli pohybu a modeli snímania vystupuje aj mapa prostredia.

$$p(x_t \mid x_{t-1}, u_t, m) \tag{2.16}$$

$$p(z_t \mid x_t, m) \tag{2.17}$$

Podľa toho, či sa jedná o feature-based alebo location-based mapu, je potrebné prispôsobiť oba modeli. Viac reprezentácia mapy algoritmus nijako neovplyvňuje.

Keďže vzorkovou reprezentáciou je možne aproximovať ľubovoľnu pravdepodobnostnú ditribúciu, je MCL schopná riešiť globálnu loklizáciu a s malými úpravami aj kidnapped robot problém. Rozdiel medzi MCL pre position tracking a globálnu lokalizáciu je iba v inicializovaní vzoriek. Pri position trackingu sú pozície všetkých vzoriek inicializované v malom okolí skutočnej vopred známej pozície. Úlohou loklizácie je iba vysporiadať sa s nepresnosťami pohybu, za pomoci senzorových dát. Vykonávaním pohybov sa zhluk vzoriek rozptyľuje v

rôznych smeroch podľa modelu pohybu 5. Ak by sme ignorovali výstupy senzorov, vzorky by sa po niekoľkých krokoch roztrúsili na všetky možné voľné pozície. Spracovanie senzorových dát sa však spravuje váhy vzoriek tak, aby vzorky na nepravdepodobných pozíciach zanikali. Vhodný počet vzoriek je pre pasition tracking nižší ako pre globálnu lokalizáciu a pre globálnu loklizáciu nižší ako pre kidnapped robot problem. Návody ako odhadnúť vhodný počet vzoriek za behu možno nájsť v [6], [17].

2.6 Simultánna lokalizácia a mapovanie

Problém *simultánnej lokalizácia a mapovania (SLAM)* kladie otázku, či je možné, aby mobilný robot nasadený do neznámeho prostredia na neznámu pozíciu postupne vytváral mapu konzistentnú s prostredím a zároveň určoval svoju pozíciu vzhľadom na túto mapu. [4]. Zložitosť SLAM problému vyplýva z vysokorozmerného stavového priestoru zloženého z premenných popisujúcich pozíciu robota a premenných popisujúcich stav prostredia. Faktory, ktoré najviac ovplyvňujú zložitosť mapovania sú:

- veľkosť prostredia
- presnosť pohybov a snímania
- štruktúra prostredia (cykly, podobajúce sa regióny, ...)

Formálny pravdepodobnostný popis problému simultánnej lokalizácie a mapovania je určenie posteriórnej pravdepodobnostnej ditribúcie

$$bel(x_{1:t}, m) = p(x_{1:t},\ m \mid u_{1:t},\ z_{1:t}) \tag{2.18}$$

Hlavné kritéria podľa ktorých sa rozdeľujú existujúce riešenia SLAMu sú:

- *Reprezentácia mapy prostredia*
  - feature-based mapy
  - location-based mapy
- *Algoritmus použitý na výpočet belief*
  - Kalmanov filter a jeho derivácie
  - Rao-Blackwellized particle filter

2.6.1 FastSLAM

FatSLAM je skupina metód pre problém SLAMu, ktoré sú založené na Rao-Blackwellized particle filtri. Prvý algoritmus z tejto skupiny bol uvedený v práci [11] v roku 2002 a znamenal výrazný posun v pravdepodobnostných riešeniach SLAMu, keďže ako prvý dokázal reprezentovať nelineárny model pohybu a negaussovskú distribúciu pozície.

Vysokorozmerný stavový priestor SLAMu robí základnu verziu particle filtra, v ktorom by každá častica obsahovala jednú možnú polohu robota a jednú možnú mapu prostredia, nepoužiteľnou. Redukciu množstva potrebných voriek je možné dosiahnuť aplikovaním *Rao–Blackwell teorémy*, podľa ktorej je združená pravdepodobnosť $p(x1, x2)$ rozložená na súčin použitím *product rule*

$$p(x1, x2) = p(x2 \mid x1) * p(x1) \tag{2.19}$$

Ak je možné vyjadriť $p(x2 \mid x1)$ analitický, potrebne je samplovať iba $p(x1)$. Aplikovaním Rao-Blackwellized teorémy na problém SLAMu 2.18, dostávame

$$p(x_{1:t},\ m \mid u_{1:t}, z_{1:t}) = p(x_{1:t} \mid u_{1:t},\ z_{1:t}) * p(m \mid x_{1:t}\ , z_{1:t}) \tag{2.20}$$

Trajektória je reprezentovaná váhovanými vzorkami a updatovaná particle filtrom. Mapa prostredia je počítaná analiticky za predpokladu, že trajektória robota je korektná. Každá vzorka tak obsahuje vlastnú mapu a zároveň každá mapa je budovaná podľa trajektórie príslušnej vzorky. V prípade mapovania rozsiahlého priestoru použitím tejto základnej verzie FastSLAMu s pamäťovo náročnými gridmapami, by bolo nutné prispôsobiť, čo znamená zredukovať, množstvo vzoriek tak, aby celkové pamäťové nároky implementácie ostali v rozumných hraniciach. Ak je robot vybavený laserovými senzormi, ktorých presnoť je pomerne vysoká, použitím techniky *scan matching* je možné veľmi presne identifikovať korektné hypotézy pozície a vysoký počet vzoriek v tom prípade nie je nutný. My sa zameráme na prípad SLAMu pre roboty vybavené sonárnymi senzormi, ktorých presnoť je oproti laserovým senzormi neporovnateľne nižšia. Laserovým implementáciam stačí jedno zosnímanie okolia a jeho porovnanie s mapou, ktoré pri sonárnych implementáciach nevedie k dobrým výsledkom. Prvá práca, ktorá venovala riešeniu SLAM problému pre systémy vybavené sonárnymi senzormi, bola [18]. V tejto práci bol zároveň navrhnutý spôsob, ako sharovať medzi viacerými vzorkami jednu a tú istú gridmapu, resp. jej časť. Pri mapovaní rozsiahlého priestoru sa očakávajú aj veľké rozmery generovanej gridmapy. Vieme, že pri resamplovaní dochádza ku klonovaniu vzoriek s vysokými váhami.

Takto vznikajú viacere identické vzorky, z ktorých každá obsahuje kópiu mapy spoločnej rodičovskej vzorky. V ďalších krokoch sa v dôsledku stochastického modelu pohybu tieto mapy začnú líšiť, no zmeny sú iba lokálne, postihujúce iba malé úseky gridmapy. V práci [18] je navrhnutý spôsob sharovania mapy jej rozdelením do blokov konštantnej veľkosti a zaznamenávať zmeny každej vzorky na konkrétnych blokoch. Ďalsšie podobné spôsoby sharovania sú uvedené v prácach [5], [7].

2.6.2 Generovanie gridmapy zo sonárnych dát

Keďže sme sa vybrali za riešením SLAMu po ceste gridmáp a ultrazvukových senzorov, je potrebné nájsť spôsob, ako ich navzájom prepojiť. Budeme sa zaoberať reprezentáciou a aktualizovaním distribúcie

$$p(m \mid x_{1:t}, z_{1:t}) \tag{2.21}$$

ktorá vyjadruje pravdepodobnosť že mapa je v stave m, ak sme na pozíciach $x_{1:t} = \{x_1, x_2, ..., x_t\}$ dostali zo senzorov dáta $z_{1:t} = \{z_1, z_2, ..., z_t\}$. Ak príjmeme predpoklad, že stavy obsadenosti jednotlivych buniek m_i v gridmape $m = \{m_1, m_2, ..., m_M\}$ sú nezávisle, môžme vzťah 2.22 dekomponovať na súčin

$$p(m \mid x_{1:t}, z_{1:t}) = \prod_{i=1}^{M} p(m_i \mid x_{1:t}, z_{1:t}) \tag{2.22}$$

kde m_i je binárna náhodná premenná vyjadrujúca obsadenoť i-tej bunky gridmapy m. Tú je niekedy vhodnejšie vyjadriť v tzv. *log-odds forme*

$$l_{t,i} = \log \frac{p(m_i \mid x_{1:t}, z_{1:t})}{1 - p(m_i \mid x_{1:t}, z_{1:t})} \tag{2.23}$$

Rekonštrukcia hodnoty $p(m_i \mid x_{1:t}, z_{1:t})$ z log-odd formy je

$$p(m_i \mid x_{1:t}, z_{1:t}) = 1 - \frac{1}{1 + \exp(l_{t,i})} \tag{2.24}$$

Ak predpokladáme, že na začiatku mapovania je na každej bunke rovnako pravdepodobný stav *obsadený* a stav *voľný*, odvodíme inicializačné hodnoty pre všetky bunky

$$l_{0,i} = \log \frac{0.5}{1 - 0.5} = 0; \ \forall i = 1..M \tag{2.25}$$

Algorithm 9 Grid Mapping

Input: $\{l_{t-1,i}\}$, x_t, z_t

Output: $\{l_{t,i}\}$

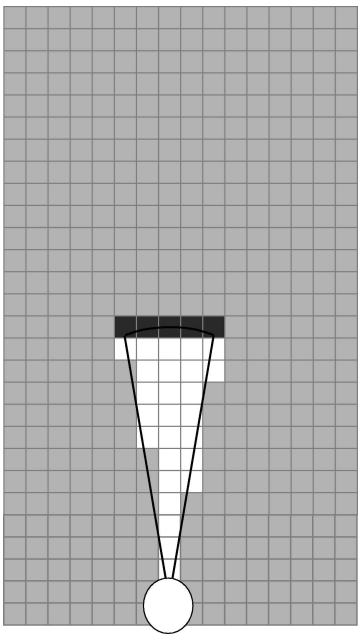
```
for i = 1 to M do
    if i in perceptual field of z_t then
        l_{t,i} = l_{t-1,i} + InverseMeasurementModel(m_i, x_t, z_t)
    else
        l_{t,i} = l_{t-1,i}
    end if
end for
```

