Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra kybernetiky

Bakalářská práce

Simultánní lokalizace a mapování v budově

PLZEŇ, 2017 Petr Štrunc

Rozsah grafických prací: **dle potřeby**

Rozsah kvalifikační práce: **30-40 stránek A4**

Forma zpracování bakalářské práce: **tištěná**

Seznam odborné literatury:

Martinez, Aaron, and Enrique Fernández. Learning ROS for robotics programming. Packt Publishing Ltd, 2013.

Vedoucí bakalářské práce: **Ing. Petr Neduchal**
Nové technologie pro informační společnost

Datum zadání bakalářské práce: **1. listopadu 2016**

Termín odevzdání bakalářské práce: **21. května 2017**

Doc. RNDr. Miroslav Lávička, Ph.D.
děkan

L.S.

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 1. listopadu 2016

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 19. května 2017

.....

podpis

Poděkování

Mé poděkování patří Ing. Petru Neduchalovi za odborné vedení, trpělivost a ochotu, kterou mi v průběhu zpracování bakalářské práce věnoval.

Abstrakt

Práce se zabývá formulací úlohy Simultánní Lokalizace a Mapování a porovnáním několika implementací, které tuto úlohu řeší. Popisuje matematické základy ke čtyřem způsobům řešení dané úlohy a blíže seznamuje s principy třech implementací řešení. Implementace jsou testovány na jednotných datech a výsledky experimentů jsou následně vyhodnoceny.

Klíčová slova: SLAM, Kalmanův filtr, částicový filtr, Graph-Based SLAM, Hector SLAM, gMapping, Google Cartographer, ROS, LIDAR

Abstract

This thesis deals with formulation of Simultaneous Localization and Mapping problem and comparison of several implementations solving this problem. It describes mathematical principles to four approaches to solve this problem and describe principles of three implementations that solves this problem. Implementations are tested with unified data and results of the experiments are evaluated after that

Keywords: SLAM, Kalman filter, particle filter, Graph-Based SLAM, Hector SLAM, gMapping, Google Cartographer, ROS, LIDAR

Obsah

Použité značení

- $\boldsymbol{x}_k$: stavový vektor popisující aktuální polohu a natočení robota
- $\boldsymbol{u}_k$: vektor řízení aplikovaného v časovém okamžiku k-1, který převede robota do stavu $\boldsymbol{x}_k$ v časovém okamžiku k
- $\boldsymbol{m}_i$: vektor popisující pozici i-tého landmarku
- $\boldsymbol{z}_{ik}$: pozorování i-tého landmarku v časovém okamžiku k
- $\boldsymbol{X}_{0:k} = \{x_0, \boldsymbol{x}_1, \ldots, \boldsymbol{x}_k\} = \{\boldsymbol{X}_{0:k-1}, \boldsymbol{x}_k\}$: historie polohy robota
- $\boldsymbol{U}_{0:k} = \{\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_k\} = \{\boldsymbol{U}_{0:k-1}, \boldsymbol{u}_k\}$: historie aplikovaného řízení
- $\boldsymbol{m} = \{\boldsymbol{m}_1, \boldsymbol{m}_2, \ldots, \boldsymbol{m}_n\}$: množina všech landmarků
- $\boldsymbol{Z}_{0:k} = \{\boldsymbol{z}_1, \boldsymbol{z}_2, \ldots, \boldsymbol{z}_k\} = \{\boldsymbol{Z}_{0:k-1}, \boldsymbol{z}_k\}$: množina všech pozorování landmarků
- $\boldsymbol{x}_{k|k-1}$: stav robota po provedení kroku predikce do časového okamžiku k
- $\boldsymbol{x}_{k|k}$: stav robota po provedení korekce v časovém okamžiku k

Úvod

Pod názvem Simultánní Lokalizace a Mapování (tzv. SLAM) se skrývá problém, jehož řešení umožňuje robotu vytvářet mapu svého okolí a lokalizovat se v ní bez jakékoli předchozí znalosti prostoru, ve kterém se pohybuje.

První diskuze o tomto problému proběhly v roce 1986 při konání IEEE Robotics and Automation Conference v San Franciscu. Ucelená struktura SLAM problému, původně pojmenovaného SMAL (simultánní mapování a lokalizace) [13], se objevila poprvé v roce 1995 v rámci International Symposium on Robotics Research [1].

Úloh, ve kterých robot potřebuje mít přehled o svém okolí je mnoho a aplikaci různých řešení SLAM problému lze najít v mnoha oborech. Za zmínku dnes stojí především samořídící automobily od společností Google a Tesla. Mezi další populární aplikace patří mapování budov za pomoci autonomních vozidel (unmanned ground vehicle - UGV) nebo letounů (unmanned aerial vehicle - UAV), navigace ponorek kolem korálových útesů, průzkum jeskyní a opuštěných dolů nebo dokonce implementace při vývoji autonomního roveru pro průzkum povrchu planetárních těles. Avšak není třeba se pro příklady těchto algoritmů natolik vzdalovat od běžného života. Ve zjednodušených verzích je lze nalézt i ve vyspělejších autonomních vysavačích a sekačkách na trávu. SLAM má v navigaci několik výhod v porovnání s GPS, např. ke správné funkci nepotřebuje kontakt se satelity. