Bakalářská práce

Lukáš Kuhajda

Akademický rok 2018/2019

Obsah

Kapitola 1

Úvod

Tato práce je věnována systémům, které pomocí měření z LIDARu (Light Detection And Ranging) utváří mapu prostředí, v němž se pohybují. Řešením tohoto problému je metoda simultánní lokalizace a mapování (SLAM), jejíž popisem se zabývá první část práce. Daný problém je založen na situaci, kdy je mobilní robot umístěn do neznámého prostředí a jeho úkolem je určovat svoji pozici a utvářet mapu. V této kapitole jsou popsány vnitřní principy a přístupy k řešení problému. Daná tematika momentálně vstupuje do podvědomí i širší veřejnosti, neboť pomalu dochází k přechodu na autonomní vozidla, která fungují na podobných principech. Autonomní vozidla však využívají i mnoho dalších senzorů, jako jsou například kamery. Tato práce je zaměřena čistě na systémy využívající 2D LIDAR, tedy senzor, měřící vzdálenosti pouze v jedné výškové hladině.

Následující kapitola se věnuje třem nejrozšířenějším 2D SLAM systémům, přesněji se jedná o gMapping, HectorSLAM a Google Cartographer. Je zde popsáno, na jakých principech každý ze systémů pracuje a jak je daný přístup do systému implementován.

Další kapitola je zaměřena na porovnání zmíněných systémů na naměřených datech. Nahrávání a zpracovávání dat probíhalo pomocí operačního systému pro roboty (Robot operation System → ROS). Díky ROSu lze snadno propojit více zařízení, na kterých běží. Ve frameworku ROS je také možné spouštět simulace, tudíž je pomocí ROSu možné získat jak data reálná, tak data simulační. Obě tyto možnosti byly v práci využity, součástí je též popis přístupu ke všem druhům nahraných dat. Data byla nahrána, aby na nich mohly být spuštěny jednotlivé systémy a na základě výsledků je poté porovnat. Pro srovnávání byla simulační a reálná data rozšířena ještě o stažený dataset z univerzity MIT. Rozsah dat je v tomto případě větší a výsledky tak ukazují jiný pohled na systémy, než je tomu při datech z menších prostor. Zmíněné srovnání je prováděno dle kritérií v sekci 4.3.1. Po provedení porovnání systémů na jednotlivých datasetech jsou uvedeny výsledky, jejich popis a diskuze nad jejich relevantností.

V páté kapitole je blíže prozkoumán systém gMapping. Jsou zde popsány důvody, které vedly k vybrání tohoto systému. Systém je podroben analýze funkčnosti v závislosti na zvolených parametrech. Součástí této kapitoly je také návrh změn na vylepšení systému, které by vedly jak k zlepšení výkonu, tak k lepší uživatelské

přívětivosti.

Poslední kapitolou je závěr, ve kterém je postupně zhodnocena celá práce. Nejdříve je však potřeba seznámit čtenáře s definicí a řešením úlohy SLAMu, jednotlivými systémy a zejména výsledky. Definice SLAM založená na teorii pravděpodobnosti bude podrobně rozebrána v další kapitole.

Kapitola 2

Simultánní lokalizace a mapování

V této sekci se nachází seznámení s problémem simultánní lokalizace a mapování. Je zde probírána jak historie, tak struktura algoritmů, na kterých jsou založeny jednotlivé systémy řešící problém SLAM.

2.1 Historie

Za počátek diskuze problému se považuje konference Robotics and Automation Conference, konaná v roce 1986. Pravděpodobnostní metody byly tehdy ještě velmi nerozvinuté, jak v robotice, tak i v umělé inteligenci. Došlo tedy pouze k debatě na dané téma.

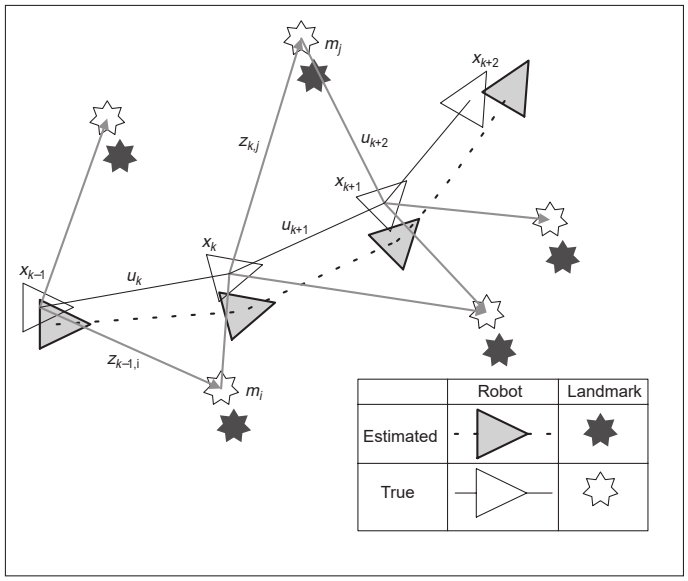
K většímu posunu kupředu se dostalo o pár let později, kdy vyšla práce pojednávající o vztahu mezi orientačními body (landmarky) a snížením geometrické nepřesnosti. Důležitým poznáním bylo zjištění, že mezi odhady landmarků na mapě je velká korelace, která je rostoucí s dalšími pozorováními.[1]

Ve stejném období vznikaly základy vizuální navigace a navigace pracující se sonarem s použitím Kalmanova filtru. Práce byly v základu podobné. Ukazovaly, že odhady landmarků získané pohybem robota prostředím, jsou v korelaci s ostatními kvůli chybě v odhadu pozice robota [2]. Je tak třeba mít stav složený z pozice robota a landmarků, tím však vznikal velký stavový vektor s náročností rostoucí v kvadrátu. V daném období byla tendence snižovat korelaci landmarků.

Později došlo k sjednocení problémů lokalizace a mapování a závěru, že snaha minimalizovat korelaci mezi landmarky byla chybná. Naopak bylo v zájmu korelaci co nejvíce zvýšit. Struktura SLAMu, a celkově první použití tohoto akronymu, byla prezentována v roce 1995 na International Symposium of Robotics Research (ISRR)[3]. Poté se v roce 1999 na ISRR odehrálo první zasedání pojednávající přímo o SLAM a došlo k představení práce dosahující dostatečné konvergence mezi SLAMem využívající Kalmanův filtr a pravděpodobnostními metodami pro lokalizaci a mapování.[4]

2.2 Formulace a struktura

Jedná se o proces, při kterém robot vytváří mapu prostředí v němž se pohybuje a na základě mapy určuje svoji pozici v prostoru. Pro určování trajektorie robota a rozložení landmarků není třeba předchozí znalosti jeho lokace, neboť odhad těchto parametrů probíhá v reálném čase. Tento proces je znázorněný na obrázku 2.1, ze kterého je vidět, že odhad nemusí být přesný. Pro metody řešící problém SLAM je tak cílem minimalizovat danou chybu, neboli drift.



Obrázek 2.1: Znázornění chyby v odhadu pozice a landmarků (z [5]).

2.2.1 Pravděpodobnostní SLAM

Cílem je získat v časovém okamžiku k odhad hustoty pravděpodobnosti

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{2.1}$$

pro každý časový okamžik k, kde $\mathbf{x}_k$ je stavový vektor popisující pozici a orientaci robota, $\mathbf{m}$ je množina landmarů, tedy mapa, $\mathbf{Z}_{0:k}$ jsou všechna pozorování landmarků, $\mathbf{U}_{0:k}$ je historie vstupů a $\mathbf{x}_0$ je počáteční stav. Jedná se tedy o sdruženou posteriorní hustotu mapy, stavu vozidla s ohledem na zaznamenané pozorování, řídící vstupy a počáteční stav robota. Pro výpočet je využit Bayesův teorém, který vyžaduje rozdělení algoritmu do dvou kroků. Prvním krokem je predikce, která využívá model pohybu robota. Tím druhým je korekce modelu pozorování, ke kterému jsou třeba data ze senzorů. Pohybový model a model pozorování tak popisují vliv vstupního řízení a pozorování.

Model pozorování popisuje pravděpodobnost zisku pozorování $\mathbf{z}_k$, pokud je známa poloha vozidla $\mathbf{x}_k$ a landmarků $\mathbf{m}$.

$$P(\mathbf{z}_k | \mathbf{x}_k, \mathbf{m}) \tag{2.2}$$

Model pohybu vozidla může být popsán jako stavový přechodový model. Přechodový stav je předpokládán jako Markovův proces, při kterém následující stav $\mathbf{x}_k$ je závislý pouze na předchozím stavu $\mathbf{x}_{k-1}$ a aplikovaném řízení $\mathbf{u}_k$ a není tak závislý ani na mapě, ani na pozorování.

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{2.3}$$

Implementace je tak ve formě dvoukrokého rekurzivního algoritmu.

Aktualizace času (predikce):

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \times P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1}. \tag{2.4}$$

Aktualizace měření (korekce):

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})}. \tag{2.5}$$

2.2.2 Struktura

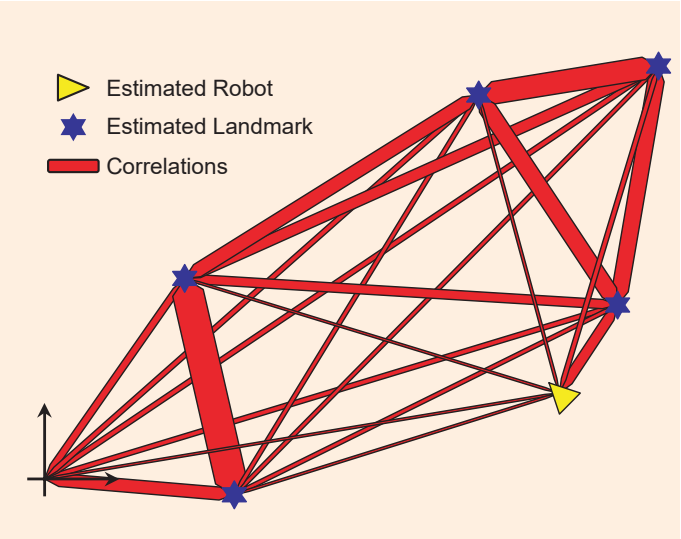
Model pozorování udává závislost polohy vozidla a pozice landmarků, z čehož vyplývá, že sdružená posteriorní pravděpodobnost nemůže být rozdělena na

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{z}_k) \neq P(\mathbf{x}_k|\mathbf{z}_k) P(\mathbf{m}|\mathbf{z}_k), \tag{2.6}$$

neboť by to vedlo k chybným odhadům. Dalším zdrojem chyb je špatný odhad pozice robota. Landmarky jsou ale silně korelované, takže chybný odhad landmarku vůči mapě nevede k chybné poloze dvou landmarků navzájem.

Velmi důležitým poznatkem bylo zjištění, že korelace mezi landmarky monotónně vzrůstá s počtem jejich pozorování (dokázáno pouze pro lineární Gaussovský případ [6]). Odhad pozice landmarku je tedy s narůstajícím počtem pozorování monotónně přesnější. Tento jev nastává díky, v podstatě, skoro nezávislému měření relativních pozic mezi landmarky. Ty jsou zcela nezávislé na natočení vozidla a úspěšné pozorovnání z tohoto bodu může nést další nezávislá měření relativních rozložení landmarků.

Pohybem v prostoru, robot získává pozorováním novou pozici známých landmarků vůči sobě a dle této informace aktualizuje jejich pozici a též svoji odhadovanou polohu. Pokud již nějaký landmark není pozorován, tak je jeho pozice aktualizována dle změny pozorovaných landmarků. Při pozorování nových landmarků dochází ke korelaci s již známými, čímž se vytváří síť. Čím častěji jsou dva landmarky pozorovány při jednom měření, tím je síla korelace větší. Opětovným projížděním prostředím je tak získána přesnější a robustnější mapa. Například násobným průchodem jedné smyčky se získá znatelně kvalitnější záznam o daném prostředí, než pouze jedním projetím. Pro lepší představu, na obrázku 2.2 se nachází ukázka korelací mezi jednotlivými landmarky a zároveň jejich korelace s aktuální pozicí robota.



Obrázek 2.2: Znázornění korelací mezi landmarky a robotem (z [5]).

2.3 Řešení problému SLAM

Při řešení je potřeba adekvátně obsáhnout jak složku modelace prostředí, tak i tvorbu pohybového modelu. Je řada možností jak tento problém řešit, například princip Monte Carlo, kdy se rozdělují hustoty pravděpodobnosti odhadu pozice robota. Další možností je Markovova lokalizace, jedná se o pravděpodobnostní formu SLAM. Nejčastější reprezentace problému ve formě stavového modelu zatíženého šumem, což vede k použití rozšířeného Kalmanova filtru (v originále *extended Kalman filter* $\rightarrow EKF$). Možností je ještě rozčlenění pohybového modelu vozidla na vzorky s obecnějším negausovským rozdělením pravděpodobnosti. V tomto případě je řeč o použití Rao-Blackwellizovaného částicového filtru. Dalším rozšířeným způsobem je Graph-based SLAM, kde jsou pozice robota vůči sobě poskládány v grafu a následně zpětně optimalizovány.

2.3.1 EKF-SLAM

EKF-SLAM je třída algoritmů využívající *extended Kalman filter*. Rozšířený Kalmanův filtr je implementací Bayesovské statistiky, která pracuje s gaussovským nelineárním systémem.
Pohyb robota je zde popsán rovnicí

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \leftrightarrow \mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \tag{2.7}$$

kde $\mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k)$ je funkce modelující pohyb robota a $\mathbf{w}_k$ jsou chyby měření.

Model pozorování je vyjádřen vztahem

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \leftrightarrow \mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{m}) + \mathbf{v}_k, \tag{2.8}$$

kde funkce $\mathbf{h}(.)$ popisuje geometrické vlastnosti pozorování a $\mathbf{v}_k$ jsou chyby měření. Tyto dvě definice jsou využity v metodě EKF k výpočtu střední hodnoty a kovariance sdruženého posteriorního rozložení

průměr:

$$\begin{bmatrix} \hat{\mathbf{x}}_{k|k} \\ \hat{\mathbf{m}}_k \end{bmatrix} = E \begin{bmatrix} \mathbf{x}_k | \mathbf{Z}_{0:k} \\ \mathbf{m} \end{bmatrix}, \tag{2.9}$$

kovariance:

$$\mathbf{P}_{k|k} = \begin{bmatrix} \mathbf{P}_{xx} \mathbf{P}_{xm} \\ \mathbf{P}_{xm}^T \mathbf{P}_{mm} \end{bmatrix}_{k|k} = E \left[\begin{pmatrix} \mathbf{x}_k - \hat{\mathbf{x}}_k \\ \mathbf{m} - \hat{\mathbf{m}}_k \end{pmatrix} \begin{pmatrix} \mathbf{x}_k - \hat{\mathbf{x}}_k \\ \mathbf{m} - \hat{\mathbf{m}}_k \end{pmatrix}^T | \mathbf{T}_{0:k} \right]. \tag{2.10}$$

Aktualizace času je pak počítána jako

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{2.11}$$

$$\mathbf{P}_{xx,k|k-1} = \nabla \mathbf{f} \mathbf{P}_{xx,k-1|k-1} \nabla \mathbf{f}^T + \mathbf{Q}_k, \tag{2.12}$$

kde $\nabla \mathbf{f}$ je Jacobián z funkce $\mathbf{f}$, která vychází z odhadu stavu $\hat{\mathbf{x}}_{k-1|k-1}$.
Aktualizace pozorování se provádí výpočtem rovnic

$$\begin{bmatrix} \hat{\mathbf{x}}_{k|k} \\ \hat{\mathbf{m}}_k \end{bmatrix} = [\hat{\mathbf{x}}_{k|k-1} \hat{\mathbf{m}}_{k-1}] + \mathbf{W}_k [\mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}, \hat{\mathbf{m}}_{k-1})] \tag{2.13}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{W}_k \mathbf{S}_k \mathbf{W}_k^T, \tag{2.14}$$

kde

$$\mathbf{S}_k = \nabla \mathbf{h} \mathbf{P}_{k|k-1} \nabla \mathbf{h}^T + \mathbf{R}_k \tag{2.15}$$

$$\mathbf{W}_k = \mathbf{P}_{k|k-1} \nabla \mathbf{h}^T \mathbf{S}_k^{-1}, \tag{2.16}$$

přičemž $\nabla \mathbf{h}$ je Jacobián z funkce $\mathbf{h}$, která vychází z odhadu stavu $\hat{\mathbf{x}}_{k|k-1}$ a odhadu mapy $\hat{\mathbf{m}}_{k-1}$.

Mezi hlavní problémy této metody se řadí konvergence, výpočetní složitost, asociace dat a nelinerita. Kvůli nelinearitě může dojít k větším nepřesnostem ve výsledku, neboť EKF-SLAM využívá lineárních modelů pro vyjádření nelineárního pohybu a modelu pozorování. Konvergence a konzistence modelu je tedy jistá pouze v lineárním případě.

Výpočetní složitost

Jedním ze základních kritérií všech algoritmů je jejich výpočetní složitost. Je tak třeba co nejvíce zjednodušit sdružený stav z pozice robota a landmarků a operace s nimi. Základním rozdělením metod redukujících výpočetní složitost, je rozlišování optimálních, konzervativních a nekonzistentních metod.

První typ, optimální metody, jsou založené na redukci daného výpočtu. Rovnicí pro aktualizaci pozorování omezují požadované výpočty. Výsledkem jsou pak odhady a kovariance, stejně tak, jako je tomu v případě plnohodnotné formy SLAM. U metod konzervativních dochází k reformulaci stavového prostoru do informační podoby. To umožňuje rozdělení výsledné matice s informacemi, což vede ke snížení výpočtů,

ale také k odhadům s vyšší neurčitostí nebo kovariancí. Konzervativní metody jsou nejčastěji implementovány v reálném použití, neboť dostatečně redukují výpočetní nároky a stále udržují dostatečnou odhadovací schopnost. Poslední možností jsou nekonzistentní metody. Jedná se o algorithmy, které mají nižší neurčitost nebo kovarianci, než algoritmy optimální. Pro řešení SLAM v praxi se ale nepoužívají.

Oddělné aktualizace

Metody využívající odděleně aktualizce vytvářejí optimální odhady. Při implementaci základní podoby aktualizace pozorování, se při každém novém měření aktualizuje jak stav vozidla, tak i mapy. To vede ke kvadratickému nárůstu složitosti s množstvím landmarků. Při této implementaci se však mapa rozdělí na submapy.

Rozlišují se dva způsoby možné implementace. První z nich pracuje na zmenšené oblasti, ale stále si drží globální referenční souřadnice, jedná se například o algoritmus *compressed EKF* (CEFK). Druhou možností je tvorba menších map s vlastním souřadnicovým rámcem neopouštějícím danou submapu. Jsou to algorithmy *constrained local submap filter* (CLSF, v překladu - omezený lokální submapový filtr). Pro další rozbor je vhodnější druhá možnost, neboť je jednodušší a při provádění operací s velkou frekvencí opakování je méně ovlivněna linearizačními chybami. Je také stabilnější a zabraňuje příliš velkému nárůstu globální kovariance.

Submapa se skládá z dvou nezávislých odhadů, které si stále udržuje. Jde o vektory x_G a x_R, kdy x_G je mapa složená z globálně referencovaných landmarků a globálně referencované pozice dané submapy a x_R je lokální submapa s lokálně referencovanou pozicí robota a lokálně referencovanými landmarky. Při získání pozorování se aktualizují pouze landmarky náležící aktuální submapě, ve které se robot nachází. Celkový globální odhad je pak získáván periodicky, zaevidováním submapy do mapy celé a použitím aktualizace omezení na společné vlastnosti obou map.

Asociace dat

Jedná se o velmi důležitý problém, neboť, i když během procesu tvorby mapy dojde pouze k jedné chybné asociaci dat, může to vézt k destabilizaci odhadu mapy. Často dokonce k pádu celého algoritmu.

Z prvu se k problému přistupovalo způsobem, kdy se každé jednotlivé zachycení landmarku porovnávalo se všemi odhady nacházejícími se v blízkém okolí. Tento individuální přístup je neproveditelný, pokud je nejistá pozice robota, tedy obvzláště v málo zaplněných prostředích.

Při popisu vzhledu je vidění jedním z hlavních způsobů snímání okolí. Podle typu senzoru se zaznamenává například tvar, barva, struktura, a tím je možné rozlišovat různé balíčky dat. To je poté využito pro přepověd dané asociace, nejčastěji pro problém s uzavřením smyčky. Metoda přinesla pokrok, jelikož počítá metriku podobnosti přes sekvenci obrazů, místo původního jednoho.

2.3.2 Rao-Blackwellizovaný částicový filtr

Forma SLAM založená na Rao-Blackwellizovaném filtru (RBPF), jinak nazývaná také jako FastSLAM, je založena na rekurzivním Monte Carlo modelu a dokáže reprezentovat nelineání stavový model. Dochází k odhadu celého posterioru, ne jenom nejpravděpodobnějšího sdružení dat, jedná se tedy o robustnější řešení než jakým je EKF.

Částice zde znázorňují trasu robota, jedna částice je tedy vzorkem v cestě. Každá částice také obsahuje mapu, kde je každý landmark reprezentovaný vlastním Gaussiánem. Díky použití částicových filtrů se jedná o jedinou formu SLAM, která řeší jak problém $full\ SLAM$, tak i problém $online\ SLAM$. Odhadováním pozice robota v každém časovém okamžiku je algoritmus označován jako online. O odhad pozice se starají právě částicové filtry.

Sdružený stav může být rozdělen zvlášť na komponentu robota a mapy. Posteriorní pravděpodobnost se tak dá odděleně počítat jako

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(m|X_{0:k}, Z_{0:k})P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0), \tag{2.17}$$

kde m je mapa, $X_{0:k}$ je trajektorie robota, $Z_{0:k}$ je sada všech již zpozorovaných landmarků, $U_{0:k}$ je historie řízení a x_0 úvodní pozice robota.

Rozdělení pravděpodobnosti zde není na jednotivých pozicích x_i, ale na celou trajektorii $X_{0:k}$ a tím se stávají jednotlivé landmarky na sobě nezávislými. Mapa je tedy reprezentována jako soubor nezávislých gaussiánů, což znamená lineární složitost oproti kvadratické u formy EKF. Hlavními ukazateli FastSLAMu je mapa, jež je počítána analyticky a vážené vzorky, jimiž je reprezentována trajektorie pohybu. Rekurzivní odhad je proveden částicovým filtrem pro stav pozice a EKF pro stav mapy.

Zpracování každého landmarku probíhá zvlášť, pozice se aktualizuje stejným způsobem jako v EKF a landmarky, které nebyly zpozorovány, zůstávají na původní pozici a neaktualizují se. Vzájemnou neprovázaností landmarků však vzniká chyba v odhadu, která s časem roste (viz. obrázek 2.3).

V čase $k-1$, je sdružený stav reprezentovaný jako

$$\{w_{k-1}^{(i)}, X_{0:k-1}^{(i)}, P(m|X_{0:k-1}^{(i)}, Z_{0:k-1})\}_i^V, \tag{2.18}$$

kde $w_k^{(i)}$ je váha vzorku i v časovém okamžiku k. Sdružený stav v čase $k-1$ je poté použit v procesu generování nové sady částic. Sada znázorňuje potenciální cesty robota, ze které je poté na základě porovnání pozorování vybrána ta nejpravděpodobnější částice.

V prvním kroku je pro každou částici vypočítáno návrhové rozložení, jež je podmíněno svojí historií. Z návrhu je odebrán vzorek x_k, který je poté sdružen k historii částice $X_{0:k}^{(i)}$. Krokem dva, dle funkce důležitosti, je stanovena váha vzorků:

$$w_t^i = \frac{a}{b}, \tag{2.19}$$

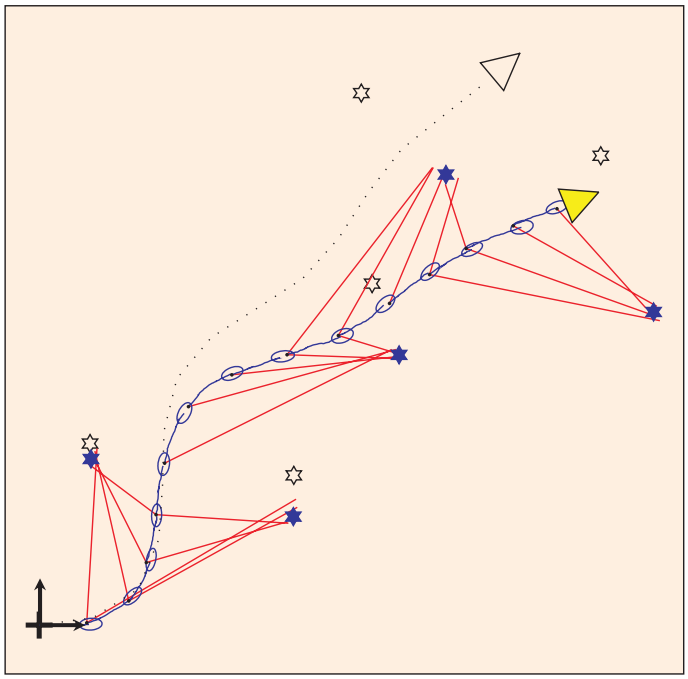
kdy a je cílové rozložení (rozložení, které je optimální pro sadu částic) a b je návrhové rozložení. Třetím krokem je případné převzorkování, které se provádí různě často

dle implementace. Posledním krokem je provedení EKF aktualizace pro každý již zpozorovaný landmark, který je při aktuálním pozorování zaznamenán.

Asociace dat

Funkcionalita systému popisovaná výše se vztahuje k datům, která mají známé sdružování. Pokud se jedná o data, pro která není tato niformace známa, je třeba systém rozšířit. Problém sdružení dat spočívá v tom, že na bázi dostupných dat není možno v časovém okamžiku k, určit proměnnou vyjadřující shodu c_k. Může se jednat například o neurčitost s pozicí, kdy má robot na základě pozorování možnost přiřadit více než jednu pozici.

Běžně bývá na měření jedna asociace dat a tedy vzorkování pouze podél cesty. Zde ale dochází i k vzorkování nad možnými rozhodnutími v průběhu cesty. Chyby ve sdruženích tedy nejsou zdaleka tak fatální, jako je tomu například u EKF. Pokud dojde k převzorkování, chybně určené částice se díky tomu můžou jednoduše nahradit.



Obrázek 2.3: Propojení odhadované trajektorie s pozorovanými landmarky (z [5]).

2.3.3 Graph-based SLAM

Graph-based SLAM je algoritmus založený na metodě nejmenších čtverců. Jedná se o offline algoritmus rozdělený do dvou částí, front-end a back-end.

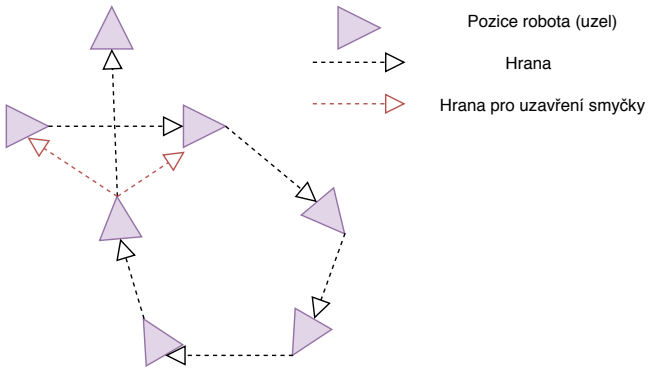
Metoda nejmenších čtverců je optimalizační metoda počítající s přeurčeným systémem, kdy je známo více rovnic, než je neznámých. Metodu představil na konci 18. století Carl Friedrich Gauss, a jak z názvu vyplývá, snaží se minimalizovat sumu chyb ve čtverci. Chyba je zde definována jako rozdíl odhadovaného stavu k aktuálnímu měření:

$$\mathbf{e}_i(x) = \mathbf{z}_i - \mathbf{f}_i(x). \tag{2.20}$$

Jedním z předpokladů pro správný průběh metody je dobrý počáteční odhad stavu. Dále, okolí minima chyby musí být hladké, bez skoků a zubů. V této oblasti se provádí lokální linearizace prvním stupněm Taylorova polynomu. Následně je vypočítána první derivace funkce chyby ve čtverci, která se položí rovna nule. Vyřeší se lineární systém, získá se nový stav, který by měl být lepší, než ten předchozí. Poté se znovu iteruje.

Tvorba pozičního grafu

Tvorba grafu je první částí metody Graph-based SLAM, tedy část vývojová (front-end). Dochází zde ke tvorbě pozičního grafu, který je zpracováván do mapy. Graf se složen ze dvou složek, a to z uzlů a hran. Uzel zde znázorňuje odhadovanou pozici robota a hrana omezení, které je mezi dvěma uzly. Toto spojení je nejisté a proměnné. Při projetí známým prostředím se vytvářejí hrany mezi uzlem novým a uzly již dříve zaznamenanými, což je využito například při uzavírání smyčky (viz. obrázek 2.4).



Obrázek 2.4: Propojení uzlů pomocí hran.

Nové hrany vznikají při splnění jedné ze dvou podmínek. Pokud je zaznamenán přesun robota z pozice x_i do x_{i+1}, hrana odpovídá naměřenné odometrii. Druhou možností je zpozorování stejného prostředí. Vzniká tak virtuální měření, ve kterém se porovnají obě tyto pozorování. Na základě rozdílu odhadované pozice při obou měřeních, se umístí nový uzel spojený odpovídající hranou(viz. obrázek 2.5).

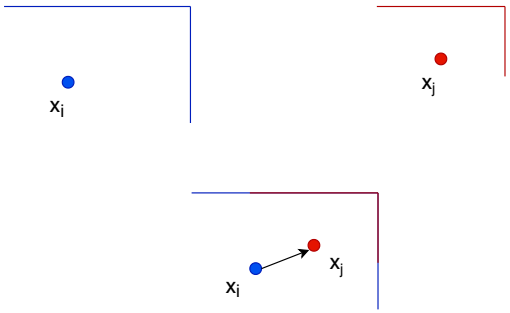
Optimalizace mapy

Druhou částí algoritmu Graph-based SLAM je back-end část, která zodpovídá za optimalizaci získaného grafu. Upravuje se zde pozice uzlů tak, aby byl získán stav, který co nejpřesněji vystihuje získaná měření. Po tomto procesu je pak možno utvořit mapu na základě známých pozic.

Rovnice pro optimalizaci vychází z metody nejmenších čtverců:

$$\mathbf{x}^* = argmin_{\mathbf{x}} \Sigma_{ij} \mathbf{e}_{ij}^T(\mathbf{x}_i, \mathbf{x}_j) \Omega_{ij} \mathbf{e}_{ij}, \tag{2.21}$$

kde $\mathbf{x}^*$ je cílové rozmístění uzlů, chyba $\mathbf{e}_{ij}$ vychází z (rovnice nahoře) a Ω_{ij} je informační matice pro danou hranu.



Obrázek 2.5: Znázornění funkce virtuálního měření.

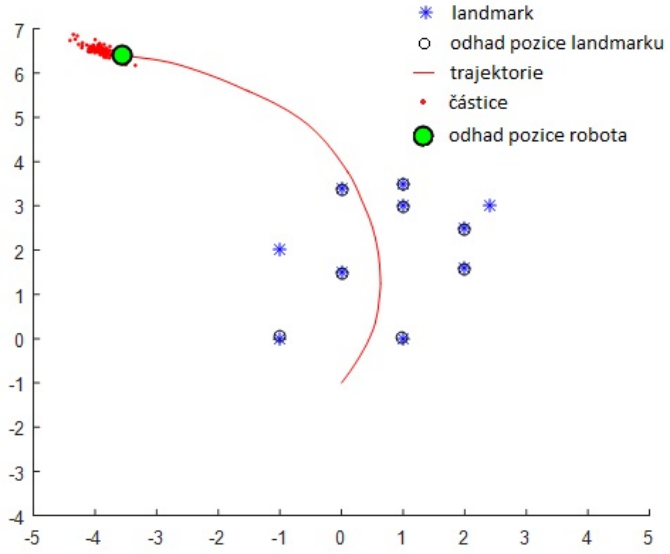
2.3.4 Reprezentace prostředí

Původně se svět modeloval jen jako soubor landmarků majících svůj určitý tvar, později však, zejména ve venkovním, podvodním a podzemním použití, se ukázala tato metoda jako nevyhovující. Hledaly se tak další metody, které by zlepšily reprezentaci prostředí. Výsledné možnosti jsou popsány v následujících odstavích.

Mapa z landmarků

V počátcích vývoje problému SLAM bylo pochopitelně potřeba vymyslet, jak reprezentovat výsledky měření a pozorování. Vznikla tedy metoda, kdy podobu mapy utvářely landmarky samotné. Když je nový landmark zpozorován, je zanesen do mapy a vytvoří se korelace k ostatním, již pozorovaným landmarkům. Odhadovaná pozice robota je také korelovaná vůči všem landmarkům.

Opětovným projížděním známého místa korelace mezi landmarky roste a mapa se zpřesňuje. Tím, že jsou všechny landmarky ve vzájemné korelaci, vytváří se pomyslná síť z těchto spojení. Tato metoda může být při správné implementaci tedy velmi přesná. Problémem je však výpočetní náročnost, která kvadraticky roste s přibývajícími landmarky. Pro menší prostředí se tedy metoda využít dá, pro větší je to ale výkonově příliš náročné.



Obrázek 2.6: Ukázka mapy z landmarků vytvořená pomocí simulace v prostředí MATLAB.

Mřížkové mapy

Metoda mřížkových map, v originále *Grid maps*, byla představena v 80. letech minulého století. Její základní myšlenkou je diskretizace spojitého prostředí do jednotlivých buněk, přičemž mřížky rozdělující prostor na buňky mají pevnou velikost. Metoda je poměrně náročná na paměť.

Důležitým faktorem je zde takzvaná obsazenost, v angličtině *occupancy*. V originále se tedy metoda nazývá *Occupancy Grid Maps*. informace o obsazení každé buňky může nabývat tří hodnot. Při inicializaci je hodnota nastavena na $m_i = 0.5$, kdy m_i představuje danou buňku. Tato počáteční hodnota znamená, že o buňce není žádná znalost. Poté, co se buňka dostane do dosahu senzoru, nabývá hodnota obsazenosti již pouze dvou hodnot. $m_i = 0$, pokud je rozhodnuto, že je buňka neobsazená a $m_i = 1$ v případě rozhodnutí o obsazenosti.

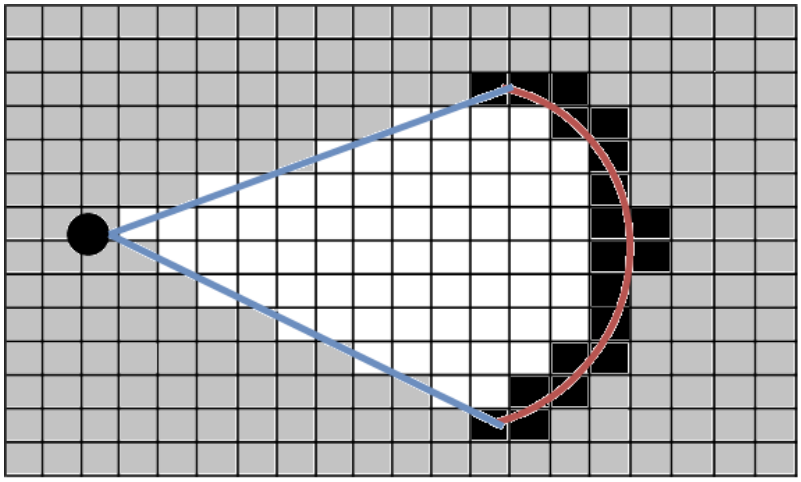
Buňky jsou na sobě zcela nezávislé a součinem pravděpodobností obsazení buněk je získáno pravděpodobnostní rozložení mapy:

$$p(m) = \Pi_i p(m_i). \tag{2.22}$$

Odhad mapy ze senzoru:

$$p(m|z_{1:t}, x_{1:t}) = \Pi_i p(m_i|z_{1:t}, x_{1:t}), \tag{2.23}$$

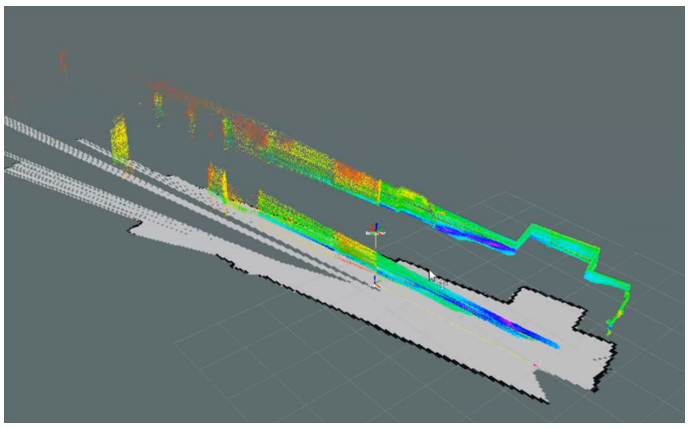
kdy $z_{1:t}$ jsou pozorování a $x_{1:t}$ pozice robota.



Obrázek 2.7: Rozložení mřížkové mapy dle aktuálního měření.

Mračno bodů

Výše zmíněné metody se vztahují především na práci s daty ve 2D. Pro zahrnutí třetí dimenze do mapy se tak využívají mračna bodů, v originálu *point clouds*. Data se získávají například pomocí kamer, či 3D LIDARů. Výsledné mapy tak vytvářejí komplexnější reprezentaci prostředí, které se skládá z milionů až miliard bodů. Jednotlivé body nesou vždy informaci o své pozici v prostoru, při využití kamery i přímo odpovídající barvu. Barevné rozlišení se dá použít i jinými způsoby. Je možné mít škálu barev, ve které přechází barvy dle zaznamenané výšky. Případně mohou barvy znázorňovat míru pravděpodobnosti, že se daný bod opravdu nachází na odhadované pozici.



Obrázek 2.8: Ukázka mapování pomocí mračna bodů.

Kapitola 3

Systémy SLAM

V této kapitole jsou popsány vybrané systémy SLAM. Výběr systémů byl proveden na základě největší rozšířenosti. Tímto kritériem byly zvoleny systémy gMapping, Hector SLAM a Google Cartographer. Níže je jednotlivě uvedena historie systému, její princip a způsob implementace.

3.1 GMapping

GMapping byl představen v roce 2007 v článku [8]. Tato metoda je založená na implementaci Rao-Blackwellizovaného filtru, kde si každá vytvořená částice nese svoji mapu prostředí. Pro praktické využití je tedy nutné částicový filtr zefektivnit, což bylo úkolem tohoto systému. Jedna možnost je zahrnout přesnost měření do návrhového rozložení, tím je získáno přesné vykreslení částic. Druhou možností, je volba vzorkovací techniky udržující rozumný počet částic. Udržuje se tak přesná mapa a snižuje se riziko vyčerpání částice, což je problém vznikající při převzorkování.

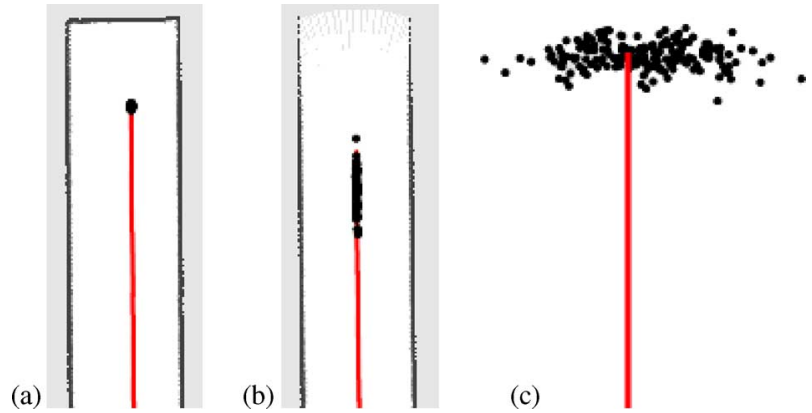
3.1.1 Mapování pomocí RBPF

Základním úkolem je odhad posteriorní pravděpodobnosti a tím zisk mapy a trajektorie pohybu. Odhad je prováděn na základě porovnání pozorování a informace z odometrie. Nejprve dochází k odhadu mapy, která je utvářena z porovnání pozorování, poté až trajektorie. Ta je získána z částic, které měly v rozhodovací době největší pravděpodobnost. Každá částice tedy reprezentuje část trajektorie.

Pro výběr správné částice je použit částicový filtr, v tomto případě Sampling Importance Resampling (SIR), který při mapování postupně zpracovává data ze senzoru, poté odometrii a aktualizuje sadu vzorků reprezentující posteriorní pravděpodobnost, která zahrnuje informace o mapě a trajektorii.

Nejprve se získají nové částice. To je provedeno odběrem vzorků z předchozí generace návrhového rozložení. V dalším kroku je částicím nastavena vážená důležitost, aby cílové rozložení nebylo rovno tomu navrhovanému. Poté dochází k převzorkování, kdy jsou částice přepisovány úměrně jejich váženým důležitostem. Nakonec se odhaduje podoba mapy, kdy je pro každou částici, na základě trajektorie vzorku a

historie pozorování, mapa vypočítána.



Obrázek 3.1: Rozložení částic dle typu prostředí.
a) konec chodby, b) cesta chodbou, c) otevřený prostor (z [9])

3.1.2 Vylepšené návrhy a adaptivní převzorkování

Pro zisk nové generace částic je třeba vykreslení vzorků z návrhového rozložení, kde platí úměra, čím lepší návrh, tím lepší výsledek. Kdyby byl návrh rovný cílovému rozložení, částice by měly stejnou váženou důležitost a převzorkování by nebylo třeba.

Návrhové rozložení typicky odpovídá odometrickému pohybovému modelu, který však není optimální a to zejména, pokud je senzor výrazně přesnější než odhad pozice. Využívá se také vyhlazování pravděpodobnostní funkce, což zabraňuje částicím v okolí významné oblasti, aby vážené důležitosti příliš poklesly. Následkem je ale zkreslení mapy. To se však dá vyřešit zahrnutím posledního pozorování do generování nových vzorků. Díky tomu se systém zaměřuje na vzorkování ve významné oblasti pravděpodobnosti pozorování.

Zlepšením návrhu se objevuje možnost získávat pro každou částici zvlášť její parametry Gaussovského návrhu a snižuje se také neurčitost výsledných hustot pravěpodobností. Porovnání pozorování má funkci maximalizace pravděpodobnosti pozorování, tvorby mapy a počáteční odhad pozice robota. Pokud je pravděpodobnostní funkce multimodální (rozložení pravděpodobnosti má více maxim, viz. obr. 3.2), například při uzavírání smyčky, porovnáním se vrací nejbližší lokální maximum pro každou částici. To může způsobit vynechání některých maxim v pavděpodobnostní funkci.

Převzorkování je velice důležitým aspektem určujícím výkon částicového filtru. Dochází k nahrazování vzorků s nízkou vahou. Je to nezbytný proces, neboť je potřeba konečného počtu částic pro aproximaci cílového rozložení. Může však odstranit dobré vzorky a ochudit tak částice, je proto důležité mít pro převzorkování

vhodné rozhodovací kritérium a provádět ho ve správný čas.

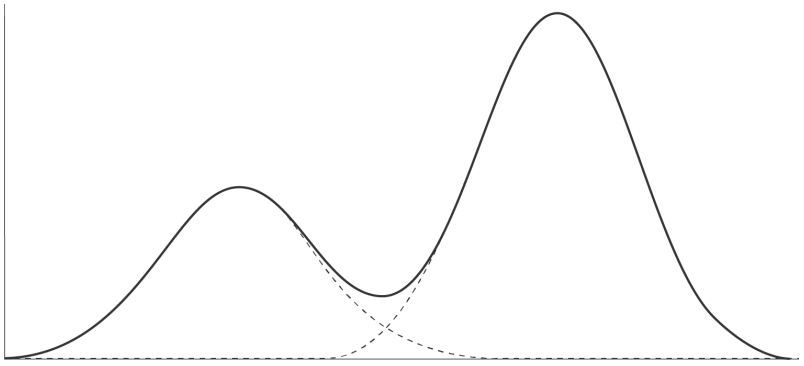
$$N_{eff} = \frac{1}{\Sigma_{i-1}^{N}(w^{(i)})^2}, \tag{3.1}$$

$w^{(i)}$ je zde normalizovaná váha částice i a N_{eff} jsou vzorky z cílového rozložení, které mají stejné váhové důležitosti. Pokud dochází ke zhoršení aproximace cílového rozložení, nastává větší odchylka vážených důležitostí.
Nastavení převzorkování dle [8] je následující:

$$N_{eff} < N/2, \tag{3.2}$$

kde N je počet částic. Tím dochází k výrazné redukci možnosti nahrazení dobrých částic a počtu převzorkování, které se vykonává pouze pokud je třeba.

Algoritmus probíhá následovně. Dojde k zisku odhadu pozice, který je re-



Obrázek 3.2: Příklad multimodálního (v tomto případě bimodální) rozložení pravděpodobnosti

prezentovaný danou částicí. Odhad je získán z předchozí pozice částice a odometrického měření od poslední aktualizace. Na základě mapy je provedeno porovnání pozorování z místa úvodního odhadu pozice, kdy se vyhledává pouze v okolí tohoto bodu. V případě selhání se pozice a váhy počítají dle pohybového modelu a následující dva kroky jsou přeskočeny. Prvním z nich je vybrání sady vzorků v okolí dané pozice, vypočítání průměrů a kovarianční matice návrhu pomocí bodového ohodnocení cílového rozložení v pozici vzorku. Druhým, potenciálně přeskočeným krokem, je zakreslení nové pozice částice z Gaussovské aproximace podle zlepšeného návrhového rozložení. Dále, a to již vždy, dojde k aktualizaci vážených důležitostí a, podle zakreslené pozice a pozorování, je aktualizována i mapa částice.

3.2 Hector SLAM

Hector SLAM je systém představený v roce 2011 v práci [9]. Jeho hlavním znakem je rychlost a nízké výpočetní nároky oproti ostatním dvěma metodám, které jsou v této práci rozebírány. Je vhodný pro implementaci v menších autonomních systémech, pro rychlý pohyb terénem a nehodí se pro uzavírání velkých smyček.

Vnitřní skladba systému je ze tří hlavních částí, a to 2D SLAM, běžící jako soft real time, 3D navigace, která je hard real time a inerciální jednotka (IMU). Hector SLAM je front-end SLAM, což znamená, že dochází k odhadu stavu robota v reálném čase a nedochází ke zpětné optimalizaci mapy.