Všeobecná schéma algoritmov updatujúcich hodnoty buniek z prichádzajúcich meraní na nových pozíciach je algoritmus *Grid Mapping*. Jeho myšlienka je jednoduchá. Bunky, ktoré sú mimo apertúry senzora si nevšíma. Bunkám, ktoré je možné senzorom zosnímať, zníži alebo zvýši hodnotu podľa výsledku podprocedúry *InverseMeasurementModel*. Tá implementuje inverzný model snímania v log-odds forme

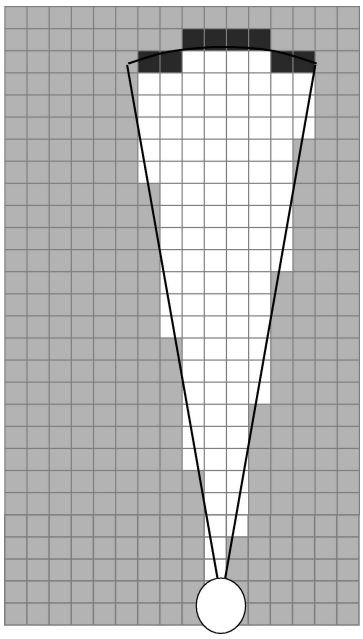
$$InverseMeasurementModel(m_i,\ x_t,\ z_t) = \log \frac{p(m_i\ |x_i, z_i)}{1 - p(m_i\ |x_i, z_i} \tag{2.26}$$

2.6.2.1 Inverzný model snímania

Práve inverzný model snímania je to, pre čo neexistuje všeobecný spôsob ako ho implementovať, ktorý by dobre fungoval s ľubovoľným typom senzorov. Jedna z takýchto všeobecných metód, ktorá by mohla po vhodnom naladení parametrov fungovať na laserových senzoroch je znázornená na obrázkoch 2.3, 2.4. Z nich je možné dostatočne pochopiť myšlienku metódy, jej pseudokód je možne nájsť v [20]. Jej základom je ohodnotenie buniek vo vzdialenosti zosnímanej senzorom konštantou z intervalu $(0.5; 1)$. Vyššia hodnota tejto konštanty znamená našu väčšiu dôveru v to, že na nameranej vzdialenosti sa skutočne nachádza nejaký objekt. Bunkám vo vnútri apertúry je priradená konštanta z intervalu $(0; 0.5)$. Nižšia hodnota znamená väčšiu dôveru v to, senzor nenameral nižšiu hodnotu preto, lebo sa na príslušnych miestach žiaden objekt nenachádza, a nie preto, že sa nachádza ale senzor pochybil.



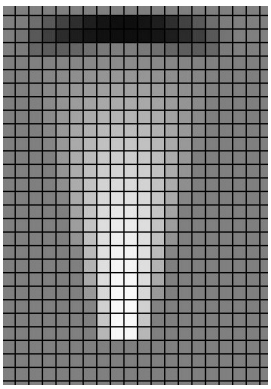
Obr. 2.3: Inverse Mesurement Model A1



Obr. 2.4: Inverse Mesurement Model A2

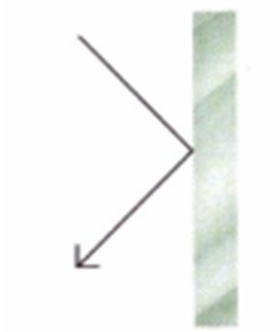
Ďalší spôsob, ktorý môžme chápať ako zovšeobecnenie predchádzajúceho, pochádza z prác [12] a [13]. Venujú sa problému ako podľa sonárnych meraní klasifikovať bunky gridmapy na prázdné, obsadené, alebo neznáme. Podobne ako v predchádzajúcej technike, aj v tejto sa bunky pred senzorom rozdelia na okrajové a vnútorné. Pre bunky z každej z týchto skupín sa určí pravdepodobosť stavu, že bunka je obsadená alebo prázdna, podľa rôznych funkcii. Ich predpisy su navrhnuté tak, aby pravdepodonosť obsadenosti buniek vyskytujúcich sa na osi senzora v namerane vzdialenosti bola navyššia a so stúpajúcim uhlom buniek od senzora klesala. Funkcia, ktorá mení bunky vnúterného regiónu a vyjadruje presvedčenie o neexistencii objektu na danom mieste, klesá so vzdialenosťou a uhlom bunky od senzora. Lokálna gridmapa vytvorená touto metódou z jednej senzorom nameranej hodnoty by mohla vyzerať zhruba ako na obrázku 2.5.

V práci je zároveň popísaný spôsob, ako lokálne gridmapy vytvorené z jednotlivých meraní zakomponovať do výslednej globálnej gridmapy a zároveň získať hodnotu vyjadrujúcu kvalitu prekrytia. Oproti prvej spomínanej metóde je táto výpočtovo náročnejšia, no lepšie reprezentuje snímanie sonárnymi senzormi.



Obr. 2.5: Iverse measurement model B

Tretia technika mapovania gridmáp, ktorú spomenieme, pochádza z práce [8]. Jej základom je zovšeobecnenie gridmáp na tzv. *response gridmapy*, ktoré sú vhodnejšie pre mapovanie prostredia s častým javom *specular reflection* 2.6. V dôsledku neho sa stáva, že hoci je prekážka v apertúre senzora v primeranej vzdialenosti, senzor ju nezachytí kvôli uhlu, ktorý zviera k nemu natočený povrch prekážky s osou senzora. Vyslaný impulz sa od povrchu odrazí v opačnom smere a echo od prekážky sa nevráti späť k senzoru. Objekt sa tak v istom smere javí ako neviditeľný, teda pre robota neexistujúci.



Obr. 2.6: Specular reflection

Rozdiel medzi klasickou gridmapu a response gridmapou je nasledovný. O bunkách klasickej gridmapy sa predpokladalo, že ak sú obsadené, odrážaju ipulzy v každom smere; ak sú voľné, impulz nevráti. Bunky response gridmapy majú ďalší rozmer, ktorým je smer. Jedna bunka sa môže v istom smere javiť voľná, v inom smere môže vracať echo k senzoru (odpovedá). Bunku gridmapy kategorizujeme ako obsadenú, ak odpovedá aspoň v jednom smere. Počet sme-

rov, na koľko je vhodné particiovať bunky, je ĺubovoľný. Zvyčajne nadobúda hodnoty od 2 po 8. Autori práce [3] ju dokonca zafixovali na hodnotu rovnakú, ako je počet senzorov robota.

Matematicke vyjadrenie modelu response gridmáp je nasledovné. Obsadenosť bunky na pozícii (i, j) vyjadruje binárna stavová premenná

$$Occ(i, j) = \{occupied,\ unoccupied\} \tag{2.27}$$

Odpovedanie bunky na pozícii (i, j) v smere ϕ vyjadruje binárna stavová premenná

$$Res(i, j, \phi) = \{response,\ no\ response\} \tag{2.28}$$

Obory hodnôt indexov sú:

- $i \in \{1, 2, ..., M_r\}$, kde M_r je počet riadkov gridmapy
- $j \in \{1, 2, ..., M_c\}$, kde M_c je počet stĺpcov gridmapy
- $\phi \in \{1, 2, ..., \Phi\}$, kde Φ je zvolený počet smerov

Poznamenajme, že predpokladáme nezávisloť buniek rovnako ako v gridmapách. Cieľom je teraz ukázať spôsob ako odvodiť pravdepodobnoť udalosti

$$Occ(i, j) = occupied \tag{2.29}$$

Takáto udaloť nastáva vtedy, ak bunka na pozícii (i, j) odpovedá aspoň v jednom smere, t.j.

$$(Occ(i, j) = occupied) \Leftrightarrow ((Res(i, j, 1) \vee (Res(i, j, 2) \vee ... \vee (Res(i, j, \Phi)) \tag{2.30}$$

Negáciou výroku, "Bunka (i, j) je okupovaná.", je výrok "Bunka (i, j) neodpovedá v žiadnom smere ϕ." Pravdpodobnosť spĺnenia prvého výroku preto získame ako rozdiel

$$p(Occ(i, j)) = 1 - \prod_{\phi=1}^{\Phi} 1 - p(Res(i, j, \phi)) \tag{2.31}$$

Tento vzťah poskytuje návod, ako v každom kroku z existujúcich hodnôt $Res(i, j, \phi)$ odvodiť $Occ(i, j)$. Pri absencii akýchkoľvek počiatočných informácii o gridmape, ju inicializujeme tak, aby stavy *occupied* a *unocupied* premennej *Occ* boli v každej rovnako pravdepodobné, čo znamená

$$p(Occ(i, j)) = 0.5 \tag{2.32}$$

Inicializačné hodnoty pravdepodobností premennej Res sú preto

$$p(Res(i,j,\phi)) = 1 - (0.5)^{\frac{1}{\Phi}} \tag{2.33}$$

Aktualizácia všetkých hodnôt $p(Res(i,j,\phi))$ po prijatí merania z na pozícii robota x funguje podľa schémy Bayesovho pravidla

$$p(Res(i,j,\phi)|z,x) = \frac{p(z|(Res(i,j,\phi),x) * p((Res(i,j,\phi)|x)}{p(z|x)} \tag{2.34}$$

Od tohto miesta sa vzdialime práci [8], z ktorej si odnášame myšlienku pridania tretieho rozmeru bunkám. Dôvodom je v nej uvedený model pohybu $p(z|(Res(i,j,\phi),x)$, javiaci sa akoby bol vytvorený a nevysvetliteľnými konštantami doladený pre jedno použitie.

Odvodenie vlastnej metódy pre aktualizáciu pravdepdobnosti ($Res(i,j,\phi)$ pre bunky ležiace v apertúre senzora začneme v regióne buniek, ktorých vzdialenosť od senzora je rovnaká ako nameraná vzdialenosť. Analýzu pre bunky s vzdialenejšie a bližšie ako nameraná vzdialenosť uvedieme neskôr.

Majme teda ľubovoľnú bunku, ktorej vzdialenosť od senzora odpovedá nameranej vzdialenosti. Odvodenie vzťahu pre update jej stavovej premennej $Res(i,j,\phi)$ z čerstvých dát x, z začneme rovnako Bayesovým pravidlom. Kvôli lepšej prehľadnosti dlhých výrazov, ktoré nas čakajú, spravíme najprv premenovania

$$R(i,j,\phi) := Res(i,j,\phi) \tag{2.35}$$

$$P := p(R(i,j,\phi|\ x,z) \tag{2.36}$$

Bayesovo pravidlo nasadené na náš problém potom vyzerá

$$P = \frac{p(z|(R(i,j,\phi),x) * p((R(i,j,\phi)|x)}{p(z|x)} \tag{2.37}$$

Prvé zjednodušenie dostávame, ak si uvedomíme že starý stav ($R(i,j,\phi)$ nezávisi od novej polohy robota x, čo znamená

$$p((R(i,j,\phi)|x) = p((R(i,j,\phi)) \tag{2.38}$$

Spojením 2.37 a 2.38 dostávame

$$P = \frac{p(z|(R(i,j,\phi),x) * p((R(i,j,\phi))}{p(z|x)} \tag{2.39}$$

Teraz marginalizujeme udalosť namerania dát z na pozícii x cez možné hodnoty $R(i,j,\phi)$

$$p(z|x) = p(z|x,R(i,j,\phi)) * p(R(i,j,\phi)) + p(z|x,\neg R(i,j,\phi)) * p(\neg R(i,j,\phi)) \tag{2.40}$$

Spojením 2.39 a 2.40 dostávame

$$P = \frac{p(z|(R(i,j,\phi),x) * p(R(i,j,\phi))}{p(z|x,R(i,j,\phi)) * p(R(i,j,\phi)) + p(z|x,\neg R(i,j,\phi)) * p(\neg R(i,j,\phi))} \tag{2.41}$$

Výraz $p(z|(R(i,j,\phi),x)$ vyjadruje pravdepodonosť namerania konkrétnej hodnoty z, za predpokladu, že pozícia robota je x a bunka na pozícii (i,j) odpovedá v smere ϕ . Senzor vráti hodnotu z jedine vtedy, ak žiadna z buniek ležiacich medzi danou bunkou a senzorom v žiadnom zo svojích smerov neodpovedá. Množinu takých buniek, ktorých vzdialenosť od senzora je menšia ako z, označíme IN_z a pravdepodobnosť toho, že žiadna z nich v žiadnom smere neodpovedá $P_{\neg IN_z}$ Z toho, čo sme povedali, dostávame

$$P_{\neg IN_z} := p(z|(R(i,j,\phi),x) = \prod_{(i,j,\phi)\in IN} \neg R(i,j,\phi) \tag{2.42}$$

Dosadením 2.42 do 2.41 dostávame

$$P = \frac{P_{\neg IN_z} * p(R(i,j,\phi))}{P_{\neg IN_z} * p(R(i,j,\phi)) + p(z|x,\neg R(i,j,\phi)) * p(\neg R(i,j,\phi))} \tag{2.43}$$

Ďalej sa zameriame na podvýraz $p(z|x,\neg R(i,j,\phi))$ vyjadrujúci pravdepodobnosť toho, že namerame hodnotu z hoci jedna z buniek v tejto vzdialenosti v smere ϕ určite neodpovedá. Aby senzor mohol namerať hodnotu z musela preto odpovedať aspoň jedna z ďalších buniek v rovnakej vzdialenosti od senzora ako riešená neodpovedajúca bunka. Navyše muselo súčastne platiť, že žiadna z buniek množiny IN_z nemohla znova odpovedať v žiadnom smere. Množinu buniek ležiacich vo vzdialenosti z od senzora s výnimkou bunky (i,j,ϕ) označíme $C_z^{i,j,\phi}$ a pravdepodobnosť toho, že niektorá z nich v niektorom smere odpovedá $P_{C_z^{i,j,\phi}}$

$$p(z|x,\neg R(i,j,\phi)) = P_{C_z^{i,j,\phi}} * P_{\neg IN_z} \tag{2.44}$$

Dosadením 2.44 do 2.43 a zopár jednoduchými úpravami dostávame

$$P = \frac{P_{\neg IN_z} * p(R(i,j,\phi))}{P_{\neg IN_z} * p(R(i,j,\phi)) + P_{C_z^{i,j,\phi}} * P_{\neg IN_z} * p(\neg R(i,j,\phi))} \tag{2.45}$$

$$P = \frac{p(R(i,j,\phi))}{p(R(i,j,\phi)) + P_{C_z^{i,j,\phi}} * (1 - p(R(i,j,\phi)))} \tag{2.46}$$

$$P = \frac{1}{1 + \frac{P_{C_z^{i,j,\phi}}}{p(R(i,j,\phi))} - P_{C_z^{i,j,\phi}}} \tag{2.47}$$

$$P_{C_z^{i,j,\phi}} = 1 - \prod_{(i,j,\phi) \in C_z^{i,j,\phi}} \neg R(i,j,\phi) = 1 - \prod_{(i,j,\phi) \in C_z^{i,j,\phi}} (1 - R(i,j,\phi)) \tag{2.48}$$

Vzťahy 2.47 a 2.48 nám dávajú návod, ako aktualizovať stavovú premennú *Res* buniek, ktorých vzdialenosť od senzora, je priblížne rovnaká ako nameraná hodnota. Bunky s väčšou vzdialenosťou sme sa rozhodli ponechať na starých hodnotách, keďže o tom či odpovedajú alebo nie, nemôžeme nič tvrdiť, medzi nimi a senzorom leží objekt, ktorý ich cloní. Pravdepodobnosť buniek s menošou vzdialenosťou sme sa rozhodli iba penalizovať nejakou konštantou z intervalu $(0, 1)$.

2.6.3 Odometry-Sonar-ResGrid FastSLAM

Názvom *Odometry-Sonar-Grid FastSLAM, (O-S-G FastSLAM)* označíme techniku riešenia SLAM problému, ktorá

- pohybové dáta získava z *odometrie*
- senzorové dáta získava iba zo *sonárnych* senzorov
- buduje *response gridmapu* prostredia
- patrí do skupiny *FastSLAM* algoritmov, čo znamená, že jej základom je Rao-Blackwellized particle filter

$$p(x_{1:t},\ m \mid u_{1:t}, z_{1:t}) = p(x_{1:t} \mid u_{1:t},\ z_{1:t}) * p(m \mid x_{1:t}\ , z_{1:t}) \tag{2.49}$$

Belief pozície je reprezentovaný váhovanými vzorkami 2.50

$$S_t = \{(x^i_{1:t},\ p(m^i \mid x^i_{1:t},\ z_{1:t}),\ w^i_t) \mid i = 1..N\} \tag{2.50}$$

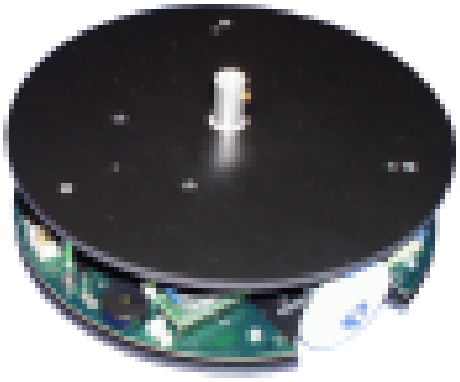
z ktorých každá obahujuje možnú trajektóriu robota a vlastnú response gridmapu prostredia. Model pohybu je možné implementovať podľa *sample motion model* 2.3.1. Pre zmeny váh vzoriek nie je nutné implemntovať model snímania, tak sme to spravili pri Monte Carlo lokalizácii. Kvalitu vzoriek možno určiť aj tým, že ohodnotíme kvalitu prekryvu novej gridmapy s jej verziou, v ktorej

bola pred zahrnutím posledného merania. Čím sa mapy viac podobajú, tým menej prekvapení prinieslo posledné meranie, tým viac odpovedá odhadovaná pozícia robota a pozícia mapy skutočnosti. Resamplovanie a určenie stavu kedy je potrebné funguje presne ako sme ho popísali v predchadzajúcich kapitolách.

Kapitola 3

Experimenty

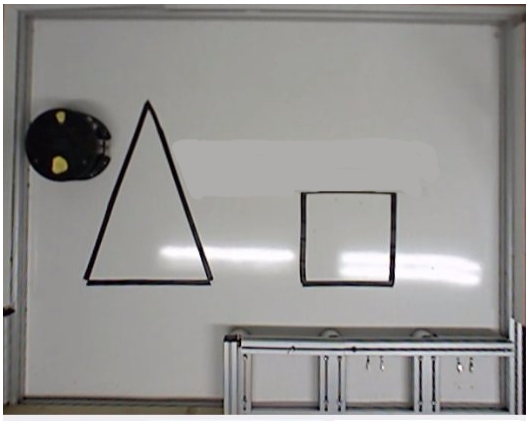
3.1 Platforma a prostredie



Obr. 3.1: Robotnačka

Platforma použitá pre odskúšanie implementácii pravdepodobnostných lokalizačných a mapovacích algoritmov bol mobilný robot *robotnačka* [22], ktorý je súčasťou robotické laboratória *Virtual Robotic Lab* [15]. Aby sme mohli sledovať a zaznamenávať dianie robota, obmedzili sme jeho svet na stôl laboratória snímaný kamerami, ktorého rozmery sú 1200 a 1500 milimetrov. Vzhľadom na priemer robotnačky, ktorý je 210 milimetrov, bol tento jej svet pomerne malý. Experimenty boli zamerané na algoritmy odvodené od particle filtra a jeho vzorkovej reprezentácie pravdepodobnostných distribúcii. Úspešnosť a nároky týchto metód (pri fixnom stupni požadovanej presnosti) výrazne závisia od veľkosti prostredia, preto nie rozsiahle prostredie virtuallabu bolo pre naše účely úplne

postačujúce. Nami používaná robotnačka zdieľa stôl s ďalšou pohybu schopnou robotnačkou. Tá bola počas experimentov nepoužívaná a prostredie sme preto uvažovali ako statické.



Obr. 3.2: Virtual Robotic Lab

Lokalizačná aj mapovacia aplikácia, ktoré sme vytvorili, môžu byť použité aj s iným typom robota ako robotnačka. Použitý robot musí spĺňať všetky z nasledujúcich požiadaviek:

- *mobilita*

  Použitý robot musí byť schopný pohybovať sa prostredím a poskytovať dáta o vykonaných pohyboch.

- *snímanie*

  Robot musí disponovať aspoň jedným senzorom na meranie vzdialeností a vedieť poskytovať namerané hodnoty.

- *komunikácia*

  S robotom sa musí dať komunikovať prostredníctvom Java API. Komunikovať pre nás znamená vysielať mu pohybové príkazy a príjimať dáta zo senzorov.

Jediným modulom našej aplikácie, ktorý je priamo závislý od použitého robota je trieda `RobotManager`. Jej inštancia zabezpečuje nadviazanie spojenia a následnú komunikáciu s robotom. Ostatné objekty aplikácie môžu ovládať robota volaním jej verejnej metódy `act(action, stepSize, dataExpectant)`, ktorá zabezpečí, že príslušný robot vykoná akciu `action` s veľkosťou kroku

`stepSize`. Ak vykonanie tejto akcie a následné zosnímanie okolia senzormi prebehne v poriadku, bude zavolaná príslušná metóda objektu `dataExpectant` s argumentami, ktorými budú odometrické a senzorové dáta. Povolené akcie sú `forward`, `left`, `right`. API robotnačky dáva odometrické dáta a očakáva hodnotu akcie, ktorú ma vykonať, v jej špecifickej jednotke, ktorou je *počet krokov*. Za uniformnu jednotku dĺžky sme v celej aplikácii zvolili milimetre. Mierou uhlov boli radiány. Príslušnú konverziu medzi týmito dvoma jednotkami zabezpečuje tiež výhradne táto trieda. V prípade použitia iného robota (prípadne iného API robotnačky) je táto trieda jediná, ktorú je potrebné prispôsobiť prepísaním a prekompilovaním a preto sme ju na tomto mieste popísali. Ďalšie moduly aplikácie budú spomenuté v nasledujúcich kapitolách. Dokumentáciu použitého Java API robotnačky, je možné nájsť na adrese `http://webcvs.robotika.sk/cgi-bin/cvsweb/~checkout~/robotika/robot/java/doc/index.html`.

3.2 Pohyb

Pohyb robotnačky zabezpečujú dva krokové motory *Microcon SX 17* a voľné oporné koliesko kvôli stabilite. Aplikácia komunikujúca s robotnačkou cez jej API získava na požiadanie jej pohybové dáta získané odometriou. Tieto dáta sú vo forme dvojice

$$(u_x, u_y) \tag{3.1}$$

kde u_x je celkový počet krokov prejdených od (re)štartu odometrie v smere kolmom na počiatočný smer robotnačky. Podobne u_y je celkový počet krokov prejdených od (re)štartu odometrie v smere vodrovnom k počiatočnému smeru robotnačky.

Model pohybu robotnačky sme implementovali podľa schémy *sample motion model* 5. Nakoľko ten očakáva na vstupe dáta o pohybe vo forme trojice

$$u = (\Delta x, \Delta y, \Delta h) \tag{3.2}$$

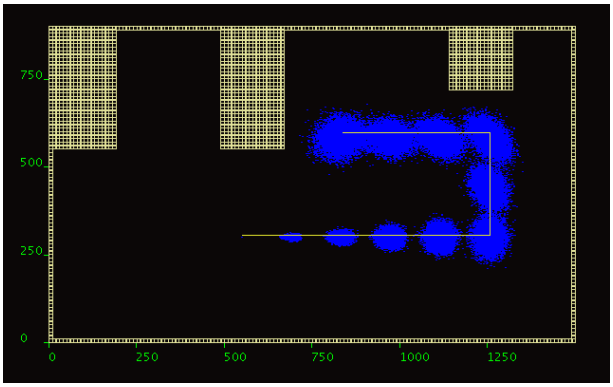
bolo potrebné do tejto podoby transformovať odometrické dáta 3.1 . Zistiť zmeny $\Delta x, \Delta y$ bolo triviálne, odhaliť zmenu natočenia pomohla zabudovaná funkcia `atan2`. Kvôli tejto transformácii sme museli zakázať robotnačiu akciu `backward`. Ak by bolo totiž cúvanie povolené, nebolo by z odometrických dát možné jednoznačne určiť, či robotnačka cúvla a natočenie nezmenila, alebo sa otočila o pravý uhol a šla dopredu. Po dvoch takýchto rôznych postupnostiach

akcii máme rovnaké dáta (u_x, u_y), no dve možnosti pre príslušné $(\Delta x, \Delta y, \Delta h)$. Druhým obmedzením, ktoré si táto transformácia vyžiadala, bolo zakázanie akcie s nulovou dĺžkou kroku. Po nej nie je možné z dát (u_x, u_y), ktoré sa samozrejme v poslednom kroku nijako nezmenili, odvodiť zmenu natočenia robota. Zmenu natočenia nie je možné zistiť z týchto dát ani ak bol robot pootočený o nenulový počet krokov, preto pohybovú akciu považujeme za kompletnu až vtedy ak sa zmenili (u_x, u_y). Kým sa aspoň jedna zložka tejto dvojice nezmení `RobotManager` neposkytuje žiadne dáta (ani senzorové, nakoľko sú zbytočné ak nevieme, či sa robot otáčal alebo nie).

Použitý formálny model pohybu 5 bolo potrebné vhodným nastavením parametrov α_i čo najlepšie zosúladiť s pohybom robotnačky. Ako sme uvádzali v kapitole 2.3.1, je hodnoty týchto parametrov možné vydolovať algoritmicky z reálneho datasetu alebo nastaviť "od oka" pozorovaním chýb, ktoré pri pohybe robota vznikajú. My sme sa rozhodli pre druhú možnosť. Po mnohých pokusoch s rôznymi hodnotami sme sa sa ustálili na hodnotách uvedených v tabuľke 3.2.

α_1	0.005
α_2	0.0005
α_3	0.2
α_4	0.5

Tabuľka 3.1: Parametre modelu pohybu



Obr. 3.3: Chyby pohybu

Na obrázku 3.3 sme znázornili, čo by sa dialo so vzorkami inicializovanými na rovnakej pozícii, ak by sme sa robotom pokúšali prejsť po plnej čiare a po každom kroku by pozície vozriek predikoval model pohybu s parametrami z tabuľky 3.2 Korekcia vzoriek v tomto príklade zanedbaná. Hodnoty parametrov modelu je možné v aplikácii prenastavovať za jej behu alebo editáciou príslušného konfiguračného súboru.

3.3 Snímanie

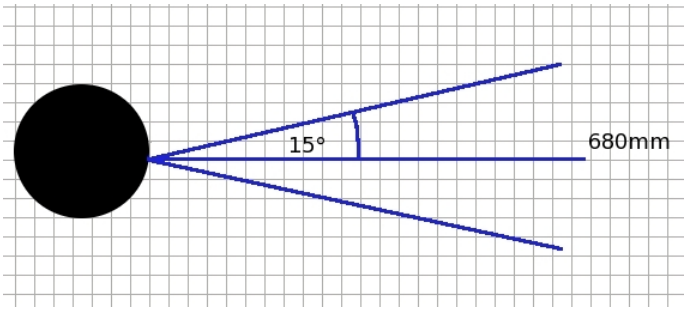
Získavanie informácii o okolí robota zabezpečujú *ultrazvukové senzory*. Ultrazuvkové senzory *(sonáre)* sú senzory na meranie vzdialeností *(prosimity sensors)*, ktoré určujú relatívnu vzdialenosť objektu prostredia od senzora. Ich princíp spočíva vo vyslaní zvukového signálu a meraniu času, kým sa od objektu odrazeny zvukový signál vráti späť k senzoru. [16]

Robotna4ka je vybavená piatimi sonármi modelu *SRF08* 3.4.

