Bakalářská práce

Lukáš Kuhajda

Akademický rok 2018/2019

Obsah

1 Úvod

Ve své bakalářské práci se věnuji systémům, které pomocí měření z LIDARu (Light Detection And Ranging) utváří mapu prostředí, v němž se pohybují. V první části práce jsem se zaměřil obecně na problém simultánní lokalizace a mapování (SLAM). Mobilní robot je umístěn do neznámého prostředí a jeho úkolem je určovat svoji pozici a utvářet mapu. Zabývám se zde vnitřními principy a různými přístupy k řešení problému. Daná tématika momentálně vstupuje do podvědomí i širší veřejnosti, neboť pomalu dochází k přechodu na autonomní vozidla, která fungují na podobných principech. Tyto automobily však využívají i mnoho dalších senzorů, jako jsou například kamery. Já se zaměřuji na systémy využívající čistě jen 2D LIDAR, tedy sezor, měřící vzdálenosti pouze v jedné výškové úrovni.

V další části své práce jsem se podrobněji věnoval třem nejrozšířenějším 2D SLAM systémům, přesněji se jedná o GMapping, HectorSLAM a Google Cartographer. Popsal jsem zde jejich základní rysy, kterými se liší od ostatních a jimiž je dostatečně charakterizována jejich funkčnost.

měření, výsledky, porovnání, změna v gmappingu, proč jsem si vybral ...

2 Simultánní lokalizace a mapování

V této sekci se nachází seznámení s problémem simultánní lokalizace a mapování. Procházím zde jak historii, tak strukturu 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 snížení geometrické nepřesnosti. Důležitým prvkem zde bylo zjištění, že mezi odhady landmarků na mapě je velká korelace, která je rostoucí s dalšími pozorováními.

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. 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 korelaci landmarků snižovat.

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). 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.

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.

2.2.1 Pravděpodobnostní SLAM

Cílem je získat v časovém okamžiku k odhadu hustoty pravděpodobnosti

$$P(\mathbf{x}_k, \mathbf{m} | \mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) \tag{1}$$

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 srduženou 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 známe polohu vozidla $\mathbf{x}_k$ a landmarků $\mathbf{m}$.

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \tag{2}$$

Model pohybu vozidla může být popsán jako stavový přechodový model. Předpokládáme přechodový stav 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.

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \tag{3}$$

Máme tak implementaci ve formě dvoukrokého rekurzivního algoritmu.

Aktualizace času (predikce):

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \int P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \times P(\mathbf{x}_{k-1}, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k-1}, \mathbf{x}_0) d\mathbf{x}_{k-1} \tag{4}$$

Aktualizace měření (korekce):

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k}, \mathbf{U}_{0:k}, \mathbf{x}_0) = \frac{P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) P(\mathbf{x}_k, \mathbf{m}|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k}, \mathbf{x}_0)}{P(\mathbf{z}_k|\mathbf{Z}_{0:k-1}, \mathbf{U}_{0:k})} \tag{5}$$

2.2.2 Struktura

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

$$P(\mathbf{x}_k, \mathbf{m}|\mathbf{z}_k) \neq P(\mathbf{x}_k|\mathbf{z}_k) P(\mathbf{m}|\mathbf{z}_k), \tag{6}$$

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 + odkaz). 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 robota v prostoru, 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 tak získáváme přesnější a robustnější mapu. Například násobným průchodem jedné smyčky se získá znatelně kvalitnější záznam o daném prostředí, než pouze jedním projetím.

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*). Jinou 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.

2.4 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

$$P(\mathbf{x}_k|\mathbf{x}_{k-1}, \mathbf{u}_k) \leftrightarrow \mathbf{x}_k = \mathbf{f}(\mathbf{x}_{k-1}, \mathbf{u}_k) + \mathbf{w}_k, \tag{7}$$

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

$$P(\mathbf{z}_k|\mathbf{x}_k, \mathbf{m}) \leftrightarrow \mathbf{z}_k = \mathbf{h}(\mathbf{x}_k, \mathbf{m}) + \mathbf{v}_k \tag{8}$$

kde funkce $\mathbf{g}(.)$ popisuje geometrické vlastnosti pozorování a $\mathbf{v}_k$ jsou chyby měření.