Daný systém se musí převézt z 3DOF přidáním naklonění a rotace na 6DOF, kdy navigační filtr spojí měření z IMU a dalších senzorů. Tím dojde k získání 3D řešení. 2D SLAM poté poskytuje informaci o poloze v prostoru. Oba odhady se aktualizují nezávisle na sobě a jsou propojeny jen velmi málo.

Kvůli znatelnému posunu odhadů integrované pozice a rychlosti, který je způsoben šumem v senzorech, jsou do systému zahrnuty další informace. Jedná se například o porovnávání snímků, snímání magnetického pole, senzor barometrického tlaku a nebo měření rychlosti kol.

Pohyb robota je popsán:

$$\dot{\Omega} = E_{\Omega}.\omega \tag{3.3}$$

$$\dot{p} = v \tag{3.4}$$

$$\dot{v} = R_{\Omega}.a + g, \tag{3.5}$$

kde $\Omega = (\phi, \theta, \psi)^T$ je informace o otáčení, stoupání a natočení. Vektor $x = (\Omega^T p^T v^T)^T$ reprezentuje 3D stav, p, v jsou pozice a rychlost platformy v navigačním rámci. Vektor $u = (\omega^T a^T)^T$ je vstupní vektor pro inerciální měření, $\omega = (\omega_x, \omega_y, \omega_z)^T$ je úhlová rychlost a $a = (a_x, a_y, a_z)^T$ je zrychlení. R_{Ω} je pak matice směrových cosinů, E_{Ω} je mapování natočení těla na deriváty Eulerova úhlu a g je vektor gravitace.

3.2.1 2D SLAM

Jako reprezentace prostředí je zde vybrána metoda mřížkových map. Odhadovaná pozice robota slouží pro transformaci pozorování na lokální souřadnicový rámec. Odhadovanou orientací a přidruženými hodnotami je z pozorování vytvořeno mračno bodů, které je předzpracováno pro odstranění odlehlých bodů. Filtrace probíhá pouze na základě souřadnic koncového bodu, kdy jsou pro porovnávání pozorování použity pouze koncové body v rámci skenovací roviny.

Jak již bylo poznamenáno, je zde použita struktura mřížkových map, to vede k omezení přesnosti a neumožnění přímého výpočtu interpolovaných hodnot a derivátů. Pro oba odhady se tedy využívá interpolační schéma, díky kterému tato možnost je. Souřadnice, které se potřebují nahradit, se aproximují pomocí čtyř nejbližších celočíselných souřadnic.

Laserové skenery jsou velice přesná zařízení, jejich měření zatěžuje minimální

šum a snímky se dají vytvářet s vysokou frekvencí. Měření pomocí laseru je tedy mnohem přesnější než to odometrické, což je jeden z důvodu, proč je odometrie v tomto systému zcela vynechána. Při zarovnávání snímků s již známou mapou není potřeba hledat žádná spojení mezi koncovými body, ale dochází k porovnávání s předchozími skeny.

Hledání transformace pro nejlepší sladění skenu s mapou:

$$\xi^* = argmin_\xi \Sigma_{i=1}^n [1 - M(S_i(\xi))]^2, \tag{3.6}$$

$M(P_m)$ je zde hodnota obsazenosti a $S_i(\xi)$ jsou souřadnice koncového bodu $s_i = (s_{i,x}, s_{i,y})^T$, přičemž ξ jsou souřadnice robota.

Optimalizace chyby měření:

$$\Sigma_{i=1}^n [1 - M(S_i(\xi + \Delta\xi))]^2 \to 0, \tag{3.7}$$

kde $\Delta\xi$ je odhad ξ.

Při tvorbě mapy dochází často k nalezení pouze lokálního minima, reprezentací mapy pomocí více rozlišení se tak toto riziko znatelně redukuje. Utváří se mřížkové mapy, kdy každá další má poloviční rozlišení, než ta předchozí. Je tak uloženo více map, které se současně aktualizují podle odhadu aktuální pozice. Díky tomuto přístupu jsou mapy konzistentní a nepotřebují převzorkování. Zarovnávání pozorování probíhá pouze na mapě s nejvyšším rozlišením a tento odhad je pak použit pro ostatní mapy. Mapy s nízkým rozlišením jsou tedy v podstatě dostupné okamžitě, a lze je hned využít pro odhad trasy.

3.2.2 Odhad 3D stavu

Pro odhad úplného 6D stavu (3D stav + informace o naklonění z gyroskopů a informace z akcelerometrů) slouží navigační filtr běžící v reálném čase. Filtr je asynchronně aktualizován při příchodu odhadované pozice z porovnání pozorování nebo jiných informací ze senzorů. Filtr je implementován ve formě EKF, jedná se tedy o nelineární filtr a jako jeho známé vstupy se berou inerciální měření. Rychlost a pozice je získávána integrací zrychlení. Aby se zabránilo nezávislému růstu odhadu stavu při nepřítomnosti měření, dochází k aktualizaci pseudo-nulové rychlosti, která se spouští při dosažení odchylky určité prahové hodnoty a zajišťuje tak stabilitu systému.

3.3 Catographer

Systém Cartographer je vyvíjen společností Google a aktuální verze implementace byla publikována v roce 2012 [10]. Jedná se o implementaci Graph-Based SLAM. Je vhodný pro tvorbu rozsáhlých map a zisk optimalizovaných výsledků v reálném čase.

Cartographer vytváří v reálném čase 2D mřížkovou mapu. Submapy se vkládají na odhadované místo a daná submapa se porovnává vůči poslední zaznamenané. Dochází tedy k hromadění globální chyby z odhadu pozice. Systém neobsahuje žádný částicový filtr a to z důvodu snížení hardwarových nároků na běh algoritmu.

Submapa se po pořízení již dále nijak nepřepisuje a je zařazena mezi ostatní k porovnávání na uzavření smyčky. Pokud dojde k blízkému odhadu pozice a zároveň dostatečné shodě snímků, přidá se omezení uzavření smyčky do optimalizačního problému, čímž je odhad pozice. Odhad se aktualizuje po několika vteřinách a uzavření smyčky je tedy okamžitě viditelné.

Rozlišují se dva možné přístupy, lokální a globální, v obou případech se jedná o optimalizaci pozice, přičemž pozice se zkládá z hodnoty na ose x, z hodnoty na ose y a z natočení robota. V systému je také možné využít jednotku IMU, která slouží k odhadu směru gravitace, což je vhodné při pohybu na nerovné ploše.

3.3.1 Lokální 2D SLAM

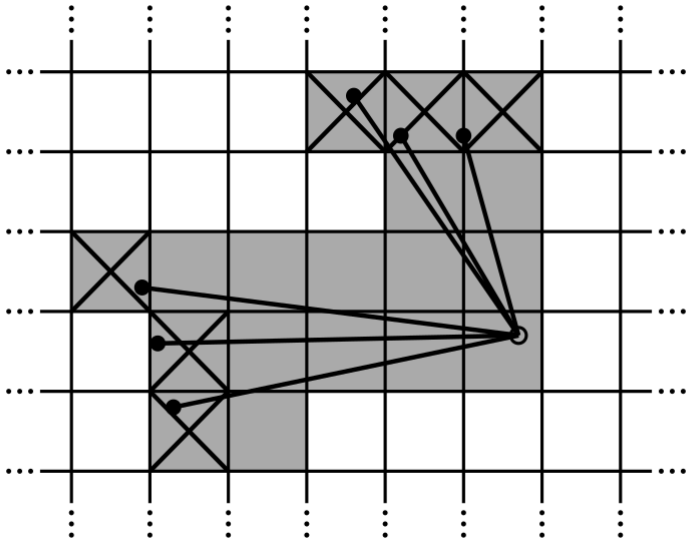
Skeny prostředí se iterativně zarovnávají se snímky již obsaženými v submapě. Submapa je tedy v postatě zachycení kousku světa, které je složené z pár skenů. Lokální chyba, vznikající při její tvorbě, je poté odstraněna v globálním přístupu. Pro každý bod mřížky je definován odpovídající pixel skládající se ze všech pixelů nejblíže danému bodu.

Při přidání skenu do pravděpodobnostní mřížky je počítána množina zasažených bodů mřížky a bodů minutých. Při zásahu se nejbližší bod mřížky vloží do množiny zásahů. Při nezaznamenání překážky jsou vloženy do množiny minutí všechny body odpovídající dráze laserového paprsku. Dosud nepozorované body mřížky mají přiřazenou pravděpodobnost minutí či zásahu, podle toho, jestli se v jedné z těchto množin vyskytují. Již pozorovaným bodům se pak aktualizují pravděpodobnosti minutí a zásahu.

Před vložením skenu do submapy se ještě využívá Ceres scan matching, který optimalizuje pozici skenu vůči submapě. Jedná se o maximalizaci pravděpodobnosti výskytu v dané oblasti.

$$argmin_{\xi} \Sigma_{k=1}^{K} (1 - M(T_{\xi} h_k)) \tag{3.8}$$

H je zde informace o bodech skenu, M je pravděpodobnostní mřížka, ξ je pozice snímání skenu a T_{ξ} je pozice skenu vůči submapě. Dochází k transormaci, kdy body skenu se transformují do submapy.



Obrázek 3.3: Vizualizace zasažení a minutí bodů v mřížce (z [10])

3.3.2 Globální 2D SLAM

Daný systém pracuje v oblasti submap s porovnáváním scan-to-scan, hromadí se v něm tedy lokální chyby. Pár snímků za sebou má však vůči sobě chybu minimální. Relativní pozice skenů se ukládají a v případě, že se submapa nezmění, všechny další páry ze skenů a submapy se předkládají k porovnávání pro uzavření smyčky. To vše běží na pozadí a pokud je nalezena shoda, dojde k uložení relativní pozice mezi optimalizační problémy.

Optimalizační problém je problém nelineárních nejmenších čtverců. Díky tomu se můžou jednoduše přidávat zbytky pro zohledňování dalších dat. Jednou za pár sekund je, pro optimalizaci pozice skenu vůči daným omezením, spuštěn Ceres scan matching. Omezeními jsou myšlena relativní pozice ξ_{ij} a kovarianční matice Σ_{ij}.

$$argmin_{\Xi^m,\Xi^s}\frac{1}{2}\Sigma_{ij}\rho(E^2(\xi_i^m,\xi_j^s,\Sigma_{ij},\xi_{ij})), \tag{3.9}$$

kde ρ je ztrátová funkce, například tedy Hubertova ztráta. Jedná se o snížení vlivu odlehlých hodnot, které přidávají nesprávná omezení do optimalizačního problému.

Při uzavírání smyčky se vyžívá také branch-and-bound scan matching, což se dá přeložit jako porovnávání větví a mezí. Zde se metoda zaměřuje na přesnou shodu pixelů. Podmnožiny možností jsou popsány jako uzly stromu, kdy kořenový uzel obsahuje všechna možné možnosti W. Potomci uzlu dohromady utváří stejný soubor možností, jako samotný rodičovský uzel. Listy jsou brány jako singlety, odpovídají jedinému proveditelnému řešení. Tento přístup dává stejné řešení jako ten předchozí, dokud je hodnota $score(c)$ vnitřních uzlů horní mezní skóre jeho prvků. Neexistuje tedy žádné řešení, než to doposud známé.

$$\xi^* = argmax_{(\xi \epsilon W)} \Sigma_{k=1}^{K} M_{nearest}(T_\xi h_k) \tag{3.10}$$

W je zde vyhledávací okno a $M_{nearest}$ je rozšíření pravděpodobnostní mřížky M na všechny R^2 zaokrouhlením argumentů do nejbližšího bodu mřížky. Rozšířená hodnota bodu mřížky ukazuje na odpovídající pixel.

Výběr uzlů probíhá prohledáváním do hloubky. Efektivnost algoritmu hodně závisí na podobě stromu, zda-li má pro výpočet dobrou horní mez a dobré aktuální řešení. Důležitým termínem je zde prahová hodnota skóre. Jedná se o hodnotu, pod níž není v zájmu jít a řešení je tedy nevyhovující. Díky tomu se nepřidávají špatné shody jako omezení pro uzavírání smyčky. Pro rychlost algoritmu je důležité rozhodování o průchodu stromem. Pro každého potomka je vypočítána horní hranice skóre a je vybrán ten nejslibnější, což je uzel s největším mezním počtem.

Každý z uzlů je popsán pomocí celočíselného pole:

$$c = (c_x, c_y, c_\Theta, c_h) \epsilon Z^4, \tag{3.11}$$

kde c_h je výška uzlu. Pokud $c_h = 0$, uzel je list.

Horní meze jsou pak počítány přes vnitřní uzly:

$$score(c) = \Sigma_{k=1}^{K} max_{(j \epsilon \bar{W})} M_{nearest}(T_{\xi_j} h_k). \tag{3.12}$$

Kapitola 4

Experimentální měření

Pro testování všech výše zmíněných systémů bylo využito frameworku *Robot Operating System* (ROS). V této části práce je tedy popsána jeho funkčnost a využití. Dále je zde porovnání implementací SLAM, jak na simulačních datech, tak i na datech reálných. Porovnání je provedeno výpočtem chyb v trajektoriích jízdy a v případě nekonvergence systému rozebráním situace, proč k danému problému došlo.

4.1 Robot Operating System

Robot operating system, zkráceně ROS, je operační systém určený pro ovládání robotů. Poskytuje vše očekávané od operačního systému, jako je například abstrakce hardwaru, nízkoúrovňové ovládání, komunikace mezi procesy a mnoho dalších funkcí. Obsahuje také nástroje a knihovny pro vytváření, získávání, psaní a spouštění kódu na více zařízeních. ROS prozatím běží pouze na platformách založených na Unixu. Software je testovaný především na systémech s Ubuntu nebo Mac OS X, přičemž s Windows není zatím kompatibilní.

Při běhu je komunikační infrastrukturou ROSu vytvářena síť volně propojených procesů, které mohou běžet na více zařízeních. Implementováno je zde několik možných komunikačních stylů. Přes služby je to synchronní komunikace RPC (*Remote Procedure Call* = vzdálené volání procedur). Je to jedna z nejstarších metod pro komunikaci programů na dálku. Obsahuje mechanismus umožňující volat z programu funkce ze vzdáleného počítače. Dalším obsaženým stylem komunikace, je asynchronní streamování dat, které běží přes topiky. Posledním je ukládání dat na *Parameter Server*, což je sdílený víceúrovňový slovník, který je přístupný prostřednictvím síťových rozhranní API. Procesy, označované jako uzly (*nodes*, nody), využívají tento server k ukládání a načítání dat za běhu.

Existuje celá řada softwarových platforem pro roboty a je těžké porovnávat, která z nich je ta nejlepší. ROS je open source systém, hlavní jeho snahou je znovuvyužitelnost napsaného kódu a multiplatformní provozuschopnost, což z něj činí nejrozšířenější systém pro roboty. Celý rámec je složený z distribuovaných procesů, nodů. Nody jsou individuáně spustitelné soubory, které se ze běhu volně propojují do peer-to-peer sítě, mohou být seskupeny do balíků a být tak jednoduše sdíleny.

Jednou z dalších snah systému ROS je co největší hubenost. Nedochází k za-

balení metody main() a tím může být kód využit i jinými frameworky. ROS je tak snadné integrovat s dalšími robotickými softwarovými frameworky, jako je například OpenRAVE nebo OROCOS. Dalším důležitým rysem je jazyková nezávislost, kdy je možné plně využívat jazyků Python, C++ a Lisp. Dále, zatím experimentálně, je možné používat určité knihovny z jazyků Java a Lua. Neméně důležitými vlatnostmi jsou také škálovatelnost, což je pro velké runtime systémy vlastnost velmi užitečná a snadné testování pomocí vestavěné jednotky *rostest*.