Obr. 3.4: Ultrazvukový senzor SRF08

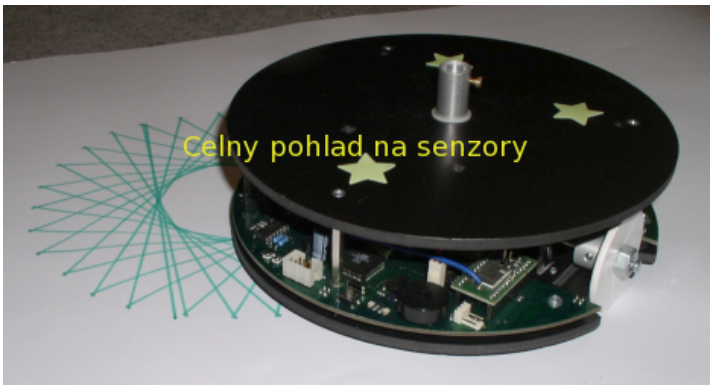
Ten, podľa jeho technickej špecifikácie [1], dokáže merať vzdialenosti od 30mm do 6000mm. Nakoľko sú senzory umiestnené vo výške iba 70mm, zvukový signál sa vo vzdialenosti priblížne 680mm takmer vo všetkých prípadoch odráža od povrchu stola hoci v tomto okolí nie je žiadna prekážka. To znamená, že maximálnou merateľnou vzdialenosťou prekážky od robota bude v naších experimentov práve hodnota 680mm. Uhol *beam aperture*, ktorý musí zvierať prekážka s osou senzora, aby bola viditeľná, závisí od jej vzdialenosti. Pre zjednodušenie sme predpokladali ich nezávisloť a defaultnú hodnotu sme zvolili na 15° rovnako v oboch smeroch od osi senzora.

Senzory sú umiestnené v polkruhu po obvode robotnačky. Jeden senzor je na čele robotnačky, zvyšné 4 sú od neho umiestnené symetricky v oboch smeroch,



Obr. 3.5: Grafický model senzora

pričom ich spojnice so stredom robotnačky zvierajú uhly 45° a 90°.



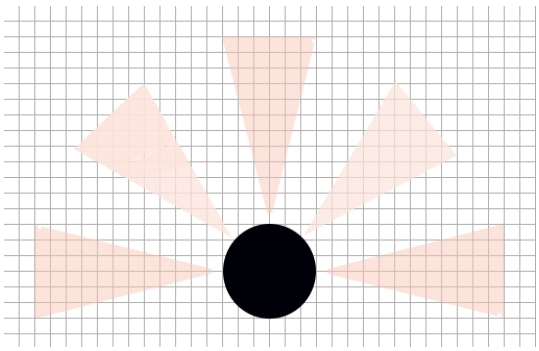
Obr. 3.6: Rozmiestnenie senzorov

Ak uvažujeme konštantnu apertúru 15° a odklon susedných senzorov 45° dostávame z toho záver, že merania senzorov sú nezávisle. 3.7 Tento záver je iba teoretický. V skutočnosti by bolo možne nájsť miesto, v ktorom umiestnená prekážka by bola zachytená oboma susednými senzormi.

Aplikácia komunikujúca s robotnačkou dostáva dáta zo senzorov vždy v podobe pätice

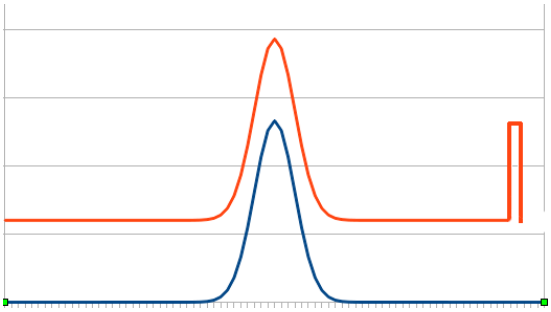
$$z = (z_0, z_1, z_2, z_3, z_4) \tag{3.3}$$

kde z_2 je hodnota v centimetroch z čelného senzora, z_0 je hodnota zo senzora otočeného o 90° v zápornom smere vzhľadom na čelný senzor, atď. Použité senzory sa ukázali byť pomerné presné, ak bola prekážka umiestnená kolmo na os senzora. Problémom však bol jav specular reflection. Ak sa robotnačka pohybovala pozdlž steny, senzor, ktorého os s ňou zvierala uhol 45°, ju nevidel. Tento nedostatok sme sa pri lokalizácii snažili dorovnať vhodným nastavením parametrov.



Obr. 3.7: Grafické znázornenie rozmietnenia senozorov

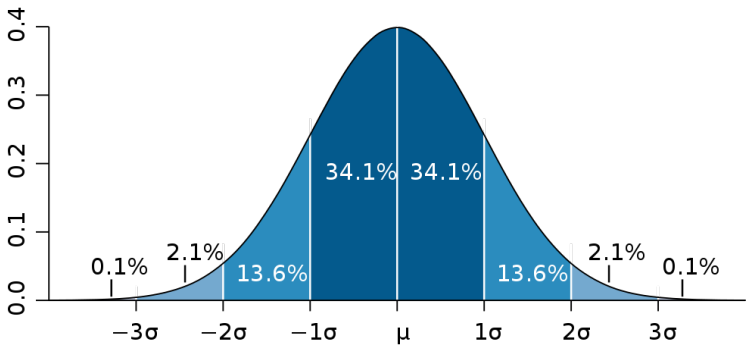
Pri implementácii modelu snímania $p(z \mid x,\ m)$ sme vychádzali z algoritmu *Beam measurement model* 7. Ten je zložený zo štyroch zložiek, z ktorých každá prispieva svojou váhou. V poradí, v akom boli zložky opísané v kapitole 2.4.1, druhá zložka vyjadrená exponenciálnou distribúciou reprezentuje chyby, ktorých príčinou sú objekty v okoli neobsiahnuté v mape. Mapa statického prostredia však obsahuje všetky objekty a keďže naše prostredie je statické, mohli sme túto zložku zanedbať. Tvar výslednej distribúcie zloženej iba z troch zložiek je znázornený na obrázku 3.8 červenou krivkou.



Obr. 3.8: Mesasurement model

Prvým parametrom tohto modelu je štandardná odchýlka σ gaussovskej zložky distibúcie. Z niekoľkých párov nameraných a skutočných vzdialeností sme určili jej hodnotu $\sigma = 20mm$. To znamená, že prehlasujeme, že 68,2 % meraní nie je od skutočnej hodnoty odlišných o viac ako 20mm.

Po odhýlke gaussovskej zložky sú ďalšími parametami modelu váhy jedtlivých zložiek. Na úspešnosť modelu mal veľký vplyv pomer váhy gaussovskej zložky w_{quass} a váhy uniformnej náhodnej zložky w_{uni}. Väčšina prác so zameraním podobným nášmu sa venuje laserovým senzorom. Ich spoľahlivoť je



Obr. 3.9: Diagram štandardnej odchýlky normálneho rozdelenia

s ultrazvukovými neporovnateľné vyššia, náhodné chyby senzora málo časté a preto váha uniformenj distribúcie malá. S nasími ultrazvukovými senzormi kvôli nežiaducému javu specular reflection sme dosiahli uspokojivé výsledky až keď sme náhodnu zložku dostatočne zosilnili.

Pre odskúšanie funkčnosti modelu snímania sme spravili zopár testov, ktorých hlavnou myšlienkou bolo skúmať, či pri fixnej pätici vymyslených nameraných dát, model snímania ohodnotí vzorky tak, že poradie vzoriek zoradených podľa nových hodnôt váh bude zodpovedať naším očakávaniam. Jednu inštancia tohto pokusu je nasledovná.

Prepokladajme, že robotnačka namerala senzorové dáta

$$z = (200, 200, 200, 200, 200) \tag{3.4}$$

Majme päť vzoriek {P1, P2, P3, P4, P5}, na tých pozíciach, že v závislosti od mapy, by pätica ideálnych senzorov mala namerať postupne na pozíciach jednotlivých vzoriek hodnoty:

$$z1 = (210, 210, 210, 210, 210) \tag{3.5}$$

$$z2 = (210, 210, 210, 220, 220) \tag{3.6}$$

$$z3 = (220, 220, 220, 220, 220) \tag{3.7}$$

$$z4 = (220, 220, 220, 240, 240) \tag{3.8}$$

$$z5 = (220, 220, 220, 400, 400) \tag{3.9}$$

Vidíme, že hodnoty sú skonštruované tak, aby prvá vzorka bola na pozícii, v ktorej sa reálne a ideálne meranie líšia v každej hodnote o polovicu štandardnej

odchýlky modelu. Meranie v druhej vzorke sa od reálneho odlišuje v dvoch hodnotách presne o odchýlku a v troch hodnotách o jej polovicu. Pri tretej vzorke je každá hodnota vzdialená od ideálnej presne o odchylku, pri štvrtej vzorke sú dve merania odlišné o jej dvojnásobok, posledné má dve merania odličné o päťnásobok ochýlky. Výstupy algoritmu, t.j. váhy jednotlivých vzoriek sumarizuje tabuľka 3.10. Ak by sme resamplovali vzorky po jednom kroku, percentuálne zastúpenie v novej množine by približne také, aké ukazuje posledný riadok tabuľky. Resamplovanie zvyčajne nenastáva po každom kroku. Váhy vzoriek sa preto v ďalších krokoch môžu zmeniť. Ich hodnoty sa však po každom kroku medzi sebou násobia bez pridávania nejakého zabúdajúceho koeficientu, preto sú všetky merania medzi dvoma resamplovaniami rovnocenné.

P1		P2		P3		P4		P5	
20	0.016802	20	0.016802	40	0.014049	40	0.014049	40	0.014049
20	0.016802	20	0.016802	40	0.014049	40	0.014049	40	0.014049
20	0.016802	20	0.016802	40	0.014049	40	0.014049	40	0.014049
20	0.016802	40	0.014049	40	0.014049	80	0.009350	200	0.008049
20	0.016802	40	0.014049	40	0.014049	80	0.009350	200	0.008000
	41.29%		28.87%		16.88%		7.48%		5.47%

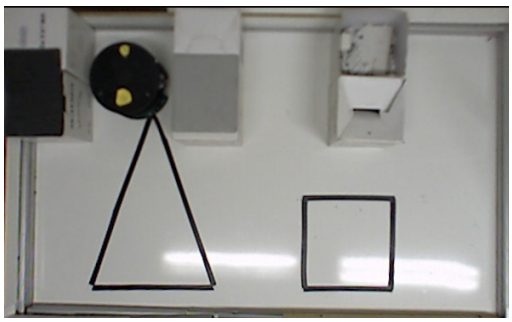
Obr. 3.10: Váhy vzoriek

Po detailnom skúmaní hodnôt váh, sme tento predchádzajúci test vykonali vo väčšom prevedení. Vytvorili sme 5000 vzoriek na náhodných pozíciach a s náhodným natočením. Robotnačku sme umiestnili na ľubovoľné miesto v prostredí a počas pokusu sme s ňou nehýbali. Dáta zo senzorov sme v každom kroku updatovali a na základe nich upravili váhy vzoriek. Po každých 4 krokoch sme množinu vzoriek resamplovali. Cieľom bolo ukázať, že náš model snímania a resammplovanie pozbavia množinu od málo vieryhodných vzoriek.

