VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Mapování 3D prostoru pomocí LIDARových dat

Mapping of 3D Space Using LIDAR Data

2015 Martin Mada

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Mada**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Mapování 3D prostoru pomocí LIDARových dat
Mapping of 3D Space Using LIDAR Data

Zásady pro vypracování:

Mapování okolního prostoru má využití v mnoha technických odvětvích. Výsledné mapy se používají od lokalizace autonomních vozidel až po realistické 3D modely budov a jiných prostor. Zajímavou a v posledních letech oblíbenou metodou je mapování pomocí LIDARového skeneru. Cílem práce je popsání nejpoužívanějších metod pro mapování a implementace jedné z těchto metod pro vytvoření 3D mapy.

Ve své práci popište a implementujte následující:
1. Stručně popište technologii LIDARu.
2. Popište problém mapování a nejpoužívanější techniky pro jeho vyřešení.
3. Po konzultaci s vedoucím práce implementujte vybranou metodu pro mapování 3D prostoru.
4. V závěru práce zhodnoťte dosažené výsledky.

Seznam doporučené odborné literatury:

[1] MONTEMERLO, Michael, Sebastian THRUN, Daphne KOLLER a Ben WEGBREIT. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. AAAI, 2002.
[2] MONTEMERLO, Michael, Sebastian THRUN, Daphne KOLLER a Ben WEGBREIT. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. IJCAI, 2003.
[3] ELIAZAR, Austin a Ronald Parr. DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks. IJCAI, 2003.
[4] THRUN, Sebastian a Micheal MONTEMERLO. The Graph SLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. IJRR, 2005.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Branislav Holý**

Datum zadání: 01.09.2014

Datum odevzdání: 07.05.2015

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 7. května 2015 .....

Abstrakt

V dnešní době má mapování okolního prostoru obrovské využití v mnoha technických odvětvích. Příkladem může být jeho použití pro lokalizaci automobilů a jiných vozidel, dále pak realistické 3D modely budov nebo jiných prostor a v neposlední řadě má velké uplatnění v robotice. Jednou z nejpoužívanějších a nejzajímavějších metod mapování je v současnosti mapování pomocí technologie LIDAR, respektive pomocí LIDARového skeneru. V této práci popisuji několik základních metod pro mapování a také implementaci jedné z nich.

Klíčová slova: Mapování, 3D, LIDAR, laser, skener

Abstract

There are a lot of different technical fields in which is mapping of 3D space used these days. A good example can be it's use for localization of cars and other vehicles, then realistic 3D models of buildings or other areas and last but not least has a big use in robotics. One of the most widely used and the most interesting method of mapping is mapping by using LIDAR technology or by using LIDAR scanner. This work includes a description of several methods, which are used for maping and also an implementation of one of these methods.

Keywords: Mapping, 3D, LIDAR, laser, scanner

Seznam použitých zkratek a symbolů

LIDAR	–	Light Detection And Ranging
3D	–	Three-dimensional
Laser	–	Light amplification by stimulated emission of radiation
CD	–	Compact disc
DVD	–	Digital video disc
GPS	–	Global Positioning System
NASA	–	National Aeronautics and Space Administration
LAS	–	Lidar Archive Standard
SLAM	–	Simultaneous localization and mapping
MLC	–	Monte Carlo localization algorithm
FastSLAM	–	Factored Solution to the Simultaneous Localization and Mapping Problem
EKF	–	Extended Kalman Filter
DP	–	Distributed particle
PCL	–	Point Cloud Library
RANSAC	–	Random sample consensus
ICP	–	Iterative closest point

Obsah

Seznam obrázků

1 Úvod

Je určitě nepopíratelným faktem, že LIDRová technologie a technologie s ní spojené, mají dnes obrovské uplatnění v mnoha oblastech vědy a techniky. Ať už tuto technologii máme ve svých autech a požíváme ji například při parkování nebo ji využíváme pro zjišťování hladin oceánů a dalších informací, které nám umožňuje použití této technologie v leteckém skenování. Tato práce se ovšem věnuje zcela jiné, velice zajímavé oblasti.

Konkrétně se jedná o využití této technologie k vytváření map budov a jiných 3D prostor. Mapování prostoru obecně je velice zajímavé a má mnoho využití v praktickém životě. Například umožňuje primitivnímu robotu orientovat se v dané místnosti, dokázat ji bez problému projít a pořídit důležité informace o prostorách bez toho, aniž by do dané místnosti musel vstoupit člověk. Toto má samozřejmě využití v případě, kdy vstup do takových prostor není pro člověka bezpečný, nebo jednoduše není možný z hlediska dostupnosti.

Tato práce ve své první části uvádí LIDAR technologii. Dozvíte se něco o její historii, jak se časem vyvíjela až do dnešní verze, dále pak její bohaté využití, a také jak vlastně funguje a z čeho se takový skenovací systém LIDARu může skládat. Druhá část práce už je zaměřena přímo na metody umožňující přesné mapování 3D prostoru. Budou zde představeny celkem čtyři známé metody, z nichž jedna bude v poslední části práce popsána i z hlediska její praktické implementace. Každá metoda obsahuje jednoduché vysvětlení její základní myšlenky, složitost jejího výpočtu a samozřejmě její srovnání s ostatními.

Cílem této práce je popsat a seznámit čtenáře se základními metodami pro mapování 3D prostoru a následná implementace jedné z těchto metod.

2 LIDAR technologie

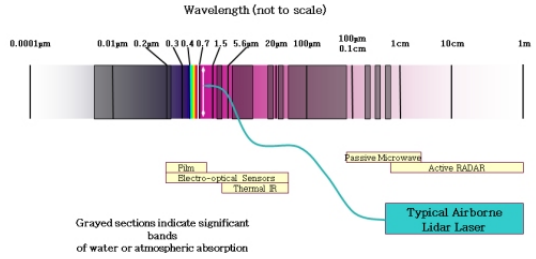
Tato kapitola slouží k seznámení čtenáře s technologii LIDAR. V první části popisuje základní rysy této technologie, její použití ve světě, zejména pak použití pro měření vzdáleností. V druhé části seznamuje čtenáře s jednotlivými částmi LIDARu a v třetí části se zabývá vývojem této technologie od historie až po současnost. Čtenář by si měl odnést základní znalosti o této technologii, které mu pomohou k lepšímu pochopení zbytku práce.

2.1 Seznámení s LIDAR technologií

LIDAR neboli *Light Detection and Ranging*, což v překladu znamená detekce světla a vzdálenosti, je technologie založená na takzvaném *remote sensing* neboli dálkovém průzkumu. Jedná se o metodu využívající světlo ve formě pulzního laseru, pro měření vzdálenosti na Zemi. Tyto světelné pulzy se v kombinaci s údaji z jiných systémů stávají skvělým nástrojem pro vytvoření 3D informace o tvaru daného prostoru, (například tvar Země při leteckých snímcích) popřípadě i vlastností jeho povrchu. Více informací zde [1].

Jak už je zmíněno výše, LIDAR technologie je založena na pulzujícím laseru. Ale co je to vlastně ten laser? Obecně laser znamená *light amplification by stimulated emission of radiation* neboli zesilovač světla pomocí stimulované emise záření. Co přesně to znamená není náplní této práce. Nám bude stačit vědět, že se jedná o zařízení generující silný proud částic s vysokou energií (fotony) v rámci velmi úzkého rozmezí vlnových délek. To dává laseru schopnost vytvořit souvislý zdroj světla pro konkrétní užití. Dnes jej můžeme vidět prakticky všude. Ať už na CD/DVD čtečkách, chirurgických či řezných nástrojích nebo právě na přístrojích pro LIDAR mapování, které nás zajímá. Zdroj laserového světla je vlastně základem celé LIDAR technologie.

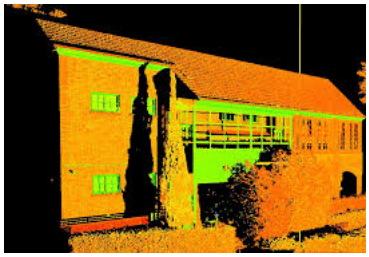
Aby byl laser dostatečně silný a použitelný musí mít již zmíněnou úzkou vlnovou délku. Ta se může v závislosti na svém využití lišit, ale obecně se pohybuje kolem 1064 nanometrů. Toto číslo si asi dokážete jen stěží představit. Pro lepší ukázku co to číslo vlastně znamená slouží následující obrázek, na němž je znázorněné celé spektrum vlnových délek a k nim přidané příslušné použití.



Obrázek 1: Ukázka spektra vlnových délek

Zdroj: ASPRS

Abychom dobře zvládli práci s LIDARem a následné mapování prostoru, musíme být schopni si poradit se základními problémy, které toto mapování obnáší. Důležité je abychom byli schopni přesně lokalizovat senzor, ze kterého měříme, a také být schopni přesně změřit čas, který laser potřebuje k tomu, aby dorazil k cílovému objektu a zpět. Každý laserový pulz vyzářený směrem k danému objektu je definován jako „sloupec světla“ elektromagnetické energie. Když tento sloupec světla dorazí k danému objektu zachytí ho snímač. Pokud je zasažený objekt „pevný“, jako například budova nebo auto, dojde k odrazu dopadané energie, ale pouze jeden odraz se zaznamená. Problém může nastat při skenování vegetace, kdy laser může projít některými objekty, jako například listem a výsledný obraz nebude přesný. Pokud se nám vše podaří mohl by výsledný obraz daného objektu vypadat například jako následující obrázek.



Obrázek 2: Příklad naskenované budovy

Zdroj: Project Surveyors

Výsledný obrázek, který můžete vidět, se skládá s velkého počtu „teček“, kde každá představuje jednu cestu laseru směrem k objektu a zpět a jeho následné zaznamenání na snímač.

Důležitou složkou při měření LIDARovými lasery je již výše zmíněný čas. Respektive čas, který je potřeba aby laser urazil vzdálenost k objektu a zpět. Jelikož se daný laser pohybuje rychlostí světla, můžeme vynásobením uplynulého času s konstantou rychlostí světla určit lehce vzdálenost od objektu a to pomocí následující rovnice.

*Vzdálenost snímače od objektu a zpět = uplynulý čas * rychlost světla*

Pokud chceme určit vzdálenost k objektu musíme výsledek samozřejmě vydělit dvěma, jelikož nám počítá vzdálenost k objektu a zpět. Tato vzdálenost je často označována jako rozsah. Pro tuto podkapitolu jsem čerpal především z [2].

2.2 Základní komponenty LIDARu

Pokud se chceme blíže seznámit s technologií LIDAR, musíme znát základní části, které LIDAR obsahuje. Samozřejmě že to není pouze laser, ale je to celá řada důležitých zařízení, které spolu musejí perfektně spolupracovat a fungovat, aby byl výsledek perfektní. V této práci bych rád uvedl pouze ty nejdůležitější části.

Hlavní částí LIDARu je samozřejmě **Laserový snímač**. Toto zařízení se dále skládá z dalších třech důležitých částí. Jeho práce spočívá ve vytváření konzistentních proudů laserových impulzů, které jsou přiváděny do takzvané „rozmezní jednotky“, kde se odrážejí od zrcadla, které může být například rotační, a tím se dostávají přímo k cílí. V této rozmezní jednotce je také optický přijímač, který zaznamenává celkový čas, který laser potřebuje ke své cestě k cíli a zpět a posílá tuto informaci do řídící jednotky. Tento stejný proces se opakuje 100 000 až 200 000 krát za sekundu. Laserový snímač bohužel není sám schopen spočítat souřadnice odražených laserových bodů, ani přesně určit polohu snímače aby mohl vytvořit 3D model souřadnic a tedy výsledný obraz.

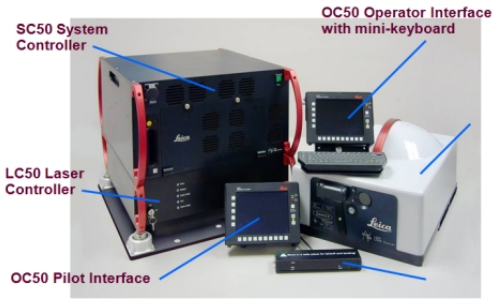
Zde přichází na řadu takzvané **Přímé Georeferencování**, které hraje klíčovou roli v celém systému. Jak už z názvu napovídá, jeho práce je přesně určit pozici snímače a jeho orientaci vzhledem k zemi.

Tomu velice napomáhá **GPS**. Jak je dnes již velice známo jedná se o satelitně řízenou radio-navigační technologii, která je dnes zakotvena v mnoha běžných osobních zařízeních. LIDAR tuto technologii využívá také a to k přesnému odvození polohy snímače v trojrozměrném prostoru, zatímco je umístěn na pohyblivém zařízení, ať už se jedná o auto, letadlo nebo robota. Samozřejmě, že se nejedná o stejné zařízení jako máme my v mobilech, ale o jeho přesně upravenou a vyladěnou verzi.

Na řadu teď přichází **Inerciální soustava**, která ke znalostem o umístění snímače a cíle přidává i poslední, velmi důležité složky a to rychlost a čas. Inerciální měřící systémy obsahují akcelerometry, které dokáží velmi přesně měřit rychlost. Jedinou nevýhodou je jeho náchylnost na změnu okolí a to hlavně vnější vliv gravitace nebo sebemenší „šťouchnutí“ může způsobit odchylky ve výpočtech.

Máme tedy spoustu údajů z různých systémů, které LIDAR využívá, ale potřebujeme je nějak zpracovat a také zajistit, že všechny složky systému odvádějí svou práci správně. K tomu je zapotřebí spolehlivé **počítačové systémy**. Tyto systémy také poté musí integrovat vypočítané údaje do použitelných a přesných výšek na zemi.

Pro ukázku jak takový přístroj, obsahující všechny výše zmíněné složky vypadá poslouží následující obrázek.



Obrázek 3: ALS-50 II Lidarový Systém

Zdroj: Leica Geosystems

Pro následující podkapitolu jsem čerpal informace z [3].