Ros se skládá ze tří úrovní pojmů. První úrovní je souborový systém, další je výpočetní graf a poslední je komunitní úroveň.

4.1.1 Souborový systém

Na úrovni souborového systému jsou v ROSu myšleny všechny prostředky, se kterými se pracuje na pevném disku. Jedná se o balíčky (*Pakages*), metabalíčky (*Metapackages*), *Package Manifests*, repozitáře, typy zpráv a typy služeb.

Balíčky jsou základní stavební jednotkou softwaru ROS. Jedná se o nejnižší strukturu umožňující tvorbu programu. Balíčky mohou obsahovat *ROS runtime* procesy, dále knihovny závislé na ROSu, datové sady, konfigurační soubory a mnoho dalšího, co lze užitečně zahrnout do jednoho souboru.

Metabalíčky se již nachází výše, je to specializovaný balíček, který slouží k reprezentaci skupiny souvisejících balíčků. Nejčastější jejich využití je držení zpětné kompatibility pro zásobníky (*Stacks*), které prošly konverzí přes *rosbuild*. Zásobníky jsou též kolekce balíčků, které seskupují určité funkce, například *navigation stack*, jenž na základě informací ze senzorů dává příkazy pro další navigaci robota v prostředí. *Package Manifest*, package.xml, poskytuje metadata o daném balíčku. Jsou v něm obsažené informace jako název, verze, popis, licenční informace, závislosti a další.

Repozitář je opět souhrn balíčků (může však obsahovat pouze jeden balíček), které sdílejí společný systém kontroly stejné verze (*Version Controlling System*, VCS). Balíčky tak mohou být společně uvolněny užitím nástoje *catkin*. Repozitáře často mapují převedené zásobníky přes *rosbuild*.

Typy zpráv a typy služeb pak již definují typy, které se smějí v ROS využívat. Pro typy zpráv je cesta k souboru *my_package/msg/MyMessageType.msg* a obsahuje datové struktury pro posílané zprávy. Definice typů služeb se nachází na adrese *my_package/srv/MyServiceType.srv* a definuje datové struktury požadavků a odpovědí.

4.1.2 Výpočetní graf

Výpočetní graf se skládá z jednotlivých procesů zpracovávajících společně sdílená data a dohromady vytvářejících peer-to-peer síť. Základními pojmy pro výpočetní graf jsou nody (*nodes*), *Master*, *Parameter Server*, zprávy (*messages*), topiky (*topics*), služby a bagy. Všechny pojmy jsou implementovány v repozitáři pro ko-

munikaci *ros_comm*.

Nody, neboli uzly, jsou procesy obstarávající výpočty. Řídicí systémy pro robota obsahují často mnoho uzlů. Pro ROS je typická snaha o co největší modularitu, každý uzel má tak jednu svoji úlohu, například zpracování laserového senzoru. Uzly jsou nejčastěji psány pomocí knihoven *roscpp* nebo *rospy*, kde písmena za *ros* znamenají, zda-li se jedná o implementaci jazyka C++ nebo Python.

Master zde zajišťuje registraci jmen a vyhledávání ve výpočetním grafu. Uzly s *Masterem* komunikují a oznamují mu své registrační údaje, bez něj by se tedy uzly navzájem nemohly najít a vyměňovat si zprávy. Při změně registračních údajů provede *Master* zpětné volání uzlů, tím je zajištěno dynamické vytváření spojení mezi uzly. Blízce související termín je *Parameter Server*, který je součástí *Mastera* a umožňuje ukládání dat do centrálního umístění.

Zprávy jsou informace posílané mezi uzly. Jsou tvořeny datovými typy jako je integer, float, boolean a tak dále a nebo ve formě pole obsahujících tyto primitivní typy.

Topik je název sloužící k identifikaci obsahu, kterým je určitá zpráva. Pokud uzel vyžaduje určitý typ dat, začne odebírat data z příslušného topiku. Pokud chce uzel data do topiku odesílat, publikuje do něj danou zprávu. Do jednoho topiku může zapisovat i číst z něj zároveň více uzlů a jeden uzel může odebírat a publikovat do více topiků. Uzly, ať publikující, či odebírající, nejsou si vědomy ostatních účastníků u daného topiku.

Komunikační model typu publikující/odběratel je vhodný pro posílání zpráv. Pro distribuované systémy, které využívají interakce typu žádost/odpověď, je to však nevyhovující. Tato interakce se tedy provádí pomocí služeb, které jsou definovány dvojicí struktur. Jedna pro žádost a druhá pro odpověď. Například uzel nabízí určitou službu pod jménem a klient této služby využívá zasláním požadavku, kdy poté čeká na odpověď.

Posledním pojmem jsou zde bagy, v originálu *Bags*, které slouží k uložení a přehrávání dat z topiků. Jedná se o důležitý mechanismus skladování dat, který je nezbytný pro vývoj a testování algoritmů.

Pro ROS jsou velmi důležitá pojmenování. Jména jsou primárním prostředkem pro vybudování velkého a složitého systému. Každy uzel, topik, služba nebo parametr má svůj název. Všechny klientské knihovny podporují přemapování názvu z příkazového řádku, běžící programy tak mohou být za běhu překonfigurovány, aby fungovaly i odlišné topologii výpočetního grafu.

4.2 Zisk dat k porovnání

Data potřebná pro porovnávání se dají získat prostřednictvím internetu. Existují veřejně přístupné nahrané datasety z MIT, Deutsche Museum a řady dalších míst. Pro vyzkoušení systémů na velkých datech je tedy daná možnost také využita. Pro porovnání implementací jsou ale převážně použity datasety získané jak ze simulace v ROS, tak i reálná data naměřená robotem *Dagu Wild Thumper* $6WD$. Získání datasetu je pro porovnání implementací zásadním faktorem. Systémy SLAM mohou samozřejmě jednotlivě běžet na aktuálně simulovaných datech, pro správné porovnání jsou však potřebná data totožná.

Datasety jsou v ROS ukládány ve formátu *bag*, kde se jsou nahrány vybrané topiky. Pro každou porovnávanou implementaci byly vytvořeny *launch* soubory, které při zavolání spouštějí uvedené uzly se specifikovaným nastavením. Soubory *launch* byly pro všechny systémy nastaveny tak, aby potřebovaly informaci o měření z LIDARu, odometrickou informaci a transformaci přepočítávající pozici robota. Tyto tři typy informací stačí při spuštění dané metody k vytvoření mapy a všeho k tomu přidruženého. Pro porovnání trajektorií je potřeba uložit ještě informaci o přesné pozici robota, neboli *ground truth*. V nahraných souborech *bag* se tedy nachází pouze tyto zmíněné čtyři informace. Výjimkou jsou data získaná robotem *Wild Thumper*. *Ground truth* by zde měla vycházet z odometrie kol, tudíž o informaci méně. Odometrie z kol robota je však natolik nepřesná, že se pro účely porovnávání nedá využít. Tato nepřesnost samozřejmě zhoršuje provedení metod, které odometrii využívají (Cartographer, gMapping), podrobnější popis dané problematiky se nachází u vyhodnocení daného typu naměřených dat.

4.2.1 Simulace v ROS

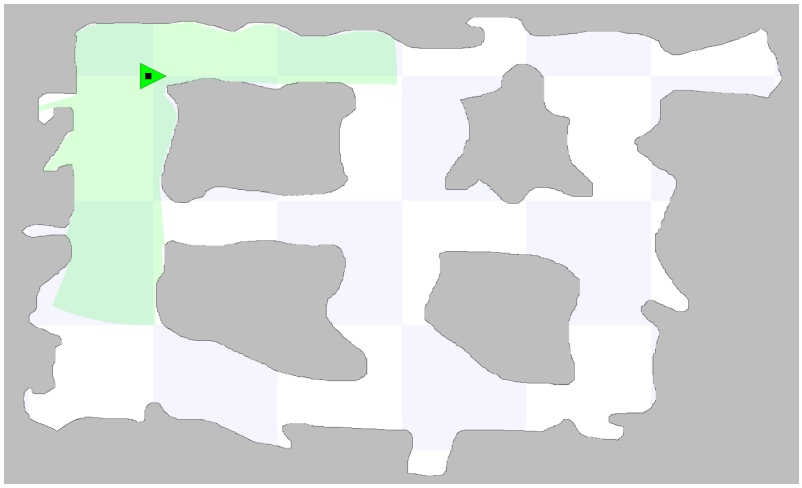
Pro zisk dat pomocí simulace jsou v systému ROS implementovány dva hlavní simulátory. Prvním z nich je *Gazebo*. Jedná se o nástroj určený pro simulaci ve 3D, stále prochází vývojem a je nejpoužívanějším simulátorem v ROS. Pro dané účely má však zbytečně veliké množství nastavení. Byl zvolen tedy nástroj *Stage*, který je starší, má méně funkcí, má ale také menší výkonostní nároky a pro simulaci robota s 2D LIDARem je naprosto vyhovující.

Simulační prostředí je rozloženo do tří souborů. Dva s příponou *inc*, kdy první obsahuje informace pro načtení mapy. Jsou zde obsaženy informace o velikosti, frekvenci obnovování, výšce překážek v ose z a různá boolean nastavení. V druhém *inc* souboru se pak nachází informace o simulačním robotovi. Byla zvolena trojúhleníková platforma, na které se nachází malý čtvercový blok simulující LIDAR. Je zde obsaženo i jeho nastavení, jako je počet paprsků, úhel rozsahu a velikost dosahu. Posledním konfiguračním souborem je soubor *wolrd*, ve kterém se nachází nastavení velikosti okna a načtení obou souborů *inc*. Při načítání souboru s nastevením mapy je zde definován obrázek obsahující mapu a s načítáním souboru o robotovi se definuje jeho pozice v prostoru. Výsledný vzhled prostředí je zobrazen na obrázku 4.1.

Pro nahrání dat poté stačí příkazem

```
rosrun stage_ros stageros file.world
```

spustit simulaci a pomocí některého z navigačních balíčků s robotem v mapě jezdit. Pro navigaci byla zvolena knihovna *teleop_twist_keyboard*, která umožňuje navigaci robota pomocí klávesnice.



Obrázek 4.1: Prostředí Stage

4.2.2 Nahrání reálných dat

K nahrání reálného datasetu byl k dispozici robot *Wild Thumper* $6WD$. Jedná se o robota s pohonem šesti kol. Hliníkové tělo má v sobě s rozestupem jednoho centimetru otvory, díky kterým se na něj dají jednoduše přidělat další součástky.

Pro řízení kol je využito Arduino, pro zpracování dat a komunikaci je zde výkonný malý počítač od společnosti NVIDIA, jehož označení je Jetson TX2 a velikost 50 x 87 milimetrů. Výpočetní výkon zajišťuje osmijádrový procesor s architekturou jádra ARMv8. Velikost operační paměti má 8 GB a úložiště 32 GB. Pro komunikaci je zde rozhraní Wi-Fi 802.11ac a Bluetooth 4.1. Jako měřící senzor je zde využit LIDAR $Hokuyo\ URG-04LX-UG01$ s detekovatelným rozsahem od dvou centimetrů do 5,6 metru. Daný sezor má rozsah snímání 240^o s úhlovým rozlišením $0{,}36^o$. Obnovovací frekvence je 100 milisekund a uvedená přesnost je 30 milimetrů.

Pro pohyb robota v prostoru byla využita možnost připojení bezdrátového ovladače *Xbox One Wireless Controller*. Přesto, že robot díky integrovanému počítači od NVIDIA má velmi vysoký výkon, tyto prostředky byly nevyužity a data byla při běhu programu posílána rovnou do počítače, kde se následně ukládala do souboru *bag*. Spojení bylo zprostředkováno Wi-Fi routerem *Ubiquiti AirCube AC*, který musí být připojen ethernetovým kabelem k danému počítači. Tento druh propojení má své jisté výhody i nevýhody. Výhodou je bezproblémový přenos, který při připojení robota rovnou na univerzitní Wi-Fi nebyl zdaleka pravidlem. Nevýhodou

tohoto řešení je dosah, který je limitován, stejně tak jako je tomu běžně u vysílačů Wi-Fi signálu. Pro dané účely to však nakonec nebyl problém a vhodným umístěním routeru byla možnost naměřit data na dostatečném prostoru.

Obrázek 4.2: Dagu Wild Thumper 6WD s připevněnými senzory

4.2.3 Uložení trajektorie

Po nahrání datasetu a spuštěním všech metod SLAM na těchto datech, je získána množina výsledků, kterou je možno porovnat. Je více možností jak implementace porovnávat, kromě trajektorie je možné srovnání map. Pro možnost s mapou, byla by třeba tvorba výsledných map ve stejném měřítku a zisk přesné výměry prostředí, ve kterém byla data nahrána. Zvláště informace o přesné podobě prostředí by byla ve velkém množství porovnávání značný problém. Pro tyto účely se tak volí možnost s trajektoriemi, kdy zisk přesné trajektorie pohybu robota, *ground truth*, je výrazně jednodušší.

Tvorba trajektorie je u každého systému zcela rozdílná. Nejjednodušší extrakce trajektorie je v případě Hector SLAM. Tato metoda sama publikuje topik s názvem *slam_out_pose*, kde je zaznamenávána aktuální pozice. Tím, že Hector SLAM neumí přepisovat předchozí výsledky různými optimalizacemi, či uzavíráním smyčky a informace pouze přidává, může se *slam_out_pose* při běhu implementace celou dobu poslouchat a data z topiku ukládat. Poslouchání topiku je prováděno příkazem `rostopic echo /topic`, což začne vypisovat data do terminálu. Přidáním za příkaz `> file.txt` se data uloží do daného soboru. Výsledný příkaz pro uložení trajektorie pomocí Hector SLAM tedy vypadá následovně:

```
rostopic echo /slam_out_pose > trajectory_hector.txt
```

U metody gMapping je zisk trajektorie složitější. Tato implementace nikam trajektorii sama nezaznamenává. Jednou z možností, jak ji tedy získat, je využít uzel obsažený v systému Hector SLAM. Přidáním uzlu s názvem *hector_trajectory_server* do spouštěného souboru *launch* je tak získána možnost extrahovat trajektorii. Neboť je gMapping systém obsahující optimalizaci cesty a mapy a umí uzavírání smyčky, je třeba trajektorii uložit až po doběhnutí dat obsažených v souboru *bag*. Takto se zavolání topiku provádí příkazem `rosservice call /topic`, uložení do souboru se provede stejně jako v případě s Hector SLAM. Příkaz pro uložení trajektorie má podobu:

```
rosservice call /trajectory > trajectory_gmapping
```