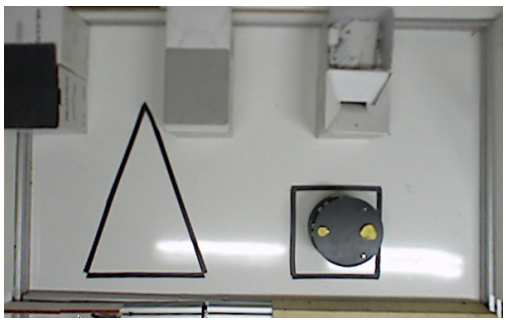
Za prvú testovaciu pozíciu robotnačky sme zovlili úzku "chodbu", ktorých bolo v prostredí 3 a boli rovnaké. Po 12 krokoch a 3 resamplovanich ostali nažive hlavne vzorky vyskytujúce sa v chodbách, čo bol pre nás uspokojivý výsledok. Druhykrát sme robotnačku umiestnili vo voľnom priestore. Po dvoch resamplovaniach zanikli vzorky v chodbách, čo nám tiež stačilo. Niektoré chronologicky zoradené etapy tohto testu znázorňuje kolekcia obrázkov 3.11 až 3.16.

3.4 Resamplovanie

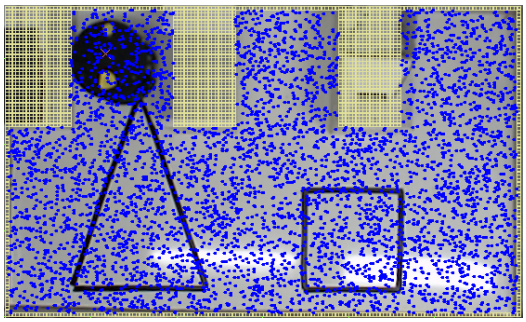
Pre resamplovanie vzoriek sme v aplikácii implementovali algoritmy *Select with replacement* 3 aj *Select with replacement in linear time* 4. Časová zložitosť



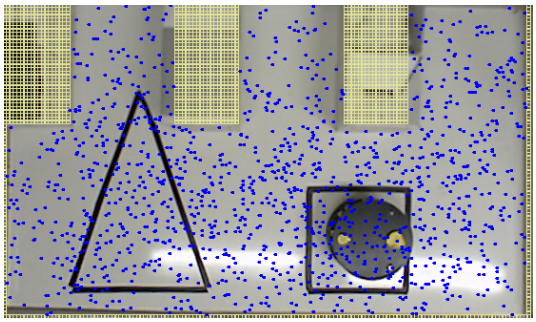
Obr. 3.11: Umiestnenie robota - A



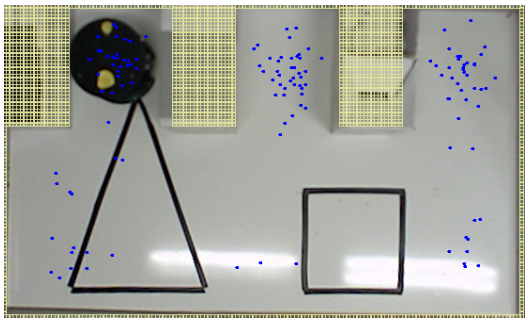
Obr. 3.12: Umiestnenie robota - B



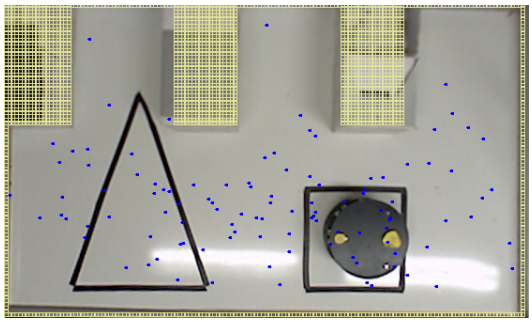
Obr. 3.13: Survival of the fittest - A1



Obr. 3.14: Survival of the fittest - B1



Obr. 3.15: Survival of the fittest - A2



Obr. 3.16: Survival of the fittest - B2

prvého je v dôsledku triedenia vygenerovaných náhodných čísel $O(Nlog(N))$. V druhom algoritme je triedenie nahradené sumovaním záporných logaritmov vygenerovaných náhodných čísel. Hrubá časova analýza zaraďuje tento algoritmus do triedy $O(N)$. Obsahuje ale oproti prvému algoritmu navyše jeden prechod poľom náhodných čísel, v ktorom nastáva delenie desatinnych čísel a časovo náročné by mohlo byť aj vytváranie logaritmov zo všetkých vygenerovaných čísel. Aby sme sa presvedčili, ktorá z implementácii bude v naších podmien-

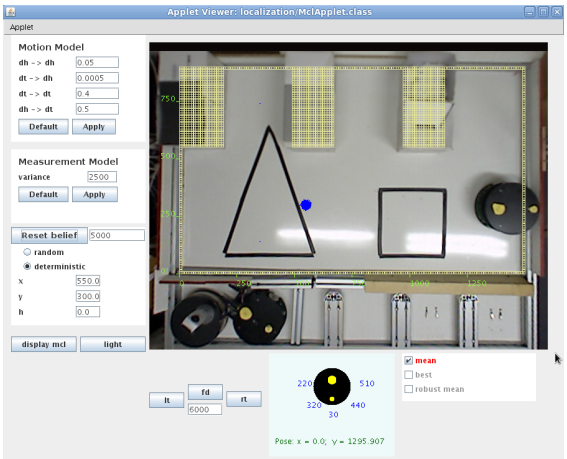
kach svižnejšia, zaznamenavali sme čas behu oboch resamplovaích procedúr pre rôzne hodnoty mohutnosti množiny vzoriek. Výsledky sú uvedené v tabuľke 3.4. Potvrdili, čo bolo očakávane na základe hrubej časovej analýzy.

Size of set	Running time [ms]	
	Select w. r.	Select w. r. in linear time
1000	3	2
10000	17	7
100000	62	41
500000	167	93

Tabuľka 3.2: Average running time of resampling algorithms

3.5 Monte Carlo Localization

Monte Carlo Localization jen algoritmus pre lokalizáciu založený na Particle filtri. V prípade, že implementácie samplovacieho modele pohybu, ľubovoľného modelu snímania a resamplovacej procedúry sú hotové, je jeho implementácia priamočiarym prepísaním pseudokódu *Particle filter* 5 do programovacieho jazyka.



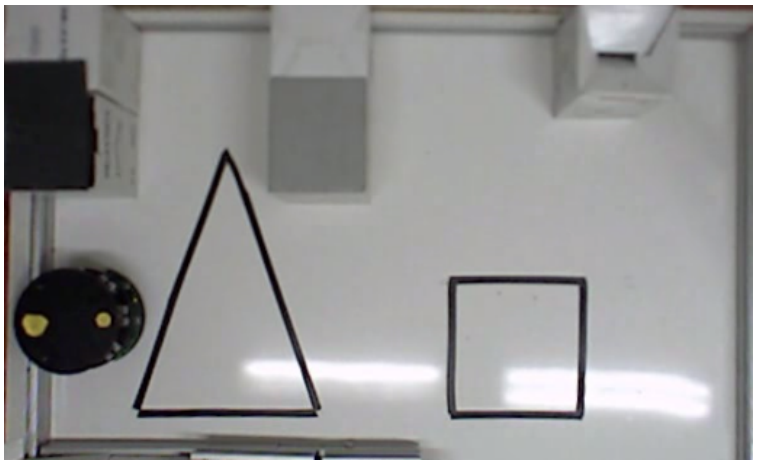
Obr. 3.17: MCL Applet

Aby bolo možné experiment lokalizácie pohodlne sledovať, rozhodli sme sa vizualizovať množinu vzoriek nad skutočným obrazom z kamery laboratória. Prezentačnú vrstvu celej aplikácie tvorí Java Applet 3.17. Je dostupný pre

každého na adrese:

`http://virtuallab.kar.elf.stuba.sk/~risko/probrob/localization`

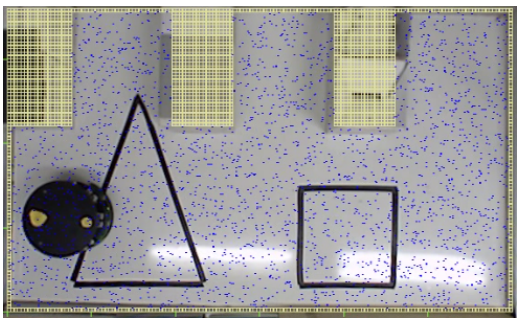
Po nastavení parametrov modelov a spôsobe inicializácie vzoriek je možné ovládať pohyb robotnačky. Robotnačka vykonaná pohyb, ak jej v tom nič nebráni. Ak je internetové spojenie dostatočné, obraz v pozadí vzoriek sa plynule mení. Applet tiež graficky znázorňuje získané odometrické a senzorové dáta, ktoré prichádzajú po vykonaní každého pohybu. Experiment tak môže prevádzať hocikto. Obmedzený je však iba na prostredie laboratória, v ktorom nemá možnosť meniť rozloženie prekážok. Jeden z naších experimentov, ktorý sme s týmto appletom vykonali, mal nasledujúci priebeh a výsledky. jeho cieľom bolo ukázať že naše implemntácie zvládajú globálnu loklizáciu a kidnapped robot problem v prostredí, ktorého skutočný vzhľad je na obrázku 3.27.