Tyto dvě definice jsou využity v metodě EKF k výpočtu průměru a kovariance sdruženého posteriorního rozložení

průměr:

$$\begin{bmatrix} \hat{\mathbf{x}}_{k|k} \\ \hat{\mathbf{m}}_k \end{bmatrix} = E \begin{bmatrix} \mathbf{x}_k|\mathbf{Z}_{0:k} \\ \mathbf{m} \end{bmatrix} \tag{9}$$

kovariance:

$$\mathbf{P}_{k|k} = \begin{bmatrix} \mathbf{P}_{xx}\mathbf{P}_{xm} \\ \mathbf{P}_{xm}^T\mathbf{P}_{mm} \end{bmatrix}_{k|k} = E \left[\begin{pmatrix} \mathbf{x}_k - \hat{\mathbf{x}}_k \\ \mathbf{m} - \hat{\mathbf{m}}_k \end{pmatrix} \begin{pmatrix} \mathbf{x}_k - \hat{\mathbf{x}}_k \\ \mathbf{m} - \hat{\mathbf{m}}_k \end{pmatrix}^T |\mathbf{T}_{0:k} \right] \tag{10}$$

Aktualizace času je pak počítána jako

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{f}(\hat{\mathbf{x}}_{k-1|k-1}, \mathbf{u}_k) \tag{11}$$

$$\mathbf{P}_{xx,k|k-1} = \nabla\mathbf{f}\mathbf{P}_{xx,k-1|k-1}\nabla\mathbf{f}^T + \mathbf{Q}_k, \tag{12}$$

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

$$\begin{bmatrix}\hat{\mathbf{x}}_{k|k}\\ \hat{\mathbf{m}}_k\end{bmatrix} = [\hat{\mathbf{x}}_{k|k-1}\hat{\mathbf{m}}_{k-1}] + \mathbf{W}_k[\mathbf{z}_k - \mathbf{h}(\hat{\mathbf{x}}_{k|k-1}, \hat{\mathbf{m}}_{k-1})] \tag{13}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{W}_k\mathbf{S}_k\mathbf{W}_k^T, \tag{14}$$

kde

$$\mathbf{S}_k = \nabla\mathbf{h}\mathbf{P}_{k|k-1}\nabla\mathbf{h}^T + \mathbf{R}_k \tag{15}$$

$$\mathbf{W}_k = \mathbf{P}_{k|k-1}\nabla\mathbf{h}^T\mathbf{S}_k^{-1}, \tag{16}$$

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, sdružování dat a nelinerita. Prvním z nich, konvergence, se projevuje postupným přechodem determinantu kovarianční matice mapy a všech podkategorií dvojic landmarků k nule. Jednotlivé odchylky landmarků pak konvergují dle původních nepřesností z odhadu pozice robota a jeho pozorování.

Výpočetní složitost je kvadraticky rostoucí s počtem zaznamenaných landmarků, neboť při každém zaznamenaném pozorování se aktualizují již uložené landmarky. Tento problém prošel vývojem a existují metody pracující v reálném čase s tisíci landmarky.

Metoda EKF-SLAM je velmi náchylná na chybné spojení pozorování se známými landmarky. Jedná se zejména o problém s uzavřením smyčky, kdy dochází k opětovnému návratu na místo, ze kterého robot začínal nebo ve kterém se již nacházel.

Nelinerita je posledním z významných problémů. Kvůli ní 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.

2.4.1 Výpočetní složitost

Neobvyklá struktura problému, sdružený stav složený z pozice robota a landmarků, je využita v řadě metod pro redukci výpočetní složitosti. Zde má pohybový model vliv pouze na stav pozice robota a model pozorování na dvojici robot-landmark. Základním rozdělením metod redukujících výpočetní složitost, je rozlišování optimálních, konzervativních a nekonzistentních metod. První typ, optimální metody, jsou založené na redukci daného výpočtu, výsledkem jsou pak odhady a kovariance, stejně tak, jako je tomu v případě plnohodnotné formy SLAM. U metod konzervativních dochází k odhadům s vyšší neurčitostí nebo kovariancí, většinou ale, i přes větší nepřesnost, jsou implementovány v reálném použití. Poslední možností jsou nekonzistentní metody a jedná se o algorithmy, které mají nižší neurčitost nebo kovarianci, než algoritmy optimální. Pro řešení SLAM v praxi se ale nepoužívají.

