VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

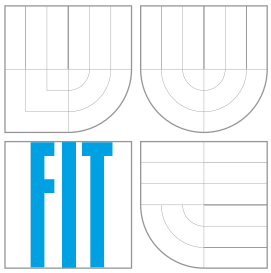
TVORBA MAPY PROSTŘEDÍ POMOCÍ ČÁSTICOVÝCH FILTRŮ A LASEROVÉHO SCANNERU

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE PETR IZRAEL
AUTHOR

BRNO 2010

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

TVORBA MAPY PROSTŘEDÍ POMOCÍ ČÁSTICOVÝCH FILTRŮ A LASEROVÉHO SCANNERU

MAP MAKING USING PARTICLE FILTERS AND LASER SCANNER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PETR IZRAEL

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. JAROSLAV ROZMAN

BRNO 2010

Abstrakt

Tato bakalářská práce se zabývá automatickou tvorbou mapy prostředí užitím laserového scanneru. Tento problém je také znám jako simultánní lokalizace a mapování. Práce popisuje častý přístup k problému užitím částicových filtrů pro tvorbu dvojrozměrných mřížkových map. Popsána je také vlastní implementace a několik jednoduchých experimentů.

Abstract

This thesis describes basics of automatic map making based on odometry and laser range data of a robot. This task is also known as Simultaneous Localization and Mapping. Approach described in this thesis uses particle filters for creating planar grid-based maps. Final chapters are concerned with own implementation and some basic experiments.

Klíčová slova

robotika, SLAM, částicový filtr, lokalizace, mapování

Keywords

robotics, SLAM, particle filter, localization, map making

Citace

Petr Izrael: Tvorba mapy prostředí pomocí částicových filtrů a laserového scanneru, bakalářská práce, Brno, FIT VUT v Brně, 2010

Tvorba mapy prostředí pomocí částicových filtrů a laserového scanneru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Petr Izrael
17. května 2010

Poděkování

Děkuji svému vedoucímu práce Ing. Jaroslavu Rozmanovi za poskytnuté konzultace a zapujčení mnohé odborné literatury.

Obsah

Kapitola 1

Úvod

Tato práce ma za cíl vysvětlit základy automatické tvorby map prostředí pomocí časticových filtrů a laserového scanneru. Tvorba map prostředí je úkol určený pro pohyblivého (mobilního) robota a je také znám pod názvem *Simultánní lokalizace a mapování/Simultaneous Localization and Mapping/SLAM*, protože úkolem robota není pouze mapovat své okolí, ale také musí být schopen v postupně vytvářené mapě určit svojí polohu, aby dokázal určit kde přesně nově zmapované prostory navazují na původní mapu.

1.1 Robotika

Klíčový rozdíl mezi mobilními a statickými roboty je, že mobilní jsou za pomocí různých mechanismů (většinou motorem poháněná kola) schopni pohybu v prostoru. Aby bylo možné z této výhody naplno těžit, je pro většinu pokročilejších aplikací těchto robotů nutné vyřešit otázku lokalizace[1] - Robot si musí být vědom, kde se v současné době nachazí. Tento problém je v robotice znám pod pojmem *lokalizace*.

Lokalizace a mapování

Roboti, kteří operují ve známém a neměnném prostředí si obvykle vystačí s danou mapou prostředí, jenž je jim napevno dána a nepřipouští v ní žádné změny. Problémem může být, pokud se prostředí mění, například se v něm pohybují lidé, nebo se pohybuje s nábytkem a tak dále. S menšími změnami prostředí si současné agoritmy využívající *pravděpodobnostní přístup* dokáží poradit.

Chceme-li vytvořit skutečně samostatné roboty, kteří jsou schopni se přizpůsobit i zcela novému prostředí, a případně se v něm dále orientovat, musíme implementovat také schopnost tvorby map prostředí. A protože mobilnímu robotu nikdy žádný věštec neprozradí, kde přesně se nachází, musíme řešit oba problémy (lokalizaci i mapování) současně.

Nutnost řešit tyto dva problémy současně je sice každým intuitivně předpokládána, vyvstává ale také zajímavý problém - Pokud chci vytvořit mapu, musím nejdřív vědět kde jsem. A naopak, pokud chci zjistit, kde jsem, potřebuji mapu. K tomuto problému se dá postavit mnoha způsoby.

V této práci popisuji pravděpodobnostní přístup k řešení pomocí tzv. částicových filtrů, jenž mají mimo robotiku dobré uplatnění také například v počítačovím vidění (*Kondenzační algoritmus/Condensation algorithm*)

[1] U jednoduchých aplikací je možné se tomuto zcela vyhnout a spoléhat se při řešení problému na jiné algoritmy[13]

Kapitola 2

Teoretický základ

2.1 Základní pojmy

Pro účely lokalizace a mapování je potřeba seznámit se se 4 pojmy:

- Pozice robota $\mathbf{x}$.
- Řídicí data u.
- Měření senzorů z.
- Mapa prostředí m.

Pozice

Pozice $\mathbf{x}$ musí být chápana nejen jako globální souřadnice v daném souřadnicovém systému, ale také úhel natočení robota. Formálně řečeno je pozice robota v dvoudimenzionálním prostředí v čase t dána vektorem: $\mathbf{x}_t = [x, y, \theta]^T$.

Řídicí data

Řídicí data u popisují interakci robota s prostředím, v našem případě výlučně pohyb. Pro řídicí data platí, že mění stav x, tedy pozici robota.

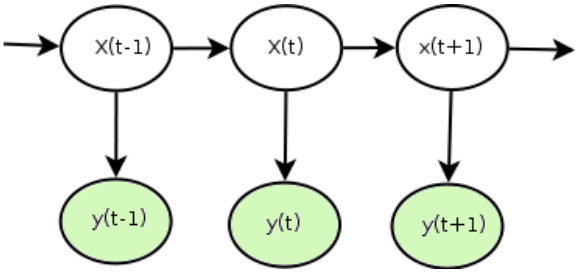
Měření senzorů

Měření senzorů z stav x nemění ale naopak ho popisují, slouží tedy k získávání informací.

2.2 Způsoby reprezentace mapy

Dva hlavní způsoby reprezentace mapy jsou[13]:

- **Mřízkové mapy s pravděpodobností obsazenosti** (*Occupancy grid maps*, dále jen OGM) - Jeden bod v mřížce odpovídá skutečné dvourozměrné ploše o velikosti dané rozlišením mapy. Čím má mapa větší rozlišení, tím více detailní prostředí dokaže přesně zaznamenat, nevýhodou ale mohou být paměťové nároky. Bod v mřížce může nabývat buď pouze hodnoty 0 nebo 1, nebo může nabývat více hodnot a je tak možné do mapy zaznamenat více informací; často se ukládá pravděpodobnost, že bod je



Obrázek 2.1: Stav x(t) není přímo zjistitelný, narozdíl od výstupu y(t), který generuje.

obsazen. Sestrojení této mapy je oproti jiným typům map jednodušší, protože se využívá přímo surových naměřených dat, např. z laserového scanneru.

- **Topologické mapy** - Reprezentují realný prostor grafem, kde každý uzel v grafu označuje konkrétní místo, na kterém se robot může nacházet a hrany označují fyzické spojení míst. Pokud používá robot pro svou lokalizaci topologickou mapu, není pro něho důležité, jaké jsou jeho přesné souřadnice, ale ve kterém uzlu se nachází. Automatická konstrukce těchto map většinou vyžaduje předchozí znalosti o uzlech vytvářeného grafu.

V této práci bude nadále vždy pod pojmem *mapa* myšlena OGM mapa.

2.3 Pravděpodobnostní přístup

Na pozici $\mathbf{x}_t$ bude v textu odkazováno také jako na *stav*. Stav je v tomto smyslu vlastnost charakterizující robota a jeho prostředí. Jinými slovy se jedná o *stav* systému robot-prostředí. Stavy, které se mění v čase se označují jako dynamické. Jelikož pro odhad pozice využíváme pravděpodobnostního přístupu, je cílem modelovat systém, který popisuje stochastické procesy[1] při změně *skutečné pozice* robota.

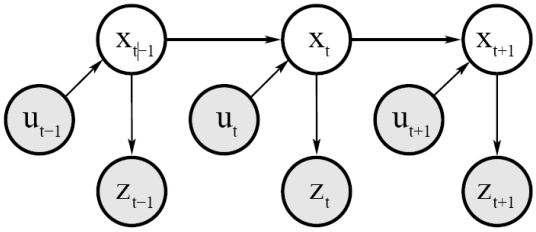
Pokud stav označíme za *kompletní*, znamená to, že proces popisující přechod z jednoho stavu na druhý splňuje podmínku, že znalost minulých stavů nijak neovlivní stavy budoucí. Procesy splňující tuto podmínku jsou známy pod názvem Markovův řetězec, podle ruského matematika Andreje Markova.

V oblasti robotiky není mnohdy možné stav zjistit přímo, ale pouze pomocí výstupu, který (a navíc ještě v drtivé většině stochasticky) generuje funkce tohoto stavu. Vše jasně zobrazuje obrázek 2.1. Pokud modelovaný systém splňuje tyto požadavky, mluví se o něm jako o *Skrytém Markovově modelu*.

Pro úspěšné získání tohoto stavu musíme tedy pochopit souvislost mezi generovaným výstupem a skutečným stavem x a vytvořit funkci inverzní, jinými slovy musíme vědět, jak z výstupu y(t) získat stav x(t).

To přesně je případ stavu pozice. Nemůžeme ji zjistit přímo, ale pouze pomocí znalostí o předcházejícím stavu (v nejhorším případě za předcházející stav můžeme považovat i stav "neznámá pozice") a výstupu, který nový stav generuje, což je v tomto případě měření senzorů (laserového scanneru). Vidíme že generovaný výstup je stochastický, protože senzory jsou nepřesné, mohou selhat, může se jim někdo připlést do cesty a podobně (viz 3.7). Na obrázku 2.2 je zobrazen model popisující systém lokalizace, vycházející z měření senzorů

[1] Stochastickým procesem se rozumí jakýkoliv (časový) vývoj, jež podléhá nederministickému chování, které můžeme popsat pravděpodobnostním rozložením



Obrázek 2.2: Vývoj stavů a měření v čase, převzato z [15]

a řídicích dat. Zajímá nás stav x_t, ten je ale zjistitelný pouze přes měření z_t. Stav x_t ovlivňuje předchozí hodnota stavu a řídicí data u_t. Pokud proto mámé nějaké znalosti o pozici předcházející nové pozici robota a víme jak řídicí data u_t ovlivnili novou pozici, je úkol odhadu nového stavu x_t ulehčen a nemusíme se spoléhat pouze na měření z_t.

Všechny přechody naznačené na obrázku 2.2 jsou ale stochastické a mnohdy měření poskytuje informace, které nejsou pro jednoznačné určení pozice dostatečné. Robot proto nikdy nevyužívá nějaký svůj nejlepší odhad pozice, ale vždy pracuje s pravděpodobnostním rozložením nad množinou všech možných pozic, respektive v praxi s aproximací tohoto rozložení. Je tedy možné elegantně vyjádřit i složitou představu o vlastní pozici. Například se robot může domnívat, že s největší pravděpodobností může být na jednom ze dvou míst, v němž u každého připouští nějakou odchylku vzniklou nejistotou při měření a k tomu může například i připustit, že s malou pravděpodobností se nachází úplně někde jinde.

Toto vůbec není neobvyklé - pokud robot nezná svou původní polohu je pravděpodobnost všech pozic stejná. Poté začne zpracovávat data ze senzorů a zjistí že se nachází v nějakém (blíže nespecifikovaném) rohu, pravděpodobnost pozic v rozích místnosti stoupne a ostatní pravděpodobnosti klesnou. Pokud se robot vyskytuje v místnosti čtvercového tvaru, připustí čtyři různé pozice.

Vnitřní znalost robota ohledně své pozice je tedy charakaterizována pravděpodobnostím rozdělením a bývá také značena $bel(x)$, což je zkratka pro anglické slovo *belief* čili víra nebo přesvědčení[15].

Kapitola 3

Lokalizace robota

Tato kapitola vysvětluje principy lokalizace robota, zejména pravděpodobnostní metody odhadu pozice známé pod názvem Monte Carlo lokalizace a klasický způsob její implementace. Tato metoda je dobře rozšiřitelná tak, aby mohla být použita pro účely simultánní lokalizace a mapování.

3.1 Odometrie

Slovo *odometrie* je složenína dvou řeckých slov *hodos*-cesta a *metron*-měření. Odometrie popisuje transformaci dat získáných z vnitřních sensorů robota do odhadu aktuální pozice. Například u robota s koly se snímá počet přechodů mezi různě zabarvenými ploškami na kole. Z těhto hodnot se díky znalosti některých dalších konstant (průměr kol...) a předchozí pozice vypočítá nová pozice. Tento proces se nekonečně v pravidelných či nepravidelných intervalech opakuje.

Dvě měření odometrie jdoucí po sobě v čase $t-1$ a v čase t označují změnu pozice a dají se proto intepretovat jako řidicí data u_t. Toto je obvykle přesnější, než kdybychom řídicí data získavali jiným způsobem[15].