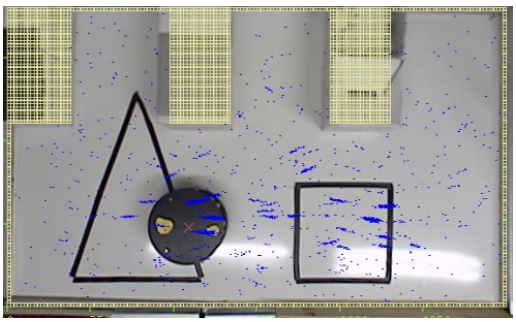
Obr. 3.18: Prostredie pre testovanie lokalizácie

Pri globálnej lokalizácii je belief robota o svojej pozícii reprezentovaný uniformnou distribúciou nad množinou všetkých možných pozícii. V jeho vzorkovej reprezentácii to znamená inicializáciu každej vzorky na na náhodnú pozíciu 3.19. Pohyby robota sme sa rozhodli ovládať ručne, hoci bolo možné prenechať ich aj na náhodne riadenie robota, snažiaceho sa iba nenáburať čelne do žiadného objektu. Update vzoriek bol vykonávaný po každom kroku spracovaním odometrických a senzorových dát. Počet krokov medzi dvoma resamplovaniami sme v tomto experimente nastavili fixne na hodnotu 4. Vývoj distribúciu vzoriek po približne každých desať korkoch znázorňujú obrázky 3.20, 3.21, 3.22.

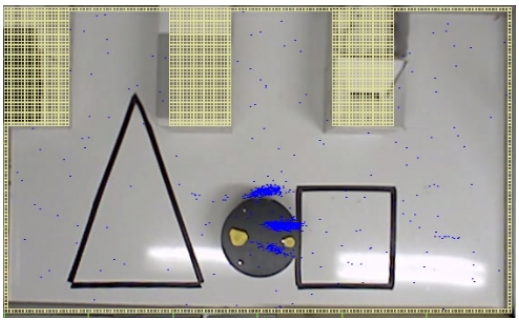
Na obrázku 3.22 je vidieť, že distribúcia vozriek dostatočne presne repre-



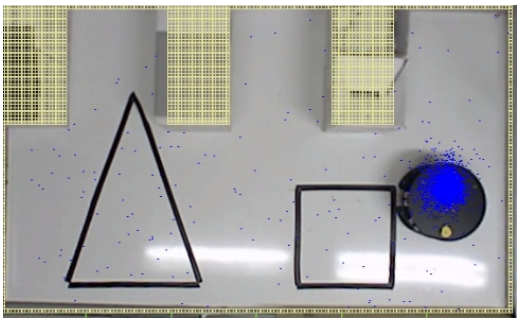
Obr. 3.19: Inicializačná distribúcia vzoriek



Obr. 3.20: Distribúcia vzoriek 1



Obr. 3.21: Distribúcia vzoriek 2

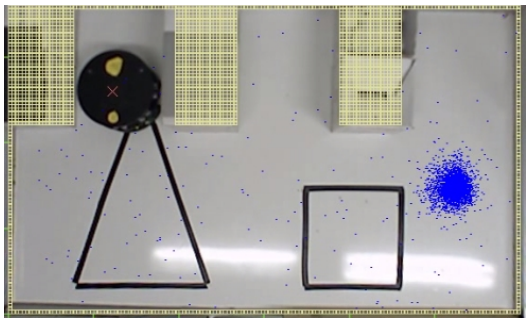


Obr. 3.22: Distribúcia vzoriek 3

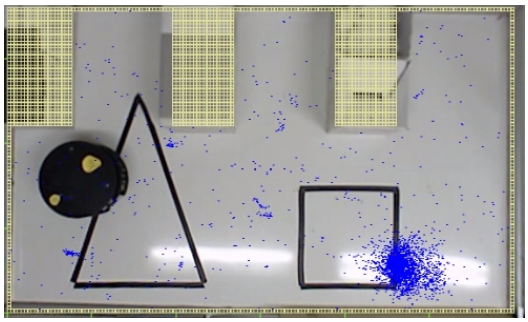
zentuje pozíciu robota. Hoci sa zhluk vzoriek zdá byť priliž roztiahnutý, je potrebné si uvedomiť, že v jeho centre je veľa vzoriek navzájom prekrytých. Váhovaný priemer vzoriek to potvrdil a globálnu lokalizáciu sme brali ako úspešne ukončenú.

Nasledovalo rúčné premiestnenie robota na inú pozíciu v prostredí. Aby bol Monte Carlo lokalizačný algoritmus schopný sa s týmto prípadom vysporiadať, bolo štandardnú schému Particle filtra rozšíriť o pridávanie malého počtu náhodnych vzoriek po každom resamplovaní. Bez nich by zhluk vzoriek sústredených okolo pozície robota pred premiestnením nebolo možné roztrúsiť, keďže resamplovanie by stále vyberalo do novej množiny iba vzorky zo zhluku. V našej implementácii bolo pridávanie náhodných vzoriek zahrnuté. Ich relatívny počet vzhľadom na mohutnosť základnej mmožiny vzoriek je možné prenastaviť. V tomto experimente bola jeho hodnota 1%. Po niekoľkých krokoch sa zhluk vzoriek roztrúsil na rôzne pozície a lokalizácia akoby bežala od začiatku. Po dostatočne veľa krokoch sa vytvoril nový zhluk okolo skutočnej pozície a po-

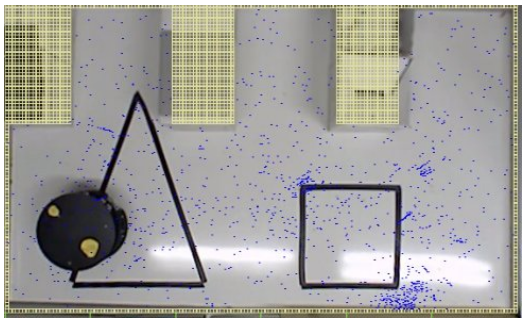
hyboval sa podľa skutočnej pozície robota, čo znamenalo úspech experimentu. Jeho priebeh zachytávajú obrázky 3.23, 3.24, 3.25, 3.26.



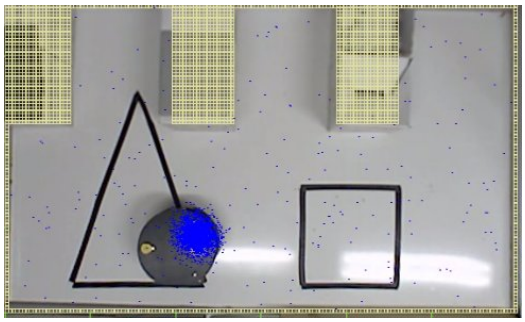
Obr. 3.23: Vzorky po prenesení 1



Obr. 3.24: Vzorky po prenesení 2



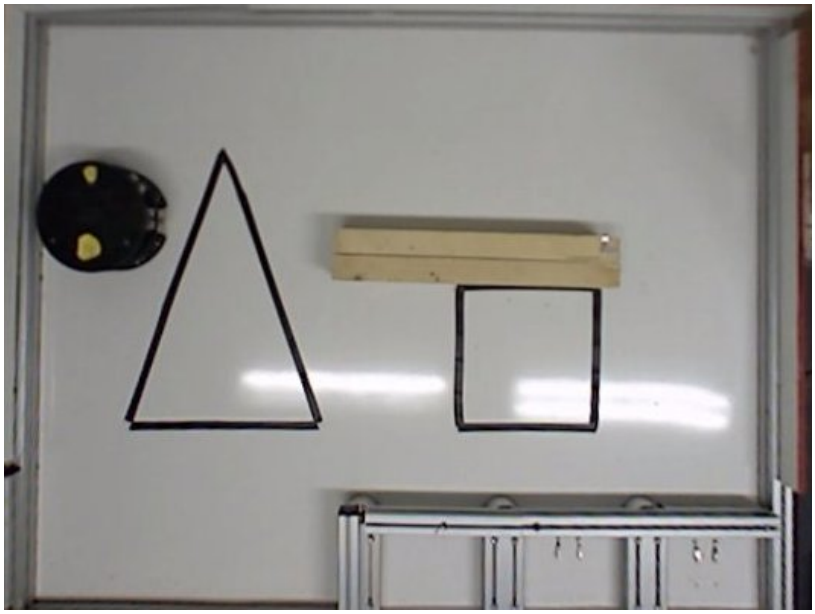
Obr. 3.25: Vzorky po prenesení 3



Obr. 3.26: Vzorky po prenesení 4

3.6 FastSLAM

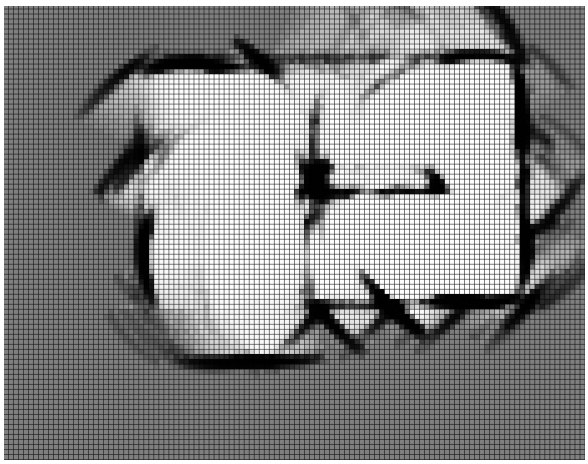
Druhá skupina experimentov, ktorým sme sa venovali súvisela s mapovaním a súčasnou lokalizáciou robota. Prostredie, ktoré sme sa rozhodli robotnačkou zmapovať, je znázornené na obrázku. Vytvorili sme si dataset (obahujúci odometrické a senzorové dáta) z jednej náhodnej prechádzky robota týmto prostredím.