2.3 Historie a vývoj

Laser na bázi dálkového průzkumu jak ho známe dnes, vznikl v roce 1970 pod NASA, za účelem pořizování leteckých snímků. Snímače původně sloužily zejména na vědecké měření, jako například měření vlastností atmosféry, hladiny vody oceánů, snímky lesů a ledovců atd. Tato technologie se ukázala jako velice přesná pro měření, ovšem v té době (80. léta) nebyla k dispozici spolehlivá technologie GPS, pro umístění a lokalizaci snímače.

Toto samozřejmě odstartovalo obrovskou poptávku právě po technologii typu GPS, což pomohlo jejímu rychlému výboji směrem kupředu. To umožnilo sbírat důležité informace ze satelitů a družic, které sloužily pro zpřesnění výpočtů. Tento rychlý vývoj technologií tykajících se laseru vedl k tomu, že v polovině 90. tých let byl vyroben LIDARový laser schopný dosáhnout až 25 000 pulsů za jedinou sekundu. Tento laser měl uspokojit především zákazníky, kteří jej používali výhradně pro topografické mapování. I když v dnešní době zní tato čísla jako směšná a nedostatečná, v 90. tých letech to znamenalo obrovský pokrok a hlavně další impulz pro vývoj lepších a lepších technologií do budoucna. Už v této době přinášely LIDARy velmi kvalitní a přesná data, které nebyly možné získat klasickou metodou pozemního průzkumu nebo fotogrammetrie[1]. Komunita lidí zabývajících se touto technologií, projevila velký zájem o LIDARová data, nejen pro mapování zemského povrchu, ale i pro mapování budov a komunikací.

Když byla LIDARová technologie a LIDARová data představena společnosti zabývající se mapováním, byl to ohromný úspěch. Do té doby se totiž, jak už bylo výše zmíněno, provádělo mapování s vysokým rozlišením pomocí fotogrammetrie. Hlavní přínos byl v tom, že při fotogrammetrii musí být daný objekt „viděn“, aby mohl být mapován, kdežto LIDARový laser měl schopnost proniknout mraky a zmapovat i objekty, které by fotogrammetrie neviděla, na druhou stranu byla tato technologie mnohem dražší a vyžadovala zkušenosti na práci s pořízenými daty. LIDAR také dokáže prohlédnout mezi stromy v hustě zalesněné oblasti, kde by jiné technologie selhaly. LIDAR se tedy s postupným stabilizováním cen a nákladů na provoz ukázal jako rychlá, přesná a atraktivní technologie pro 3D mapování.

Společně s nárůstem poptávky po LIDARu vznikaly i různé normy, pokyny a technické specifikace pro jeho použití. Byly vyvinuty normy pro zajištění kvality a přesnosti dat. Dále společnost ASPRS vyvinula „LAS“ pro výměnu LIDARových dat, které byly zpracovávány jak výrobci snímačů, vývojáři softwaru tak i koncovými uživateli.

V současné době existuje více než 200 LIDARových systémů po celém světě. Ty nejlepší systémy jsou schopné dosáhnout až 250 000 pulsů za jedinou sekundu, spravovat více impulzů najednou, zpracovávat větší výnosy z daných impulzů nebo dokonce zvládají digitalizaci zpátečního průběhu. Sběrnice dat jsou poté upraveny pro výsledné použití uživatelem. LIDAR je v současné době využíván v mnoha různých oblastech a jeho použití a vývoj do budoucna je na perfektní cestě. Pro tuto podkapitolu jsem čerpal z [4].

[1] Fotogrammetrie se zabývá rekonstrukcí tvarů, měřením rozměrů a určováním polohy předmětů, které jsou zobrazeny na fotografických snímcích.

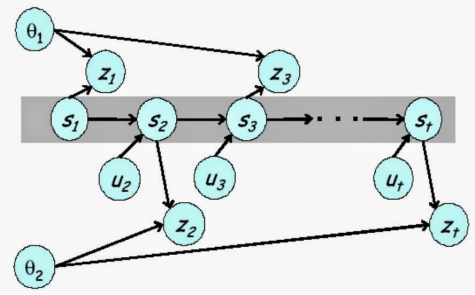
3 Metody pro 3D mapování

Následující kapitola se věnuje nejznámějším a nejpoužívanějším metodám určených pro mapování 3D prostoru za použití LIDARových dat. Čtenář by měl získat přehled o těchto metodách, jejich použití a hlavně jejich srovnání.

3.1 Metody SLAM

Tato bakalářská práce se zabývá metodami z rodiny SLAM neboli *Simultaneous Localization and Mapping*. Tyto metody v dnešní době přitahují velkou pozornost v oblasti robotiky a mapování prostředí. Řeší totiž problém budování map daného prostředí za použití určité sekvence orientačních bodů naměřených a získaných pomocí jednoduchého robota. Ovšem od té chvíle, kdy se robot začne pohybovat, aby nasbíral nové informace nastává prostor pro chybu. Je totiž nutné provádět mapování zároveň s lokalizací robota. Odtud název SLAM. Schopnost současně lokalizovat robota během jeho cesty a mapovat jeho okolí je považována, jako klíčový předpoklad pro vytvoření skutečně autonomních robotů. Toto ale není jediný problém, který metody rodiny SLAM řeší. Jak už bylo zmíněno, prostor pro chybu nastává v okamžiku pohybu robota, a nejedná se pouze o problém jeho lokalizace, jako spíše o problém šumu v jeho pohybu. Dnešní jednoduší roboti, kteří se pro základní mapování používají, mají tento šum příliš velký na to, aby bylo možno mapu sestavit pouhým posunutím snímků. A zde právě nastupují metody rodiny SLAM, které, jak se dozvíte v následujících kapitolách, právě tento problém řeší.

Každá z metod, kterým se tato práce věnuje, řeší výše zmíněný problém trochu odlišným způsobem.



Obrázek 4: Problém SLAMu

Zdroj: [7]

Na obrázku výše jde vidět pohyb robota, který se pohybuje z pozice *S1* přes sled ovládacích prvků *U1, U2,... Ut*. Jak se pohybuje, tak pozoruje význačné body v okolí své pozice. V čase *t1* pozoruje význačný bod $\theta 1$ z celkového počtu dvou význačných bodů $\theta 1$, $\theta 2$. Toto měření je označeno jako *Z1*. V čase *t2* robot pozoruje druhý význačný bod $\theta 2$ a v čase *t3* pozoruje znovu první význačný bod $\theta 1$. Problém SLAMu spočívá v

odhadu(výpočtu) umístění význačných bodů a cesty robota pomocí ovládacích prvků U a měření Z.

3.2 FastSLAM

První metoda, kterou si v této práci představíme se nazývá FastSLAM. Nenechte se zmást názvem metody. Slovo *Fast* zde nepředstavuje označení pro rychlost, ale jedná se o zkratku, která ve spojení se slovem *SLAM*, dává celkový název metody *Factored Solution to the Simultaneous Localization and Mapping Problem*. Jedná se o velice známý a významný algoritmus z rodiny SLAM.

Hlavní myšlenka tohoto algoritmu, a také jeho největší plus, jenž mu dává obrovskou výhodu oproti ostatním algoritmům, je rozdělení hlavního problému SLAM, čili lokalizace robota a zároveň mapování okolí na dvě části. To dělá FastSLAM algoritmus velice efektivní. Jednoduše nemusí řešit obě tyto části dohromady, ale dokáže je vypočíst nezávisle na sobě. První část se tedy zabývá správným určením polohy robota a jeho cesty. Druhá část poté řeší odhadování[2] pozice význačných bodů, na základě dat pořízených robotem. Toto rozdělení, jak už bylo zmíněno, velice zvyšuje efektivnost a přesnost tohoto algoritmu a označuje jej jako algoritmus nezávislý na problému SLAM.

Samotný výpočet je poté prováděn pomocí takzvaných upravených Parlicle filtrů [8], které slouží k odhadování bodů na cestě robota, a tedy i jeho polohy. Tyto filtry upřesňují svůj výpočet v případě, kdy robot daným místem jede podruhé. Způsob, jakým tyto filtry fungují je velice zajímavý a přesný. Každý z Particle filtrů si totiž udržuje normalizovaný soubor vzorků $S = s_1, ..., s_m$, které se nazývají částice. Slovo částice bude v této práci zaznívat častěji, jelikož Particle filtry se ukázaly jako velice efektivní a požívají jej i ostatní metody. Každá z těchto částic poté obsahuje K Kalmanových filtrů, čili filtrů, které dokáži predikovat a upřesnit polohu bodů na základě předešlých výpočtů, v našem případě na základě předešlých dat z robota. Jednoduše řečeno, robot při druhém průchodu danou oblastí odhaduje a upřesňuje polohu bodů na základě výpočtů, které získal, když tudy jel poprvé. Více o Kalmanově filtru zde [5]. Výsledný algoritmus je poté instancí Rao-Blackwellizedova filtru [6]. Při naivní implementaci výše zmíněné myšlenky, se dostáváme k algoritmu, jehož časová náročnost bude $O(MK)$, kde M je počet částic ve filtru a K je počet význačných(orientačních) bodů.

Co se týče testování tohoto algoritmu v praxi, tak výsledky byly v některých případech velice zmatené. Experimentální výsledky pomocí zkušebního robota totiž ukázaly, že počet význačných bodů K není vždy přímo úměrný k počtu potřebných částic M, pro vytvoření přesné mapy. Jednoduše se v některých případech snížil počet potřebných částic M a v některých se zase zvýšil, při přidání většího počtu význačných bodů K.

Nyní, když známe základní myšlenku a způsob jakým se provádí v praxi, můžeme přejít k samotnému výpočtu FastSLAMu. Než k tomu ale dojde, musíme si zavést pár pojmů a jednotek se kterými budeme počítat. Zavedeme si proměnou nt, která bude reprezentovat index význačného bodu, na který se robot dívá v čase t. Například na ob-

[2] V této práci bude slovo odhad zaznívat namísto slova výpočet velice často, jelikož počítač vždy odhaduje a nikdy nepočítá a samotné vzorce pro dané výpočty, počítají pouhou pravděpodobnost výskytu daného jevu.

rázku 4 máme $n1 = 1, n2 = 2, n3 = 1$, neboli v čase $t1$ pozoruje význačný bod $\theta 1$, v čase $t2$ význačný bod $\theta 2$ a v čase $t3$ opět význačný bod $\theta 1$.

3.2.1 Číselná reprezentace

Víme tedy, že FastSLAM používá Particle filtry. Víme také, že každá z jeho částic obsahuje K Kalmanových filtrů. Tím jsme tedy zvládli teoretickou reprezentaci tohoto algoritmu. Jak ale vypadá ta číselná? Začneme naší úvahu na případě, kde známe všechny $n^t = n1, ...nt$ a také všech K význačných bodů. Z toho co víme, tedy z podmíněné nezávislosti na problému SLAM vyplývá, že celková pravděpodobnost výskytu význačných bodů a pozic robota, získaná pomocí měřeních z^t a kontrolních příkazů u^t, může být početně vyjádřena následujícím způsobem.

$$p(s^t, \theta | z^t, u^t, n^t) = p(s^t | z^t, u^t, n^t) \prod_k p(\theta_k | s^t, z^t, u^t, n^t) \tag{1}$$

Na této rovnici lze krásně vidět rozdělení problému SLAM na dvě části. Slovy řečeno, celý problém může být rozdělen na $K + 1$ problémů, kde jeden problém je odhadování pravděpodobnosti pozice robota během jeho cest s^t a K problémů odhadování pozice K význačných bodů závislých na odhadnuté cestě. Tato faktorizace je velice přesná a vždy aplikovatelná na SLAM problém.

První část pravé strany rovnice nám tedy zobrazuje, jakým způsobem FastSLAM implementuje odhad pravděpodobnosti výskytu bodů na cestě robota(jeho lokalizaci) $p(s^t | z^t, u^t, n^t)$, za použití již zmíněného Particle filtru. Tento postup použití filtrů je nejen přesný, jak už bylo výše zmíněno, ale také dokáže poskytnout dobrý odhad i v případě nepříliš lineárního pohybu robota. Což v dnešní době je velice dobrá vlastnost. Ne, že by roboti byli tak nepřesní co se týče jejich pohybu, ale požadovaná přesnost u map roste s konkurencí, a tak každý výkyv je nutné opravit. Druhá část pravé strany rovnice poté řeší zbylou část problému SLAM, a to odhad pozice význačných bodů $p(\theta_k | s^t, z^t, u^t, n^t)$, která je tedy realizována Kalmanovými filtry. Jak již zaznělo v úvodní kapitole o metodě FastSLAM, tak každá částice obsahuje K Kalmanových filtrů. Každý filtr slouží k odhadu jednotlivých význačných bodů. Je tomu tak proto, jelikož každý odhad pozice význačných bodů je logicky závislý na předešlém výpočtu, čili na odhadu pravděpodobnosti pozice robota, a proto má každá částice v daném Particle filtru svůj vlastní, lokální výpočet význačných bodů. Celkově nám to tedy pro M částic a K význačných bodů dává MK Kalmanových filtrů

3.2.2 Particle filtr pro odhadnutí cesty robota

Nyní už tedy máme zase o něco lepší představu o tom, jak FastSLAM pracuje, ale je třeba jít trochu do podrobností. Pojďme si přiblížit jednotlivé části rovnice 1. Začneme tou pro výpočet pravděpodobnosti výskytu bodů na cestě robota $p(s^t | z^t, u^t, n^t)$. Opět připomínám, že pro tento účel slouží Particle filtry, ale v tomto konkrétním případě se jedná o Particle filtry, které jsou velice podobné takzvanému MLC algoritmu, jenž je aplikací Particle filtru na problém určení pozice robota. Podobnost těchto algoritmů spočívá v

tom, že si oba v každém časovém úseku uchovávají množinu částic, která poté reprezentuje odhad pravděpodobnosti výskytu bodů na cestě robota, označenou jako S_t. Každá jednotlivá částice $s^{t,[m]} \in S_t$ poté reprezentuje odhad cesty robota:

$$S_t = \{s^{t,[m]}\}_m = \{s_1^{[m]}, s_2^{[m]}, ...s_t^{[m]}\}_m \tag{2}$$

Ve vzorci je použita proměnná $[m]$, čímž je myšlena m-tá částice dané množiny, kterou si každý algoritmus uchovává. Výpočet množiny částic S_t je prováděn postupně, od množiny S_{t-1} v čase $t-1$ s ovládacím prvkem u_t a měřením z_t. Jak již bylo zmíněno, každá částice má svůj vlastní lokální výpočet, a tedy každá částice $s_t^{[m]}$ v S_{t-1} je použita pro generování pravděpodobnostního odhadu pozice robota v čase t:

$$s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]}) \tag{3}$$

Samozřejmě, že nemůžeme provést výše uvedený odhad jen tak. Je k tomu nutné znát informace o pohybu robota, které nám v tomto případě poskytuje pohybový model, bez kterých by počítání nemělo smysl. Když tedy máme veškeré informace potřebné k výpočtu a podaří se nám odhad získat, tak jej přidáme do dočasné množiny částic společně s cestou $s^{t-1,[m]}$. Jednoduše řečeno, když každá částice počítá svůj odhad, tak jeho výsledek uloží do společné množiny částic. Po této fázi, se za předpokladu, že množina částic v S_{t-1} je distribuována podle $p(s^{t-1}|z^{t-1}, u^{t-1}, n^{t-1})$, je pak nová částice distribuována podle: $p(s^t|z^{t-1}, u^t, n^{t-1})$. Tato distribuce je obecně označována jako *návrhová distribuce* Particle filtrování. S tímto pojmem se ještě setkáme v dalších kapitolách.