3.1.1 Chyba odometrie

Nejnaivnější implementace lokalizace by (za předpokladu, že je známa počáteční pozice robota) pozici robota vyčetla přímo z dat odometrie. Tímto ale získáme pouze *velmi* nepřesný odhad pozice, což je způsobeno mnoha faktory.

Mezi zdroje chyb mimo běžnou (a také závažnou) chybu vzniklou zaokrouhlovaním a diskretizací (prostoru i času), které odometrie robota s koly není schopna postihnout patří například:

- Rozdílné tření různých povrchů podlah - Důležitý zdroj chyb, přítomný i pokud zamezíme výskytu níže zmíněných. Kolečka se na každém povrchu chovají jinak. Tření může ovlivnit např. i teplota, tlak, špína na podlaze a spousta dalších těžce předvídatelných faktorů
- Sklon podlahy - Chyba vzniká kdykoli je úhel sklonu podlahy jiný než úhel natočení koleček.
- Nerovnosti podlahy - Robot může různě a nepředpovídatelně poskakovat po podlaze

- Naražení do překážky - kolečka se točí, odometrie je tedy přesvědčena, že robot se hýbe, ve skutečnosti ale stojí na místě
- „Únos robota“ - Pokud robota zvedneme a položíme na libovolné jiné místo, robot se v žádném případě nemůže spoléhat jen na odometrii.

Tento seznam je pouze ilustrační, protože vždy závisí na konkrétnímu způsobu pohybu a implementaci odometrie. Závěr je ale jednoznačný - spoléhat se pouze na odometrii je pro spolehlivé určení pozice naprosto nedostatečné.

3.2 Senzory

Chceme-li polohu robota určit přesněji, musíme využít senzorů. Senzory jsou sice také nepřesné, při použití vhodných algoritmů je ale spojení hodnot odometrie s měřením dostatečně kvalitních senzorů pro určení pozice dostatečné.

Měření senzorů v časte t značím výrazem z_t. V následujícím textu v duchu zadání předpokládám, že senzorem je vždy laserový scanner.

3.3 Bayesův filtr

Absolutním základem zjištění nového pravděpodobnostního rozdělení pozice robota je důležitý poznatek teorie pravděpodobnosti - tzv. Bayesův teorém, který popisuje vztah mezi podmíněnou pravděpodobností jevu a jeho opačnou podmíněnou pravděpodobností. Matematický zápis Bayesova teorému je:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)} \tag{3.1}$$

$p(A|B)$ je pravděpobonost jevu A, pokud nastal jev B. Tento zápis značí podmíněnou pravděpodobnost. Následující odvozování bylo převzato z [15].

Teorém můžeme rozšířit tak, že B ozačuje více jevů. Za A dosadíme pozici robota a za B dosadíme všechny získané řídicí hodnoty a všechna měření senzorů a teorém aplikujeme na poslední získané měření z_t. Získáme tedy:

$$p(x_t|u_t 1:t, z_{1:t}) = \frac{p(z_t|x_t, z_{1:t-1}, u_{1:t})p(x_t|z_{1:t-1}, u_{1:t})}{p(z_t|z_{1:t-1}, u_{1:t})} \tag{3.2}$$

Kde zápis $z_{1:t}$ ozačuje všechna měření od času 1 po čas t se dá rozepsat jako $z_1, z_2, z_3 ... z_t$. Obdobně pro další zápisy s dvojtečkou.

Tato rovnice se dá v mnoha směrech upravit, pokud připustíme zjednodušení, že stav x je *kompletní*. Připouštímě také, že jsou v něm zahrnuta všechna minulá měření a řídicí data. Toto zjednodušení není nijak zvlášť přehnané, protože právě z měření a řidích dat se vždy nová pozice vypočítává. Opomíjíme tím ale velké množství dalších faktorů, například zanedbáváme změnu prostředí.

Další úprava se týká jmenovatele, jenž je pro všechny možné stavy x_t konstantní. Proto na něj můžeme pohlížet jako na normalizační konstantu η, jež slouží k normalizaci výsledku na 1.

Výše popsanými úpravami získáváme:

$$p(x_t|u_{1:t}, z_{1:t}) = \eta\, p(z_t|x_t)\, p(x_t|z_{1:t-1}, u_{1:t}) \tag{3.3}$$

Nyní nastává klíčová úprava, jež definuje princip Bayesova filtru pro určení nové polohy robota. Výraz $p(x_t|z_{1:t-1}, u_{1:t})$ můžeme pomocí Věty o úplné pravděpodobnosti upravit takto:

$$p(x_t|z_{1:t-1}, u_{1:t}) = \int p(x_t|x_{t-1}, z_{1:t-1}, u_{1:t})\, p(x_{t-1}|z_{1:t-1}, u_{1:t})\, dx_{t-1} \tag{3.4}$$

Zde vidíme, že pravděpodobnost stavu x_t je podmíněna pravděpodobností stavu x_{t-1} a opět můžeme předpokládát, že stav x je kompletní a po dosazení do rovnice 3.3 získáme konečnou podobu rovnice:

$$p(x_t|z_{1:t}, u_{1:t}) = \eta\, p(z_t|x_t) \int p(x_t|x_{t-1}, u_t)\, p(x_{t-1}|z_{1:t-1}, u_{1:t-1})\, dx_{t-1} \tag{3.5}$$

neboli

$$bel(x_t) = \eta\, p(z_t|x_t) \int p(x_t|x_{t-1}, u_t)\, bel(x_{t-1})\, dx_{t-1} \tag{3.6}$$

Z této rovnice získáváme algoritmus známý jako Bayesův filtr.[15] Algoritmus popisuje transofrmaci rozdělení pravděpodobnosti stavu x_{t-1} na rozdělení pravděpodobnosti stavu x_t.

Algoritmus 1: Bayesův filtr

Vstup: $bel(x_{t-1}), u_t, z_t$
Výstup: $bel(x_t)$

foreach x_t **do**
  $\overline{bel}(x_t) = \int p(x_t|x_{t-1}, u_t)\, bel(x_{t-1})\, dx_{t-1}$
  $bel(x_t) = \eta\, p(z_t|x_t)\, \overline{bel}(x_t)$
end
return $bel(x_t)$

Jak je vidět, pro realnou implementaci výše zmíněného algoritmu je nutno vyřešit několik problémů:

1. Způsob, jakým budeme reprezentovat rozdělení pravděpěpodobnosti pozice robota $bel(x_t)$, tedy znalosti o vlastní pozici.
2. Výpočet $p(x_t|x_{t-1}, u_t)$ - tj. implementaci pohybového modelu robota
3. Výpočet $p(z_t|x_t)$ - tj. implementaci modelu senzorů

3.4 Částicové filtry

Na první z výše zmíněných bodů dávají odpověď částicové filtry. V principu se využívá náhodného výběru vzorků z funkce hustoty pravděpodobnosti, popisující přechod z jednoho stavu do následujícího. Poté se pomocí ohodnocující funkce přiřadí jednotlivým vzorkům váhy, které určují pravděpodobnost,že daný vzorek je blízký skutečnému stavu. [15, 4]. Částicové filry také v našem případě definují způsob, kterým je možno reprezentovat libovolné pravděpodobnostní rozložení pozice robota tak, aby s ním bylo možno efektivně pracovat na číslicovém počítači. Rozložení je reprezentováno množinou M částic (vzorků), kde každá částice představuje jednu možnou konkrétní pozici:

$$\chi_t = \{x_t^{[1]}, x_t^{[2]}, \ldots, x_t^{[M]}\}$$

Částicové filtry v praxi

Výhoda použití částicových filtrů pro lokalizaci tkví v jejich univerzálnosti, snadné implementaci a v možnosti teoreticky reprezentovat libovolné rozdělení pravděpodobnosti včetně stavu jako „neznámá pozice“, kdy jsou částice rovnoměrně rozprostřeny po celém stavovém prostoru. Je tedy možné aplikovat částicové filtry i na problémy, kde robot nezná svoji počáteční pozici. Robustnost částicových filtrů je zaplacena zvýšenými nároky na výpočetní výkon v porovnání s jednoduššími metodami, například Kalmanovým filtrem. Aplikace částicových filtrů do oblasti lokalizace je známa pod názvem *Monte Carlo Lokalizace (MCL)* [4]. Tento algoritmus bude spolu s principem částicových filtrů popsán v následující sekci.

3.5 Monte Carlo Lokalizace

Následující sekce byla z vélké části (včetně obrázků) převzata z [15].

Základní algoritmus MCL Lokalizace pracuje v nekonečné smyčce ve třech fázích, akce v každé fázi se aplikuje na všech M částic:

1. **Predikce** - Náhodné vzorkování nové částice $\mathbf{x}_t$ využitím řidicích dat, předchozí polohy částice a pohybového modelu robota. Jinými slovy vzorkování z hustoty pravděpodobnosti $p(x_t|x_{t-1}, u_t)$. Predikci odpovídá řádek 2 algoritmu 2.

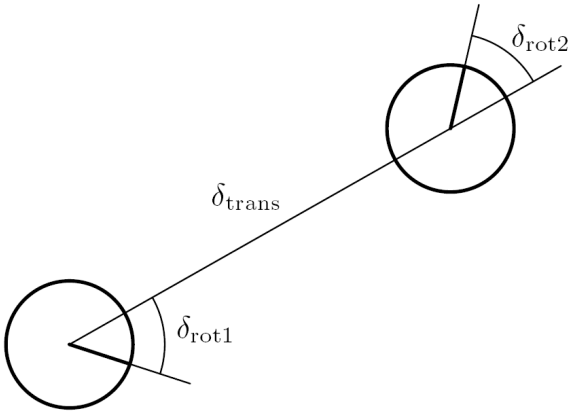
2. **Korekce** - Každé částici je přiřazena váha, která určuje pravděpodobnost, že částice odpovídá realné pozici. Zde se využívá měření senzorů. Korekci odpovídá řádek 3 algoritmu 2.

3. **Převzorkování** - Částice s nízkým váhovým ohodnocením se vypustí, naopak částice s vysokou pravděpodobností se znásobí. Celkový počet nových částic je typicky opět M. Tato fáze je náznačena na řádcích 6-9 v algoritmu 2. Tento základní MCL algoritmus reprezentuje pozici přímo množinou částic, kdežto většina reálných implementací algoritmů (např. v [12, 16]) reprezentuje pozici množinou vážených částic, které převzorkovává jen pokud jsou váhy velké části částic příliš malé.

Algoritmus 2: Monte Carlo lokalizace

Vstup: Množina částic χ_{t-1}, u_t, z_t
Výstup: χ_t

1 **for** $k = 1$ *to* M **do**
2 $\quad x_t^{[k]} = \textbf{sample\_motion\_model}(u_t, x_{t-1}^{[k]})$
3 $\quad w_t^{[k]} = \textbf{measurement\_model}(z_t, x_t^{[k]}, m_{t-1}^{[k]})$
4 $\quad \bar{\chi}_t = \bar{\chi}_t + \langle x_t^{[k]}, w_t^{[k]} \rangle$
5 **end**
6 **for** $k = 1$ *to* M **do**
7 $\quad$ draw i with probability $\propto w_t^{[i]}$
8 $\quad$ add $x_t^{[k]}$ to χ_t
9 **end**
10 **return** χ_t



Obrázek 3.1: Znázornění změny pozice robota z času $t-1$ do času t a rozklad této změny na tři dílčí části δ_{rot1}, δ_{trans} a δ_{rot2}

3.6 Pravděpodobnostní model změny pozice

Zde popsaný způsob odhadu změny (tj. predikce) lze interpretovat jako funkci **sample_motion_model** z algoritmu 2. Jedná se o standardní a snadno implementovatelnou metodu. Důmyslnější metoda na bázi scan-matchingu je představena v kapitole o simultánní lokalizaci a mapování v sekci 5.2.2. Tato metoda má ale přesto značný význam. Následující text (i s obrázky) byl volně převzat z [15].