Obr. 3.27: Prostredie pre mapovacie experimenty

Pri implementácii sme vychádzali z úvah popísaných v kapitole 2.6.3. Vzorkami sme reprezentovali trajektórie robota a každá vzorka obsahovala vlastnú response gridmapu vybudovanú na základe trajektórie robota a zosnímaných dát. Nakoľko nebolo naše prostredie extrémne rozsiahle, pamäťové nároky veľkého počtu gridmáp boli zvládnuteľné, a preto nebolo nutné zaoberať sa nejakou technikou sharovania gridmáp alebo jej častí medzi viacerými vzorkami. Model pohybu sme použili rovnaký a s rovnakými parametrami ako pri lokalizácii. Inverzný model snímania, ktorý riadi tvorbu gridmapy z nameraných dát a pozícii, sme implementovali postupne všetkými troma spôsobmi spomenutými v kapitole 2.6.2.1.

Prvý spôsob, ktorý všetkým bunkám apertúry senzora vo vzdialenosti od senzora menšej ako je nameraná hodnota priraďuje jednú konštantnú hodnotu a bunkám vo vzdialenosti rovnakej ako je nameraná hodnota inú konštantnú hod-

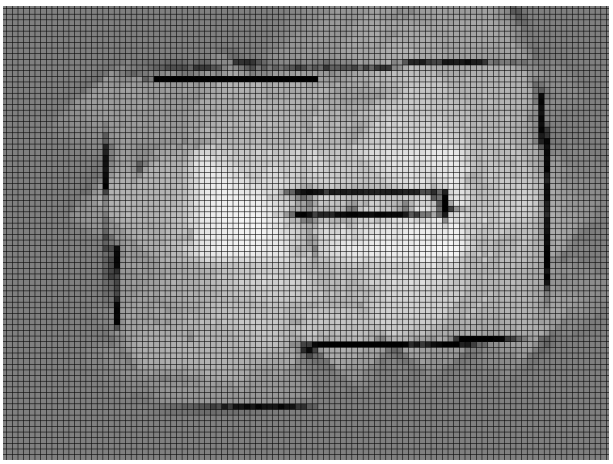


Obr. 3.28: Mapa A

notu, neviedol k uspokojivým výsledkom. Jedna z množstva máp, ktoré počas experimentu vznikli, je znázornená na obrázku 3.28. Vidíme, že dlhé krajné steny sú na niektorých miestach rozbité. Príčinou toho je práve jav specular reflection. Ak sa robot pohybal popri týchto stenách stále bol jeden jeho senzor v takej pozícii, že jeho vyslané impulzy na stenu sa k nemu nevrácali.

Zlepšenie sme očakávali od druhej metódy, ktorá bunkám v spmínaných regiónoch neprirad'uje konštanty, ale rôzne hodnoty závislé od pozície bunky v apertúre. K zlepšeniu ale nedošlo. Hlavný problém, specular reflection, ostával nevyriešený.

Výrazné zlepšenie prišlo zmenou klasických gridmáp na response gridmapy. Pre ich aktualizovanie sme implementovali algoritmus, tak ako sme ho odvodili v závere kapitoly 2.6.2.1. Bunky sme rozdelili do 4 smerov, čo znamenalo 4-násobne viac potrebnej pamäte na uchovávanie gridmáp. Jedna z máp je znázornená na obrázku 3.29. Je na nej možné vidieť pomerne celistvé obrysy stien.



Obr. 3.29: Mapa B

3.7 Popis komponentov aplikácie

Z niekoľkých programovacích jazykoch, ku ktorým existuju softvérové knižnice umožňujúca ovládanie robotnačky, sme sa rozhodli naprogramovať aplikáciu v Jave. Stručný popis jej hlavných modulov je nasledovný.

Jadro aplikácie je nezávislé od robotnačky, či prostredia. Tvoría ho triedy balíkov

- `particlebase`

  Obsahuje definície základných dátových štruktúr a iterfejsov potrebných pre implementáciu partiklového filtra 2.

  - `Pose`
  - `Particle`
  - `MapManagerInterface`
  - `MotionModelInterface`
  - `MeasurementModelInterface`
  - `ResamplerInterface`

- `mcl`

  Obsahuje implementáciu algoritmu Monte Carlo Localization 8, pričom vyžaduje štruktúry a implemntácie interfejsov prvého balíka.

- `fastslam`

  Obsahuje implementáciu algoritmu FastSLAM, pričom vyžaduje štruktúry a implemntácie interfejsov prvého balíka.

Druhá časť aplikácie je jej *grafický interfejs* vo forme appletu. Umožňuje ovládať robotnačku, prenastavovať parametre modelov a mapu prostredia. Priebeh algoritmov vizualizuje nad skutočným obrazom z kamery laboratória. Ten sa refrešuje v nastaviteľných intervaloch. Použiť ho pre iných robotov alebo iné prostredia je možné iba prepísaním a prekompilovaním tried

- `RobotManager`

  Zabezepčuje komunikáciu s robotom. Poskytuje informácie o pohybe robota a výsledkoch jeho meraní.

- `LabImageManager`

  Zabezepčuje zobrazovanie obrazu laboratória.

- `Config`

  Obsahuje rôzne konfiguračné parametre ako rozmery prostredia, dlžku kroku robota a iné.

Kapitola 4

Záver

V práci sme od popísali niektoré z metód pravdepodobnostnej robotiky. Začali sme zadefinovaním základných pojmov a cez všeobecné schémy algoritmov sme sa dostali až k implentáciam Particle filtra schopným riešiť lokalizáciu a mapovanie. V praktickej časti sme vytvorili Java Applet umožňujúci každému ovládať robotnačku, sledovať dianie v laboratóriu prostredníctvom plynulého obrazu z kamery, experimentovať s parametrami modelov lokalizačného algoritmu Monte Carlo a sledovať grafickú reprezentáciu belief pozície nad skutočným obrazom. Pre problém SLAMu s použitím ultrazvukových senzorov sme si odvodili pravidlá pre aktualizáciu zovšeobecnených gridmáp. Experimentamie sme otestovali viacero metód tvorby gridmáp.

Zoznam obrázkov

Literatúra

[1] *SRF08 Ultra sonic range finder.* Technical Specification.

[2] James Carpenter, Peter Clifford, and Paul Fearnhead. An improved particle filter for non-linear problems. pages 2–7, 2004.

[3] L. S. Dinnouti, A. C. Victorino, and G. F. Silveira. Simultaneous localization and map building by a mobile robot using sonar sensors. In *Proc. International Congress of Mechanical Engineering*, Sao Paulo/SP, Brazil, 2003. Article 1996.

[4] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2, 2006.

[5] Austin Eliazar and Ronald Parr. Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks, 2003.

[6] Dieter Fox, Wolfram Burgard, and Frank Dellaert Sebastian. Monte carlo localization: Efficient position estimation for mobile robots. In *In Proc. of the National Conference on Artificial Intelligence (AAAI*, pages 343–349, 1999.

[7] Giorgio Grisetti, Gian Diego Tipaldi, Cyrill Stachniss, Wolfram Burgard, and Daniele Nardi. Fast and accurate slam with rao-blackwellized particle filters. *Robotics and Autonomous Systems*, 55(1):30–38, 2007.

[8] Andrew Howard and Les Kitchen. Generating sonar maps in highly specular environments.

[9] P. Jensfelt, O. Wijk, D. J. Austin, M. Andersson, and Patric Jensfelt. Experiments on augmenting condensation for mobile robot localization. In *In*

Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 2518–2524, 2000.

[10] Adam Milstein. Occupancy grid maps for localization and mapping, 2008.

[11] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.

[12] H. P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *1985 IEEE International Conference on Robotics and Automation*, pages 116–121, St. Louis, Missouri, March 1985. IEEE Computer Society Press.

[13] Hans P. Moravec. Sensor fusion in certainty grids for mobile robots. *AI Magazine*, 9(2):61–74, 1988.

[14] Rudy Negenborn. Utrecht university robot localization and kalman filters, 2003.

[15] Pavel Petrovič, Andrej Lúčny, Richard Balogh, and Dušan Ďurina. Robotické laboratórium so vzdialeným prístupom. In *8th International Conference on Automation and Robotics in Theory and Practice (Robtep)*, volume 2006, pages 389–394. SjF TU Košice, 2006.

[16] Md. Jayedur Rashid. Parameterized sensor model and handling specular reflections for robot map building, 2006.

[17] Ioannis M. Rekleitis. A particle filter tutorial for mobile robot localization. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, 3480 University St., Montreal, Québec, CANADA H3A 2A7, 2004.

[18] Christof Schröter, Hans joachim Böhme, and Horst michael Gross. Memory-efficient gridmaps in rao-blackwellized particle filters for slam using sonar range sensors.

[19] Dan Simon. Kalman filtering, 2001.

[20] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[21] Greg Welch and Gary Bishop. An introduction to the kalman filter, 1995.

[22] Dušan Ďurina, Pavel Petrovič, and Richard Balogh. Robotnačka - the drawing robot. *Acta Mechanica Slovaca*, 10(2-A):113–116, 2006.