Tento proces distribuce je poté nutné provést pro všechny částice, a díky toho po vygenerování M částic tímto způsobem, získáme novou množinu S_t, která vznikla z dočasné(původní) množiny. Každá částice $s^{t,[m]}$ je vyhotovena s pravděpodobností, která je přímo úměrná takzvanému *importance factor* $w_t^{[m]}$, který se počítá následovně:

$$w_t^{[m]} = \frac{p(s^{t,[m]}|z^t, u^t, n^t)}{p(s^{t,[m]}|z^{t-1}, u^t, n^{t-1})} \tag{4}$$

Výsledný vzorek množiny S_t je distribuovaný v závislosti na jeho přiblížení se k touženému $p(s^t|z^t, u^t, n^t)$. Toto přiblížení je přesnější a přesnější, jak se počet částic M blíží k nekonečnu.

3.2.3 Odhadnutí polohy význačných bodů

Už tedy víme, jak FastLSLAM řeší určení polohy robota a můžeme tedy přejít k druhé části výpočtu rovnice 1. Jak už bylo výše zmíněno, výpočet polohy K význačných bodů $p(\theta_k|s^t, z^t, u^t, n^t)$ je prováděn pomocí K Kalmanových filtrů. Víme už, že každá částice si provádí odhad pozice význačných bodů, pomocí svého lokálního výpočtu, který je prováděn na základě odhadu pozice robota. To má za důsledek, že jednotlivé Kalmanovy filtry jsou takzvaně připojeny k jednotlivým pozicím částic v St. Jinak řečeno, celková

pravděpodobnost výskytu bodů na cestě robota a pravděpodobnost výskytu význačných bodů je v algoritmu FastSLAM reprezentována pomocí množiny:

$$S_t = \{s^{t,[m]}, \mu_1^{[m]}, \Sigma_1^{[m]}, ..., \mu_K^{[m]}, \Sigma_K^{[m]}\}_m \tag{5}$$

Kde $\mu_1^{[m]}$ a $\Sigma_1^{[m]}$ jsou průměr a kovariance[3] Gaussovy reprezentace k-tého význačného bodu θ_k vázaného na m-tou částici. Ve skutečnosti je $\mu_1^{[m]}$ dvojrozměrný vektor a $\Sigma_1^{[m]}$ je matice o rozměru 2x2. Tímto tedy dostáváme výsledný výpočet pravděpodobnosti výskytu význačného bodu, který je závislý na faktu zda $n_t = k$ nebo ne. Neboli zda byl význačný bod θ_k pozorován v čase t nebo nikoliv. V případě, že ano, tedy v případě, že platí $n_t = k$ dostáváme rovnici:

$$\begin{aligned} & p(\theta_k|s^t, z^t, u^t, n^t) \\ \overset{Bayes}{\propto} & p(z_t|\theta_k, s^t, z^{t-1}, u^t, n^t)p(\theta_k|s^t, z^{t-1}, u^t, n^t) \\ \overset{Markov}{=} & p(z_t|\theta_k, s_t, n_t)p(\theta_k|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \end{aligned} \tag{6}$$

V opačném případě, tedy případě že $n_t \neq k$, tak jednoduše necháme Gaussovu reprezentaci nezměněnou:

$$p(\theta_k|s^t, z^t, u^t, n^t) = p(\theta_k|s^{t-1}, z^{t-1}, u^{t-1}, n^{t-1}) \tag{7}$$

FastSLAM algoritmus implementuje aktualizaci rovnice jenž můžeme vidět výše 6, za použití rozšířeného Kalmanova filtru(EKF). Tento filtr používá linearizovanou verzi perceptuálního modelu $p(z_t|s_t, \theta, n_t)$. EKF používaný pro FastSLAM, je velice podobný EKF používaného pro SLAM v tom, že se blíží modelu měření pomocí lineární Gaussovy funkce.

Ovšem to neznamená, že použití Kalmanových filtrů v algoritmu FastSLAM a v algoritmu SLAM je stejné. Je zde totiž jeden výrazný rozdíl, mezi použitím Kalmanových filtrů ve FastSLAM algoritmu a tradičním SLAM algoritmem, a to ten, že aktualizace ve FastSLAM algoritmu zahrnují pouze dvoudimenzionální Gaussian[4](pro dva parametry lokace význačného bodu), zatímco ve SLAM založeném na EKF, je použit Gaussian o velikosti $2K + 3$ (K význačných bodů a 3 parametry pozice robota). Tento výpočet má také časové výhody, kdy lze v algoritmu FastSLAM použít v konstantním čase, zatímco ve standardním SLAMu je zapotřebí čas kvadratický na K.

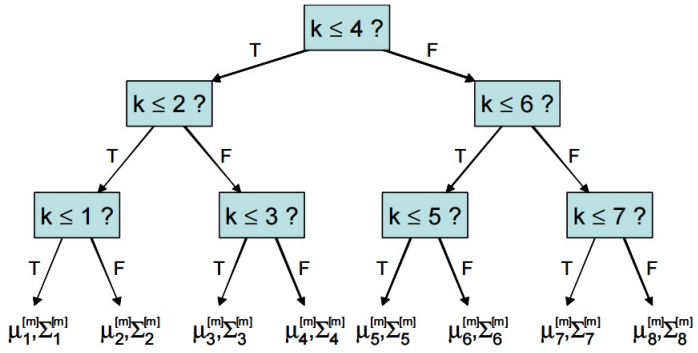
3.2.4 Efektivní implementace

Nyní se dostáváme k popisu efektivní implementace a s ní spojeným výpočtem časové složitosti algoritmu FastSLAM. Pokud vše provedeme tak, jak je výše popsáno, tak bude tento algoritmus vyžadovat čas, který bude samozřejmě závislý na počtu význačných

[3]Míra vzájemné vazby mezi dvěma náhodnými veličinami
[4]Myšleno Gaussovo rozdělení pravděpodobnosti

bodů K, pro každou iteraci Particle filtru implementovanou naivně. Závislost na počtu význačných bodů je způsobena jejich propojení s částicemi, kde každá částice řeší vlastní výpočet dané pozice význačného bodu. Pokaždé, když částice získá daný odhad, tak ho přidá do souboru částic S_t, do kterého je vlastně kopírována. Neboli aktualizuje pozici, na kterou je kopírována svým výsledkem. Tomuto procesu se říká *převzorkování*. Když si uvědomíme, že každá částice obsahuje K těchto odhadů pozice význačných bodů a částic máme M, tak celkový proces kopírování(převzorkování) vyžaduje $O(MK)$ času. To může znít jako velké číslo, nicméně většině tohoto kopírování se dá vyhnout.

Přístup, jenž je popisován v této práci, umožňuje vykonat FastSLAM algoritmus v podstatně kratším čase a to za $O(MlogK)$. Toto výrazné zkrácení času je umožněno díky výborné myšlence, která provádí reprezentaci Gaussianů v každé částici pomocí binárního stromu[5]. Obrázek níže ukazuje přesně takový strom jedné částice v případě reprezentace 8-mi význačných bodů v rámci jedné částice.



Obrázek 5: Strom reprezentující 8 odhadů význačných bodů v rámci jedné částice

Zdroj: [7]

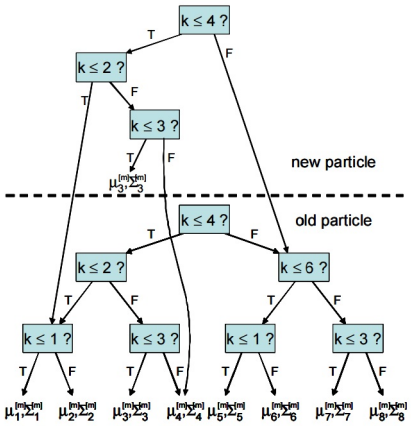
Již výše definované Gaussovy parametry $\mu_k^{[m]}$ a $\Sigma_k^{[m]}$ jsou chytře umístěny na listech stromu. Jednoduše řečeno, při potřebě dosažení těchto parametrů, budeme potřebovat čas roven $\log K$. Toto nám zajišťuje základní definice binárního stromu.

Samotné využití stromu, nám zatím moc neříká. Pojďme si ukázat jeho funkci na následujícím příkladě. Budeme tedy předpokládat, že robot používající FastSLAM, obdržel nový řídící příkaz u_t a pořídil nové měření z_t. Potom tedy každá nová částice patřící do S_t se bude lišit od jí odpovídající částice patřící do S_{t-1} ve dvou směrech. Zaprvé bude mít odlišný odhad cesty robota vypočítaný pomocí rovnice 3 a zadruhé Gaussian s indexem n_t bude rozdílný v souladu se vzorcem 6.

Teď je samozřejmě otázkou, zda když na daný příklad použijeme tuto novou myšlenku reprezentace Gaussianů, tak bude kopírování nové částice do daného stromu efek-

[5]Binární strom je strom ve smyslu používaném v teorii grafů. Jedná se o orientovaný graf s jedním vrcholem (kořenem), z něhož existuje cesta do všech vrcholů grafu. Každý vrchol binárního stromu může mít maximálně dva orientované syny a s výjimkou kořene právě jednoho předka. Kořen předka nemá.

tivnější, než při použití starého principu nebo ne. Na tuto otázku je jednoduchá odpověď. Při kopírování jedné částice nám totiž stačí upravit pouze jednu cestu ve stromu, která reprezentuje všechny Gaussiany a nemusíme tedy upravovat strom celý. Typický příklad je zobrazen na následujícím obrázku, kde v horní části jde vidět tvorba nové částice a v dolní části se nachází částice původní.



Obrázek 6: Generování nové částice ze staré při úpravě pouze jednoho Gaussianu

Zdroj: [7]

Příklad generování částice na obrázku předpokládá, že $n_t = 3$, to znamená, že pouze Gaussovy parametry $\mu_3^{[m]}$ a $\Sigma_3^{[m]}$ budou aktualizovány. A tudíž místo generování celého stromu úplně od začátku, se pouze vytvoří nová cesta vedoucí k Gaussianu $n_t = 3$. Tato cesta tvoří nekompletní strom. K dokončení tohoto stromu, musíme pro všechny větve, jenž patří k dané cestě, zkopírovat všechny korespondující ukazatele ze stromu do nové částice. To způsobí, že dané větve budou ukazovat na stejný podstrom, jako ukazovaly ve stromu původním. Jednoduše generování takového neúplného stromu zabere čas, odpovídající $O(logK)$. Navíc přístup k jednotlivým Gaussianům zabere také čas odpovídající $O(logK)$, jelikož počet kroků potřebných k dosažení listu je roven délce cesty (která je podle definice logaritmická). To znamená, že obojí, jak generování, tak přístup ke stromu, zabere čas $O(logK)$. Jelikož v každé aktualizaci kroku M, je vytvořena nová částice, celková aktualizace vyžaduje čas roven $O(MlogK)$. Tyto výsledky dělají reprezentaci pomocí binárního stromu velmi efektivní.

3.2.5 Asociace dat

V reálném světě to samozřejmě nemusí být takové, jak je to popsáno výše. Většinou ne všechny význačné body se dají identifikovat, nebo o nich nemáme žádné informace, a tím pádem celkový počet těchto bodů K není tak snadné získat. V takovémto případě musí robot vyřešit problém s asociací dat, mezi momentálním pozorováním význačných

bodů z_t a mezi souborem význačných bodů v mapě θ. Také musí rozhodnout, zda měření odpovídá novému, dosud neobjevenému význačnému bodu. V tom případě by musela být mapa odpovídajícím způsobem rozšířena(aktualizována). Tento problém je ve většině algoritmů typu SLAM, založených na EKF, řešen pomocí takzvané metody *maximum likelihood*. Více o této metodě zde [9].

3.2.6 Shrnutí metody

FastSLAM algoritmus byl v této práci představen jako velmi efektivní řešení současných problémů při mapování prostoru za pomocí jednoduchého robota. Tento algoritmus používá Rao-Blackwellizedovu reprezentaci pravděpodobnosti výskytu daných jevů(polohy robota, překážek), Particle filtry pro odhad pozice a cesty robota a Kalmanovy filtry pro odhad pozice význačných bodů. Je založen na podmíněné nezávislosti na problému SLAM, na rozdíl od jiných algoritmů, které v tomto ohledu mohou selhat. Tuto nezávislost osobně vidím, jako největší plus celého algoritmu. Odhady význačných bodů jsou efektivně reprezentovány za pomocí binárního stromu, který, jak bylo výše ukázáno, je také velice efektivním přínosem pro FastSLAM algoritmus. Aktualizace pravděpodobnosti výskytu vyžaduje čas roven $O(MlogK)$, kde M je počet částic a K je počet význačných bodů. Toto je výrazný rozdíl oproti potřebě $O(K2)$ času u běžného SLAM algoritmu založeného na Kalmanových filtrech. Experimenty ukazují, že FastSLAM je schopen sestavit mapu s mnohem vetším počtem význačných bodů než starší metody. Pro tuto kapitolu jsem čerpal z [7]

3.3 FastSLAM 2.0

Algoritmus s názvem FastSLAM 2.0 logicky vychází z chyb a nedokonalosti svého předchůdce, tedy FastSLAMu. Abychom mohli říct, co je na novém FastSLAMu dobré, musíme si nejprve říct, co je na tom původním algoritmu špatné, tedy co tento algoritmus vlastně vylepšuje. Hlavní problém, nacházející se v původním FastSLAMu, je ten, že v tomto algoritmu je pozice $s_t^{[m]}$ vzorkována v závislosti na shodě s předpovědí, která vychází z pohybového příkazu u_t, jak je ukázáno zde 3. Tento vzorec ovšem nebere v potaz příslušné měření z_t pořízené v čase t. Přístup tohoto typu je ovšem poněkud problematický v případě, že šum v pohybu vozidla je příliš velký vzhledem k šumu měření neboli náš kontrolní příkaz u_t nevykonal přesně to, co měl. V takové situaci vzorkované pozice ve většině případů spadnou do oblasti s nízkou pravděpodobností výskytu a budou následně s velkou pravděpodobností vymazány pří takzvaném převzorkování. Bohužel mnoho robotických systému z reálného světa je charakterizováno poměrně vysokým pohybovým šumem, a proto bylo potřeba přijít s nápadem, který tento problém efektivně řeší.

3.3.1 Vzorkování pozice

FastSLAM 2.0 přichází právě s jedním takovým nápadem: Pozice jsou nyní vzorkovány na základě jak pohybových příkazů u_t, tak pořízených měření z_t. Což by ve výsledku

mělo minimalizovat nepřesnosti způsobené šumem v pohybu robota. To je formálně znázorněno pomocí následujícího vzorkovacího rozdělení, které tentokrát bere v úvahu měření z_t.

$$s_t^{[m]} \sim p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) \tag{8}$$

Když toto rozdělení srovnáme s 3, zjistíme, že zahnutí měření z_t má smysl pouze v případě, zahrneme-li náš současný odhad polohy význačného bodu získaného z proměnných $s^{t-1,[m]}, u^{t-1}, z^{t-1}, n^{t-1}$. Takže v podstatě rozdíl od FastSLAMu je pouze v tom, že zde počítáme s měřením z_t, nicméně tato změna má významné důsledky. Tím pádem totiž návrhové rozdělení je produkt dvou faktorů. První je nám již známý $s_t^{[m]} \sim p(s_t|u_t, s_{t-1}^{[m]})$ a druhý je pravděpodobnost měření z_t.

$$p(s_t|s^{t-1,[m]}, u^t, z^t, n^t) = \eta^{[m]} \int \underbrace{p(z_t|\theta_{n_t}, s_t, n_t)}_{\sim N(z_t; g(\theta_{n_t}, s_t), R_t)} \underbrace{p(\theta_{n_t}|s^{t-1,[m]}, z^{t-1}, n^{t-1})}_{\sim N(\theta_{n_t}; \mu_{n_t,t-1}^{[m]}, \Sigma_{n_t,t-1}^{[m]})} d\theta_{nt} \tag{9}$$

$$\underbrace{p(s_t|s_{t-1}^{[m]}, u_t)}_{\sim N(s_t; h(s_{t-1}^{[m]}, u_t), P_t)}$$

Rovnice výše zobrazuje právě ono zmíněné rozdělení návrhové distribuce na produkt dvou faktorů, a to: již známé stavové distribuce $p(s_t|s_{t-1}^{[m]}, u_t)$ a pravděpodobnosti měření z_t. Počítaní toho druhého z již zmíněných faktorů zahrnuje integraci nad možnou pozicí význačného bodu θ_{n_t}.

Nicméně vzorkování přímo z této distribuce je nemožné z jednoho hlavního důvodu: Tato rovnice nemá analytické řešení. To znamená, že neexistuje způsob, jak se pomocí úprav této rovnice dostat k tíženému výsledku. Naštěstí, je zde způsob, jak dosáhnout analytického řešení této distribuce a to tím, že g je upřesněno pomocí lineární funkce(h může zůstat nelineární):

$$g(\theta_{n_t}, s_t) \approx \hat{z}_t^{[m]} + G_\theta * (\theta_{n_t} - \mu_{n_t,t-1}^{[m]}) + G_s * (s_t - \hat{s}_t^{[m]})$$

kde $\hat{z}_t^{[m]} = g(\hat{\theta}_{n_t}^{[m]}, \hat{s}_t^{[m]})$ znázorňuje předpokládaný výsledek měření, $\hat{s}_t^{[m]} = h(s_{t-1}^{[m]}, u_t)$ předpokládanou pozici robota a $\hat{\theta}_n^{[m]} = \mu_{n,t-1}^{[m]}$ předpokládanou pozici význačného bodu. Matice G_θ a G_s jsou Jacobiho determinanty[11] proměnné g. Po upřesnění se návrhová distribuce8 stává Gaussianem s následujícími parametry:

$$\Sigma_{s_t}^{[m]} = [G_s^T Q_t^{[m]-1} G_s + P_t^{-1}]^{-1} \tag{10}$$

$$\mu_{s_t}^{[m]} = \Sigma_{s_t}^{[m]} G_s^T Q_t^{[m]-1} (z_t - \hat{z}_t^{[m]}) + \hat{s}_t^{[m]} \tag{11}$$

3.3.2 Shrnutí

Tato kapitola se může na první pohled jevit jako nic moc říkající. Ovšem musíme si uvědomit, že se jedná pouze o vylepšení metody FastSLAM, takže vše co platilo pro tuto metodu platí i zde, akorát došlo k několika úpravám. Jednou z hlavních změn této metody oproti FastSLAMu je, jak již bylo výše zmíněno, přidáním měření z_t do výsledného vzorkování pozice $s_t^{[m]}$, což minimalizuje chyby způsobené šumem v pohybu robota. Přidáním měření z_t nebude samozřejmě ovlivněno pouze vzorkování, ale i další výpočty, které s ním pracují. FastSLAM 2.0 tedy přináší spíše vylepšení metody než novou metodu jako takovou.

Experimenty s touto metodou ukázaly úžasné výsledky při použití poměrně malého počtu částic a co se týče výkonnosti daleko překonal svého předchůdce. Tedy, když si uvědomíme, že již FastSLAM byl označen za velmi přesnou a efektivní metodu, tak nápad FastSLAMU 2.0 byl podle mého názoru krok výborným směrem. Pro tuto kapitolu jsem čerpal z [12].

3.4 DP-SLAM

Na řadu přichází algoritmus, který je v některých věcech velice podobný algoritmům FastSLAM a FastSLAM 2.0 a v některých se naopak velice liší. Jedná se o algoritmus založený na stejné myšlence jako algoritmy předešlé, a to na podmíněné nezávislosti na SLAM problému. Nicméně, tento algoritmus je na rozdíl od FastSLAMu založený čistě na laseru a nedělá žádné odhady(výpočty) význačných bodů. Čili pracuje se snímky, které dostal od robota a neprovádí s nimi žádné úpravy, před použitím samotného algoritmu. Tento algoritmus se také vyhýbá problémům s asociací dat pomocí vytváření mnohem přesnějších map místo rozptýlených význačných bodů, a tím dává dohromady asociaci dat s lokalizací. DP-SLAM používá Particle filtry pro reprezentaci jak pozice robota, tak možné konfigurace výsledné mapy.

Ovšem naprostou novinkou, a podle mě velice efektivní a zajímavou, se kterou tento algoritmus přichází, a která ho naprosto odlišuje od metod ostatních, je použití nového způsobu reprezentace mapy, který je nazván distribuované částice(DP), podle kterého dostal také tento algoritmus jméno. Pomocí této reprezentace, je DP-SLAM schopen udržet a aktualizovat stovky až tisíce „kandidátských map“(map, ze kterých se poté skládá výsledná mapa) a pozic robota v reálném čase, během jeho pohybu prostředím. Což je velká změna oproti předešlým algoritmům.

Další věc, která tomuto algoritmu přidává na efektivitě a dělá z něj kvalitní nástroj pro mapování prostředí, je jeho maximální možná složitost za jeden laserový „zásah“ (nasnímání jednoho bodu). Tato složitost je maximálně log-kvadratická v počtu částic, které si udržujeme v rámci Particle filtru a lineární v oblasti zaznamenávání laserem. Více o složitosti tohoto algoritmu v pozdějších kapitolách. Nicméně v praxi je potřeba poměrně méně času. Hodně na času ubere fakt, že tato technika používání DP, v podstatě nedělá žádné předpoklady ohledně prostředí, které snímá. Vše bere tak jak je a přesto je dostatečně, nebo spíše velice přesná na to, aby uzavřela smyčku chodeb dlouhou několik desítek metrů s ostrými, kolmými hranami s minimálními chybami. Tato přesnost je

dosažena díky, již výše zmíněnému mapovacímu procesu, bez potřeby explicitních algoritmů na opravu smyčky. Samozřejmě, že by DP-SLAM o tyto explicitní algoritmy, které by upřesňovaly výsledky, mohl být ve finále doplněn, ovšem jejich použití by bylo spíše časově nevýhodné, při použití dostatečného množství částic.

3.4.1 Particle filtry pro lokalizaci

Abychom lépe pochopili výhody spojené s tímto algoritmem, musíme si ukázat, jak se dané situace řeší obecně, a až potom přejdeme k ukázce a vysvětlení distribuovaného mapování částic, které je pro tento algoritmus typické. Začneme tedy s Particle filtry, konkrétně s jejich využití pro lokalizaci a mapování. Particle filtr je přirozený přístup k problematice lokalizace, kde pozice robota je neznámý stav, který má být pozorován. Změna stavu je v tomto případě pohyb robota a pozorování je čtení dat pořízených senzory robota při průjezdu prostředím.

Změna stavu v čase je řešena takzvaným pohybovým modelem, se kterým jsme se již setkali při zjišťování odhadu pozice robota v metodě FastSLAM. Obvykle pohyb označený odometrií[6], se bere jako základ pro pohybový model, jelikož se jedná o spolehlivé měření počtu otočení koleček robota. Nicméně odometrie je notoricky nepřesné měření co se týče **aktuálního** pohybu robota, dokonce i v těch nejlepších podmínkách. Navíc většina prostředí tyto podmínky ani zdaleka nemá a různé skluzy a posuny kol robota a také nerovnosti terénu mohou způsobit značné chyby, které se rychle nakupí. Pohybový model se liší napříč druhy robotů a terénů, ale obecně se skládá z lineárního posunu pro vysvětlení systematických chyb a Gaussova šumu. Takže pro odometrické změny x, y a θ, Particle filtr aplikuje chybový model a získá pro částici i:

$$\begin{aligned} x_i &= a_x * x + b_x + \mathcal{N}(0, \sigma_x) \\ y_i &= a_y * y + b_y + \mathcal{N}(0, \sigma_y) \\ \theta_i &= a_\theta * \theta + b_\theta + \mathcal{N}(0, \sigma_\theta) \end{aligned} \tag{12}$$

Proměnné a a b jsou v tomto případě lineární korekce, pro vysvětlení konzistentních chyb v pohybu robota. Funkce $\mathcal{N}(O, \sigma_\theta)$ vrací náhodný šum z normálního rozdělení[7], se střední hodnotou 0 a standardní odchylkou σ, která je odvozena experimentálně a může záviset na velikosti proměnných x,y a θ.

Po každé další simulaci musíme vážit(kontrolovat) částice, které máme uložené, na základě aktuálního pozorování okolí naším robotem. Robot si ve své paměti ukládá mapu, která se následně používá čistě pro výpočet lokalizace. Pozice, jenž je popsána jednotlivými částicemi, odpovídá určitému bodu a orientaci na mapě. To vše nám velmi zjednodušuje práci s daty, jelikož je velmi snadné určit, co budou senzory robota vracet v rámci

[6] Základem odometrie je znalost geometrického modelu robota. Tyto modely se liší zejména tím, jakých druhů pohybu jsou roboti schopni.

[7] Normální neboli Gaussovo rozdělení je jedno z nejdůležitějších rozdělení pravděpodobnosti spojité náhodné veličiny. Slovo „normální“ se vztahuje k staršímu významu „řídící se zákonem, předpisem nebo modelem“

dané mapy. Předpokládá se totiž, že chyby snímače jsou normálně rozděleny, tudíž pokud první překážka na mapě zaznamenána laserem má vzdálenost d a nahlášena vzdálenost robotem je d', potom hustota pravděpodobnosti pozorování rozporu $\delta = d' - d$, je normálně rozdělena s průměrem 0. Pro klasické experimenty předpokládáme standardní odchylku v laserových měřeních 5 cm. Vzhledem k modelu a pozici, je každé čtení senzoru korektně bráno, jako nezávislé pozorování. Celková hodnota pravděpodobnosti každé částice je potom:

$$P_i = \Pi_k P(\delta_{ik}|s_i, m) \tag{13}$$

kde δ_{ik} je označení pro rozdíl mezi očekávanou a vnímanou vzdáleností pro snímač P a částici i.

3.4.2 Naivní SLAM

Zde si ukážeme jak danou problematiku řeší naivní SLAM. To nám pomůže zase o něco lépe vidět přínos algoritmu DP-SLAM. Pří použití Particle filtru pro SLAM, každá částice je vlastně reprezentace jedné specifické trajektorie prostředím a má k sobě přiřazenu jednu specifickou mapu. Pokud je částice převzorkována, celá mapa je považována za součást skrytého stavu a je překopírována na novou částici. Jestliže je mapa obsazená matice o velikosti M, a P částic je udržováno v Particle filtru, pak při ignorování ceny pro lokalizaci, musí být vykonáno $O(MP)$ operací pro pouhé kopírování mapy. Pro počet částic potřebných k dosažení přesné lokalizace v přiměřeně velkém prostředí, by naivní přístup vyžadoval několik GB dat o pohybu robota, nutných k jednotlivé aktualizaci.