Pokud jsou řídící data odvozeny z odometrie, každou z M částic můžeme aktualizovat například následujícím způsobem:

Pohybový model založený na odometrii

Výpočet nové částice je docela přímočarý. Předpokládáme, že každá změna pozice se dá rozložit na tři dílčí části - rotace do směru pohybu (δ_{rot1}), pohyb (δ_{trans}) a rotace do natočení v nové pozici (δ_{rot2}). Tyto hodnoty se vypočítají takto:

$$\delta_{rot1} = atan2(\bar{y}_t - \bar{y}_{t-1}, \bar{x}_t - \bar{x}_{t-1}) - \theta_{t-1} \tag{3.7}$$

$$\delta_{trans} = \sqrt{(\bar{x}_t - \bar{x}_{t-1})^2 + (\bar{y}_t - \bar{y}_{t-1})^2} \tag{3.8}$$

$$\delta_{rot2} = \bar{\theta}_t - \bar{\theta}_{t-1} - \delta_{rot1} \tag{3.9}$$

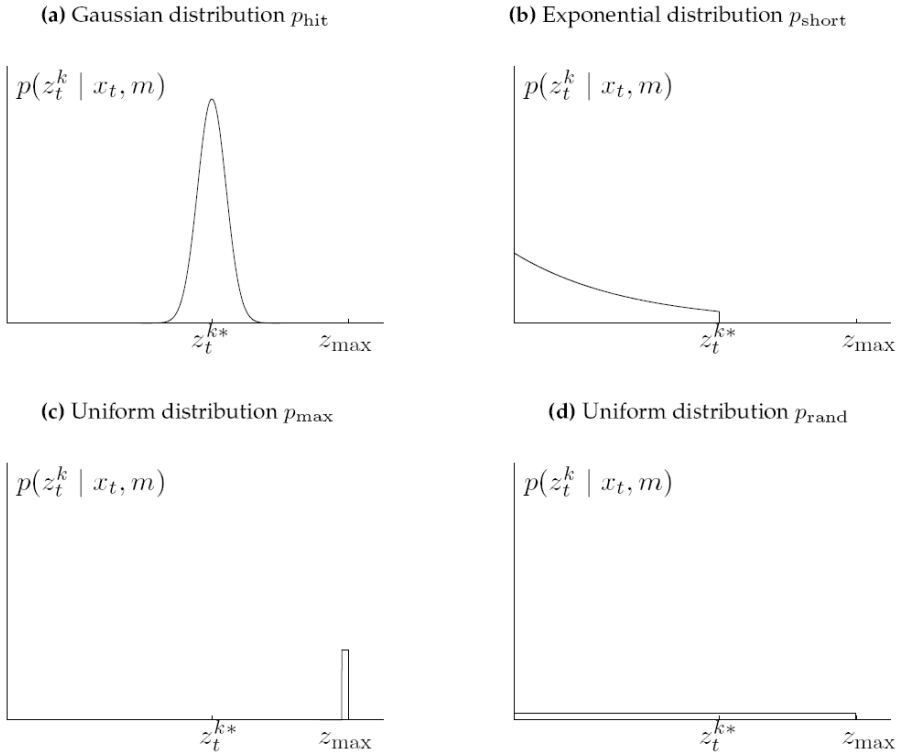
Kde $\bar{x}$,$\bar{y}$ a $\bar{\theta}$ určuje x-ovou/y-ovou složku pozice a rotaci podle odhadu odometrie v čase uvedeném v dolním indexu. Nepřesnost je modelována přičtením určitého šumu, který má popisovat chyby odometrie (viz 3.1.1) a kterému většinou definujeme gaussovo rozložení:

$$\hat{\delta}_{rot1} = \delta_{rot1} + \varepsilon_{\alpha_1|\delta_{rot1}|+\alpha_2|\delta_{trans}|} \tag{3.10}$$

$$\hat{\delta}_{trans} = \delta_{trans} + \varepsilon_{\alpha_3|\delta_{trans}|+\alpha_4|\delta_{rot1}+\delta_{rot2}|} \tag{3.11}$$

$$\hat{\delta}_{rot2} = \delta_{rot2} + \varepsilon_{\alpha_1|\delta_{rot2}|+\alpha_2|\delta_{trans}|} \tag{3.12}$$

Zde proměnná ε označuje náhodnou proměnnou s gaussovým rozložením se střední hodnotou 0. Rozptyl definuje dolní index proměnné. Čtyři parametry α_1 až α_4 jsou závislé na konkrétním prostředí a realizaci odometrie. Velmi podrobný rozbor chyby odometrie lze najít v [12]. Význam parametrů ukazuje následující tabulka:



Obrázek 3.2: Čtyři části modelu laserového scanneru

α_1	Rotační chyba - určuje, jak chybu rotace ovlivňuje velikost změny rotace
α_2	Driftová chyba - určuje, jak chybu rotace ovlivňuje právě ujetá vzdálenost
α_3	Translační chyba - určuje, jak chybu posunu ovlivňuje právě ujetá vzdálenost
α_4	Určuje, jak chybu posunu ovliňuje změna rotace

Nyní již máme všechno co potřebujeme pro výpočet nové částice $\mathbf{x_t} = (\mathbf{x}'\, \mathbf{y}'\, \theta)$:

$$x' = x + \hat{\delta}_{trans} \cos(\theta + \hat{\delta}_{rot1}) \tag{3.13}$$
$$y' = y + \hat{\delta}_{trans} \sin(\theta + \hat{\delta}_{rot1}) \tag{3.14}$$
$$\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2} \tag{3.15}$$

x,y a θ označují odpovídající složky pozice minulé (vstupní) částice.

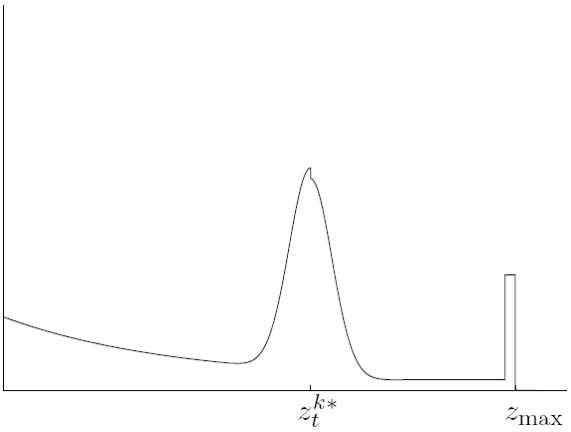
3.7 Model pravděpodobnosti měření založený na ray castingu

Zde popsaný způsob odhadu pravděpodobnosti měření lze interpretovat jako funkci **measurement_model** z algoritmu 2. Volně převzato (včetně obrázků) z [15].

Každé částici se přiřadí váha odpovídající věrohodnosti částice, tedy pravděpodobnosti, že se částice blíží skutečné pozici. Algoritmus pracuje s mapou prostředí, proto se spíše než $p(z_t|x_t)$ uvádí, že počítáme $p(z_t|x_t, m)$.

Model měření laserovým scannerem

Laserový scanner obvykle v jednom kroku vrací K hodnot dílčích měření. Jedno dílčí měření z_t^k odpovídá vyslání laserového paprsku do konkrétního úhlu a jeho následné zachycení.



Obrázek 3.3: Funkce hustoty pravděpodobnosti měření

Typický laserový scanner skenuje prostor pod úhlem 180° a každý další paprsek je vyslán v o jeden stupeň jiném úhlu, takže $K = 181$.

Musíme tedy spočítat $p(z_t^k|x_t, m)$ pro každé z K dílčích měření a následně tyto hodnoty mezi sebou vynásobit a získáme $p(z_t|x_t, m)$. Především musíme znát mapu prostředí a musíme vědět, jak se laserový scanner opravdu chová a jak se liší výsledek měření od skutečné vzdálenosti překážky. Protože pozice je dána částicí a mapu známe (nebo alespoň její část, pokud řešíme SLAM problém), zbývá zjistit skutečnou vzdálenost od překážky. Tu zjistíme technikou zvanou *ray casting*, která spočívá ve vystřelení imaginárního paprsku v naší mapě z místa domnělé pozice robota ve stejném směru, jako byl vyslán laserový paprsek. Tím pro dané dílčí měření zjistíme "reálnou" vzdálenost od překážky, kterou značíme z_t^{k*}. Tuto hodnotu použijeme pro sestrojení funkce hustoty pravděpodobnosti měření. Ta se skládá ze čtyř elementárních funkcí, jež jsou pro větší názornost na obrázku 3.2.

1. Gaussovo rozložení p_{hit} se středem v bodě z_t^{k*} popisuje úspěšné měření s malou odchylkou, jež je dána nepřesností scanneru, atmosférickými efekty apod.

2. Exponenciální rozložení p_{short} popisuje chybu měření vzniklou nečekanými objekty v trajektorii laserového paprsku.

3. Velmi úzké rovnoměrné rozložení p_{max} soustředěné kolem bodu z_{max} modeluje případy, kdy scanner absolutně selhal v měření a vrátil tedy maximální dosah scanneru (z_{max})

4. Rovnoměrné rozložení p_{rand} definované na celém intervalu $(0, z_{max})$, modelující ostatní těžko vysvětlitelné chyby měření.

Výsledná funkce hustoty pravděpodobnosti je znázorněna na obrázku 3.3.

Kapitola 4

Mapování

4.1 Vytváření OGM map

Při vytváření mapy řešíme problém formálně definovaný $p(m|z_{1:t}, x_{1:t})$, předpokládáme tedy, že v každém kroku známe svou pozici. Tohle je silně v rozporu s výše zmíněným způsobem lokalizace, který pro určení pozici potřebuje právě mapu. Řešení tohoto problému bude vysvětleno v následující části, nyní můžeme např. předpokládát, že náš robot má bezchybnou odometrii a pozice je tedy vždy známa.

Standardní přístup zjednodušuje problém na zjištění pravděpodobnosti jednotlivých bodů v mapě a řešíme tedy problém pro každý bod v mapě zvlášť:

$$P(m_i|z_{1:t}, x_{1:t})$$

$P(m_i)$ zde odpovídá pravděpodobnosti, že bod m_i je obsazen ($P(m_i) = Occupied$). Výpočet pravděpodobnosti obsazenosti každého bodu nezávisle na jeho okolních bodech může u některých druhů senzorů způsobovat problémy. Není to ale naštěstí případ laserového scanneru, jehož paprsek v jednom okamžiku protíná vždy pouze jeden bod. [1].

Princip mapování ukazuje následující algoritmus:

Algoritmus 3: OGM Mapování

Vstup: x_t, z_t
Výstup: $l_{t,i}$

```
foreach cell m_i do
    if m_i in perceptual field of z_t then
        l_{t,i} = l_{t-1,i} + inverse_sensor_model(m_i, x_t, z_t) - l_0
    else
        l_{t,i} = l_{t-1,i}
    end
end
return l_{t,i}
```

Algoritmus v zájmu ošetření zaokrouhlovacích chyb pro pravděpodobnosti blízké 0 a 1 a také pro zvýšení výpočetní rychlosti pracuje s tzv. *log šancí (log odds)*, která je definována

[1] Pokud je pro tvorbu mapy prostředí použit sonar, je toto zjednodušení poněkud problematické a byla proto navrhnuta jiná metoda mapování - viz [14]

takto:

$$l_{t,i} = \log \frac{P(m_i|x_{1:t}, z_{1:t})}{1 - P(m_i|x_{1:t}, z_{1:t})} \tag{4.1}$$

Skutečnou pravděpodobnost lze vypočítát pomocí vzorce:

$$P(m_i = Occupied|z_{1:t}, x_{1:t}) = 1 - \frac{1}{1 + e^{l_{t,i}}} \tag{4.2}$$

Výraz l_0 odpovídá počáteční log-šanci obsazenosti bodu. Takto můžeme nadefinovat apriorní pravděpodobnost obsazenosti. Pokud např. věříme, že mapovaný prostor obsahuje velmi mnoho volného prostoru můžeme apriorní pravděpodobnost nadefinovat menší než 0,5. Často se ale apriorní pravděpodobnost ignoruje a stanovuje se jednoduše na 0,5. Její log-šance je tedy 0 ($\log(\frac{0,5}{1-0,5}) = \log(1) = 0$), což algoritmus dále zjednodušuje a také nepatrně zvyšuje jeho rychlost (máme o jednu operaci součtu méně).

Všechny body na mapě, které protíná vyslaný laserový paprsek jsou aktualizovány. Pravděpodobnost obsazenosti v bodech, kterými paprsek proletěl klesne a naopak pravděpodobnost obsazenosti v bodě od kterého se paprsek odrazil stoupne. Konkrétní model ale musí být definován ve funkci **inverse_sensor_model**, která implemetnuje $P(m_i|x_t, z_t)$ a výsledek opět jako log-šanci:

$$\textbf{inverse\_sensor\_model}(m_i, x_t, z_t) = \log \frac{P(m_i|x_t, z_t)}{1 - P(m_i|x_t, z_t)} \tag{4.3}$$

Funkce je opakem běžného senzorového modelu, představeném v části 3.7, protože se v ní nezjišťuje pravděpodobnost z_t při znalosti dané m_t, ale pravděpodobnost určité oblasti m_t při měření z_t.

Tato sekce byla volně převzata z[15].

4.2 Inverzní senzorový model

Výpočet $P(m_i|x_t, z_t)$ se v případě použití laserového scanneru počítá pouze v bodech, jež tvoří úsečku, která vychází z bodu pozice robota v odpovídajícím úhlu a končí v bodě, kde byla detekována překážka. Ostatní body mapy zůstavají nezměněny. Přestože mohou existovat i složitější metody, které aktualizují i v nejbližším okolí této úsečky, domnívám se, že pro dostatečně přesný scanner nejsou příliš zapotřebí.