Prvním přístupem pro redukci výpočetní složitosti, je omezení požadovaného výpočtu rovnicí aktualizace pozorování. Výpočet časové aktualizace může být omezen metodami využívající rozšířený stav, výpočet stavu pozorování pak metodami

oddělujícími rovnice dané aktualizace a obě tyto omezení vedou k redukci výpočtů, typicky jsou to optimální algoritmy. Další možností, je reformulace stavového prostoru do informační podoby, která umožnuje rozdělení výsledné matice s informacemi pro snížení výpočtů, což bývají algoritmy konzervativní. Obvykle je díky nim znatelně redukována výpočetní složitost a stále je zachována dostatečně dobrá odhadovací schopnost. Posledním přístupem, o kterém se zmíním, je submapping. Jde o rozdělování mapy na regiony, kdy následné aktualizace se týkají pouze dané oblasti a s určitou periodou poté i v rámci celé mapy.

2.4.2 Oddělné aktualizace

Jedná se o metody vytvářející 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. V této metodě se však mapa rozdělí na submapy.

Rozlišujeme 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, což jsou algorithmy *constrained local submap filter* (CLSF, v překladu - omezený lokální submapový filtr). Pokračovat budu v rozboru druhé možnosti, 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.

Logaritmus submapy 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 vlatnosti obou map.

2.4.3 Rozčlenění stavového prostoru

V této metodě se vyjadřuje stavový odhad $\hat{x}_k$ a matici kovariance P_k v informační formě pomocí matice informací $Y_k = P_k^{-1}$ a vektoru informací $\hat{y}_k = Y_k\hat{x}_k$. Je výhodné pro mapy s větším měřítkem, kdy spousta nediagonálních prvků bude velmi blízkých nule, což vede k možnosti nastavení těchto prvků na hodnotu nula. Může tím však vznikat malá ztráta optimality při vzniku map.

Rozšíření stavu je rozčleňovací operace vedoucí k přesnému rozčlenění informační formy. Předpokládáme, že podmnožina stavů x_i obsahuje většinu stavů mapy a po rozčlenění dosahuje pouze konstantní složitosti v čase. Můžeme tedy získávat přesné řešení díky rozšiřování stavu novým odhadem pozice robota v každém kroce a zachovat všechny předchozí pozice. Nenulové nediagonální prvky jsou pouze ty, které jsou spojené napřímo s měřenými daty.

Dále sem musíme zahrnout marginalizaci, jež je nezbytná pro odstranění předchozích stavů pozice. Máme možnost marginalizovat všechny předchozí stavy, což vede na

zhuštěnou matici informací, což je nežádoucí. Správnou volbou ukotvení pozice můžeme marginalizovat velkou čast pozic, aniž bychom vyvolaly nadměrnou hustotu matice informací.

2.5 Rao-Blackwellizedův částicový filtr

Forma SLAM založená na Rao-Blackwellizedově filtru (RBPF), jinak nazývaná také jako FastSLAM, je na bázi rekurzivního 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 $online\ SLAM$. Tato forma počítá posterior pro celou cestu, tím je tedy brána jako řešící problém $full\ SLAM$. Odhadováním pozice robota v každém časovém okamžiku je algoritmus označován jako online. O odhad pozice se starají právě částicové filtry.

Sdružený stav může být jako faktor komponent robota a podmíněných komponent mapy

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(m|X_{0:k}, Z_{0:k})P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0), \tag{17}$$

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 partikulárním 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.

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

$$\{w_{k-1}^{(i)}, X_{0:k-1}^{(i)}, P(m|X_{0:k-1}^{(i)}, Z_{0:k-1})\}_i^V, \tag{18}$$

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 kroce je pro každou částici vypočítáno návrhové rozložení, jež je podmíněno svojí specifickou 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:

$$w_t^i = \frac{a}{b}, \tag{19}$$

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

různě často dle implementace. Krokem posledním je provedení EKF aktualizace pro každou již zpozorovanou částici, která je při aktuálním pozorováním zaznamenána.

2.5.1 Sdružování neznámých dat

Funkcionalita systému popisovaná výše se vztahuje k datům, která mají známé sdružování. Pokud se jedná o data, pro která danou informaci nemáme, 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.

2.6 Asociace dat

Jedná se 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 úpté využito pro přepověd dané asociace, nejčastěji pro problém s uzavřením smyčky. Pokrok této metody přišel s počítáním metriky podobnosti přes sekvenci obrazů, místo původního jednoho.