3.4.3 Distribuované mapování částic

Jak jste si jistě všimli v podkapitole výše, naivní SLAM dělá až příliš mnoho práce, která zabírá až příliš mnoho paměti a tudíž jeho použití není příliš efektivní. Pro celkové zjednodušení celého procesu, nám poslouží právě distribuované mapování částic. Pojďme si nyní ukázat, jak toto mapování dokáže uspořádat jednotlivé částice do speciální datové struktury, aby věděly o změnách, jenž byly provedeny před nimi a mohly s těmito informacemi pracovat, bez nutnosti přístupu k informacím, které pro ně nejsou zrovna důležité. Pokaždé, když je částice vzorkována v iteraci i, aby produkovala nástupce na iteraci $i + 1$, nazýváme částici i-té generace *rodičovskou* a částici generace $i + 1$ nazýváme *dítětem*. Dvě děti se stejným rodičem se nazývají *sourozenci*. Čili si mezi sebou vytvoří jakési *rodinné vztahy*. Předpokládejme nyní, že laser skenuje oblast o velikosti $A \ll M$ a bere v úvahu dva sourozence s_1 a s_2. Každý ze sourozenců odpovídá jiné pozici robota a bude provádět maximálně A aktualizací mapy, kterou dědí od svého rodiče. Takže sourozenci s_1 a s_2 se mohou lišit maximálně v A pozicích v rámci dané mapy.

Samozřejmě, že při prezentaci problému pomocí DP, nastává prostor pro otázku, zda není výhodné a lepší, zaznamenávat vzniklé rozdíly mezi dílčími mapami jednotlivých částic. Například vytvořit list změn, který by obsahoval změny, jenž každá částice udělá vzhledem ke své rodičovské mapě. Jednoduše uchovávat rozdíly mezi dítětem a rodičem. Tento způsob řešení efektivního vytváření aktualizací map, by sice daný problém

vyřešil a zefektivnil by jeho celkový proces, nicméně by na druhou stranu zhoršil počítání lokalizace robota. Důvodem zmíněného zhoršení, by byla nutnost pracovat nad celým rodokmenem současné částice (zjistit všechny kroky k jejímu vypočtení), a také konzultace nad celým obsahem daného listu, obsahující dané změny pro každou částici. To vše by byla práce navíc při sledování dráhy robota skrz mapu k odhalení překážek. Složitost této operace by byla lineární v závislosti na počtu iterací Particle filtru. Hlavní výzva tohoto algoritmu tedy spočívá ve schopnosti poskytnutí takové datové struktury, která by umožnila efektivní aktualizace mapy, a také by zároveň nezhoršila, ba naopak by zefektivnila i samotnou lokalizaci robota, se složitostí, která by byla nezávislá na počtu iterací Particle filtru. A přesně to přináší Distribuované mapování částic neboli také *DP-Mapping*, které je vysvětleno dále pomocí dvou pojmů reprezentující dvě datové struktury, a to takzvaný *Rodový strom* a samotná *Mapa*.

3.4.3.1 Udržování rodového stromu částic

Začněme nejdříve s rodovým stromem a konkrétně s jeho udržováním, které je velice důležité. A přitom základní myšlenka udržování rodového stromu částic, je poměrně jednoduchá. Samotný strom je tvořen počáteční částicí, která tvoří jeho kořen, ze kterého pak vedou cesty k potomkům této částice. Každá částice si udržuje ukazatel na svého rodiče a má přiděleno unikátní ID. Nakonec si každá částice uchovává čtvercovou mřížku(část mapy), kterou aktualizuje(aktualizovala).

To, jak nám tento strom pomáhá při lokalizaci robota, bude vysvětleno později. V této části bude vysvětleno jak zařídit, aby strom zůstal velikostně ohraničen, bez ohledu na počet iterací použitého Particle filtru.

Toto ohraničení si udržujeme tím, že jednoduše odstraníme uzly stromu, které nepotřebujeme, aby se nám nenakupily a daný strom nebyl více na obtíž, než k užitku. Konkrétně odstraníme ty uzly, které jsou tvořeny částicemi, jenž nemusí mít žádné děti. Takové částice můžeme ze stromu odstranit, jelikož jejich odstraněním nepřijdeme o žádnou důležitou informaci. Samozřejmě toto může způsobit, že i rodič této částice bude nyní bez dítěte a tudíž rekurzivním voláním funkce pro odstranění částice bez dětí, odstraníme i jeho, dokud neodstraníme všechny takové částice. Tím se zbavíme takzvaných mrtvých větví našeho stromu. Po našem „prořezání“ se stromem je jasné, že nám v něm zbyly pouze ty částice, které jsou předchůdci současné generace částic.

Náš strom ovšem stále obsahuje více informací, než je potřeba aby si pamatoval. Naštěstí tento algoritmus přichází i s řešením tohoto problému. Už jsme odstranili částice, které neměly děti. Nyní přicházejí na řadu ty částice, které mají potomka pouze jednoho. Ty odstraníme jednoduše tím, že necháme zhroutit danou větev stromu. Tato akce, zhroucení zbytečných větví stromu, bude mít za následek sloučení rodičovských aktualizací a aktualizací od dětí nad danou mapou. Využití tohoto zhroucení se budeme věnovat v další kapitole. Použitím celého výše zmíněného postupu na celý strom(na všechny jeho uzly), dostaneme minimální podobu rodového stromu, který obsahuje několik potřebných a lehce prokazatelných vlastností(informací).

3.4.3.2 Reprezentace mapy

Jak už bylo zmíněno v kapitole zabývající se obecným vysvětlením distribuovaného mapování částic, tak hlavní výzva pro reprezentaci mapy, je vymyslet takovou datovou strukturu, která by umožňovala efektivní aktualizace mapy a zároveň efektivní lokalizaci(Pokud Vám to zní hodně podobě problému SLAM, tak máte pravdu). Klasický naivní přístup úplné mapy pro každou částici je, jak již bylo ukázáno, velmi neefektivní. Druhý přístup, který je o něco méně naivní, udržuje historii aktualizace každé částice, což je ale také neefektivní, jelikož to představuje závislost na počtu iterací Particle filtru.

Tento algoritmus nabízí řešení reprezentace map pomocí sdružování částic s mapami, místo sdružování map s částicemi. DP-mapování tedy udržuje pouze jednu obsazenou mřížku (částice jsou distribuované po mapě). Na rozdíl od klasické obsazené mřížky, každý čtverec této mřížky ukládá vyvážený strom(jako je například červenočerný strom[10]). Strom je přichycen na ID částice, jenž provedly změny na obsazenosti daného čtverce.

Základní inicializace této mřížky tedy probíhá, jako inicializace matice prázdných stromů. Když potom nějaká částice provede pozorování nad nějakým čtvercem dané mřížky, zaznamená se její ID a výsledky jejího pozorování do příslušného stromu mřížky. Toto je velice výhodné a efektivní, jelikož tato metoda záznamu umožňuje každé částici chovat se tak, jakoby měla vlastní mapu. Dále je to také výhodné v případě, kdy chceme zjistit hodnotu měření v daném čtverci mřížky. Jelikož částici stačí zkontrolovat její rodový strom a podívat se, zda nějaký z jejich předků nemá aktuálnější pozorování pro daný čtverec. Pokud žádný z předků do daného čtverce nic nezapsal, pak částice označí tuto pozici jako neznámou.

Pojďme se nyní vrátit k tomu, jak jsme nechali zhroutit větev stromu, jenž obsahovala částice pouze s jedním dítětem. Zde má totiž toto zhroucení své využití. Za prvé, jak již byli zmíněno, tak při zhroucení je sada čtverců aktualizovaných dítětem, sloučena do rodičovy sady. Zadruhé, pro každý čtverec navštívený dítětem, změníme ID klič uložený ve vyváženém stromu tak, aby odpovídal rodiči. Pokud nastane situace, že jak rodič, tak dítě udělaly aktualizaci nad daným čtvercem, rodičova aktualizace je nahrazena aktualizací dítěte. Dítě je poté odstraněno ze stromu a rodičovo vnouče se stane jeho přímým dítětem. Toto zajišťuje, že počet položek uložených ve vyváženém stromu v každém čtverci mřížky je $O(P)$. Přičemž P označuje počet částic.

3.4.4 Výpočetní složitost

Celou dobu je tady zmiňováno, jak je naivní přístup špatný a v čem všem ho algoritmus DP-SLAM předčí. Ovšem jedná pěkná věc na naivním přístupu o udržování kompletní mapy pro každou částici je jednoduchost. Pomineme-li náklady na blokové kopírování map, tak vyhledávání a změny na mapě lze provádět v konstantním čase. V tomto ohledu se může na první pohled zdát mapování distribuovaných částic jako méně efektivní. Nicméně ve skutečnosti jsou DP mapy mnohem lepší než naivní přístup.

Nyní se podíváme na samotné vyhledávání. Vyhledávání na DP mapě vyžaduje srovnání mezi předky částice a vyrovnaným stromem v daném čtverci mřížky. Ukážeme

si složitost na příkladě. Nechť D označuje hloubku rodového stromu, tedy maximální délku rodokmenu částice. Přesněji řečeno, jelikož není zaručeno, že rodový strom bude vyvážený, D může být rovno $O(P)$. Nicméně v praxi tento případ téměř nikdy nenastane, bylo zjištěno, že $D \approx O(logP)$, jelikož povaha převzorkování částice vede k velmi vyváženým rodovým stromům. Proto můžeme dokončit naše vyhledávání po pouze D přístupech do našeho stromu. Vzhledem k tomu, že vyvážený strom může pojmout pouze P položek a jednotlivé hledání zabere $O(logP)$ času, přistupování k jednotlivým čtvercům mřížky v mapě může být vykonáno v $O(DlogP)$ čase.

Pro lokalizaci bude každá částice muset dělat $O(A)$ přístupů k mapě. Jelikož každá částice musí přistoupit k celému zkoumanému prostoru pro svou vlastní mapu, bude potřeba $O(AP)$ přístupů, což dává lokalizaci s DP mapováním složitost $O(ADPlogP)$.

Pro dokončení celkové analýzy musíme vzít v úvahu dva následující detaily: Cenu vkládání nových informací do mapy a cenu udržování rodového stromu. Vzhledem k tomu, že požíváme vyvážený strom pro každý čtverec mřížky tak obojí, jak vkládání, tak mazání v rámci naší mapy zabere $O(logP)$ za jeden vstup. Každá částice může udělat maximálně $O(A)$ nových položek, které budou třeba odstranit jen jednou. Tím pádem celkový proces přidávání nových položek může být proveden v čase $O(ADlogP)$ pro jednu částici nebo $O(ADPlogP)$ celkově.

Může se zdát, že složitost udržování rodového stromu, je poměrně velká, ale není tomu tak. Zbývá tedy ukázat, že postarat se o udržování rodového stromu má svou cenu oprávněně. Konkrétně je třeba ukázat, že zhroucení uzlů stromů, jenž nemají žádné děti, nepřesahuje čas $O(ADPlogP)$. To nemusí být na první pohled zřejmé, vzhledem k tomu, že postupné operace hroucení mohou udělat soubor aktualizovaných čtverců pro jeden uzel v rodovém stromu tak velký, jako je celková mapa. Přesto však cena těchto operací bude $P(ADPlogP)$. Zaprvé je dobré zvážit cenu sloučení listu upravených čtverců dítěte s rodičovým listem. Pokud dítě upravilo n čtverců, musíme vykonat $O(nlogP)$ operací pro porovnání položek dítěte s položkami rodiče kvůli případným duplicitám.

Posledním krokem, který je nutný udělat, je aktualizace všech ID pro všechny položky dětí. Toho dosáhneme vymazáním původních ID a vložením kopie nového s rodičovým ID. Náklady na tuto operaci jsou opět $O(nlogP)$. Musíme si uvědomit, že každá položka mapy uložená v rodovém stromě částic má potenciálních D kroků při kterých může být zhroucena během kontroly stromu a jeho počtu dětí na každém uzlu, jelikož D je celkový počet uzlů mezi její prvotní pozicí a kořenem a žádné další uzly už se nikdy mezi těmito pozicemi nevytvoří. Při každé iteraci particle filtru, P částic každá vytvoří A nových položek s potenciálem D. Takže celkový potenciál při každé iteraci je $P(ADPlogP)$.

Celková výpočetní složitost algoritmu DP-SLAM může být tedy shrnuta takto: Pro particle filtr, který udržuje P částic, laser který skenuje A čtverců mřížky a rodový strom částic s hloubkou D, vyžaduje DP-SLAM:

- $O(ADPlogP)$ *operací pro lokalizaci, vycházející z:*
  - *P částic kontrolujících A čtverců mřížky*
  - *A vyhledávání o ceně* $O(DlogP)$ *za jeden čtverec mřížky*

- *$O(APlogP)$ operací pro vkládání nových dat do stromu, vycházející z:*
  - *P částic vkládající informace na A čtvercích mřížky*
  - *Vkládání o ceně $O(OlogP)$ za každý kousek informace*
- *Udržování rodového stromu částic s cenou $O(ADPlogP)$, vycházející z:*
  - *Ceny $O(logP)$ potřebné k odstranění pozorování nebo jeho posunutí o stupeň výše v rodovém stromu*
  - *Maximální potenciál ADP zaveden při každé iteraci*

3.4.5 Shrnutí metody

Byl zde představen algoritmus s názvem DP-SLAM, jako algoritmus založený na Particle filtrech určený pro lokalizaci a mapování. Dále bylo ukázáno, čím se tento algoritmus výrazně liší od algoritmů předešlých. Hlavní výhoda, je jeho nezávislost na předem získaných informacích, konkrétně na předem určených pozicích význačných bodů, což ho také zbavuje nutnosti řešit problémy s asociací dat, které vznikají kvůli těmto význačným bodům. Tento algoritmus je schopen, za použití velmi efektivních datových struktur, udržet několik tisíc map v reálném čase, poskytujících vysoce přesné mapování. DP-SLAM byl ozkoušen na skutečném robotovi, kde poskytl detailní mapu daného prostředí.