Skutečný stav políčka na mapě může být buď Obsazeno (Occupied/Occ) nebo Volno (Empty). Na každém bodu mapy je uložena pouze pravděpodobnost, že místo je obsazeno. Pravděpodobnost, že místo je volné se jednoduše dopočítá:

$$P(m_i = Empty) = 1 - P(m_i = Occupied)$$

Pro aktualizaci pravděpodobnosti obsazenosti bodu můžeme použít Bayesův vzorec ve tvaru[11]:

$$P(m_i = Occ|x_t, z_t) = \frac{P(z_t|m_i = Occ, x_t)P(m_i = Occ)}{P(z_t|m_i = Occ, x_t)P(m_i = Occ) + P(z_t|m_i = Empty, x_t)P(m_i = Empty)} \tag{4.4}$$

Tento vzorec použijeme na všechny body mapy ležící v úsečce, které protíná měření. Logicky předpokládáme, že $P(z_t|m_i = Occ)$ je velké, pokud m_i je koncový bod úsečky (tj. místo, kde byla změřena překážka) a naopak tušíme, že pravděpodobnost obsazenosti

bodu, kterým laserový paprsek proletěl, bude malá. Tyto pravděpodobnosti je možné v nejjednodušším případě určit konstantou. Například:

$$P(z_t|m_i = Occ, m_i\, je\, koncovy\, bod) \quad = P_{occ} \quad = 0,8 \tag{4.5}$$

$$P(z_t|m_i = Occ, m_i\, neni\, koncovy\, bod) \quad = P_{free} \quad = 0,1 \tag{4.6}$$

Pokud algoritmus pracuje s body v mapě uloženými jako log-odds (viz algoritmus 3), je aktualizace pravděpodobnosti každého bodu velmi triviální. Rovnici 4.4 vůbec nemusíme počítat, přesto získáme stejné výsledky. Stačí nadefinovat funkci **inverse_sensor_model** a to např. tak jednoduše, jak ukazuje tento algoritmus (se zjednodušením převzato z [15]):

Algoritmus 4: inverse_sensor_model

Vstup: m_i, x_t, z_t

if z_t *neprotíná* m_i **then**
  return l_0
end
if m_i *je koncový bod* **then**
  return l_{occ}
else
  return l_{free}
end

l_{occ} a l_{free} jsou konstatny, které odpovídají log-šancím pravděpodobností P_{occ} a P_{free} z rovnic 4.5 a 4.6.

Je vidět, že použití log-šancí velmi zjednodušuje výpočet. Místo složitého výpočtu rovnice 4.4, postačí pro aktualizaci bodu mapy jedna operace součtu.

Kapitola 5

Simultánní lokalizace a mapování

5.1 Obecný algoritmus

Implemetnace SLAMu s využitím částicových filtrů je ve své nejjednodušší podobě velmi přímočará. Jedná se prakticky o rozšíření MCL lokalizace tak, že jedna částice kromě vlastní pozice a váhy obsahuje také svoji dosud vytvořenou mapu. Při vytváření mapy se pozice dá považovat za známou a je dána pravě pozicí částice. Pozice v kontextu Simultánní lokalizace a mapování označuje relativní pozici k místu, kde robot začal mapovat.

Algoritmus 5 popisuje princip obecný způsob jakým se SLAM dá pomocí částicových filtrů řešit. Tento algoritmus ukazuje společnou kostru algoritmů z mnoha různých zdrojů ([5, 8, 6, 7])

Algoritmus 5: SLAM s využitím částicových filtrů

Vstup: Množina částic χ_{t-1}, u_t, z_t
Výstup: χ_t

$\chi_t = \emptyset$
for $k = 1$ *to* M **do**
  $x_t^{[k]} =$ **predict_new_position**()
  $w_t^{[k]} = w_{t-1}^{[k]}$.**assign_weight**()
  $m_t^{[k]} =$ **update_map**()
  $\chi_t = \chi_t + \langle x_t^{[k]}, m_t^{[k]}, w_t^{[k]} \rangle$
end
normalize_weights()
if *Resampling is advantageous* **then**
  resample()
end
return χ_t

M označuje celkový počet částic. Horní index proměnných určuje konkrétní částici, která se právě používá.

Existující varianty algoritmů se od sebe liší pravě rozdíly ve funkcích **predict_new_position**, **assign_weight**, **update_map**, **resample** a také ve způsobu rozhodování, kdy má k převzorkování vůbec dojít. Význam těchto funkcí je následující:

- **predict_new_position** - Odhad nové pozice částice. Nejjednodušší je odhadovat po-

mocí pravděpodobnostního modelu pohybu založeném na odometrii (viz 3.6)

- **assign_weight** - Přiřazení váhy částici. Násobí se s původní váhou částice. Otázkou je jak vypočítat váhu částice, když neznáme celou mapou prostředí, ale pouze její dosud vytvořenou část. Obecně ale platí, že se počítá pravděpodobnost měření při znalosti pozice ($p(z_t|x_t, m_{t-1})$).
- **update_map** - Aktualizace mapy. Provádí se způsobem představeným v předchozí sekci. Existují ale i zcela odlišné metody [11], některé navržené speciálně pro SLAM [5].
- **normalize_weights** - Normalizování vah. Jejich suma musí být 1.
- **resample** - Převzorkovaní. Částice s vysokou váhou zde nahrazují částice s nízkou váhou. Pro tento účel existuje řada algoritmů, zajímavějším problémem je ale způsob, jakým se rozhoduje, zda převzorkování provést, či ne.

5.2 Predikce - odhad nové pozice

Množství částic, které algoritmus potřebuje se odvíjí právě od kvality odhadu změny pozice. Jedná se tedy o klíčovou část algoritmu. Pokud změnu pozice dokážeme odhadnout s dobrou přesností vystačíme si s malým množstvím částic, které pokrývají malou plochu.

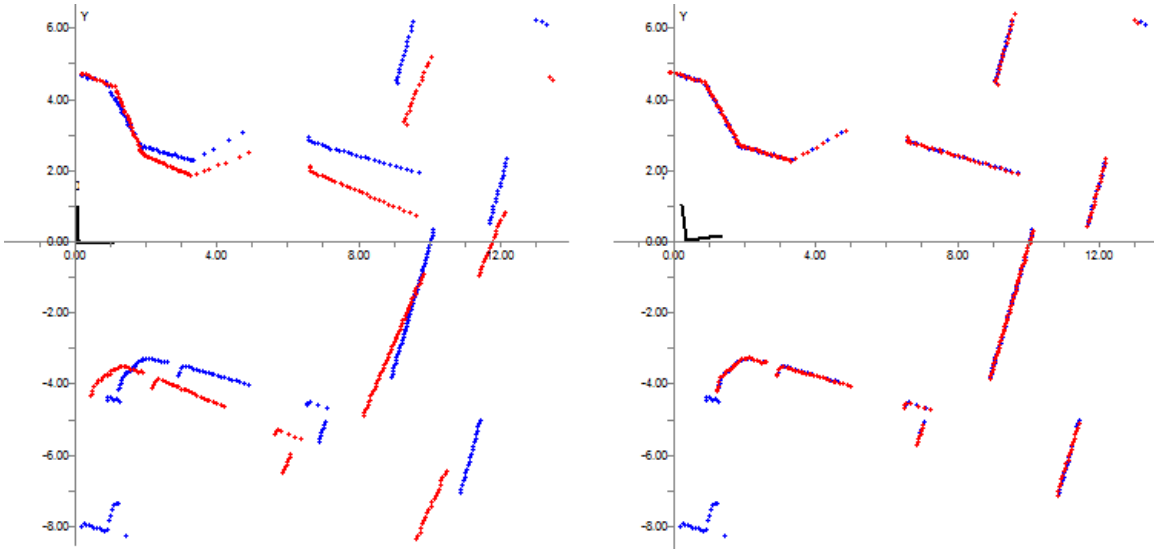
5.2.1 Klasický pohybový model

Standardní způsob odhadu pozice vyžaduje již několikrát zmíněný pravděpodobnostní model pohybu robota. Většina částic, které generujeme tímto způsobem má nízkou váhu a jedná se tedy o špatné odhady. Potřebujeme jich tedy velké množství, což představuje značnou nevýhodu. Částice s velmi nízkým váhovým ohodnocením jsou navíc prakticky zcela zbytečné a budou při nejbližším převzorkování nahrazeny částicemi s vyšší váhou. V případě lokalizace to není velký problém, protože definitvní odhad pozice se může určit váženým průměrem všech částic a převzorkování navíc nepředstavuje nic čemu bychom se chtěli vyhýbat, protože ho lze uskutečnit v relativně krátké době. Pokud ale částice obsahují i mapu, která může mít velikost i několik MB, je převzorkování velmi pomalé a je dobré ho uskutečnit jen tehdy, je-li to skutečně nezbytné a hlavně s co nejměnším počtem částic.

5.2.2 Odhad pomocí scan-matchingu

Existují naštěstí lepší metody odhadu pozice. Snad všechny z nich se spoléhají na metody známé jako scan-matching. Scan-matching je metoda při které se hledá korespondence mezi dvěma po sobě jdoucími měřeními z_{t-1} a z_t (viz obrázek 5.1), případně korespondence mezi všemi minulými měřeními $z_{1:t-1}$ a současným měřením z_t. Konkrétní implementace scan-matchingu může být realizována například algoritmem Iterative Closest Point. Popis tohoto algoritmu překračuje rámec této práce, je ale možné jej naimplementovat velmi efektivně a uplatnění proto nachází i při lokalizaci v reálném čase[2].

Vstupem zarovnávacích algoritmů je kromě dvou množin bodů také počáteční odhad. Zde je nejvhodnější použít odhad pozice pomocí odometrie, algoritmus je ale tak robustní, že počáteční odhad často vůbec nemusí být použit. Toto jsem si sám ověřil, ale algoritmus musel být nastaven tak, aby hledal korespondeci i pro vzdálenější body. Výpočet tedy trvá déle a není samozřejmně úplně spolehlivý. V monotonním prostředí bez významných prvků,



Obrázek 5.1: Obrázek vlevo ukazuje dvě měření. z_{t-1} je znázorněno modře a z_t červeně. Obě měření jsou zaneseny do grafu s pozicí robota v bodě [0;0]. Obrázek vpravo ukazuje výsledek scan-matchingu. Červené měření bylo zarováno tak, aby co nejvíc odpovídalo modrému. Z tohoto zarovnání je možné odhadnou změnu pozice.

jako je například velmi dlouhá prázdná chodba, se může lehce stát, že dvě po sobě jdoucí měření vypadají prakticky shodně. V takovém případě je možné, že i když se robot mohl posunout o mnoho centimetrů, žádný pohyb není možné pouze z porovnání měření dobře rozpoznat.

Výstupem často není pouze jedna nejlepší změna pozice ale gaussovo rozložení se středem v nejpravděpodobnější pozici. Výběrem náhodných vzorků z tohoto rozložení získáváme efektivní způsob odhadu nové pozice, snadno použitelný v částicovém filtru.

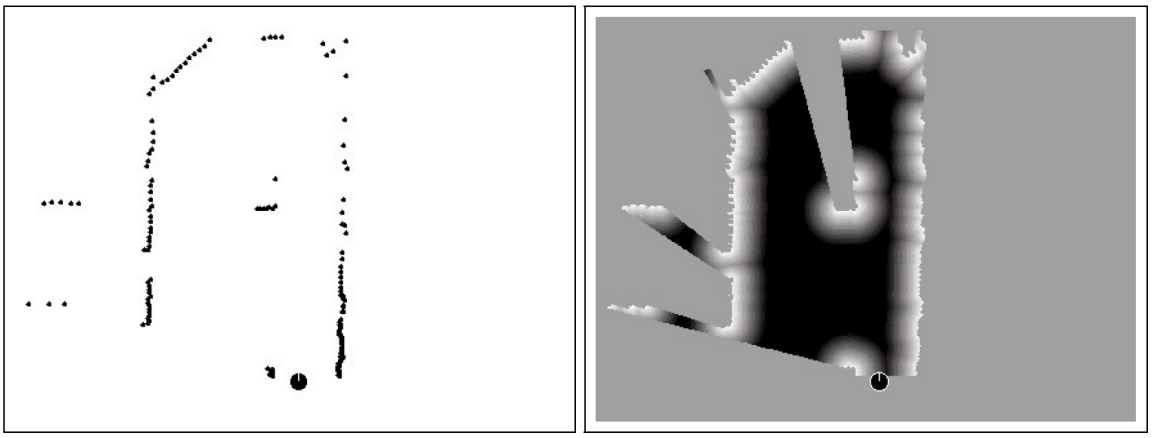
5.3 Korekce - Přiřazení vah

Váha částice se v každém kroku aktualizuje tak, že se vynásobí její předchozí váha s číslem odpovídajícím pravděpodobnosti měření. Pravděpodobnost měření se mění s pozicí částice a pro její výpočet není vhodné použít model založený na ray castingu.

Ray casting je běžný způsob výpočtu pravděpodobnosti měření, pokud máme hotovou a přesnou mapu prostředí. V případě že ale mapu teprve tvoříme, není použití ray castingu vhodné. Při mapování (obzvláště při vysokém rozlišení mapy) se totiž často stává, že například již několik metrů vzdálená zeď se do mapy v jedné iteraci mapování nezanese jako spojitá čára, ale jako několik izolovaných bodů, mezi kterými je volná, nezmapovaná plocha. Ray casting ale předpokládá dokonalou mapu prostředí, a může se proto stát, že paprsek letí mezi těmito izolovanými body a do žádného se přímým zásahem nestrefí. V blízkém okolí obsazených bodů ale intuitivně očekáváme vyšší pravděpodobnost naměření překážky.