2.6.1 Multihypoziční asociace dat

Jedná se o metodu nepostradatelnou pro robustní sběr dat v přeplněném prostředí, kdy se vytváří oddělené odhady trasy pohybu pro každou asociační hypotézu. Tato funkce je však silně limitována dotupným výpočetním výkonem. Dále je metoda vyžívána při implementaci robustního SLAMu ve velkých prostředích, kdy je při uzavírání smyčky vytvořena hypotéza pro smyčku uzavřenou i pro stále neuzavřenou. Tím se bere v potaz, že je prostředí pouze podobné.

2.7 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.

2.7.1 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.

2.7.2 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 paměťové zrdoje.

Důležitým faktorem je zde takzvaná obsazenost, v angličtině *occupancy*. V originále se tedy metoda často nazývá *Occupancy Grid Maps*. Pravděpodobnost obsazení každé buňky může nabývat tří hodnot. Při inicializaci je hodnota pravděpodobnosti nastavena na $P(m_i) = 0.5$, kdy m_i je daná buňka. Tato počáteční hodnota znamená, že o buňce nemáme žádnou znalost. Poté, co se buňka dostane do dosahu senzoru, nabývá pravěpodobnost obsazenosti již pouze dvou hodnot. $P(m_i) = 0$, pokud je buňka neobsazená a $P(m_i) = 1$ v případě obsazenosti.

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

$$p(m) = \Pi_i p(m_i) \tag{20}$$

Odhad mapy ze senzoru:

$$p(m|z_{1:t}, x_{1:t}) = \Pi_i p(m_i|z_{1:t}, x_{1:t}), \tag{21}$$

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

3 GMapping

3.1 Úvod

Jako velice podstatný úkol mobilního robota, bereme schopnost tvorby mapy. Ta může být kvalitně vytvářena, pokud existuje dobrý odhad pozice a pozice zase správně získávána, pokud je dostatečně kvalitní mapa. Efektivním řešením je tedy Rao-Blackwellizedův částicový filtr, který se může ještě vylepšit. Jednou možností, 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.

Návrhové rozložení je získáváno vyhodnocováním pravděpodobnosti pozice robota z kombinace informací ze senzoru a odometrie. Poslední pozorování je využito pro tvorbu nových částic, odhad stavu se tedy provádí na základě více informací, než jen z odometrie.

3.2 Mapování pomocí RBPF

Základním úkolem je odhad posteriorní pravděpodobnosti a tím zisk mapy a trajektorie pohybu. Odhad je prováděn díky pozorování a informaci z odometrie. Nejprve dochází k odhadu mapy, která je utvářena z 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 případěm typ SIR (Sampling Importance Resampling), který při mapování postupně zpracovává data ze senzoru, poté odometrii a aktualizuje sadu vzorků reprezentující posteriorní pravděpodobnost zahrnující informace o mapě a trajektorii.

Celý proces začíná ziskem nových částic odběrem vzorků z předchozí generace návrhového rozložení. Dále 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.

3.3 RBPF s vylepšenými návrhy a adaptivním převzorkováním

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 nebylo by třeba převzorkování.

Typicky návrhové rozložení 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, přílišnému poklesu vážených důležitostí. Následkem je ale zkreslení mapy. To se však dá vyřešit zahrnutím posledního pozorování do

generování nových vzorků. Díky tomu dochází k zaměření se na vzorkování ve významné oblasti pravděpodobnosti pozorování.

Zlepšením návrhu se objevuje možnost získávat pro každou částici zvlášť její parametry Gaussiánského návrhu a snižuje se také neurčitost výsledných hustot pravěpodobností. Porovnávač pozorování určuje režim významné oblasti pravděpodobnostní funkce pozorování. Také má většinou funkci maximalizace pravděpodobnosti pozorování, tvorby mapy a počáteční odhad pozice robota. Pokud je pravděpodobnostní funkce vícerežimová, například při uzavírání smyčky, porovnávač vrací pro každou částici nejbližší maximum, což může způsobit vynechání některých maxim v pavděpodobnostní funkci.

Převzorkování je velice důležitým aspektem určujícím výkon částicového filtru. Dochází k nahrazování vzorků s nízkou váhou, těmi s váhou vysokou. Je to nezbytný proces, neboť je potřeba konečného počtu částic pro aproximaci cílového rozložení. Může však odtranit dobré vzorky a ochudit tak částice, je proto důležité mít pro převzorkování vhodné rozhodovací kritérium a provádět ho ve správný čas.

$$N_{eff} = \frac{1}{\Sigma_{i-1}^{N}(w^{(i)})^2} \tag{22}$$

$w^{(i)}$ 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.
Základní nastavení převzorkování je následující:

$$N_{eff} < N/2, \tag{23}$$

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

Algoritmus probíhá následovně. Dojde k zisku odhadu pozice, který je reprezentovaný danou částicí. 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 vybrání sady vzorků v okolí dané pozice, vypočítání průměrů a kovarianční matice návrhu bodovým hodnocením 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.

4 Catographer

4.1 Úvod

Systém Cartographer je vyvíjen společností Google. Je vhodný pro tvorbu rozsáhlých map a zisk optimalizovaných výsledků v reálném čase. Při porovnávání snímků typem scan-to-scan dochází rychlému hromadění globálních chyb, v případě porovnávání stylem scan-to-map dochází, při správném odhadu pozice a kvalitnímu snímku z LIDARu, k velké redukci chyb, což vede k větší efektivnosti a robustnosti algoritmu. Dále se využívá porovnávání přesnosti pixelů. Jde o metodu redukující hromadění lokálních chyb a je vhodná při řešení problému uzavírání smyčky.

4.2 Přehled

Cartographer vytváří v reálném čase 2D mřížkovou mapu (grid mapu) s rozlišením na 5 cm. 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, tím je odhad pozice. Odhad se aktualizuje po několika vteřinách a uzavření smyčky je tedy okamžitě viditelné.

4.3 Lokální 2D SLAM

Rozlišujeme 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 obsažená jednotka IMU (inertial measurment unit), která slouží k odhadu směru gravitace, což má využití při pohybu na nerovné ploše.

Skeny prostředí se iterativně zarovnávají se snímky již obsaženými v submapě. Submapa je tedy v postatě zachycení kousku světa, který je složený 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 minutí je vložen bod mřížky sdružený se všemi pixely, které jsou protínány jedním paprskem mezi počátkem skenování a každým snímacím bodem. Nepřidávají se sem body již přidané do množiny zásahů. Doposud 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.

$$argmin_{\xi}\Sigma_{k=1}^{K}(1 - M(T_{\xi}h_k)) \tag{24}$$

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.

4.4 Uzavírání smyčky

Daný systém pracuje v oblasti submap s porovnáváním scan-to-scan, hromadí se v něm tedy lokální chyby, ale pár snímků za sebou má 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 můžeme jednoduše přidávat zbytky pro zohledňování dalších dat. Jednou za pár sekund je, pro optimalizaci pozice skenu vůči daným omezením, spuštěn Ceres scan matcher. Omezeními jsou myšlena relativní pozice ξ_{ij} a kovarianční matice Σ_{ij}.

$$argmin_{\Xi^m,\Xi^s}\frac{1}{2}\Sigma_{ij}\rho(E^2(\xi_i^m, \xi_j^s, \Sigma_{ij}, \xi_{ij})), \tag{25}$$

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.

$$\xi^* = argmax_{(\xi\epsilon W)}\Sigma_{k=1}^{K}M_{nearest}(T_{\xi}h_k) \tag{26}$$

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 tuple integerů:

$$c = (c_x, c_y, c_\Theta, c_h)\epsilon Z^4, \tag{27}$$

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

Horní meze jsou pak počítány na vnitřních uzlech, zde je zájem o výpočetní úsilí a kvalitu spojení.

$$score(c) = \Sigma_{k=1}^{K} max_{(j\epsilon\bar{W})} M_{nearest}(T_{\xi_j} h_k) \tag{28}$$

5 Hector SLAM

5.1 Úvod

Hlavním znakem systému Hector SLAM, je jeho rychlost a nízké výpočetní nároky oproti předchozím dvěma typům, které jsou v této práci rozebírány. Je tedy 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 jednotka IMU, která se nachází i v systému Cartographer. Hector SLAM je braný jako front-end SLAM, což znamená, že dochází k odhadu stavu robota v reálném čase. Back-end SLAM optimalizuje poziční graf vzhledem k omezením mezi pozicemi.