Tvůrci toho algoritmu věří, že tento algoritmus byl prvním svého druhu, který dokázal dosáhnout takové přesnosti pro daný typ dat a bez jakékoliv znalosti předem zadaných informací o prostoru. Nicméně tento algoritmus má také své limity. V případě použití laserových snímačů s fixně danou výškou skenování, nemusíme mít dostatečné informace o prostředí. DP-SLAM nabízí velmi jednoduchou reprezentaci mapy, v niž se předpokládá, že každá buňka mřížky je buď viditelná nebo zcela neprůhledná pro laser. Toto samozřejmě vytváří diskrétní chyby na okraji objektů a zmatené výsledky pro velmi malé objekty. Ovšem algoritmus DP-SLAM používá velmi malé buňky a tím tyto chyby minimalizuje, nicméně nelze je zcela eliminovat. Například při skenování stolu s nářadím, ze kterého by visely například kabely, by mohlo dojít k právě této chybě, kdy by naskenování kabelů způsobilo nárůst částic a chyby v lokalizaci a snížili by tak výkon algoritmu. Pro tuto kapitolu jsem čerpal z [13].

3.5 GraphSLAM

Poslední metodou, které se tato práce bude věnovat, je metoda s názvem GraphSLAM. Jedná se o metodu, kterou jsem si zvolil pro svou implementaci. Postup implementace metody ovšem nebude zahrnut v rámci této kapitoly, ale bude popsána v poslední kapitole práce. V této kapitole jde pouze o seznámení se se základní myšlenkou metody GraphSLAM, jako to bylo u metod předešlých.

Stejně jako předchozí metody, tak i GraphSLAM získal svůj název na základě toho, jakým způsobem řeší problém mapování prostoru. Jedná se o algoritmus určený k mapování, který používá takzvané *řídké grafy* vazeb. Základní myšlenka tohoto algoritmu

je velice jednoduchá: GraphSLAM získává z dat pořízených robotem soubor takzvaných *měkkých vazeb*, které jsou reprezentovány již zmíněným řídkým grafem. Z tohoto grafu, respektive z daných vazeb, je algoritmus schopen vypočítat vše potřebné, pro přesné určení mapy a cesty robota v rámci globálního odhadu. Vazby, jenž se v řídkém grafu nacházejí, jsou obecně nelineární, ale v průběhu jejich řešení jsou linearizována. V této kapitole bude GraphSLAM popsán jednak jako algoritmus pro tvorbu řídkých grafů nelineárních vazeb, a také jako metoda pro vyplnění řídké matice lineárními vazbami.

Při aplikaci algoritmu GrapSLAM na problém rozsáhlého mapování zjistíme, že je tento algoritmus schopen zvládnout velké množství prvků, a také spolupracovat s informacemi pořízenými pomocí GPS a použití těchto informací v mapování.

3.5.1 Mapování problému SLAM do Grafu

Pokud chceme lépe porozumět, jak využít grafy při pracování s algoritmem GraphSLAM, musíme si vysvětlit, jak se s těmito grafy pracuje obecně.

3.5.1.1 Offline SLAM problém

Nejdříve si ukážeme takzvaný Offline SLAM. Offline problém je charakterizován schopností shromažďovat veškerá data během mapování a přeměnit tato data v mapu, jakmile robot ukončí svou operaci.

Než začneme naše vysvětlení, budeme si muset zavést pár základním značení (proměnných), které budeme používat. Stejně, jako jsme to dělali i v metodách dřívějším, i tady budeme označovat čas pomocí proměnné t. Pozici robota v čase t poté označíme jako x_t. Dále budeme používat označení $x_{1:t}$, pro soubor pozic robota, na kterých byl od času 1, až po čas t. Mapu(svět) si potom označíme jednoduchou zkratkou m. Budeme předpokládat, že mapa bude časově neměnná, a proto k ní nebudeme přidávat index času. V tomto případě budeme mapu brát jako soubor prvků mj.

K dosažení mapy nějakého prostředí, robot musí být schopen „vnímat“. Jeho měření v čase t označíme jako z_t. Obvykle robot dokáže vnímat více vlastností a prvků kolem sebe, než pouze jeden, a proto si každé individuální měření označíme z_t^i. Následující funkce h ukazuje, jak takové měření vzniká:

$$z_t^i = h(x_t, m_j, i) + \varepsilon_t^i \tag{14}$$

V tomto případě, je ε_t^i Gaussova náhodná proměnná, udávající šum v měření s nulovou střední hodnotou a kovariancí Q_t. Proměnná m_j udává prvek mapy, zaznamenaný $i - tm$ měřícím paprskem v čase t.

Jak už bylo zmíněno, GraphSLAM může využívat pro svou práci informace pořízené pomocí GPS. V tom případě můžeme z funkce h zcela odebrat proměnnou m_j udávající prvek mapy, jelikož tuto informaci mu je schopno GPS poskytnout.

$$z_t^i = h(x_t, i) + \varepsilon_t^i \tag{15}$$

A konečně se dostáváme k samotnému pohybu robota. Robot ve SLAMu změní svou pozici za pomocí kontrolních příkazů. Příkaz uplatněný mezi časy $t-1$ a t je označován jako u_t. Přechodný stav je znázorněn pomocí následující funkce g:

$$x_t = g(u_t, x_{t-1}) + \delta_t \tag{16}$$

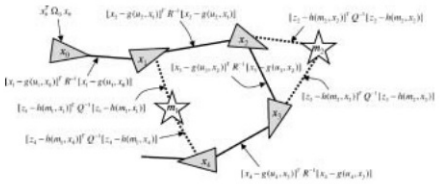
kde $\delta_t \sim N(0, R_t)$ znázorňuje šum v kontrolním příkazu. Funkce g může být brána jako kinetický model robota. Po znalosti těchto funkcí a pravidel, můžeme říci že výpočet pravděpodobnosti pro Offline SLAM, je daný následujícím výpočtem pravděpodobnosti výskytu nad cestou robota $x_{1:t}$ a mapy m.

$$p(x_{1:t}, m|z_{1:t}, u_{1:t}) \tag{17}$$

Jedná se o výpočet pravděpodobnosti výskytu nad celou cestou robota $x_{1:t}$ mapou m, místo počítání pouze pro současnou pozici x_t. Bylo zaznamenáno, že v reálných situacích je velice obtížné vypočíst pravděpodobnost pro více rozměrové mapy m.

3.5.1.2 Základní myšlenka GraphSLAMu

Nejlépe si tuto myšlenku ukážeme rovnou na příkladu, konkrétně na jednoduchém obrázku. Na obrázku níže je znázorněn GraphSLAM algoritmus. Je na něm ukázán graf, který GraphSLAM získává ze čtyř pozic robota označených jako x_1, x_2, x_3, x_4 a dva význačné body mapy m_1 a m_2. Spoje mezi pozicemi robota a význačnými body jsou rozděleny na dva druhy, pohybové spoje a spoje měření. Pohybové spoje logicky spojují jakékoliv dvě sousední pozice robota a spoje měření spojují pozici robota s význačným bodem, který byl z dané pozice pozorován. Každá hrana v daném grafu se vztahuje k nelineární vazbě. Jak se dozvíte později, tyto vazby reprezentují záporný logaritmus pravděpodobnosti měření a pohybového modelu, odtud pochází myšlenka o takzvaných *informačních vazbách*. Přidáním takových vazeb do grafu je pro GraphSLAM triviální, není třeba žádných složitých výpočtů.



Obrázek 7: GraphSLAM

Zdroj: [14]

Pro výpočet pravděpodobnosti výskytu bodů pro mapu, GraphSLAM linearizuje soubor vazeb. Výsledkem této linearizace je řídká informační matice[8] a informační vektor.

[8] Řídká matice je matice, která obsahuje převážně nulová čísla.

Řídkost této matice, umožňuje algoritmu GraphSLAM požít takzvaný algoritmus *eliminace proměnných*, což slouží k transformování grafu na nový, mnohem menší, definující pouze pozici robota. Pravděpodobnost výskytu cesty robota, je potom vypočítána použitím standardně odvozených technik.

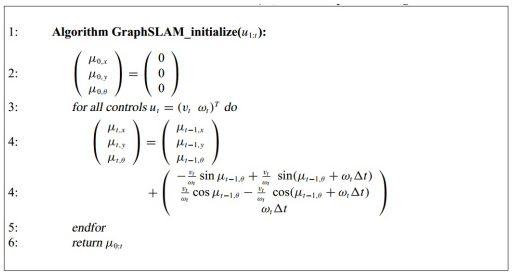
3.5.2 GraphSLAM algoritmus

Nyní přichází na řadu přesný popis výpočetních kroků algoritmu GraphSLAM. Tento algoritmus by se dal rozdělit na čtyři menší algoritmy které, jak si ukážeme níže, mají každý na starost jeden z hlavních kroků, při celkovém výpočtu algoritmu. Tyto algoritmy se nazývají **GraphSLAM initialize**, který slouží, jak už název napovídá k inicializaci informační matice a vektoru, dále pak algoritmus **GraphSLAM lienarize**, který také neslouží k ničemu jinému, než k sestavení informační matice a informačního vektoru, čili jejich naplnění. Třetí algoritmus, který zde bude popsán nese jméno **GraphSLAM reduce**. Tento algoritmus se stará o redukci dimensionality informační matice a informačního vektoru. A poslední algoritmus se nazývá **GraphSLAM solve**, který zprostředkovává finální výpočet.

Hlavní problém implementace, který zde bude popsán a vyřešen, je přeměna získaného výsledku podmíněné pravděpodobnosti $p(z_t^i|x_t, m)$ a $p(x_t|u_t, x_{1-t})$ tak, aby se dal zapsat jako jeden řádek informační matice, která je spolu s informačním vektorem klíčová pro algoritmus GraphSLAM. Hlavní problém této přeměny je fakt, že prvky informační matice jsou všechny lineární, tudíž před vložením výsledků do dané matice, je třeba tato data linearizovat. K provedení této linearizace potřebujeme počáteční odhad $\mu_{0:t}$ pro všechny pozice $x_{0:1}$.

Samozřejmě, že existuje celá řada řešení, které by vedly ke správnému určení počáteční μ vhodné pro naši linearizaci vstupních dat pro informační matici. Za všechny můžeme uvést například EKF SLAM, který můžeme pustit v rámci našeho algoritmu a použít přímo jeho odhad pro linearizaci. GraphSLAM má ovšem pro tento případ své vlastní mechaniky, jak dosáhnout tíženého výsledku. Tato technika, respektive tento algoritmus, se nazývá **GraphSLAM initialize** a prování odhad pomocí zřetězení pohybového modelu robota $p(x_t|u_t, x_{1-t})$. Vstupem tohoto algoritmu jsou kontrolní příkazy řídící robota $u_{1:t}$ a jeho výstupem je potom samozřejmě požadovaný soubor odhadů pozic $\mu_{0:t}$. První pozice robota, tedy pozice, kde robot začínal se svým měřením se inicializuje na nulu a pozice následující se počítají rekurzivně za pomoci rychlého pohybového modelu. Jelikož nás zajímá pouze vektor středních hodnot pozic $\mu_{0:t}$, protože potřebujeme výsledek, který bude možno vkládat na řádek informační matice, **GraphSLAM initialize** používá pouze deterministickou část pohybového modelu. Také nebere v tomto odhadu v úvahu žádné měření.

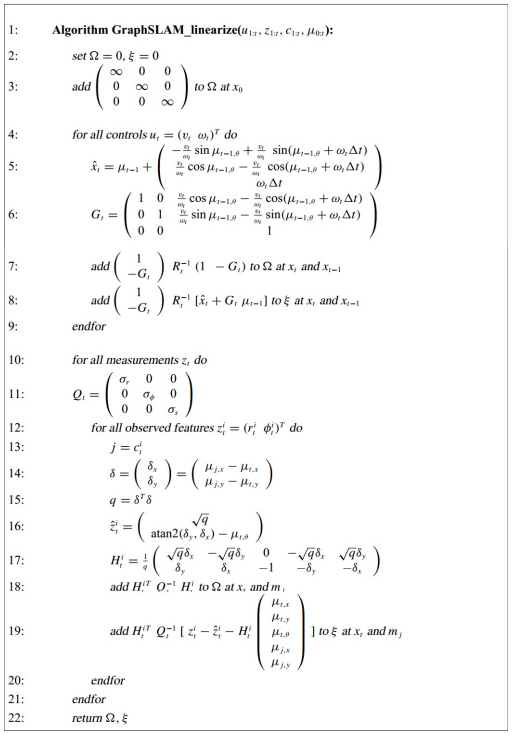
Jakmile jsme tedy získali počáteční $\mu_{0:t}$, je nyní GraphSLAM algoritmus schopen sestavit již zmíněnou dvojici, a to informační matici Ω a jí odpovídající informační vektor ξ. Tento algoritmus pouze sestavuje zmíněné prvky, čili je inicializuje na základní hodnoty, o jejich naplnění se stará následující algoritmus. Toto vše je dosaženo pomocí linearizace spojení v daném grafu. Jak přesně probíhá výše popsaný výpočet je znázorněno na následujícím obrázku, popisujícím algoritmus **GraphSLAM initialize**.



Obrázek 8: GraphSLAM initialize

Zdroj: [14]

Na řadu přichází algoritmus **GraphSLAM lienarize**. Tento algoritmus dostává jako vstupní data kontrolní příkazy $u_{1:t}$, měření $z_{1:t}$ a související proměnné $c_{1:t}$ a nakonec výstup algoritmu předešlého, čili $\mu_{0:t}$. Algoritmus poté postupně sestaví informační matici Ω a informační vektor ξ skrz linearizaci, pomocí lokálního přidávání dílčích matic, v souladu s informacemi získanými z každého měření a každého kontrolního příkazu. Opět je toto zobrazeno pomocí následujícího obrázku.