5.3.1 Likelihood field

Přesnějších výsledků dosáhneme, pokud použijeme metodu nazvanou Likelihood field[15]. Zde se nesleduje dráha paprsku, ale zjistí se pouze koncový bod měření a pravděpodob-



Obrázek 5.2: Vlevo první měření, ze kterého byla postavena mapa. Na pravém obrázku je znázorněno odpovídající pravděpodobnostní pole. Čím světlejší místo, tím větší pravděpodobnost naměření překážky na daném místě. Obrázek převzat z [15]

nost měření je nepřímo úměrná se vzdáleností koncového bodu od nejbližšího obsazeného bodu v mapě (viz obrázek 5.2). Protože dráha paprsku není sledována, je tato metoda efektivnější než ray-casting, pro hotové mapy se může kompletní pravděpodobnostní pole navíc jednoduše předpočítat a tím se dosáhne výrazného urychlení lokalizačního algoritmu.

Nevýhodou této metody oproti klasickému modelu založeném na ray castingu je ignorování fyzikálních vlastnostní laserového paprsku. Metodou likelihood field zkoumáme pouze koncový bod měření a ignorujeme tak, jestli se nějaké překážky nachází mezi robotem a koncem paprsku. Ačkoliv se to může zdát být na první pohled velká nevýhoda, algoritmus dává v praxi dobré výsledky.

Pro funkčnost algoritmu je vpřípadě OGM mapy nutné provést několik drobných úprav. Například zvolit hranici, určující od jaké hodnoty pravděpodobnosti obsazenosti budeme bod považovat opravdu za obsazený.

5.4 Převzorkování

Algoritmus 5 pracuje s množinou vážených částic. Převzorkování je označení procesu kdy částice s nízkou váhou jsou nahrazeny částicemi s vysokou váhou. Kdy je ale skutečně nejlepší převzorkovat částice? Převzorkování je časově velmi náročný proces. Jak již bylo dříve zmíněno, je důležité mít co nejlepší odhad pozice a tedy co nejmenší počet částic, aby bylo převzorkování co nejrychlejší a a nemuselo se vykonávat tak často.

Převzorkování musí být prováděno velmi opatrně a zřídka také proto, aby se nestalo, že se zahodí dobrá částice. Tento problém je ve skutečnosti velmi častý. Může se vyskytnout i opačný problém, kdy částice s na první pohled špatnou mapou je zachována.

5.4.1 Počet efektivních částic

Nejlepší způsob, jak určit, kdy je vhodné převzorkovávat je výpočtem tzv. efektivního počtu částic N_{eff}, který byl poprvé představen v publikaci [10]. Tato hodnota je definována takto:

$$N_{eff} = \frac{1}{\sum_{i=1}^{M} (w^{[i]})^2} \tag{5.1}$$

Pokud váhy všech částic jsou shodné, N_{eff} se rovná M (počtu všech částic). Čím větší jsou odchylky jednotlivých vah, tím menší je také N_{eff}. Intuitivně se tato hodnota opravdu dá vyložit jako počet efektivních (dobrých) částic.

V každé iteraci částicového filtru tedy můžeme vypočítat N_{eff} a rozhodnout se jestli převzorkovávat právě na základě této hodnoty, například tehdy když N_{eff} je menší než $M/2$.

5.4.2 Select with Replacement

Metod převzorkování existuje několik, zde popíši pouze nejrozšířenější a nejjednodušší metodu zvanou Select with Replacement. Volně převzato z [12].

Tento algoritmus vybírá částice které budou vybrány pro znásobení s pravděpodobností odpovídající jejich váze. Nejdříve se vytvoří pole, obsahující průběžnou sumu vah částic. Je zřejmé, že posledním prvkem tohoto pole bude hodnota 1. Dále se vytvoří M náhodných čísel v rozsahu $(0; 1)$ a tyto čísla se uloží seřazeně do pole. Na konec tohoto pole přidáme ještě číslo 1. Obě pole obsahují rostoucí posloupnost čísel od nuly do jedné. Počet náhodných čísel, které se vyskytují v intervalu mezi dvěma průběžnými sumami určje, kolikrát se částice na daném indexu znásobí. Pokud má částice malou váhu, její efekt na průběžnou sumu je malý a není tedy příliš pravděpodobné, že přežije, protože v odpovídajícím intervalu se nejspíše žádné náhodné číslo nevygenerovalo. Následující algoritmus dává formálnější vysvětlení (pole začínají na indexu 0):

Algoritmus 6: Select with Replacement

Vstup: Pole všech normalizovaných vah $W[M]$
Výstup: Pole obsahující indexy částic po převzorkování $Index[M]$

$Q[] = cumsum(W)$ cumsum - vytvoření průbežné sumy
$t[] = rand(M)$
$T[] = sort(t)$
$T[M] = 1; i = 0; j = 0;$
while $i < M$ **do**
    if $T[i] < Q[j]$ **then**
        $Index[i] = j$
        $i = i + 1$
    else
        $j = j + 1$
    end
end
return *Index*

Kritickou částí algoritmu je seřazení pole, minimální složitost algoritmu je $O(M \log M)$. Exitují také rychlejší nebo za jistých podmínek robustnější algoritmy pro převzorkování. Několik jich je uvedeno v [12].

Časově nejnáročnější část převzorkování je ale její poslední část, a to samotná kopie částic. Kopie desítek map o velikosti několik MB může být opravdu dlouhý proces. V [5] byl navržen speciální způsob reprezentace map, který dokáže proces převzorkování výrazně urychlyt. Pokud ale používáme málo částic, není to vyloženě zapotřebí.

Kapitola 6

Implementace

V této kapitole je popsán způsob implementace výše zmíněné teorie do funkčního programu. Následující sekce popisuje první pokusy o implementaci některých zde uvedených algoritmů. Program, který je hlavním cílem bakalařské práce je popsán v dalších sekcích a je mu také věnována samostatná kapitola týkající se několika experimentů a popisující vzniklé mapy. Tento program dokáže načítat datové sady v jednom z často používaných formátů a vytvořit z nich mapu prostředí. Tyto datové sady obsahují pouze sekvenci odometrických dat a měření laserovým scannerem.

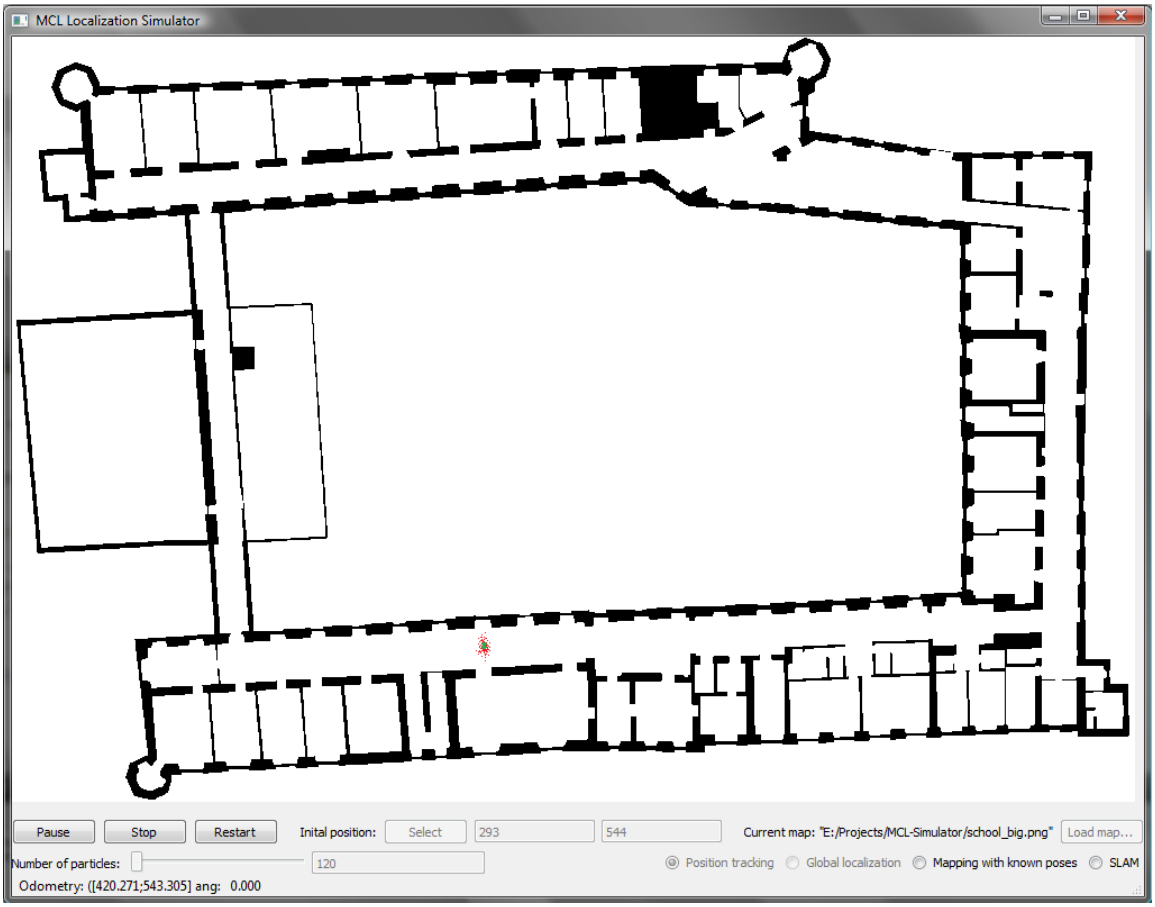
6.1 První pokusy

Jako úplně první mezicíl jsem si stanovil vytvořit jednoduchý simulátor robota, díky kterému bych si prokazatelně ověřil, že mnou naimplementované algoritmy jsou funkční. V tomto simulátoru jsem prováděl veškeré počáteční experimenty s algoritmem MCL, který jsem použil jako solidní základ pro budoucí částicový SLAM. Po mnoha pokusech se mi podařilo naprogramovat funkční MCL lokalizaci schopnou jak sledovaní pozice, tak globalní lokalizace. Tento simulátor využívá pouze knihovny Qt pro tvorbu uživatelského prostředí a interakci s uživatelem. Žadné další knihovny nebyly použity. Ukázka grafického rozhraní simulátoru je na obrázku 6.1

Dalšími pokusy jsem se MCL lokalizi snažil rozšířit na funkční SLAM. Zde jsem si uvědomil mnohé problémy, které se s tímto pojí. Například jak špatně funguje ray casting nad neúplnými mapami nebo jak nevhodné je odhadování pozice pouze pomocí pohybového modelu. V tomto okamžiku jsem se rozhodl většinu dosavadního kódu zahodit a začít znovu použitím knihovny, která implementuje přinejmenším scan-matching. Potenciálními kandidáty byla knihovna Carmen (Carnegie Mellon Robot Navigation Toolkit) a knihovna MRPT (Mobile Robot Programming Toolkit). Knihovna Carmen je velmi oblíbená a byla použita v několika pracech, které jsem studoval (např. [7, 8]). Nakonec jsem si ale zvolil knihovnu MRPT, která je na rozdíl od Carmenu objektově orientovaná a multiplatformní. Nevýhodou této knihovny je její v současné době slabší dokumentace, její použití je ale na druhou stranu velmi intuitivní.

6.2 smp-slam

Pokusy o tvorbu map v simulovaném prostředí jsem odsunul stranou a rozhodl jsem se pracovat již přímo s realnými datovými sadami, což mi dávalo jistotu, že naimplementované



Obrázek 6.1: Jednoduchý simulátor naprogramovaný v C++/Qt běžící v módu sledovaní pozice pomocí algoritmu MCL. Načtená mapa je zjednodušenou mapou Fakulty informačních technologií (areál Božetěchova 2)

algoritmy si poradí opravdu v reálných podmínkách.

Rozhodl jsem se vytvořit program, který dokáže vytvářet kvalitní mapy tak rychle, aby jej bylo moci použít i v realném čase se skutečným robotem. Bylo tedy nutné použít efektivní algoritmy.

Program byl vytvořen v jazyce C++ s použitím knihovny MRPT (Mobile Robot Programming Toolkit). Tuto knihovnu jsem si vybral, protože obsahovala implementaci algoritmu ICP, jenž bych zřejmně sám naprogramovat nedokázal. Tato knihovna obsahuje i implementaci několika jiných zde popsaných algoritmů a některé z nich jsem se rozhodl využít.

Knihovna MRPT využívá mnohých funkcí z knihovny pro tvorbu uživatelského rozhraní wxWidgets a grafické knihovny OpenGL. Díky tomu bylo velmi jednoduché vytvořit pěkný 3D náhled zobrazující průběh mapování.

Program jsem pracovně navzal smp-slam (Scan-matched particle SLAM).

6.3 Odhad pozice

Pro odhad pozice jsem využil možností ICP algoritmu knihovny MRPT. Tento odhad knihovna vrací v datové struktuře, která reprezentuje gaussovo rozložení a je z ní možné

zabudovanou metodou jednoduše vybrat náhodný vzorek. Tato vlastnost velmi usnadnila implementaci aktualizaci pozice. Pokud by algoritmus vracel pouze jeden nejlepší odhad pozice namísto pravděpodobnostního rozložení, bylo by zapotřebí toto rozložení vytvořit kolem vrácené pozice uměle.