5.2 Přehled

Daný systém se musí převézt z 3DOF pomocí naklonění a rotace na 6DOF, kdy navigační filtr spojí měření z inerciální jednotky (IMU) a dalších senzorů. Tím dojde k získání 3D řešení a díky 2D SLAMu je obsažena informace 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, zahrnujeme do systému další informace. Jedná se například o porovnávání snímků, snímání magnetického pole, senzor barometrického tlaku a nebo měření rychlosti kol.

Pohyb robota je popsán:

$$\dot{\Omega} = E_{\Omega}.\omega \tag{29}$$

$$\dot{p} = v \tag{30}$$

$$\dot{v} = R_{\Omega}.a + g \tag{31}$$

$\Omega = (\phi, \theta, \psi)^T$ je zde 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.

5.3 2D SLAM

Jako reprezentace prostředí je zde vybrána osvědčená metoda mřížkových map. Odhadovaná pozice robota slouží pro transformaci skenu na lokální stabilizovaný souřadnicový rámec. Odhadovanou orientací a přidruženými hodnotami je ze skenu vytvořen oblak bodů, který je pro odstranění odlehlých bodů předzpracováván. Filtrace probíhá pouze na základě souřadnic koncového bodu, kdy jsou pro porovnávání skenu 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 ta možnost je. Jakoukoliv souřadnici, kterou potřebujeme nahradit, aproximujeme pomocí čtyř nejbližších integerových souřadnic.

Laserové skenery jsou již 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 při nejlepším sladění skenu s mapou:

$$\xi^* = argmin_\xi \Sigma_{i=1}^n [1 - M(S_i(\xi))]^2 \tag{32}$$

$M(P_m)$ 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:

$$\Sigma_{i=1}^n [1 - M(S_i(\xi + \Delta\xi))]^2 \to 0, \tag{33}$$

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 součastně aktualizují podle odhadu aktuální pozice, který je generovaný zarovnávacím procesem. Díky tomuto přístupu jsou mapy konzistentní a nepotřebují převzorkovávání. Zarovnávání skenů probíhá pouze na mapě s nejvyšším rozlišením a tento odhad je pak, jak je uvedeno, použit pro ostatní mapy. Mapy s nízkým rozlišením jsou tedy v podstatě dostupné okamžitě, a lze je hned použít pro odhad trasy.

5.4 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 porovnávače 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 pozici aktualizujeme integrací zrychlení. Aby se ale zabránilo nezávislému růstu odhadu stavu při nepřítomnosti měření, dochází k aktualizace pseudo-nulové rychlosti, které se spouští při dosažení odchylky určité prahové hodnoty a zajišťuje tak stabilitu systému.

Při integraci systému jsou přítomny základní dva celky, 2D SLAM a 3D stavový odhad, které spolu musejí komunikovat v obou směrech. Jejich fungování neni

nijak synchronizováno, odhad stavu běží typicky s vyšší frekvencí. Odhadovaná pozice z EKF je brána jako počáteční odhad pro optimalizaci porovnávání. Opačným směrem je použit pro spojení pozice ze SLAM s celkovým odhadovaným stavem kovarianční průsečík (CI - covariance interseption).

6 Experimentální měření

Pro testování všech výše zmíněných systémů jsem využil framework *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.

6.1 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 *nodes*, 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. *Nodes* 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 zabalení 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.

6.1.1 Souborový systém

Na úrovni souborového systému jsou v ROSu myšleny všechny prostředky, se kterými se pracuje na pevném disku. Jedná se o balíčky (*Pakages*), *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.

Metapackage 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*. *Stacks* 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 VCS (*Version Controlling System*), což je systém sdílející stejnou verzi. Balíčky tak mohou být společně uvolněny užitím nástoje *catkin*. Repozitáře často mapují převedené *Stacks* 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 datvé struktury pro posílané zprávy. Definice typů služeb se nachází na adrese *my_package/srv/MyServiceType.srv* a definuje datové struktury požadavků a odpovědí.

6.1.2 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 *nodes*, *Master*, *Parameter Server*, zprávy, topiky, služby a bagy. Všechny pojmy jsou implementovány v repozitáři pro komunikaci *ros_comm*.

Nodes, v překladu uzly, jsou procesy obstarávající výpočty. Řídicí systémy pro robota obsahují často mnoho uzlů. Pro ROS je typická snaha o co největší modularitu, každý *node* má tak jednu svoji úlohu, například zpracování laserového senzoru. *Nodes* 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 *nodes*. 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 chce *node* určitý typ dat, začne odeírat 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 *node* může odebírat a publikovat do více topiků. *Nodes*, 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 *node* nabízí určitou službu pod jménem a klient tétu 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.