Obrázek 9: GraphSLAM linearize

Zdroj: [14]

Na řádku 2 **GraphSLAM lienarize** inicializuje informační prvky. Hodnota nekonečna, na hlavní diagonále následující matice, nastavuje hodnotu počáteční pozice x_0 na $(0, 0, 0)^T$. To je velmi důležité, jelikož v opačném případě, by výsledná matice byla jednotková. Od řádku 4 po řádek 9 dochází k integrování kontrolních příkazů. Co přesně dělají další řádky tohoto algoritmu se můžete dozvědět zde[14].

Výsledkem algoritmu **GraphSLAM lienarize** je tedy informační matice Ω a informační vektor ξ. Jak už bylo několikrát zmíněno, matice Ω je řídká. Nenulové prvky obsahuje pouze na hlavní diagonále, která se skládá z dílčích matic. Dále nenulové prvky obsahuje mezi následujícími pozicemi robota a mezi pozicemi a prvky mapy. Tento algoritmus je časově lineární v t.

Dalším krokem algoritmu GraphSLAM spočívá v redukci dimensionality informační matice a vektoru. To je dosaženo za pomocí algoritmu **GraphSLAM reduce**. Tento algoritmus bere jako vstup matici $\widetilde{\Omega}$ a vektor $\widetilde{\xi}$ a vrací redukovanou matici a vektor. Tato transformace je dosažena pomocí odebrání prvků m_j. Složitost tohoto algoritmu je opět lineární v t.

Posledním krokem GraphSLAM algoritmu je počítání středové hodnoty a kovariance pro všechny pozice na cestě robota, a také střední hodnoty odhadů pozic význačných bodů v mapě. Toto je dosaženo posledním algoritmem a to **GraphSLAM solve**. Návratová hodnota tohoto algoritmu je tedy průměr vypočítaný pro cestu robota a význačné body na mapě a také kovariance, ovšem ta pouze pro cestu.

Kvalita výsledků algoritmu GraphSLAM závisí na kvalitě počátečních odhadů (počátečních informací), spočítaných algoritmem **GraphSLAM initialize**.

3.5.3 Shnutí metody

Byl zde představen algoritmus s názvem GraphSLAM, řešící specifickou verzi problému SLAM a to takzvaný offline SLAM. Offline problém je charakterizován schopností shromažďovat veškerá data během mapování a přeměnit tato data v mapu, jakmile robot dokončí danou operaci. GraphSLAM toho docílí mapováním dat do řídkého grafu vazeb, které jsou následně mapovány do informační formy. Ta je poté redukována, za použití specifické transformace, která odebere proměnné(prvky) mapy. GraphSLAM poté obnoví mapu z odhadnuté pozice pomocí sekvence oddělených optimalizačních problémů.

Experimentální výsledky ve velkém městském prostředí ukázaly, že přístup algoritmu GraphSLAM skutečně vede k vytváření funkceschopných map. Dále ukázaly, že je skutečně snadné používat informace s jiných měřících zařízení, jako například GPS. Pro tuto kapitolu jsem čerpal z [14].

4 Vlastní implementace metody GraphSLAM

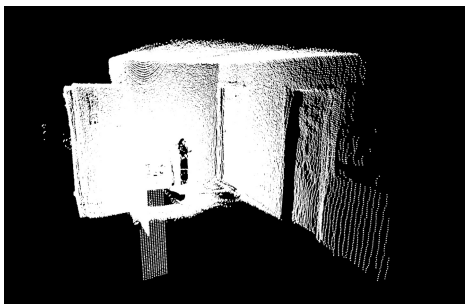
Během této kapitoly bude popsán postup implementace jedné z výše zmíněných metod. konkrétně se jedná o metodu GraphSLAM. Kapitola zahrnuje jak samotnou implementaci metody GraphSLAM, tak i přípravu na ní. Veškerou teorii o této metodě najdete v předchozí kapitole.

4.1 Vizualizace dat

Něž bylo vůbec možné začít s implementací nějakého algoritmu, bylo nutné udělat vizualizaci dat, které jsem měl k dispozici. Jednalo se o celkem 9 snímků jedné chodby. Každý z těchto snímků obsahoval přibližně 256 000 záznamů laseru, to znamená 256 000 bodů se třemi souřadnicemi(x, y a z), takovými, jak je zaznamenal laser.

Pro vizualizaci takového typu dat, jsem na doporučení vedoucího práce, zvolil knihovnu s názvem *Point Cloud Library*. Tato knihovna se obecně zabývá prací právě s laserovými snímky, jejich zobrazení a také následnou prací s nimi. Pro můj konkrétní případ jsem použil jejich *PCLVisualizer*, který je popsán a vysvětlen zde [15].

Po upravení kódu daného vizualizéru pro vlastní potřebu, jsem dostal jeden z obrazů chodby, jak ho zaznamenal(viděl) robot. Ukázka jednoho takového snímku lze vidět na následujícím obrázku.



Obrázek 10: Příklad skenované chodby

Když jsem tedy vyřešil problém vizualizace dat, mohl jsem se pustit do druhého problému, a to nalezení význačných bodů v dané chodbě. Ke knihově PCL se ještě vrátíme později, kdy ji budeme potřebovat.

4.2 Nalezení rohů

Jak už bylo zmíněno, nejen výše, ale i v kapitole zabývající se teorií ohledně metody GraphSLAM, tak tato metoda potřebuje pro svůj chod znát souřadnice význačných bodů. Pro svůj případ jsem jako význačné body zvolil rohy místnosti, respektive body, ve kterých se protínají roviny stěn. To znamená, že to mohly být i body, které nejsou reálnými rohy dané místnosti, ale nacházejí se na průsečíku zmíněných rovin.

Jako první věc jsem tedy musel najít stěny, respektive jejich obecnou rovnici roviny. Opět se zde nebavíme o reálných stěnách, ale patří zde i strop, podlaha, dveře, a jiné plochy, která svým tvarem připomínají rovinu. K nalezení těchto rovin jsem použil metodu s názvem *RANSAC* neboli *Random sample consensus*, která funguje na principu odhadování parametrů daného matematického modelu z řady výchozích prvků. Jedná se o nedeterministickou metodu, čili vytváří výsledek pouze s nějakou pravděpodobností, která roste spolu s počtem iterací. Jednoduše řečeno si na začátku zvolíme model který chceme hledat. V mém případě to byla rovina. Poté řekneme RANSACu, aby tento model hledal v objektu, který jsme mu poskytl, tedy v daném skenu místnosti. RANSAC poté v závislosti na mnou zvolených parametrech, konkrétně na maximální povolené chybě a počtu iterací, hledá tento model. Jak přesně tento algoritmus funguje se dozvíte zde [16].

Ovšem jak již bylo zmíněno, tato metoda nevrací přesné roviny. Je to způsobeno nepřesným zaznamenáváním informací pomocí laseru, kde jednotlivé body jsou vychýleny z roviny v řádech několika milimetrů. Což nezní jako velké číslo, ale když jsem po získání rovin metodou RANSAC, chtěl udělat průnik těchto rovin, tak jsem narazil na problém při počítání obecných rovnic těchto rovin, kdy při náhodném výběru tří bodů těchto rovin nastaly případy, kdy byly vybrány body, jenž ve výsledku netvoří požadovanou rovinu, jelikož jeden z nich byl vychýlen, což ve výsledku vychýlilo celou rovinu.

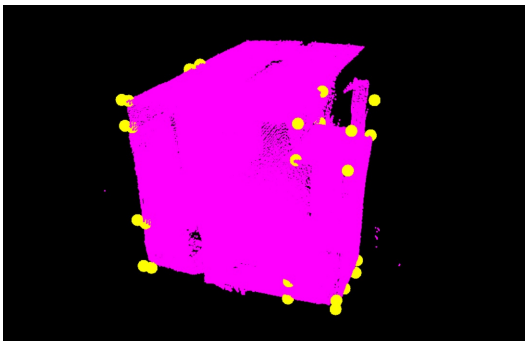
Jedno řešení tohoto problému bylo použití takzvané metody *nejmenších čtverců*. Tato metoda dokáže upřesnit tyto roviny dostatečně na to, abych byl schopen vypočítat jejich průsečík. Více o metodě nejmenším čtverců zde [17]. Pro výpočet metody RANSAC a metody nejmenších čtverců jsme použil část kódu z knihovny PCL. Jejich algoritmus obsahoval obě zmíněné metody a dokázal také vypočíst obecnou rovnici roviny každé z nalezených rovin.

Samozřejmě, že tuto funkci obsahující výpočet rovin jsem volal několikrát, abych získal co nejvíce rovin. Konkrétně jsem jich hledal 10. Problém byl, že když algoritmus rovinu našel, zapsal, uložil a začal hledat další rovinu, nastal případ, kdy našel rovinu, kterou měl již uloženou. Tak místo desíti požadovaných rovin našel pouze 4 nebo 3. Tuto chybu jsem odstranil pomocí nápadu odstraňovat již nalezené roviny z původního snímku. Čili pokaždé, když algoritmus rovinu našel, si ji uložil a „vyřezal“ ji z původního snímku, takže při druhém hledání ji už neviděl. Tímto jsem docílil požadovaných desíti rovin a mohl jsem začít hledat rohy.

Rohy jsem tedy získával z průsečíku tří rovin, čili jsem počítal soustavu tří rovnic o třech neznámých. Pro tento účel, jsem si pomocí matematické knihovny *Eigen* [18] vytvořil matici, která obsahovala pod sebou v řádcích zapsané obecné rovnice rovin. Vše jsem měl tedy připraveno, abych mohl počítat danou soustavu rovnic. Vybral jsem si způsob počítání pomocí *Gaussovy eliminační metody*, která mi spolehlivě vracela souřadnice průsečíku rovin. Tuto funkci jsem pak volal celkem 120 krát, jelikož jsem měl celkem 10 rovin, tak jsem musel testovat všechny možné kombinace jejich průniku, což mi dalo celkem velký počet bodů(rohů).

Nyní jsem musel vyřešit jiný problém. Vzhledem k již zmíněným chybám laseru, zde vznikaly body, které byly i několik desítek metrů vzdálené od dané chodby, jelikož roviny nebyly přesně rovné a protnuly se až mimo můj snímek. Tuto „chybu“ jsem opravil

pomocí vytvoření jakési pomyslné krabice kolem daného snímku a body nacházející se za úrovní této krabice, jsem jednoduše smazal. Po tomto promazání zbytečných bodů, jsem již dostával celkem přijatelné výsledky. Na obrázku níže můžete vidět rohy, které můj algoritmus našel na snímku, který odpovídá obrázku v kapitole 4.1.



Obrázek 11: Příklad nalezených rohů

Na obrázku jdou vidět body, které jakoby nebyly přímo rohy v dané místnosti, a některé dokonce levitují v prostoru. Nejedná se o chybu, ale o body, které vznikly průnikem rovin, jenž mají průsečík v přiměřené vzdálenosti od snímku. Body, které vznikly příliš blízko sebe (viz levý horní roh a levý dolní roh místnosti na obrázku), jsou výsledkem průniku roviny dveří, které jsou mírně vysunuty vzhledem ke stěně. Možná se zdá, že některé body jsou zbytečné a budou i překážet v dalším výpočtu, ale není nic lehčího, než tyto body v následujícím výpočtu ignorovat, a zaměřit se na ty, které potřebuji.

Nyní jsem tedy měl potřebné rohy dané místnosti a zbývalo zjistit, který roh z tohoto snímku je rohem ze snímku druhého. Čili jsme tuto funkci na získání rohů musel aplikovat na všech 9 snímků. Poté jsem získal jejich rohy a na řadu přišla takzvaná metoda *ICP*, která na základě získaných rohů dané snímky s drobnou chybou vyrovnala a tedy šlo vidět, který bod patří ke kterému. Metoda *ICP* je poměrně přesná a efektivní. Funguje na principu minimalizace rozdílu mezi dvěma *cloudy*[9] a snaží se tedy, o co největší sblížení těchto cloudů. Tato metoda se dá aplikovat jak na 2D, tak na 3D prostředí, což z ní dělá velice využívaný nástroj. Postup této metody spočívá ve vybrání si cíle a poté základu. Základem je cloud, který se během provádění algoritmu *ICP* nebude měnit, a bude naopak poskytovat informace o tom, jak se má cílový cloud posunout. Jednoduše vyrovnáme cílový cloud podle základového. Stejně jako metoda *RANSAC*, tak i tento algoritmus přímá parametry v podobě počtu iterací a maximální chyby, na jejichž základě se poté provede. Hlavním cílem tohoto algoritmu je tedy najít co možná nejlepší rotační matici a matici posunu, pro co nepřesnější přiblížení a vyrovnání cloudů. Více informací o metodě ICP se dozvíte zde [19].

[9]Cloud je označení pro soubor bodů. V našem případě se jedná o soubor obsahující souřadnice bodů, jak je zaznamenal laser

4.3 GraphSLAM

Když jsem tedy získal potřebné informace, mohl jsem začít implementovat samotnou metodu GraphSLAM. Jak už bylo zmíněno v kapitole, která se zabývá teoretickým popisem metody GraphSLAM, tak pro tuto metodu je klíčové získat informační matici a jí odpovídající informační vektor. To byl také můj následující krok.

Rozhodl jsem se daný problém řešit pro každou osu zvlášť, čili jsem si vytvořil celkem 3 matice a jim odpovídající 3 vektory pro osy x, y a z. Matice jsem implicitně nastavil na velikost 1x1 a přidáváním hodnot pozic robota a význačných bodů, se matice zvětšovala a postupně naplňovala. Tento postup jsme zvolil i pro informační vektory. Hlavní problém bylo zjistit, zda význačné body, jenž přidávám do dané matice a vektoru, nebyly již přidány. To jsem ošetřil jednoduchou funkcí, která mi vracela index daného význačného bodu, který jsem pak dále používal a kontroloval, zda význačný bod, který přidávám, nemá stejný index, jako body předešlé. Podobný problém byl v rámci jednoho snímku, kde některé rohy se zobrazovaly až příliš blízko sebe a mohly by tedy ohrozit celkový výpočet. Vyřešil jsem to pomocí funkce, která body(respektive jejich pozice), jenž se nacházely v mnou zadané vzdálenosti od sebe sečetla a vydělila dvěma, čímž jsem získal jeden bod, který se nacházel přibližně mezi původními body. Když jsem měl tedy naplněné jak matice, tak vektory, nezbývalo už nic jiného, než je mezi sebou vynásobit a získat ta výsledný vektor obsahující nejlepší možné odhady pro pozice robota a význačných bodů. Výsledný vektor se tedy vypočítal pomocí následujícího vzorce.

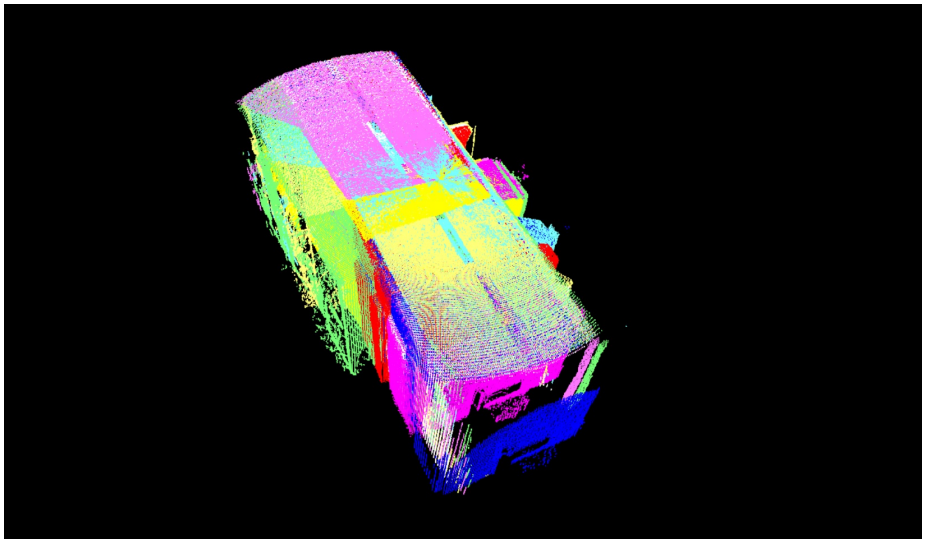
$$\mu = \Omega^{-1} * \xi \tag{18}$$

Kde μ představuje výsledný vektor, Ω informační matici a ξ informační vektor. Další obrázek(postup), nacházející se níže, ukazuje již konkrétní příklad výpočtu výsledného vektoru, kde známe pozice robota, pozice význačných bodů, chybu v pohybu robota a chybu v měření. Budeme zde uvažovat jednorozměrný prostor (Naplnění matice a vektoru pro jednu z os). Jako počáteční pozici si určíme $x_0 = -3$. V našem příkladě udělá robot 2 další „kroky". První krok bude $x_1 = x_0 + 5$ a druhý bude $x_2 = x_1 + 3$. Dále budeme uvažovat, že robot na své cestě viděl jeden význačný bod L_0 a to se vzdáleností $z_0 = 10$ z počátečního místa, vzdáleností $z_1 = 5$ po prvním pohybu a vzdáleností $z_2 = 2$ z poslední pozice. Poslední věcí, kterou zde známe jsou chyby měření a chyba pohybu, které budou v tomto případě obě rovny jedné.

$$\begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \\ L_0 \end{array}\begin{pmatrix} -3 \\ 2 \\ 5 \\ 7 \end{pmatrix} = \begin{array}{c} \begin{array}{cccc} x_0 & x_1 & x_2 & L_0 \end{array} \\ \begin{array}{c} x_0 \\ x_1 \\ x_2 \\ L_0 \end{array}\begin{pmatrix} 3 & -1 & 0 & -1 \\ -1 & 3 & -1 & -1 \\ 0 & -1 & 2 & -1 \\ -1 & -1 & -1 & 3 \end{pmatrix}^{-1} \end{array} \cdot \begin{array}{c} \\ x_0 \\ x_1 \\ x_2 \\ L_0 \end{array}\begin{pmatrix} -18 \\ -3 \\ 1 \\ 17 \end{pmatrix} \tag{19}$$

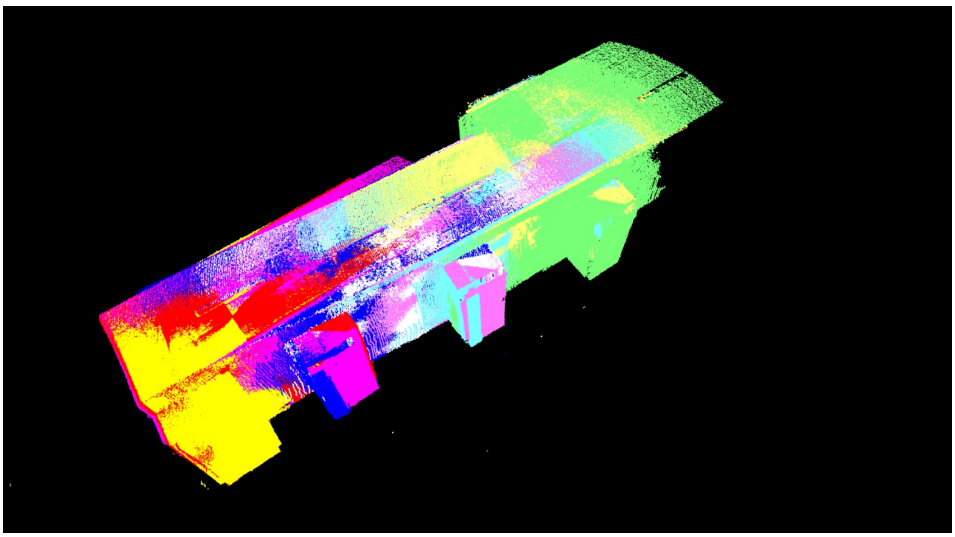
Stejným způsobem jsem i já postupoval pří spočítání výsledného vektoru pro všech 9 snímků(pozic robota). Následující obrázky ukazují, jak vypadaly snímky před použitím algoritmu, v jednotlivých částech algoritmu a nakonec, jak vypadá výsledný obraz

chodby, který algoritmus GraphSLAM vytvořil. První obrázek je tedy zobrazení všech devíti snímků, jak byly na začátku.



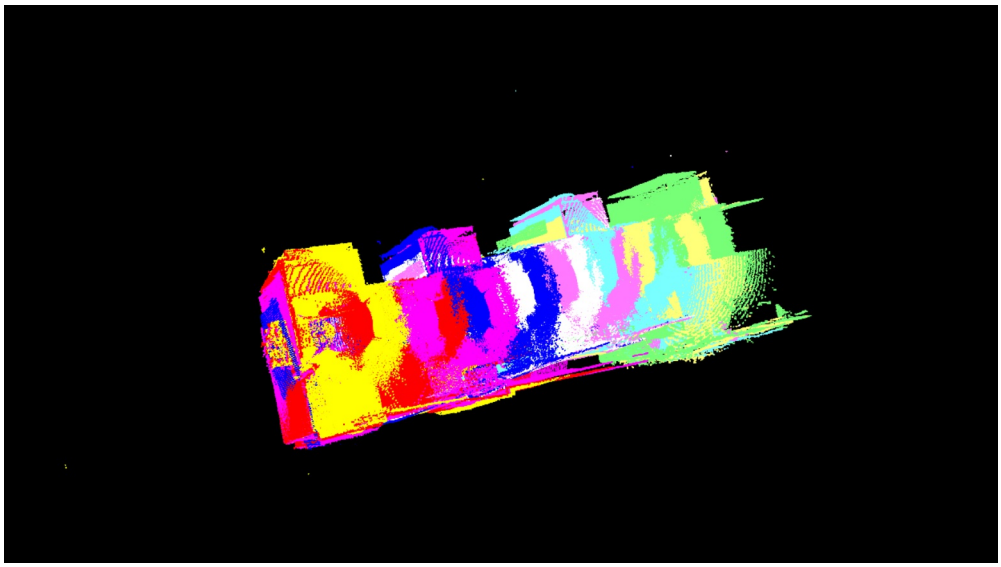
Obrázek 12: Obraz chodby bez úprav

Na obrázku lze zřetelně vidět, jak jsou snímky nezarovnané, například u modrého a fialového snímku vidíme výrazný posun. Také vychýlení způsobené chybou v pohybu robota je zde snadno rozpoznat. Další obrázek už ukazuje snímky po aplikaci metody ICP, která je vyrovnala, ovšem s nějakou chybou, kterou se právě snaží GraphSLAM odstranit.



Obrázek 13: Obraz chodby po metodě ICP

Když tyto dva obrázky porovnáte, lze vidět, že metoda ICP dané snímky celkem slušně zarovnala. Nám to ale nestačí a proto poslední obrázek ukazuje výsledné zarovnání pomocí metody GraphSLAM.



Obrázek 14: Výsledné zarovnání metodou GraphSLAM

Tento obrázek je již finální verzí mé práce. Ukazuje zarovnaný obraz chodby, který se skládá z celkem devíti původních snímků. Největší rozdíl je na snímku s modrou barvou, který to zarovnalo přesně se zadní zdí, tak jak má být. Šlo o srovnání v jednotkách milimetrů, takže změny na ostatních snímcích nemusejí být tak výrazné.

5 Závěr

Hlavním cílem práce je seznámení čtenáře s technologií LIDAR, konkrétně s její aplikací na mapování 3D prostoru. Dále pak popsání základních metod určených pro zmíněné mapování a v neposlední řadě popis vlastní implementace jedné z těchto metod.

V první části práce byla teoreticky popsána technologie LIDAR. Bylo zde uvedeno její široké využití, základní komponenty a také historie a vývoj. Druhá část práce se poté věnovala seznámení čtenáře se čtyřmi základními metodami pro mapování prostoru. Byly zde uvedeny základní myšlenky těchto algoritmů, jejich časové složitosti a srovnání s ostatními algoritmy. Náplní poslední části práce bylo popsání postupu vlastní implementace zvolené metody, řešení problémů, které při implementaci nastaly a zhodnocení výsledků.

6 Reference

[1] What is LIDAR?. [online]. [cit. 2014-10-14]. Dostupné z: http//oceanservice.noaa.gov/facts/lidar.html

[2] SCHUCKMAN, Karen a Mike RENSLOW. Introduction to Lasers and Lidar. [online]. 2014 [cit. 2014-10-14]. Dostupné z: https://www.e-education.psu.edu/lidar/l1_p3.html

[3] SCHUCKMAN, Karen a Mike RENSLOW. Lidar System Components. [online]. 2014 [cit. 2014-10-17]. Dostupné z: https://www.e-education.psu.edu/lidar/l1_p5.html

[4] SCHUCKMAN, Karen a Mike RENSLOW. History of Lidar Development. [online]. 2014 [cit. 2014-10-17]. Dostupné z: https://www.e-education.psu.edu/lidar/l1_p4.html

[5] KLEEMAN, Lindsay. Understanding and Applying Kalman Filtering. [online]. [cit. 2015-04-08]. Dostupné z: http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated3/kleeman_kalman_basics.pdf

[6] ABBEEL, Pieter. Rao-Blackwellized Particle Filtering. [online]. [cit. 2015-04-08]. Dostupné z: http://www.cs.berkeley.edu/~pabbeel/cs287-fa12/slides/RBPF.pdf

[7] MONTEMERLO, Michael, Sebastian THRUN, Daphne KOLLER a Ben WEGBREIT. FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem. [online]. s. 6 [cit. 2015-04-08]. Dostupné z: http://www.cs.cmu.edu/~mmde/mmdeaaai2002.pdf

[8] THRUN, Sebastian Thrun. Particle Filters in Robotics. [online]. [cit. 2015-04-09]. Dostupné z: http://robots.stanford.edu/papers/thrun.pf-in-robotics-uai02.pdf

[9] Maximum likelihood. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-11]. Dostupné z: http://en.wikipedia.org/wiki/Maximum_likelihood

[10] DEMAINE, Erik. Introduction to Algorithms: Balanced Search Trees. [online]. [cit. 2015-04-13]. Dostupné z: http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-046j-introduction-to-algorithms-sma-5503-fall-2005/video-lectures/lecture-10-red-black-trees-rotations-insertions-deletions/lec10.pdf

[11] LETO, Jonathan. 5 Minute Math Lesson: What is a Jacobian and what is it good for?. [online]. [cit. 2015-04-15]. Dostupné z: http://leto.net/x/2008/11/5-minute-math-lesson-what-is-a.html

[12] MONTEMERLO, Michael, Sebastian THRUN, Daphne KOLLER a Ben WEGBREIT. FastSLAM 2.0: An Improved Particle Filtering Algorithm for Simultaneous Localization and Mapping that Provably Converges. [online]. [cit. 2015-04-15]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.57.6159&rep=rep1&type=pdf

[13] ELIAZAR, Austin a Ronald PARR. DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks. [online]. [cit. 2015-04-15]. Dostupné z: http://people.ee.duke.edu/~lcarin/Lihan4.21.06a.pdf

[14] MONTEMERLO, Michael a Sebastian THRUN. The GraphSLAM Algorithm with Applications to Large-Scale Mapping of Urban Structures. [online]. [cit. 2015-04-18]. Dostupné z:http://robots.stanford.edu/papers/thrun.graphslam.pdf

[15] PCLVisualizer. Dostupné z: http://pointclouds.org/documentation/tutorials/pcl_visualizer.php.

[16] RANSAC: Random sample consensus. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-22]. Dostupné z: http://en.wikipedia.org/wiki/RANSAC

[17] MAŘÍK, Robert. Metoda nejmenších čtverců. [online]. [cit. 2015-04-22]. Dostupné z: http://user.mendelu.cz/marik/prez/mnc-cz.pdf

[18] Eigen. Dostupné z: http://eigen.tuxfamily.org/index.php?title=Main_Page

[19] EGGERT, D.W., A. LORUSSO a R.B. FISHER. Estimating 3-D rigid body transformations: a comparison of four major algorithms. [online]. [cit. 2015-04-22]. Dostupné z: http://www.cs.duke.edu/researchers/artificial_intelligence/temp/eggert_rigid_body_transformations.pdf