Nejsložitější postup získávání trajektorie je v případě systému Cartographer. Dříve se dala tato informace získat pomocí dvou příkazů. Aktualizací Cartographeru před dvěma lety však zavedl Google pro ukládání dat nový formát *pbstream*, pro práci s nimž není příliš dobrá veřeně dostupná dokumentace. Cartographer při běhu publikuje uzel s názvem */trajectory_node_list*, který nese v daném časovém okamžiku informaci o všech bodech trajektorie. Při dokončení výpočtů ze spuštěného *bag* souboru je tedy k uložení dostupná finální trajektorie. Součástí této informace jsou však pouze souřadnice v prostoru a chybí tak informace o natočení, či časovém momentu, kdy robot v daném místě byl. Pro jednodušší porovnání implementací by to stačilo, ale pro správné porovnávání dle [11] by to bylo nedostatečné. Nakonec pro předevení souboru *pbstream* do souboru *bag* posloužil kód, který přidal do své implementace Cartographeru jeden výkumník Googlu [12]. V běžné implementaci Cartographeru se tento kód nevyskytuje. Využitím tohoto kódu již byla možnost trajektorii získat a to následujícím postupem. Nejprve je třeba nechat proběhnout všechny výpočty z nahraného souboru *bag*, pro lepší orientaci si bude označen jako *measured.bag*. Po doběhnutí výpočtů z *measured.bag* je potřeba ukončit výpočty Cartographeru zavoláním

```
rosservice call /finish_trajectory 0,
```

kdy 0 označuje číslo trajektorie. Tímto příkazem se zahrnou do optimalizace trasy a mapy i částice, na které při doběhnutí dat z *measured.bag* ještě nedošlo. Následujícím příkazem

```
rosservice call /write_state state.pbstream,
```

se uloží aktuální stav Cartographeru do souboru *pbstream*. Poté je třeba se dvěma parametry zavolat přidaný uzel s názvem *pbstream_trajectories_to_bag*, který má dva vstupní parametry. První parametr je název souboru *pbstream* a parametr druhý je název souboru *bag*, do kterého se data uloží. Tento nový *bag* soubor bude pro orientaci nazýván jako *processed.bag*. Finálním krokem je z *processed.bag* získání trajektorie, která je zde uložena v topiku */tf*. To se provede již použitým příkazem `rostopic echo /topic`, za který se přidá argument `-b "bag_file"`, čímž se řekne, že se nechce poslouchat aktuálně publikovaný topik, ale topik z určeného souboru *bag*. Za to se pro uložení do vybraného souboru ještě přidá `> file.txt`. Výsledný příkaz je tedy v podobě:

```
rostopic echo /tf -b processed.bag > trajectory_cartographer.txt
```

4.2.4 Příprava dat

Data ze všech systémů jsou v stejném tvaru, počátek mají v bodě [0,0,0], informace o natočení má shodný počátek a i rozsah hodnot, který se pohybuje na intervalu $(0, 1\rangle$. Stejně je tomu tak i v případě *ground truth*, s jediným rozdílem, že počátek trajektorie není v bodě [0,0,0]. Úprava tak spočívala v posunutí všech bodů tak, aby byl počátek v tomto bodě.
Druhou a poslední úpravou, byla změna intervalu popisující natočení robota. Nově zvolený rozsah je $(-\pi, \pi\rangle$. Důvodem je problém se změnou natočení v okolí hraniční hodnoty. Při hodnotě natočení blízké jedné a otočením se na hodnotu blízké nule, vzniká velký rozdíl hodnot, který nepopisuje vzniklou situaci. S novým intervalem se může přeskočit od hodnoty blízké π do hodnoty blízké $-\pi$, což při práci s absolutními hodnotami dává přesnou informaci o změně natočení.

4.3 Vyhodnocení výsledků

V této kapitole se nachází popis kritéria, dle kterého jsou porovnávány systémy SLAM. Následně jsou zde uvedeny výsledky z jednotlivých měření a jejich rozbor.

4.3.1 Srovnávací kritérium

Pro srovnání algoritmů SLAM je dle [11] zvoleno porovnávací kritérium ve tvaru

$$\varepsilon(\delta) = \varepsilon_{trans}(\delta) + \varepsilon_{rot}(\delta), \tag{4.1}$$

kde $\varepsilon_{trans}(\delta)$ je chyba určení pozice robota a $\varepsilon_{rot}(\delta)$ je chyba natočení robota.

Rovnice pro výpočet chyby pozice se dá dále rozepsat jako:

$$\varepsilon_{trans}(\delta) = \frac{1}{N}\Sigma_{i,j} f_{trans}(\delta_{i,j} \ominus \delta^*_{i,j})^2, \tag{4.2}$$

kde N je celkový počet bodů trajektorie, $\delta_{i,j}$ je relativní pozice robota dle daného algoritmu SLAM v čase i a j, přičemž i a j jsou bezprostředně se následující časové záznamy. Pozice v daných časových okamžicích z ground truth je označena jako $\delta^*_{i,j}$. Funkce $f_{trans}()$ je určena ve formě Euklidovské normy:

$$f_{trans}(\delta_{i,j} \ominus \delta^*_{i,j}) = ||\delta_{i,j} \ominus \delta^*_{i,j}||. \tag{4.3}$$

Rovnice pro určení chyby natočení robota je ve tvaru:

$$\varepsilon_{rot}(\delta) = \frac{1}{N}\Sigma_{i,j} f_{rot}(\delta_{i,j} \ominus \delta^*_{i,j})^2, \tag{4.4}$$

kde všechny proměnné mají stejný význam jako u chyby určení pozice a funkce $f_{rot}()$ má předpis:

$$f_{rot}(\delta_{i,j} \ominus \delta^*_{i,j}) = |min(2\pi - |\delta_{i,j} \ominus \delta^*_{i,j}|, |\delta_{i,j} \ominus \delta^*_{i,j}|)|. \tag{4.5}$$

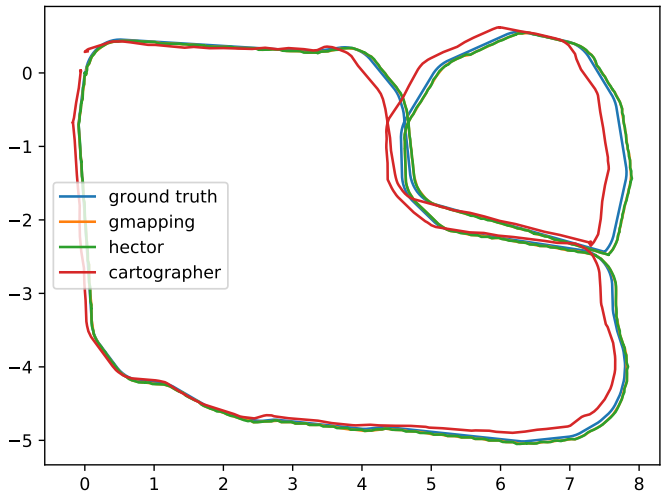
Tento přístup, narozdíl od běžného porovnávání vzdálenosti bodů v prostoru, nehledí na přímý rozdíl pozice bodů v daném čase, ale na změnu relativních pozic v časovém úseku.

4.3.2 Porovnání výsledků ze simulátoru Stage

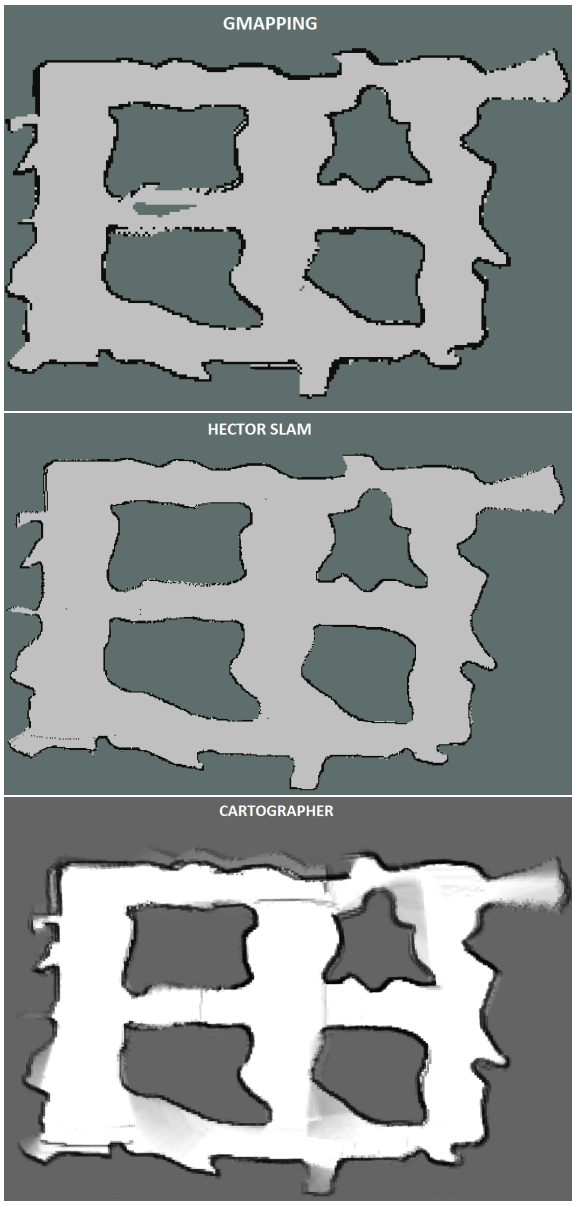
Pro porovnání výsledků, nahraných pomocí rozhraní Stage bylo vytvořeno několik obrázkových map. Přesnost algoritmů byla u všech map přibližně stejná. Názorné porovnání je tedy provedeno pouze na jedné, kde při průjezdu vznikly dvě smyčky. K porovnání jsou využity rovnice z předchozí sekce.

	$\varepsilon_{trans}(\delta)\ [m]$	$\varepsilon_{rot}(\delta)\ [rad]$	$\varepsilon(\delta)$
GMapping	0.00006	0.00004	0.00010
Hector	0.00004	0.00023	0.00027
Cartographer	0.00086	0.00062	0.00149

Z výsledků je zřejmé, že všechny algoritmy proběhly v pořádku a s vysokou přesností. Nejlepších výsledků v tomto případě dosáhl systém gMapping. Byla zde i prokázána důležitost zvoleného kritéria porovnávání. Například při srovnání trajektorií pomocí metody *Root Mean Square Error* (RMSE) dosáhl nejlepších výsledků algoritmus Hector SLAM. RMSE je porovnání trajektorií, kde se počítá pouze s rozdílem bodů v daném časovém okamžiku.



Obrázek 4.3: Grafické srovnání vytvořených trajektorií



Obrázek 4.4: Vykreslené mapy jednotlivými systémy

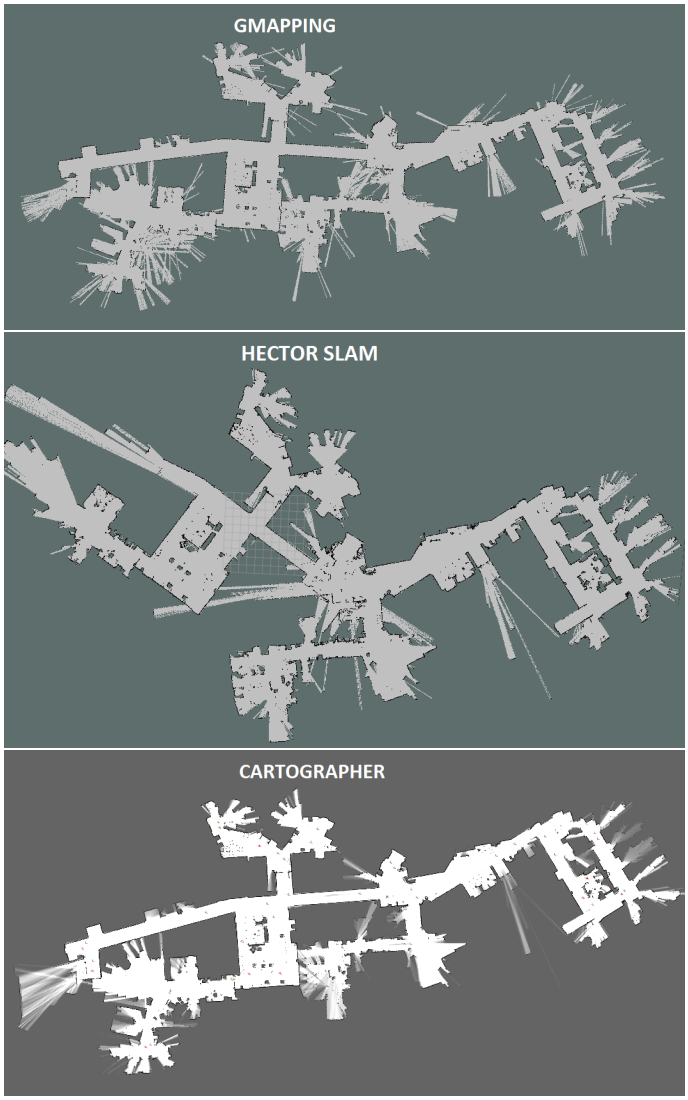
4.3.3 Porovnání výsledků ze staženého datasetu

Pro porovnání systémů byla potřeba najít dataset, který v sobě obsahuje všechny potřebné informace pro jejich vykonání. Další podmínkou, která musela být splněna, aby testování mělo srovnatelné výsledky, je nutnost znalosti *ground truth* pro daná data. Potřebné informace obsahují datasety nahrané v MIT Stata Center, kde k mnohým z nich je možno ještě stáhnout zmíněné *ground truth*. Pro porovnání tak byl zvolen jeden z těchto datasetů.

Ground truth má rovnou informaci o natočení v rozsahu $\langle -\pi, \pi \rangle$, nebyla v tomto ohledu tedy potřeba žádná změna. Data však byla natočená přibližně o 30^o

a posunuta mimo počátek [0,0,0]. Stažená data byla tedy přepočítána tak, aby bylo natočení a počátek stejný. Porovnávání pak bylo provedeno dle rovnic v kapitole 4.3.1.

	$\varepsilon_{trans}(\delta)$ $[m]$	$\varepsilon_{rot}(\delta)$ $[rad]$	$\varepsilon(\delta)$
GMapping	0.02002	0.00698	0.02701
Hector	0.00860	0.00144	0.01005
Cartographer	0.00049	0.00148	0.00197

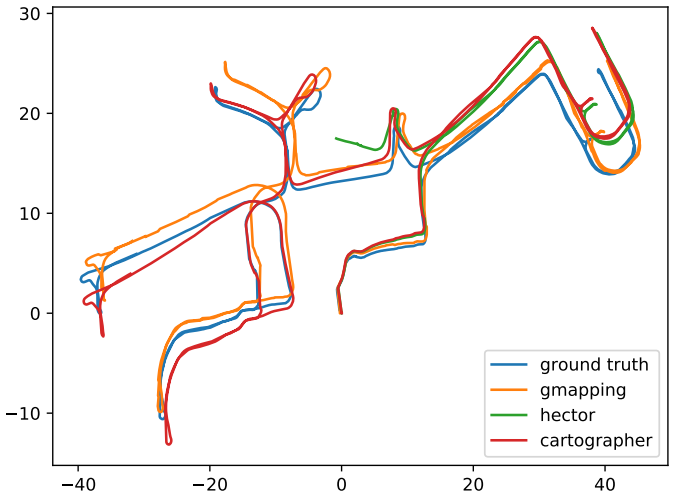


Obrázek 4.5: Vykreslené mapy jednotlivými systémy pro zkrácený dataset

Z výsledků je patrné, že v tomto případě si nejlépe vedl Cartographer. Za ním je dle tabulky Hector SLAM a poslední je gMapping. Při pohledu na mapu je však vidět, že trajektorie Hector SLAMu je neúplná a na konci silně vychýlená. Je to z

důvodu, že při průjezdu předcházejícím výběžkem, došlo vždy k dislokaci robota, následně i v dalších částech měl problémy v orientaci i přes změny v řadě nastavení. Trajektorie Hector SLAMu by tak byla pro porovnání naprosto nevyhovující a je proto ukončena v bodě, do kterého se systém jevil správně. Grafické znázornění na obrázcích 4.6, 4.5

Výše zmíněné výsledky a mapy jsou však získány z datasetu, ve kterém byla použita pouze část trajektorie obsažená ve staženém datasetu. Důvod byl, že i přes nastavení Cartographeru určené pro tyto data, nedokázal si tento systém částečně poradit s jednou chodbou a došlo tak k posunu velké části trajektorie a mapy. Dané místo je znázorněno na mapách náležících obrázku 4.7, výsledné trajektorie jsou na obrázku 4.8.