6.1.3 Komunitní úroveň

Jedná se o zdroje, které umožňují různým skupinám vývojářů, vyměňovat si software a znalosti. Spadají sem distribuce, což jsou kolekce verzovaných stacků, které lze nainstalovat. Mají dost podobnou úlohu jako distribuce Linuxu, usnadňují tedy instalaci kolekce softwaru a následné udržování konzistentní verze. Dále sem patří repozitáře, kdy různé instituce mohou vyvíjet své vlastní softwarové komponenty robota a sdílet je tak pomocí federované sítě úložišť. Velmi důležitým zdrojem je *The ROS Wiki*, což je hlavní fórum pro dokumentaci informací o ROSu. Kdokoliv přihlášený na svůj účet tak může přidávat nové, či upravovat a aktualizovat již vzniklé dokumentace, psát návody a mnoho dalších věcí užitečných pro komunitu.

6.2 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 jsem tedy danou možnost také využil. Pro porovnání implementací jsem ale převážně používal své datasety získané jak ze simulace v ROS, tak i reálná data naměřená robotem *Dagu Wild Thumper* 6*WD*. Získání datasetu je pro porovnání implementací zásadním faktorem. Systémy SLAM mohou samozřejmě jednotlivě běžet na aktuálně simulovaných datech, pro porovnání jsou však potřebná data stejná.

Datasety jsou v ROS ukládány ve formátu *bag*, kde se jsou nahrány vybrané topiky. Pro každou porovnávanou implementaci jsem vytvořil *launch* soubory, které při zavolání spouštějí uvedené *nodes* se specifikovaným nastavením. Soubory *launch* jsem pro všechny systémy nastavil tak, aby potřebovali 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 mnou 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.

6.2.1 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 mé účely má však zbytečně veliké množství nastavení. Zvolil jsem 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.

Své simulační prostředí jsem rozložil do tří souborů. Dva s příponou *inc*, kdy první obsahuje informace pro načtení mapy. Jsou zde obsaženy informace o velikosti, frekvenci obnovování, výšku překážek v ose z a různá boolean nastavení. V druhém *inc* souboru se pak nachází informace o simulačním robotovi. Zvolil jsem trojúhleníkovou platformu, na které se nachází malý čtvercový blok simulující LIDAR. Je zde osaženo i jeho nastavení, jako je počet paprsků, úhel rozsahu a vzdálenost 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.

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. Osobně jsem si zvolil knihovnu *teleop_twist_keyboard*, která umožňuje navigaci robota pomocí klávesnice.

6.2.2 Nahrání reálných dat

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

Pro řízení kol je využito Arduino, pro zpracování dat a komunikaci je zde superpočítač 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 jsem využil možnost připojení bezdrátového ovladače *Xbox One Wireless Controller*. Přesto, že robot díky integrovanému počítači od NVIDIA má velmi vysoký výkon, tyto prostředky jsem nevyužil a data při běhu programu posílal rovnou do počítače, kde jsem je ukládal do souboru *bag*. Ke spojení byl využit Wi-Fi router *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 mé účely to však nakonec nebyl problém a vhodným umístěním routeru se mi podařilo naměřit data na dostatečném prostoru.

6.2.3 Vytažení 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 *topic* 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 *topicu* ukládat. Poslouchání *topicu* 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 nezaznamenává. Jednou z možností, jak ji tedy získat, je využít *node* obsažený v systému Hector SLAM. Přidáním *node* s názvem *hector_trajectory_server* do spouštěného souboru *launch* je tak získána možnost extrahovat trajektorii. Neboť je gMapping systém obsahující optimalizac cesty a mapy a umí uzavírání smyčky, je třeba trajektorii uložit až po doběhnutí dat obsažených v souboru *bag*. Takto se zavolání *topicu* 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 zskávání trajektorie je v případě systému Cartographer. Dříve se dala tato informace získat pomocí dvou příkazů. Aktualizací Cartographeru před dvěmi lety však zavedl Google pro ukládání dat nový formát *pbstream*, pro práci s nimž není příliš dobrá veřeně dostupná dokumentace. Cartographer při běhu publikuje *node* 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 (KUMMERLE - On Measuring the Accurancy of SLAM algorithms) by to bylo nedostatečné. Nalezl jsem nakonec soubor od vývojáře pro Google, který dokáže ze souboru *pbstream* udělat soubor *bag*, ve verzi Cartographeru spravované přímo Googlem ale tento soubor obsažen není. (*https* : *//code.research.uts.edu.au/*13110756*/cartographer_ros/blob/master/cartographer_ros/cartographer_r*
Využitím tohoto kódu jsem již byl schopen trajektorii získat a to následujícím postupem. Nejprve je třeba nechat proběhnout všechny výpočty z nahraného souboru *bag*, pro lepší orientaci si ho zde označím 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 *node* s názvem *pbstream_trajectories_to_bag*, přičemž první parametr je název souboru *pbstream* a parametr druhý je název souboru *bag*, do kterého se data uloží. Tento nový *bag* soubor nazvu jako *processed.bag*. Finálním