Jako referenční mapu pro ICP algoritmus jsem se rozhodl nevybrat pouze předchozí měření, ale všechna předchozí měření. V průběhu mapování se tak vytváří vedle OGM mapy mapa druhá, obsahující pouze prostý seznam bodů. Každé částici proto přiřazuji vedle OGM mapy i tento seznam bodů, který používám pouze jako referenční mapu pro ICP algoritmus. Algoritmus pak dává lepší výsledky, než pokud by se zarovnávalo jen na základě předchozího měření[1]. Nevýhodou tohoto řešení, jak jsem později s nelibostí zjistil, je stoupající časová náročnost výpočtu zarovnání s tím, jak se tento seznam bodů rozrůstá. Částečné řešení poskytuje opět knihovna MRPT, která umožňuje fůzi bodů, které se v mapě vyskytují blízko sebe. Body jsou navíc v tomto seznamu uloženy v efektivní datové struktuře (kd-tree), která dokáže rychle najít k libovolnému bodu jeho nejbližšího souseda, což je operace, kterou algoritmus ICP používá velmi často.

Pěknou ukázku, jak funguje algoritmus ICP pro zarovnávání dvou měření je možno shlédnout v programu RawlogViewer, jenž je součástí knihovny MRPT. Je zde možné vybrat datovou sadu a po kliknutí na ikonku „ICP“ z ní vybrat dvě měření, jenž se pomalu v několika iteracích zarovnají (volba „Animate step by step“).

6.4 Mapování

Knihovna MRPT obsahuje třídu reprezentující OGM mapu a mnoho užitečných funkcí pro práci s touto třídou. Jedná se například o aktualizaci mapy (algoritmus 3), dynamické zvětšovaní mapy, alokaci a dealokaci paměti a kopírovaní mapy (použito při převzorkování). Jedna z funkcí umožňovala také zjistit pravděpodobnost měření při dané pozici robota s použitím několika různých metod, mezi nimiž nechyběl ray casting ani likelihood field.

Práce s touto knihovnou byla v této oblasti velmi intuitivní a dobře vyřešená.

6.4.1 Frekvence mapování

V rámci optimalizace rychlosti tvorby mapy je také možné v konfiguračním souboru zvolit možnost, která lehce upraví mapovací algoritmus. Běžné chování algoritmu je, že v jedné iteraci se lokalizuje robot a poté se zanesou nová měření do mapy. Program umožňuje volitelnou úpravu algoritmu, která měření do mapy nezanáší při každé iteraci, ale pouze při každé obecně n-té iteraci. Rychlost algoritmu se takto zvýší a vzniklé mapy jsou dokonce kvalitnější a přesnější. Toto nečekané zlepšení kvality si vysvětluji dvěma faktory.

Za prvé tím, že každá aktualizace OGM mapy ovlivňuje výsledek vracený při výpočtu pravděpodobnosti měření. Může například nastat situace, kdy se jedna částice v jedné iteraci posune na místo, ve kterém je malá pravděpodobnost měření. Poté se mapa zaktualizuje. V další iteraci již tato pravděpodobnost ale bude opět velká, protože mapa, která se pro výpočet této pravděpodobnosti počítá, byla nesprávně pozměněna. Dá se tedy říci, že části mapy, jež byly aktualizovány dávněji by ideálně výpočet pravděpodobnost měření měly ovlivňovat víc, než nově zmapované části, protože podléhají menší kumulované chybě.

[1] Je to znatelné a výhodné zejména v případě, kdy robot vstupuje do známého prostředí z neznámého místa - tzv. uzavírání smyčky. Pouze ale za předpokladu, že je snížena frekvence mapování. Toto je v práci dále vysvětleno.

Snížením frekvence mapování tento ideální výpočet pravděpodobnosti měření sice aproximuji jen velice hrubě, výsledky jsou ale přesto lepší.

Méně častá aktualizace se týka i mapy referenční pro scan-matching, což z podobného důvodu zvyšuje kvalitu odhadu pozice, zejména při uzavírání smyčky prostředí. Více podrobností je uvedeno v kapitole týkající se experimentů.

6.5 Souhrn použitých algoritmů

V následující tabulce jsou pro přehlednost uvedeny všechny hlavní algoritmy použité v mojí implementaci.

popis	**zvolené řešení**	**implementace**
SLAM obecně	Částicový filtr	Vlastní
Odhad změny pozice	Scan-matching (ICP)	MRPT
Vypočítaní pravděpodobnosti měření	Likelihood Field	MRPT
Algoritmus převzorkování	Sample with Replacement	Vlastní
Kdy převzorkovávat?	Výpočet N_{eff}	Vlastní

6.6 Grafické rozhraní a ovládání

Program při spuštění očekává předání dvou parametrů z příkazové řádky. První parametr vybírá vstupní datovou sadu určenou pro zpracování (formát viz 6.7.1). Druhým parametrem uživatel specifikuje název výstupní mapy. Program v současnosti podporuje ukládání hotové mapy jako obrázek ve formátu png. Intenzita pixelu odpovídá pravděpodobnosti obsazenosti. Tmavší pixely mají vyšší pravděpodobnost a naopak.

Veškeré ostatní nastavení uživatel zadává přes konfigurační soubor `config.ini`. Mezi nejdůležitější nastavení patří určení počtu částic, rozlišení mapy a zvolení režimu mapování.

Bezprostředně po zpracovaní parametrů programu a zpracovaní konfiguračních údajů program započne tvorbu mapy. Průběh této tvorby může uživatel sledovat v okně s 3D náhledem na vytvářenou mapu (viz obrázek 7.5). Toto okno je také jediným oknem, které program vytváří. Dodatečné informace o mapování může uživatel sledovat v konzolovém okně, ze kterého byl program spuštěn.

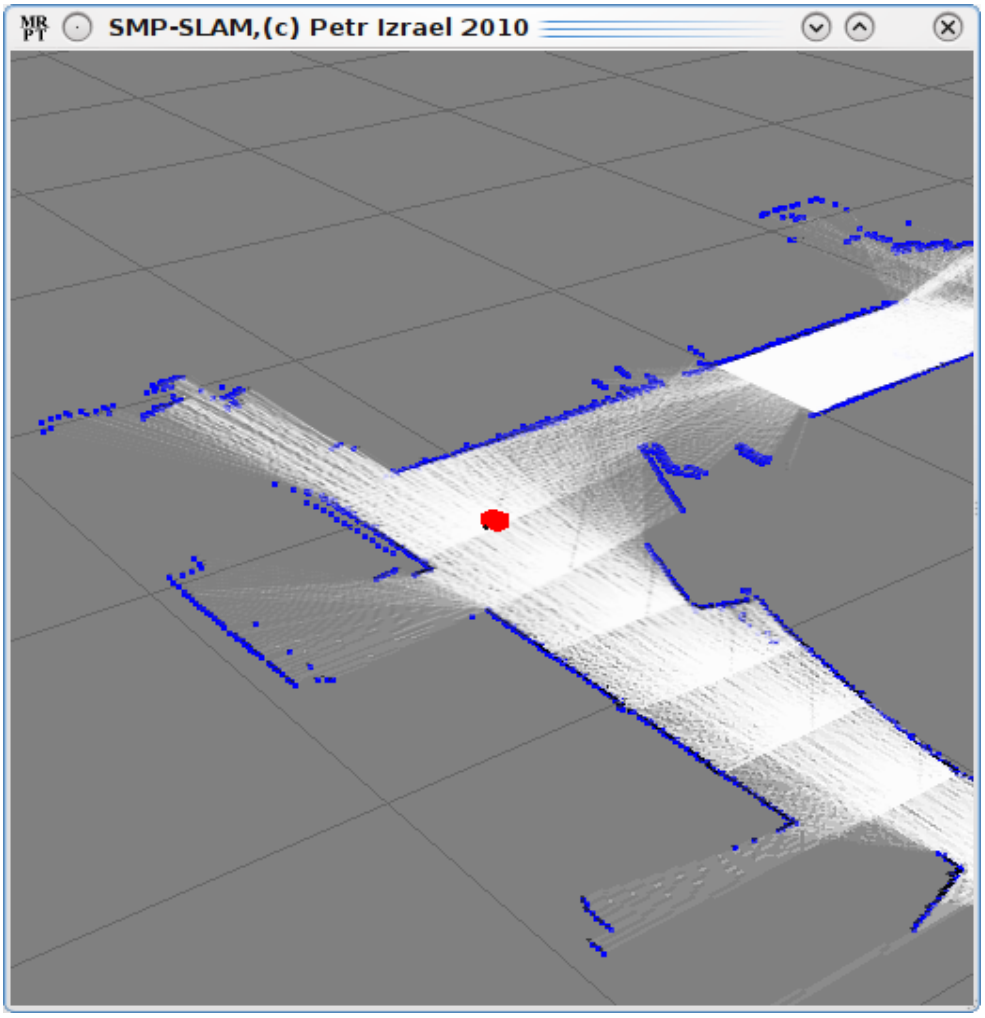
Grafický náhled zobrazuje vždy mapu částice s nejvyšším váhovým ohodnocením. Pro jeho vytvoření bylo zapotřebí pouze několika funkcí knihovny MRPT, která velice jednoduše vytváření podobných náhledů umožňuje.

6.6.1 Režimy mapování

Mapování může být spušťeno ve třech základních řežimech:

1. Pouze odometrie: Pozici robota určuje přímo odometrie. Takto vznikají obvykle velmi nepřesné mapy, připomínající skutečné mapované prostory pouze velmi vzdáleně. Tento režim slouží pro demonstraci nepřesnosti odometrie.

2. Pouze scan-matching: Pozice robota se urči pouze na základě scan-matchingu. První iterace scan-matchingu je inicializována odometrickým odhadem. Takto mohou vzniknout relativně dobré mapy, ale pouze jednoduchých prostředí. V tomto režimu je



Obrázek 6.2: Náhled na hlavní a jediné okno mapovacího program. Zobrazena je část mapy s nejvyšším váhovým ohodnocením v okamžiku uzavírání smyčky. Modré body překrývající OGM mapu představují referenční mapu pro scan-matching.

velký problém uzavírání velkých smyček v prostředí a mohou tak vzniknout topologicky nepřesné mapy.

3. Částicový filtr: Hlavní a výchozí režim,vytvářející nejpřesnější mapy.

6.7 Datové sady

Rozsáhlý archív datových sad lze najít naříklad na internetových stránkách [9] nebo [3]. Na první zmíněné stránce se nachází mnoho datových sad, bohužel každá z nich má odlišný formát. Odhadem nejčastějším formátem je zde vnitřní formát knihovny Carmen. Já si ale zvolil knihovnu MRPT. Ta má také svůj vlastní formát datových sad, tzv. "rawlog". Velké množství datových sad v tomto formátu lze najít ve druhém zmíněném zdroji.

6.7.1 Formát rawlog

Soubor ve formátu MRPT rawlog je binární sekvence dvojic datových typů `CActionCollection` a `CSensoryFrame`. První jmenovaný typ funguje jako kolekce informací o akcích robota. Nás zajíma konkrétně informace o pohybu, tento datový typ prakticky vždy obsahuje odometrický odhad změny pozice. `CSensoryFrame` obsahuje seznam měření robota. Může tedy obsahovat i záznamy z více laserových scannerů nebo jiných typů senzorů (sonar, kamera...). Rawlog je možné knihovními funkcemi načítat a extrahovat z něj pouze odometrická data a laserové měření.

6.7.2 Podpora jiných formátů

Protože v rawlogu jsem našel víc datových sad, než pro otestování funkčnosti programu opravdu potřebuji, neimplementoval jsem podporu žádných jiných formátů. Zdrojový kód je ale navržen tak, aby bylo možné bez větších potíží program rozšířit o podporu jiných formátů, či dokonce o získávání dat ze skutečného robota. Program je navržen objektově, podporu jiných formátů je možno do programu začlenit děděním z jisté abstaktní třídy a definováním virtuální metody, jenž má data obstarávat.

6.8 Možná vylepšení

Zde představuji pár možných vylepšení programu, které jsem neimplementoval, ale rozhodně stojí za úvahu v případě dalšího pokračování této práce.

1. Paralerizace - Algoritmus by bylo možné poměrně jednoduše paralelizovat, každé jádro procesoru by se staralo o přidělený počet částic. Nárust výpočetního výkonu by tak byl znatelný.

2. Vylepšení predikce - Předpokládám, že kombinace scan-matchingu s pohybovým modelem je nejlepším způsobem odhadu pozice. Možný způsob implementace této myšlenky byl představen v [7] a výsledky jsou pozoruhodné, potřebný počet částic pro generování srovnatelně kvalitních map byl podle autorů až o jeden řád menší než při řešení pouze využitím scan-matchingu.

6.9 Existující programy

Při práci na tomto projektu jsem se často inspiroval články popisují již hotová řešení problému. Jedná se zejména o publikace [8, 6, 7]. Jména autorů těchto článků (Sebastian Thrun, Wolfram Burgard, Giorgio Grisetti a další) se často objevují ve spojitosti s robotikou, lokalizací a SLAMem a jedná se o tvůrce mnoha inovativních řešení v této oblasti. Všechny tři zmíněné články popisují řešení, které pochopitelně dávají lepší výsledky než moje implementace. Společným jmenovatelem těchto prací je odhad pozice založený na scan-matchingu a použití knihovny Carmen pro implementaci programu. Zdrojové kódy těchto programů jsou dostupné na internetové stránce [1][2].