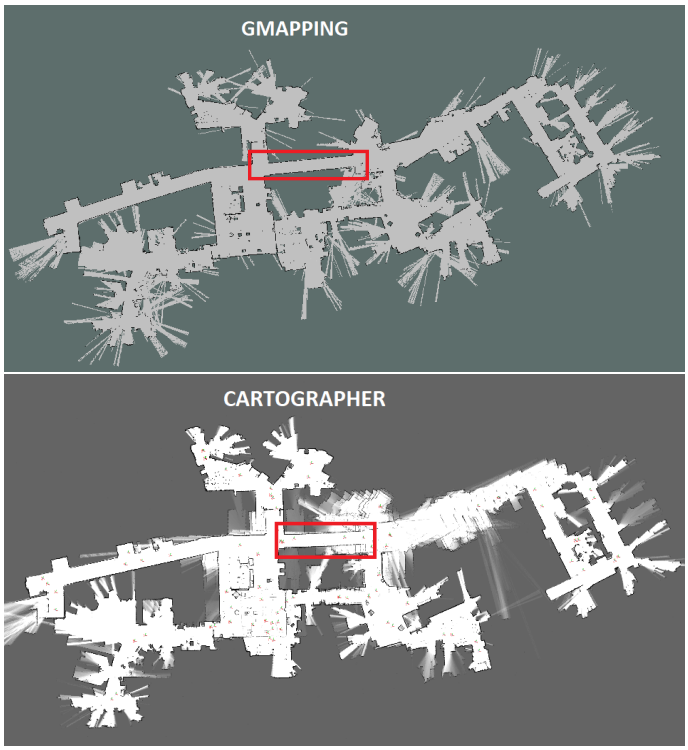


Obrázek 4.6: Grafické srovnání vytvořených trajektorií pro zkrácený dataset

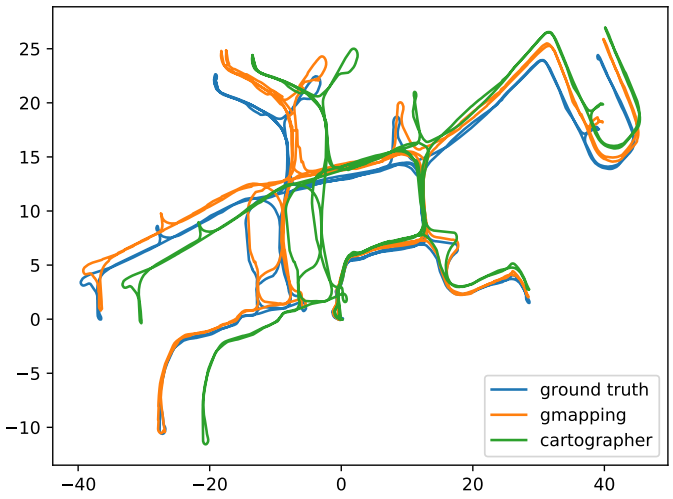
Výsledky z nezkráceného datasetu jsou následující.

	$\varepsilon_{trans}(\delta)\ [m]$	$\varepsilon_{rot}(\delta)\ [rad]$	$\varepsilon(\delta)$
GMapping	0.02601	0.00831	0.03432
Cartographer	0.00292	0.00158	0.00451

Je vidět, že výsledky se u obou porovnávaných systémů mírně zhoršily, chyba v pozici u Cartographeru poté výrazněji, což je zaviněno zmíněným posunem. I tak má však Cartographer opět lepší výsledek, neboť srovnávací kritérium nepočítá rozdíl jednotlivých bodů od sebe, ale poměřuje přírůstek mezi dvěma časovými okamžiky. Přechody v trajektorii jsou u Cartographeru tedy přesnější, než ty u gMappingu.



Obrázek 4.7: Vykreslené mapy jednotlivými systémy pro celý dataset



Obrázek 4.8: Grafické srovnání vytvořených trajektorií pro celý dataset

4.3.4 Porovnání dat nahraných robotem

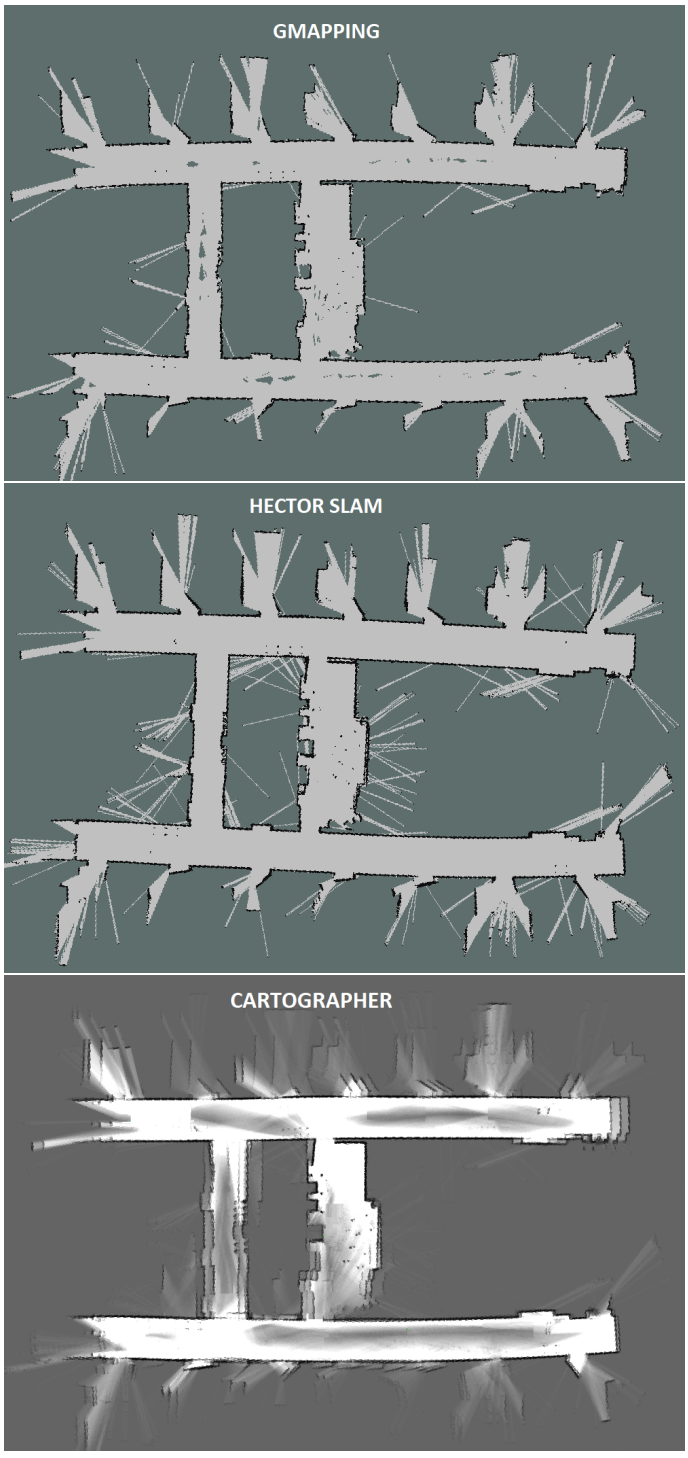
Data pro toto porovnání jsou nahrána na Fakultě aplikovaných věd ZČU v budově NTIS. Jak je již výše zmíněno, odometrie zaznamenána robotem je v tomto případě nepoužitelná. Přesto, že vše v něm nahrané je nastaveno dle specifikace výrobce, informace je velmi nepřesná. Robot sám při jízdě dopředu lehce zatáčí doprava. Informace o dopředném pohybu je ale jinak ukládána správně, při zatáčení však robot zaznamenává informaci o mnohem větším zatočení, než k jakému došlo. Ztráta odometrie má na každý ze systémů rozídlný dopad.

Hector SLAM, který odometrickou informaci vůbec nepoužívá, není tedy tímto omezením nijak ovlivněn a jeho průběh je standardní. Cartographer již odometrii využívá, lze ale vypnout sledování odometrie a používat odometrii systémem dopočítávanou. Použitím tohoto nastavení se Cartographer stává náchylný na podlouhlé prostory bez větších rozdílů, typickým příkladem jsou chodby. GMapping bezpodmíněčně odometrickou informaci vyžaduje. Informace silně zkreslená tak průběh programu velmi komplikuje.

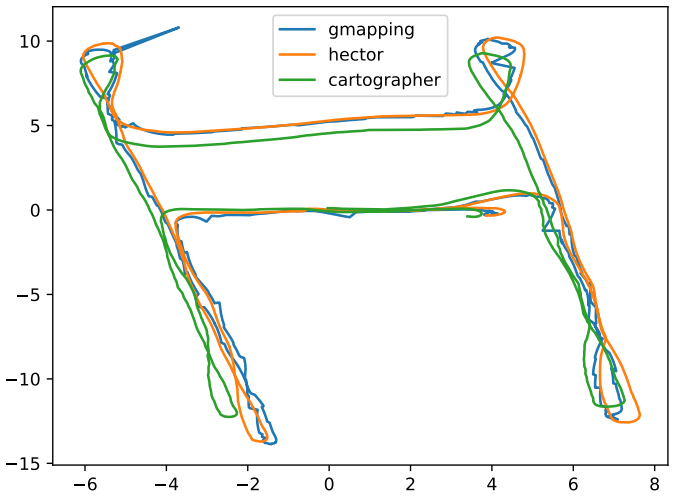
Při pohledu na vytvořené mapy (obrázek 4.9) je znát nedostatek s odometrií pouze u Cartographeru, kdy jedno projetí celého prostoru bylo nedostatečné a mapa není na pohled příliš kvalitní. Pro srovnání tak byl nahrán ještě nový dataset, kdy bylo daným prostorem projeto dvakrát a výsledek je znatelně lepší (obrázek 4.11).

Pohled na trajektorie je však již více vypovídající o nedostatcích spojených s nekvalitní odometrickou informací. Pro Hector SLAM se opět nic neměnilo a trajektorie je v pořádku. U Cartographeru je vidět, že kvůli občasným problémům s rozeznáním posunu po chodbách, je trajektorie značně kratší. Největší problém v oblasti trajektorie je znatelný u systému gMapping, kdy informace o přetáčení vytváří na trajektorii po celou dobu velké výchylky a zuby. Algoritmus si s tím však dokáže i tak poradit a výsledná mapa tím ovlivněna není.

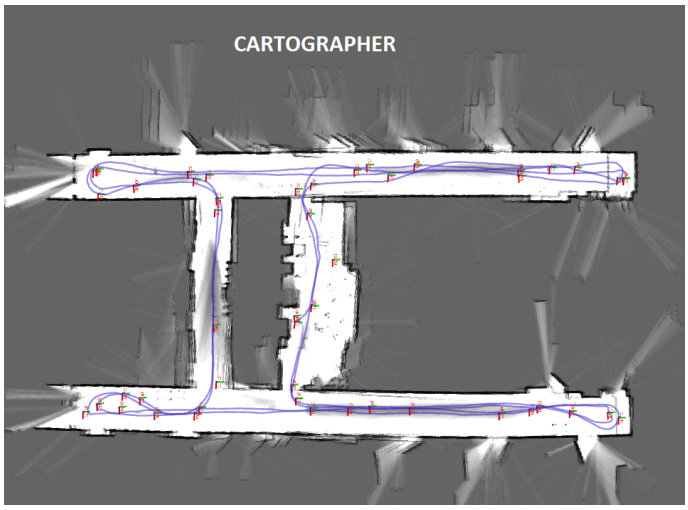
Bez znalosti *ground truth* je třeba porovnat algoritmy na základě získaných map. Měřením poměru vzdáleností mezi jednotivými body v prostoru a porovnáním s těmito body ve vytvořených mapách, vychází pro tento případ jako nejlepší algoritmus Hector SLAM, v těsném závěsu poté gMapping. Pro srovnání je přiložen i graf s trajektoriemi 4.10.



Obrázek 4.9: Vykreslené mapy jednotlivými systémy pro nahraná data robotem



Obrázek 4.10: Trajektorie jednotlivých systémů vytvořené na datech nahraných robotem



Obrázek 4.11: Mapa vytvořená systémem Cartographer při dvojném projetí protoru

Kapitola 5

Průzkum vybraného systému

Pro důkladnější prozkoumání vybraného systému byl zvolen gMapping, neboť je nejstarší, má aktivní komunitu a stále se využívá. Tím je zajištěno rychlé ustabilizování systému po případných aktualizacích. Pro bližší seznámení a případné návrhy změn vychází tedy nejlépe.

5.1 Změny v konfiguraci systému

V předchozích měřeních bylo vždy použito defaultní nastavení systému. Toto nastavení nebylo třeba měnit, neboť gMapping vždy provedl výpočty dle očekávání a nikdy se nestalo, že by jako jediný selhal. Pro zkoumání vlivu nastavení na systém, byly postupně měněny různé parametry. Jako testovací data byl zvolen zkrácený dataset z MIT Stata Center.

Kritériem pro testování vlivu nastavení na algoritmus byla pro tento případ zvolena odlišná rovnice, která porovnává přímo polohu jednotlivých bodů trajektorie s *ground truth* (*Root Mean Square Error*). Důvodem je absence extrahovatelné trajektorie přímo ze systému gMapping. Při vývoji tak nebyl kladen důraz na srovnání dle [11], ale hlavně na to, aby částice tvořící mapu měli správné umístění. Rovnice použitá pro srovnání je ve tvaru:

$$e(t, g) = \sqrt{\frac{1}{N}\Sigma_i(||t_i - g_i||)^2}, \tag{5.1}$$

kde t je vypočtená trajektorie, g je *ground truth*, N je počet vzorků trajektorie a i udává časový okamžik.

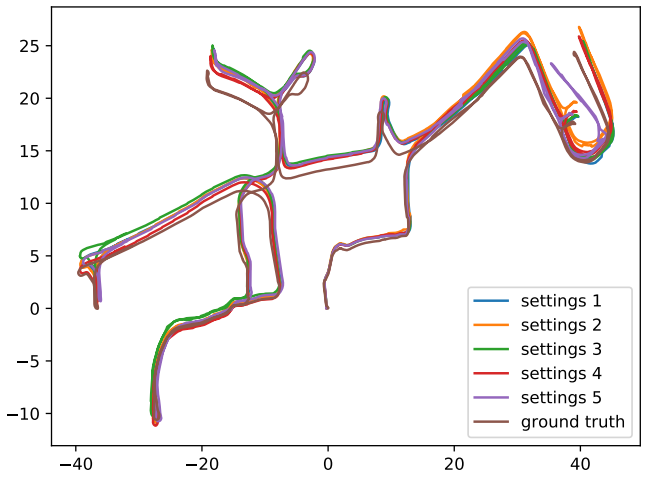
V násedující tabulce jsou defaultní hodnoty parametrů, které byly upraveny pro účely porovnání. Ostatní parametry byly ponechány na defaultních hodnotách.

Parametr	**Hodnota**	**Význam**
linearUpdate	1.0	ujetá vzdálenost pro nové měření [m] (1)
angularUpdate	0.5	zaznamenané otočení pro nové měření [rad]
particles	30	počet částic tvořený v každém kroku (2)
llsamplerange	0.01	rozsah translačního vzorkování pravděpodobnosti (3)
llsamplestep	0.01	krok v translačním vzorkování pravděpodobnosti
lasamplerange	0.005	rozsah úhlového vzorkování pravděpodobnosti
lasamplestep	0.005	krok v úhlovém vzorkování pravděpodobnosti

Číselná hodnota vpravo u prvního parametru v dané sekci znázorňuje označení dané skupiny. Provedená měření se prováděla vždy se změnou parametrů pro celou skupinu. Výsledky pro první skupinu:

$linearUpdate$	$angularUpdate$	$e(t, g)$
0.1	0.05	1.318
0.5	0.25	1.519
1	0.5	1.436
2	1.0	1.132
4	1.25	1.460

Z této tabulky je vidět, že ve většině případů nemá dané nastavení velký vliv na průběh programu. S nastavením na $linearUpdate = 2$ a $angularUpdate = 1$ je však dosaženo znatelného zlepšení. Snížením vzorkování dochází ke snížení výkonových nároků, ale stále je udržena dostatečná frekvence pozorování pro správné sestavení mapy. Pro tento případ tedy dané nastavení zlepšuje průběh programu.