krokem je z *processed.bag* získání trajektorie, která je zde uložena v *topicu* $/tf$. 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 *topic*, ale *topic* z určeného souboru *bag*. Za to se pro uložení do vybraného souboru ještě přidá $> file.txt$. Výsledný příkaz je tedy v podobě:

$$rostopic\ echo\ /tf\ -b\ processed.bag > trajectory\_cartographer.txt$$

6.2.4 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.

6.3 Vyhodnocení výsledků

6.3.1 Srovnávací kritérium

Pro srovnání algoritmů SLAM jsem dle (Kummerle) zvolil porovnávací kritérium ve tvaru

$$\varepsilon(\delta) = \varepsilon_{trans}(\delta) + \varepsilon_{rot}(\delta), \tag{34}$$

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:

$$\varepsilon_{trans}(\delta) = \frac{1}{N}\Sigma_{i,j} trans(\delta_{i,j} \ominus \delta^*_{i,j})^2, \tag{35}$$

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 $trans()$ je určena ve formě Euklidovské normy:

$$trans(\delta_{i,j} \ominus \delta^*_{i,j}) = ||\delta_{i,j} \ominus \delta^*_{i,j}|| \tag{36}$$

Rovnice pro určení chyby natočení robota je ve tvaru:

$$\varepsilon_{rot}(\delta) = \frac{1}{N}\Sigma_{i,j} rot(\delta_{i,j} \ominus \delta^*_{i,j})^2, \tag{37}$$

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

$$rot(\delta_{i,j} \ominus \delta^*_{i,j}) = |min(2\pi - |\delta_{i,j} \ominus \delta^*_{i,j}|, |\delta_{i,j} \ominus \delta^*_{i,j}|)| \tag{38}$$

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.

6.3.2 Porovnání výsledků ze Stage

Pro porovnání výsledků, nahraných pomocí rozhraní Stage, jsem si vytvořil několik obrázkových map. Přesnost algoritmů byla u všech map přibližně stejná. Názorné porovnání jsem tedy udělal pouze na jedné, kde při průjezdu vznikly dvě smyčky. K porovnání jsem využil rovnice z předchozí sekce.

	Cartographer	**GMapping**	**Hector**
Chyba určení pozice robota [m]	0.00086	5.7225×10^{-5}	4.12745×10^{-5}
Chyba určení orientace robota [rad]	0.00062	4.45249×10^{-5}	0.00023
Chyba určení stavu robota	0.00149	0.00010	0.00027

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.

6.3.3 Porovnání výsledků nahraných robotem

Data jsem pro toto porovnání nahrával v pátém patře budovy Fakulty aplikovaných věd ZČU v části NTIS. Jak jsem již výše zmínil, 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 rodílů, typickým příkladem jsou chodby. GMapping bezpodmíněčně odometrickou informaci vyžaduje. Informace silně zkreslená tak průběh programu velmi komplikuje.

Při pohledu na vytvořené mapy je znát nedostatek s odometrií pouze u Cartographeru, kdy jedno projetí celého prostoru bylo nedostatečné a mapa není na pohled příliš kvalitní. Pro srovnání jsem však nahrál ještě nový dataset, kdy jsem projel daný prostor dvakrát a výsledek je znatelně lepší.

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* musím porovnat algoritmy pomocí získaných map. Měřením poměru vzdáleností mezi jednotivými body v prostoru a porovnáním s těmito body ve vytvořených mapách, vyhodnocuji pro tento případ jako nejlepší algoritmus Hector SLAM, v těsném závěsu poté gMapping.