[2] Táto stránka obsahuje základní inforamace o problematice SLAMu a zejmena je sbírkou několika různých funkčních implementací. Obshuje také odkazy na datové sady a další užitečné informace.

Kapitola 7

Experimenty a výsledky

Funkčnost a robustnost mého programu jsem testoval na datových sadách dostupných na [3]. Program jsem testoval na sestavě Intel Core(TM) 2 Duo 2,2Ghz; 4 GB RAM.

Následující testy byly provedeny nad datovou sadou „Interior of the Intel Research Lab in Seattle“, která byla podle autorů článku [7] nahrána za 45 minut. Veškeré výsledky, které byly získany pod tuto časovou hranici nazančují možnost použití algoritmu v reálném čase. Tato sada se ukázala pro svoji rozmanitost prostředí vhodnou pro mnoho experimentů a testů. Výsledky tvorby map z dalších datových sad jsou k dispozici na přiloženém CD.

7.1 Vliv frekvence mapování na kvalitu map a rychlost

Jak bylo načnačeno v sekci 6.4.1, velmi značný vliv na výsledek algoritmu a jeho rychlost má frekvence mapování. Tou se rozumí, jak často se provádí aktualizace map. Lokalizace se provádí v každé iteraci částicového filtru, kdežto mapování můžeme provádět pouze v některých (n-tých) krocích, což má na kvalitu algoritmu velmi dobrý vliv. Tento fakt se mi z počátku zdál být nelogický a důvody jsem hledal až zpětně, poté co jsem tyto experimenty provedl. Obrázek 7.1 jasně znázorňuje o jak velké zlepšení se jedná. Následující tabulka shrnuje výsledky pokusů. Každý pokus byl proveden minimálně dvakrát, protože výsledná mapa je do jisté míry dílem náhody. Uvedené výpočetní časy jsou průmerné. Obrázky dokumentující pokus (7.1 a 7.2) byly vybrány tak, aby nejlépe vystihovaly obecné chyby, které se při tvorbě s daným nastavením vyskytovaly.

Je tedy zřejmé, že vysoká frekvence mapování není výhodná, protože výpočet trvá déle a mapy jsou nekvalitní. Příliš nízká frekvence ($n > 10$), již je ale pochopitelně také nevhodná, zejména pokud robot mapuje rychle a nezdržuje se dlouho na jednom místě. Ideální je najít vhodnou frekvenci. Hodnota $n = 5$ se zdá být vhodná a při dalších experimentech jsem frekvenci nastavoval obvykle takto. Pochopitelně vždy do jisté míry záleží na datové sadě. Tedy na tom, jak se robot během mapování chová a pohybuje, v jakých časových intervalech jsou aktualizována odometrická data a měření a podobně. Více uživatelsky přijatelné by také bylo, kdyby se namísto n zadávala vzdálenost, kterou robot musí urazit, aby byla provdená nová aktualizace mapy.

Na možnost snížení frekvence mapování můžeme také pohlížet jako na výhodnou výměnu. Mapy jsou při snížené frekvence sice kvalitnější ve smyslu jakési topologické přesnosti, jsou ale méně kvalitní ve smyslu množství obsažené informace (mnoho měření se zkrátka zahazuje a v mapě neprojeví). Nejideálnější je pochopitelně mapovat v každém kroku. Takto je ale nutné použít velké množství částic a výpočetní čas tohoto přístupu je příliš

M	n	čas	subjektivní kvalita mapy
1	1	4:42	Mapa je velmi nekvalitní, netopologická, nedává smysl. Velmi vzdáleně připomíná skutečný prostor.
1	5	4:10	Mapa dobře připomíná skutečné prostory, obsahuje ale řadu významných nepřesností
1	10	3:42	Výsledky se velmi lišily pokus od pokusu, obecně ale byly dosti špatné. Prostory, ve kterých se robot nezdržoval dlouho jsou zmapovány pouze velmi povrchně.
20	1	54:06	Zvýšení počtu částic s mapováním v každém kroku přináší sice zlepšení, mapa je ale pořád velmi špatná. Vidíme ale, že některé místnosti, kde se robot nezdržoval dlouho, jsou zmapovany lépe, než je tomu při nízké frekvenci mapování.

Tabulka 7.1: Výsledek experimentu vlivu frekvence mapování na kvalitu a rychlost. M označuje počet částic. n je perioda mapování (1/frekvence). Čas uveden v [m:s].

dlouhý. Na základě provedených experimentů těchto i dalších odhaduji, že výpočet kvalitní mapy s mapováním v každém kroku by mohl trvat i 8 hodin. Oproti tomu kvalitní mapa s mapováním v každém pátém kroku byla vypočítána již za 42 minut, jak je možno vidět v tabulce 7.2.

7.2 Zvyšování počtu částic

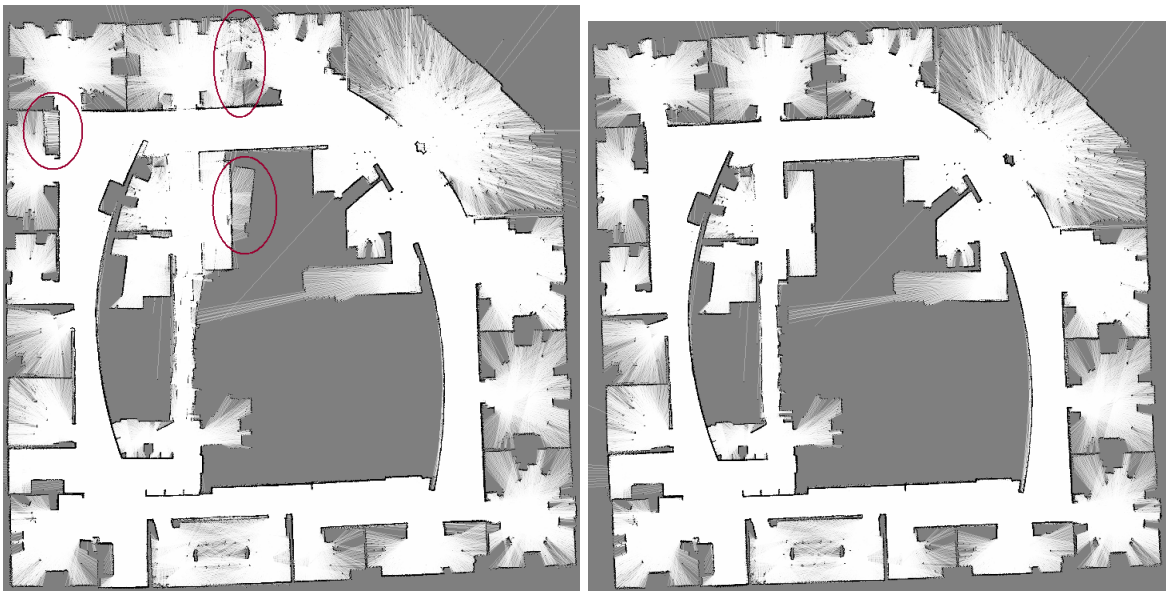
Počet částic má rozhodující vliv na kvalitu map. Bohužel taky značně negativní vliv na výpočetní čas. Následující pokusy byly provedeny s následujícím nastavením: frekvence mapování je 1 každých 5 iterací, rozlišení mapy je $4cm^2$ a hranice N_{eff} pro převzorkování je 0,6 krát počet částic.

M	čas	subjektivní kvalita mapy
20	00:13:24	Mapa na pohled dobrá, uzavření velké smyčky ale nebylo příliš dobré.
70	00:42:45	Mapa je poměrně kvalitní. Ačkoliv je poznat, že odhadnutá trajektorie robota nebyla zcela přesná, mapa je nakonec topologicky zcela vpořádku, uzavření velké smyčky (40x40 metrů) bylo provedeno odhadem (subjektivním) s chybou kolem 4-8 centimetrů.
150	01:28:38	Mapa je v porovnání s předchozí ještě o něco lepší. Mnoho částí je zmapováno přesněji, mapa je „ostřejší“, uzavření smyčky vypadá takřka bezchybně. Lépe vypadá také chodba vedoucí napříč středem laboratoře

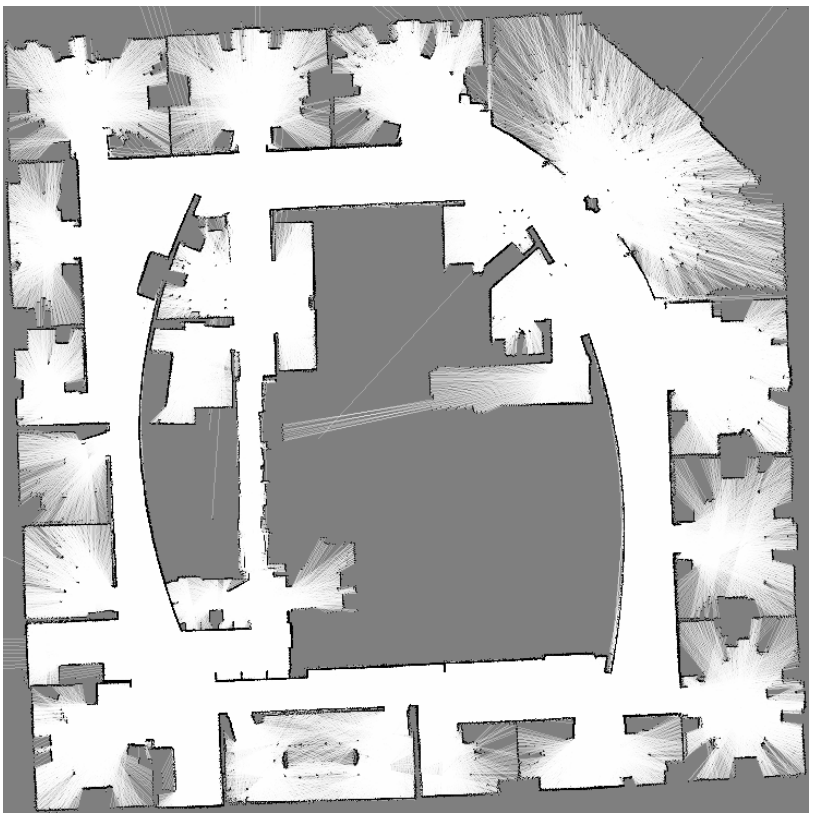
Tabulka 7.2: Výsledek experimentu vlivu počtu částic na kvalitu a rychlost. M označuje počet částic. Čas výpočtu uveden v [h:m:s].

Obrázek 7.1: Srovnání kvality výsledné mapy při rozdílném nastavení frekvence mapování. Oba obrázky byly vygenerovány s použitím pouze jedné částice. Úspěch algoritmu je tak zcela závislý na kvalitě predikce (odhadu pozice). Obrázek vlevo zobrazuje mapu vytvořenou s mapováním v každé iteraci. Druhý obrazek byl vytvořen s mapováním pouze v každé páté iteraci ($n = 5$). Z tohoto obrázku je již jasně patrné, jak zhruba mapované prostředí vypadá. Rozlišení map je $4cm^2$.

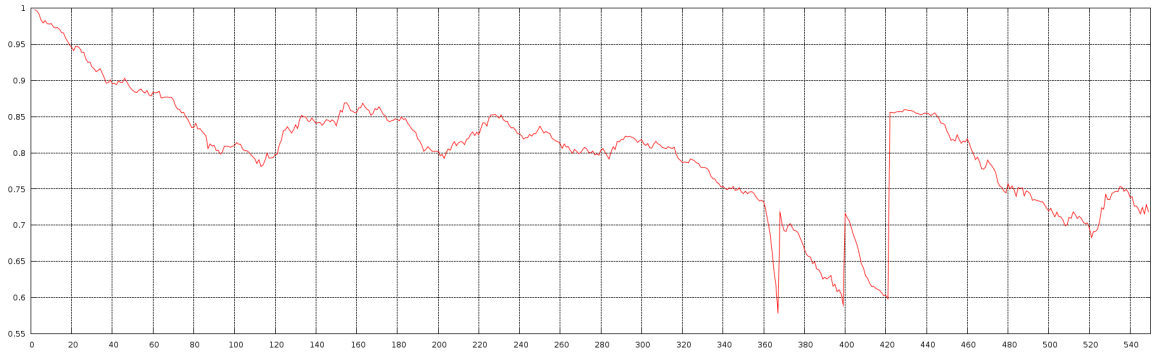
Obrázek 7.2: Obrázek vlevo: V tomto případě byla opět použita jedna jen částice a $n = 10.$, mapa je velmi nekvalitní vlivem až příliš nízké frekvence mapování. Mapa napravo vznikla mapováním v každé iteraci, odhad pozice byl tedy horší. Bylo použito 20 částic pro vylepšení kvality. To ale stále nebylo dostatečné.



Obrázek 7.3: Vlevo nejlepší mapa vzniklá použitím 20 částic. Červeně jsou zvýrazněné hlavní nepřesnosti mapy. Vpravo již velmi kvalitní mapa vzniklá použitím 70 částic.



Obrázek 7.4: Kvalitní mapa vzniklá použitím 150 částic



Obrázek 7.5: Vývoj N_{eff} v čase. Prvních 550 iterací z celkových asi 2600.

7.3 Vývoj N_{eff} v čase a uzavírání smyček

Zajímavé je sledovat, jak se chová proměnná N_{eff} v průběhu mapování. Podle vzhledu grafu lze v první řadě přibližně určit, jak dobře algoritmus odhaduje novou pozici. Čím rychlejí klesá N_{eff}, tím horší je odhad nové pozice[7]. Samozřejmně pouze pokud porovnáváme rozdílné způsoby odhadů pozic s jednou stejou metodou výpočtu pravděpodobnosti měření. Pokud by teoreticky všechny částice odhadovali pozici naprosto dokonale, byly by jejich váhy stejné a N_{eff} by bylo vždy rovné počtu částic. Naopak pokud by odhad pozice byl založen na velké náhodě, jako je tomu například v případě použití pohybového modelu, váhy částic by byly velmi rozdílné a N_{eff} by rychle klesalo. Graf 7.5 zobrazuje vývoj N_{eff} v prvních 550 iteracích algoritmu při mapování. Pokud N_{eff} klesne pod kritickou hranici, která byla v tomto příkladě stanovena na $0,6$, dochází ke převzorkování a N_{eff} se v tomto okamžiku skokově zvýší.

K velkému poklesu N_{eff} dochází zejména při uzavírání smyček, čili ve chvíli, kdy robot navštíví známé prostředí, ale vstupuje do něj z neznámého místa. Po dobu uzavíraní smyčky N_{eff} klesá velmi rychle. Tato je na grafu vidět přibližně od iterace č.360 po iteraci č.420. V této chvíli robot uzavírá velkou smyčku laboratoře. Mnoha částicím klesá váha, protože byly špatně lokalizované, což nebylo možné ověřit, dokud se smyčka neuzavřela. Po uzavření smyčky obvykle N_{eff} výrazně stoupne.

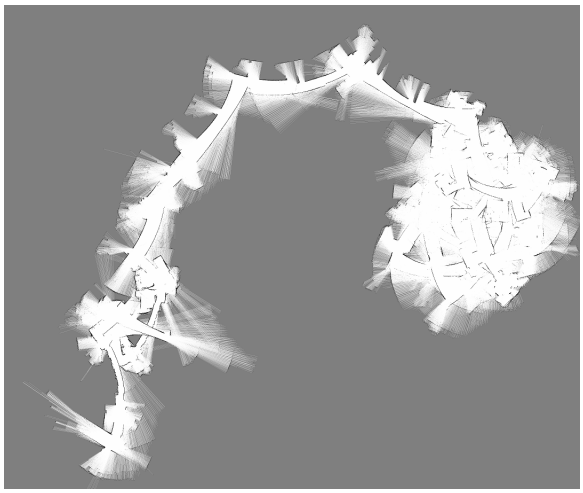
Kontrolou bylo nicméně zjištěno že kolem 10% částic po uzavření smyčky obsahuje mapy, které mají smyčku uzavřenou špatně. Zbytečne se tedy plýtvá výpočetním časem a částicemi pro ukládání a další aktualizaci očividně nedobrých map. Může se navíc stát, že tyto špatné mapy dlouhý čas odpočívají nevyužity, ale algoritmus jim později přiřadí dobrou váhu (většinou pokud se dlouho mapují jiné prostory, které se špatnou částí mapy nesouvisí). Tento problém jsem se snažil vyřešit jiným nastavením parametrů algoritmu likelihood field, nicméně toto k zdárnému řešení nevedlo. Dalším řešením by mohlo být detekovat smyčku přímo podle chování jednotlivých pravděpodobnostní měření (nikoliv vah; váhy kromě pravděpodobnosti měření ovlivňuje i váha v předchozím kroku). Dá se například předpokládat, že při uzavíraní smyčky klesá aritmetický průměr pravděpodbnosti, či je jinak ovlivněna jiná statistická veličina. Sledováním této veličiny by mohlo být možné přesněji detekovat okamžik uzavírání smyčky a následně se rozhodnut vypustit částice s nízkou pravděpodobností měření.

U velmi dobrých algoritmů, jako např. [7] klesá podle autorů N_{eff} velmi pomalu a jediný okamžik kdy klesne rychle je práve ve chvíli, kdy se uzavírá smyčka. Převzorkování se tak vykonává opravdu pouze při uzavírání smyček a nikdy jindy. Tím je zabráněno tomu, aby

byly zahozeny dobré částice.

7.4 Naivní přístup

Pro úplnost zde uvádím, jak vypadá mapa vytvořena s odhadem pozice založeným čistě na odometrii, bez využití jakýchkoli lokalizačních technik, scan-matchingu či částicových filtrů.



Obrázek 7.6: Mapa s pozicí robota určenou pouze odometrií. Skutečné prostory nepřipomíná snad ani vzdáleně.

7.5 Zhodnocení pokusů

Zde popsané pokusy považuji za nejdůležitější, ačkoliv k vyladění algoritmu jich bylo provedeno mnohem více. Mezi další zjištění patří, že scan-matching, tak jak jsem jej použil já - tedy zarovnání aktuálního měření vůči celé historii měření, dokáže poskytnout opticky lepší uzavření smyčky. Z vizualizace tvorby mapy je mnohy patrné, že v případě ne uplně přesného uzavření smyčky (např. když je použito málo nebo jen jedna částice) robot při uzavírání nápadně skokem svoji pozici změní. Tímto se pozice uměle upraví tak, aby odpovídala dřívějším měřením. Chyba v mapě tak existuje, je ale méně nápadná a opticky zanedbatelná. Toto ovšem platí pouze při snížení frekvence mapování. Při mapování v každé iteraci prakticky není možné z výhody, že má robot k dispozici celou historii měření jako referenční mapu pro scan-matching, těžit. A to proto, že je okamžitě přepsána novým měřením. Přesto nechci tvrdit, že toto řešení je nejvhodnější. Dokonce se obávám, že tyto skoky v pozici při uzavíraní smyček jsou zodpovědny za přežití oněch zhruba 10% špatných částic během převzorkování. Při malém množství částic to ale kvalitu map bez pochyby zvyšuje. Bohužel, čím víc se referenční mapa rozrůstá, tím pomalejší je scan-matching. U velmi velkých map může tedy dojít k tomu, že jedna iterace algoritmu se zpomalí natolik, že by již nebylo možné použít algoritmus v reálném čase s robotem. Tento problém při porovnávání nového měření pouze s jedním posledním neexistuje.

Bylo také ověřeno, že algoritmus zpravidla podává lepší výsledky s větším počtem částic. Doba trvání algoritmu je na počtu částic závislá přibližně lineárně. Dvojnásobné zvýšení

počtu částic prodlouží výpočet přibližně dvakrát. Pro hrubou představu o vzhledu mapovaného prostředí obvykle také postačí použití pouze několika málo částic, člověk chyby v mapě rozpozná a dokáže si je odmyslet.

Kapitola 8

Závěr

V této práci jsem shrnul základy simultánní lokalizace a mapování použitím laserového scanneru pro vytváření mřízkových map s pravděpodobností obsazenosti. Při řešení tohoto problému se nemůžeme spolehávat při určování pozice robota pouze na odometrii, jenž podléhá velkému množství chyb. Byl proto představen pravděpodobnostní přístup a zejména algoritmus Monte Carlo lokalizace (MCL). Tento algoritmus funguje dobře nad kompletními mapami prostředí a je založen na částicovém filtru, kdy jednotlivé částice mohou být taženy i z velmi nepřesného pohybového modelu robota a výsledky jsou přesto při použití většího množství částic kvalitní.

MCL je možné pro účely SLAMu rozšířit tak, že každé částici přiřadíme vlastní mapu, jenž je postupně aktualizována. Toto řešení je velmi paměťově a časově náročné, je proto nutné redukovat počet částic na minumum. Častým vylepšením je zpřesnění predikce použitím scan-matchingu, kdy je pravděpodobnosti rozložení pozice určeno nejen z odometrických dat, ale je také zpřesněno porovnáním současného měření s minulými.

V práci byla také představena vlastní implementace algoritmu, kdy aktuální měření porovnávám s celou historií minulých měření. Toto řešení umožňuje tvořit opticky dobře vypadající mapy s použitím malého množství částic. Program byl implementován v jazyce C++ a je založen na knihovně MRPT. Je schopen načítat datové sady v jednom z používaných formátů a zdrojový kod je předpřipraven pro implementaci načítání dalších, případně použití přímo s realným robotem. Rychlost algoritmu je dostačující pro použití v reálném čase.

Nevýhodou implementovaného řešení je zvyšující se časová náročnost každé další iterace, pokud se mapa příliš rozrůstá do velikosti. V takovém případě by bylo vhodnější zvolit jednodušší formu scan-matchingu. Program by mohl být také vylepšen o kvalitnější odhad pozice, jenž by spojoval scan-matching s pohybovým modelem. Nabízí se také zvýšení výkonu paralerizací algoritmu, jenž by měla být v tomto případě relativně bezproblémová.

Práce na projektu byla v mnoha směrech obohacující. Zejména jsem rád, že jsem se dozvěděl mnohé o perspektivním oboru, jakým robotika bezesporu je. Zdrojem kvalitních informací byly kromě konzultací s Ing. Jaroslavem Rozmanem výlučně anglicky psané publikace. Zejména velké množství volně dostupných odborných článků a výborná kniha Probabilistic Robotics - [15]. Anglická terminologie byla zpočátku těžkým oříškem, nyní ale můžu být rád, ze jsem si svou technickou angličtinu o poznání vylepšil.

Literatura

[1] kolektiv autorů: OpenSLAM.org. http://www.openslam.org.

[2] kolektiv autorů: Iterative Closest Point. [online], navštíveno 6. 5. 2010.
URL http://en.wikipedia.org/wiki/Iterative_Closest_Point

[3] kolektiv autorů: MRPT Downloads: Repository of Robotic Datasets. navštíveno 7.5. 2010.
URL http://babel.isa.uma.es/mrpt/index.php/Downloads

[4] Dellaert, F.; Fox, D.; Burgard, W.; aj.: Monte Carlo Localization for Mobile Robots. 1999.

[5] Eliazar, A.; Parr, R.: DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks. 2003.

[6] Grisetti, G.; Stachniss, C.; Burgard, W.: Improving Grid-based SLAM with Rao-Blackwellized Particle Filters by Adaptive Proposals and Selective Resampling. *Proceedings of the 2005 IEEE International Conference*, 2005: s. 2443–2448.

[7] Grisetti, G.; Stachniss, C.; Burgard, W.: Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. *IEEE Transactions on Robotics*, 2006.

[8] Hähnel, D.; Fox, D.; Burgard, W.; aj.: A Highly Efficient FastSLAM Algorithm for Generating Cyclic Maps of Large-Scale Environments from Raw Laser Range Measurements. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[9] Howard, A.; Roy, N.: The Robotics Data Set Repository (Radish). 2003.
URL http://radish.sourceforge.net/

[10] Liu, J. S.: Metropolized Independent Sampling with Comparisons to Rejection Sampling and Importance Sampling. 1996.

[11] Murphy, R. R.: *Introduction to AI Robotics*. MIT Press, 2000, iSBN 978-0262133838.

[12] Rekleitis, I. M.: A Particle Filter Tutorial for Mobile Robot Localizazation. Technical Report TR-CIM-04-02, Centre for Intelligent Machines, McGill University, Montreal, Quebec, Canada, 2004,
http://www.cim.mcgill.ca/~yiannis/particletutorial.pdf.

[13] Siegwart, R.; Nourbakhsh, I. R.: *Introduction to Autonomous Mobile Robots*. MIT Press, 2004, iSBN 978-0-262-19502-7.

[14] Thrun, S.: Learning Occupancy Grid Maps With Forward Sensor Models. *Autonomous Robots, 15:111-127*, 2003.

[15] Thrun, S.; Burgard, W.; Fox, D.: *Probabilistic Robotics*. MIT Press, 2005, iSBN 978-0-262-20162-9.

[16] Thrun, S.; Fox, D.; Burgard, W.; aj.: Robust Monte Carlo localization for mobile robots. 2001.

Příloha A

Obsah CD

- **mapy/** - Složka obsahující některé datové sady dostupné na [3] spolu s hotovými mapami, které byly vytvořeny mojím programem. Číslo v závorce u názvu souborů map udává počet použitých částic.
- **program/** - Obsahuje zdrojové kódy programu včetně souboru `CmakeLists.txt`. Ten slouží programu CMake pro jednoduché generování unixového makefilu, či projektových souborů v jiném formátu (CodeBlocks, Visual Studio aj...). Vnořená složka **bin/** obsahuje již vygenerovaný makefile a spustitelnou verzi programu pro Linux (přeloženo na Ubuntu 9.10).
- **tex/** - Obsahuje zdrojové kódy textu bakalářské práce (LaTeX)
- **mrpt-0.8.1.zip** - Zdrojový kód knihovny MRPT. Dostupné na oficiálních stránkách MRPT.
- **MRPT-0.8.1-win32.exe** - Instalátor knihovny MRPT pro platformu Windows.
- **xizrea00-bp.pdf** - Tato bakalářská práce ve formátu pdf
- **readme.txt** - Dodatečné informace k překladu programu a jeho použití.