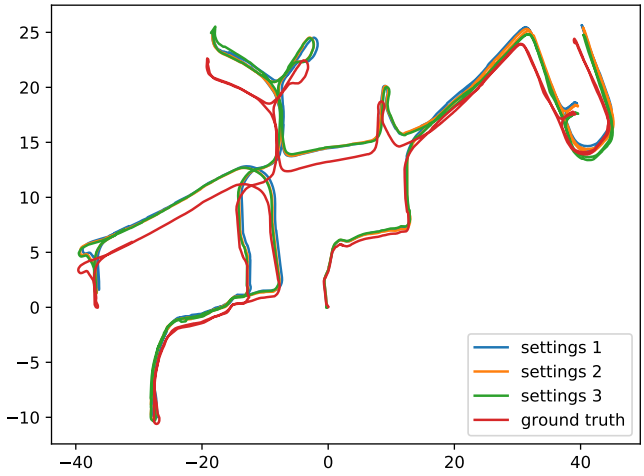


Obrázek 5.1: Získané trajektorie pro první skupinu nastavení

Pro druhou sadu nastavení se měnil pouze parametr *particles*:

particles	$e(t, g)$
15	1.567
30	1.436
45	1.433

Ze zdejších výsledků je znatelné, že se průběh programu s počtem vytvářených částic zlepšoval. Jedná se o logický krok, kdy by mělo docházet k úměře počtu částic a přesnosti trajektorie. Čím více částic je produkováno, tím je větší šance, že je vytvořena částice bližší reálným souřadnicím. Naproti tomu se s přibývajícími částicemi zvyšují výkonnové nároky programu, proto se nemůže nastavovat tento parametr na příliš vysoké hodnoty.



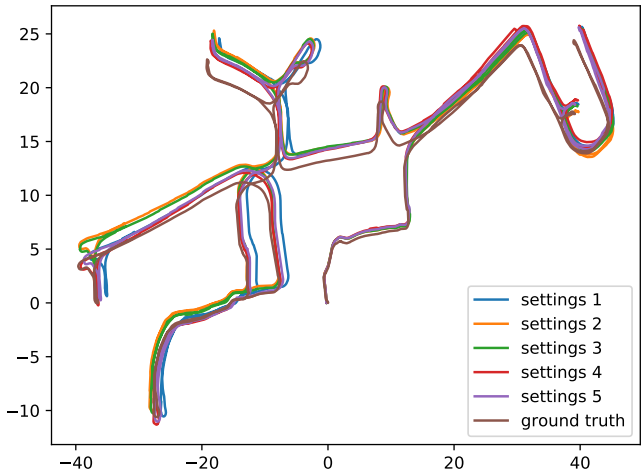
Obrázek 5.2: Získané trajektorie pro druhou skupinu nastavení

V poslední testované skupině se měnily parametry pro vzorkování pravděpodobnosti:

llsamplerange	*llsamplestep*	*lasamplerange*	*lasamplestep*	$e(t, g)$
0.001	0.001	0.0005	0.0005	1.742
0.005	0.005	0.001	0.001	1.568
0.01	0.01	0.005	0.005	1.436
0.1	0.1	0.05	0.05	1.256
0.5	0.5	0.2	0.2	1.250

Z posledních výsledků je patrné, že zmenšením oblasti vzorkování není v tomto případě dosaženo lepších výsledků. Důvodem je rychlejší degenerace částic, které jsou vytvořeny v menší oblasti. Pokud je tedy tato oblast určená špatně, všechny tyto částice selžou.

Z těchto informací lze dojít k závěru, že systém je defaultně dobře nastavený na výkonější zařízení. Jednotlivá nastavení lze upravit tak, aby algoritmus poskytnul lepší výsledek, při kombinaci více parametrů dohromady však již vzniká riziko zvýšení výkonnových nároků natolik, že má změna nastavení na program opačný vliv.



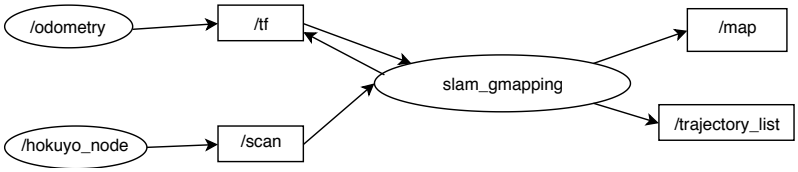
Obrázek 5.3: Získané trajektorie pro třetí skupinu nastavení

5.2 Návrh změn v systému

Systém gMapping je v současné podobě kvalitní systém. Je však možné stanovit vylepšení, která mohou vést k větší přesnosti, stabilitě či uživatelské přívětivosti.

Jako první věc, kterou je dobré uvést, je výše zmíněná neprodukce extrahovatelné trajektorie. Jako řešení se zde nabízí implementace systému pro publikování trajektorie do topiku. Topik by mohl mít podobnou strukturu jako */trajectory_node_list* v Cartographeru. Ten, jak je popsáno v kapitole 4.2.3, nemá však pro potřebné porovnávání dostatek informací. Pro případ gMappingu by tak mohla být do určitého topiku publikovaná zpráva o aktuální trajektorii, která by nesla více informací. Obsah topiku by tak obsahoval vždy hlavu, ve které by byl uvedený aktuální čas. K tomu by byly připojeny pod sebou dle času seřazené částice tvořící aktuální trajektorii. K těmto částicím by byla přidružena vždy informace o simulačním čase, kdy byla částice vytvořena, její souřadnice v prostoru a informace o natočení. Při přijímání dat z topiku by tak bylo možné v průběhu programu získávat srovnání, jak se trajektorie v průběhu času optimalizovala. Při zavolání po doběhnutí datasetu by pak byla poskytnuta rovnou hotová trajektorie. Pro lepší představu je na obrázku 5.4 vyobrazený výpočetní graf pro takto upravený systém, kde /odometry posílá zaznamenanou odometrii, /tf obsahuje informace o souřadných soustavách, /scan obsahuje data ze senzoru, která jsou přijímána z /hokuyo_node. GMappingem jsou tato data zpracována. Produktem zpracování je běžně pouze mapa /map, v tomto případě i trajektorie zaznamenávaná do /trajectory_list.

Druhým možným přístupem by bylo řešení pomocí funkce *rosservice*. Volání trajektorie by tak probíhalo stejným způsobem, jak tomu bylo přímo u gMappingu v kapitole 4.2.3. Rozdíl by však spočíval v tom, že trajektorie získaná přístupem z 4.2.3 se zpětně neupravuje. Vybrané částice tak utvářejí trajektorii, i když už byly při optimalizaci programem nahrazeny. Pokud by však tato zpráva byla publikována přímo gMappingem, daný problém by se odstranil.

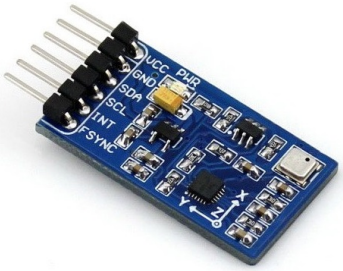


Obrázek 5.4: Znázornění odebíraných a publikovaných topiků při implementaci extrahovatelné trajektorie.

Další prospěšnou změnou by pro systém byla možnost upravovat parametry za běhu programu. Pro výše provedené testy nebyla taková úprava potřeba, neboť se jak reálný, tak simulační robot pohybovali po celou dobu ve stejném prostředí. Při uvážení, že by se však robot přemisťoval mezi uzavřenými a otevřenými prostory, tato možnost by byla jistě užitečná. Při běžném průběhu, kdy uživatel robotem sám jezdí, může program vypnout a spustit gMapping s jiným nastavením, což není příliš hezké řešení. Pro autonomního robota by to však mohl být větší problém. V případě, že vyjede do jiného typu prostředí mohl by si sám přenastavit parametry tak, aby

odpovídaly uloženému nastavení pro dané prostředí.

Zlepšení systému by bylo jistě dosaženo i integrací jednotky IMU, která by zajišťovala zlepšení odhadu pozice. V jednotce IMU jsou obsaženy gyroskopy, akcelerometry a magnetometry, které zjišťují stoupání, vybočení a rolování. Zaznamenáváním zmíněných hodnot je tedy získán lepší přehled o pohybu robota, než pouze z odometrie kol a tím tedy zlepšení odhadu pozice. Příklad, jak tato jednotka může vypadat, je na obrázku 5.5. [13]



Obrázek 5.5: Jednotka IMU určená pro použití s mikročipy Arduino.

Další potenciální změnou, která se již nevztahuje přímo k procesům v gMappingu, by byla integrace kamerového vidění, například monokulární kamery. Robot by snímal prostředí jak LIDARem, tak kamerou. V případě kamerových dat by se snažil rozeznat předměty v prostoru a následně předat informaci o tom, jaký typ předmětu to je. Například při detekci židle by byla vyslána informace, že se daný předmět nemá zanášet do mapy, případně z mapy vymazat. Do místa, kde je židle postavena, by se ještě přidala značka substituující židli. Při opětovném projetí by pak robot věděl, že v daném místě by se mohla židle nacházet a v případě, že tam stále nějaký předmět je, rovnou ho označit jako židli. Při velké množině objektů v paměti by se tak ušetřilo mnoho výpočetního výkonu. "Pohyblivé"předměty jsou v mapě navíc a při jejich přemístění může mít robot problém poznat stejné místo. Pokud by tedy v mapě zaneseny nebyly a robot měl opět jak LIDAR, tak kameru, sám by opět vyhodnotil, že předměty jsou v prostoru navíc, a počítal by tak s mapou předměty neobsahující. Pokud by mu tak byla poskytnuta mapa nějakého takto "vyčištěného"prostředí, byl by v něm schopen určit svoji pozici bez ohledu na předměty, které v daném prostoru již vůbec nejsou, jsou přemístěny nebo se tam dříve ani nenacházeli. Jedno z možných zapojení se nachází na obrázku 5.6. [14]

Obrázek 5.6: Příklad sestavy kamera-LIDAR.

Kapitola 6

Závěr

Cílem této práce bylo bližší seznámení se systémy produkujícími mapu na základě dat z 2D LIDARu. Práce začíná u obecných pojmů a uvedením do problematiky simultánní lokalizace a mapování. Následně jsou popsány jednotlivé přístupy, kterými se dá tento problém řešit. Těmito přístupy je rozšířený Kallmanův filtr, Rao-Blackwellizovaný částicový filtr a grafová reprezentace.

V další části je již popisováno, jak jsou dané přístupy implementovány do reálných systémů, které řeší problém SLAM. Popsány a náledně porovnány jsou systémy gMapping, Hector SLAM a Cartographer. Zmíněné systémy byly vybrány z důvodu jejich rozšířenosti a kvality zpracování.

Pro provádění simulací, nahrávání dat a získávání výsledků byl použit framework sloužící k programování a manipulaci s roboty (ROS). V prostředí ROS se, díky početné komunitě a kvalitní dokumentaci, pracovalo dobře. Návody jsou ve velké většině případů aktualizované a tak stále funkční. Při hledání problémů, které nepokrývá dokumentace, je možnost využít komunitní úroveň ROSu a problém tak vyřešit. Při práci se samotnými třemi systémy, byla učiněna řada zjištění ohledně jejich výkonu a stability.

Prvním a i nejvíce rozebíraným systémem byl gMapping. Má kvalitně zpracovanou dokumentaci a je nejvíce prověřený časem. Stabilita systému byla po celou dobu dobrá. Například, i přes celkem složitou situaci s daty popsanými v sekci 4.3.4, dokázal na defaultní nastavení parametrů utvořit kvalitní mapu ve všech případech. Bližší popsání nedostatků je rozebráno v kapitole 5.2.

Druhým popisovaným a zkoumaným systémem je Hector SLAM, který je vytvořen především pro rychlý pohyb v prostoru a rychlé rozhodování o nadcházející cestě. Z těchto informací je patrné, že je kladen důraz na co nejnižší výkonnostní nároky. Zatímco na menších a dobře rozeznatelných prostorech běží rychle a kvalitně, ve složitějších oblastech má velké nedostatky. Několikrát zmíněný je problém s chodbami, kde pro Hector SLAM není dostatek záchytných bodů pro indikaci pohybu vpřed. Daná problematika se dá částečně odstranit častějším měřením a zapisováním pozorování. Daná změna má však za následek zhoršení jiných vlastností. Například se zhoršuje orientace při rychlém otočení (viz. graf 4.6). Celkově byla třeba některá nastavení více upravovat a i tak nebylo zaručeno získání správného výsledku. V uvedených testech většinou Hector SLAM dokázal vytvořit kvalití mapu

a přesnou trajektorii. Ve spoustě případů však docházelo k destabilizaci systému a znedhodnocení celého výsledku.

Posledním rozebíraným systémem je Google Cartographer. Jedná se o nejnovější a nejpropracovanější systém ze všech uvedených. Tento aspekt byl ale ve velkém množství případů na škodu. Z hlediska stability, systém vytváří přesnou a kvalitní mapu, kdy se projevuje důležitost back-end procesu, optimalizace mapy. Problémem však je, že finální podoba systému vznikla před dvěma lety, části dohledatelné dokumentace přitom stále odkazují na předchozí verze. Například v sekci 4.2.3 je uveden Cartographerem publikovaný topik */trajectory_node_list*. Ten je správně zobrazen ve výpočetním grafu v oficiální dokumentaci Cartographeru, nikde se však o něm nedá dohledat žádná informace. Při komplikovanosti systému se dá absence spolehlivé dokumentace považovat za významný problém pro každého člověka, který se bude chtít se systémem seznámit. Hledání řešení pak může zabrat velké množství času, což se projevuje zejména při snaze systém spustit.

Finální porovnání systémů a zhodnocení jejich použitelnosti je následující. V případě potřeby pouze navigačních informací a pro rychlou tvorbu mapy, nejlepší volbou je Hector SLAM. Stejně tomu tak je v případě menších uzavřených prostor s větším množstvím záchytných bodů. S takovými prostory samozřejmě nemají problém ani gMapping a Cartographer. Pro rozsáhlá území by měl být nejlepší Cartographer. Pokud tedy člověk věnuje dostatek času zkoumáním nastavování a poté jeho změnami a testováním, jedná se o nejkvalitnější možnost. Ve zkoumaném rozsahu však gMapping vždy splnil své úkoly a větší smyčky také detekoval a uzavřel. Pro celkovou jednoduchost v nastavování, rozsah použití a odladěnost systému, je na základě této práce gMapping hodnocen jako ten nejspolehlivější.

Díky svým kvalitám tak byl gMapping vybrán pro bližší prozkoumání. Byly na něm testovány změny konfigurace (kapitola 5.1), na které se objevila odpovídající reakce. Při práci s gMappingem se však přišlo na to, že i zde je prostor pro zlepšení. V kapitole 5.2 jsou posány možnosti, jak systém z hlediska výkonu i uživatelsky vylepšit. Z pohledu uživatele se jedná o možnost extrakce trajektorie a změnu parametrů za běhu programu. Z hlediska výkonnostního pak integrace jednotky IMU. Na konci sekce je ještě uvedena možnost integrace kamery. V tomto případě se jedná o výrazné rozšíření systému, ne pouze o zlepšení. Tato změna by tak posunula systém ještě na vyšší úroveň, což může být zajímavý námět na budoucí práci.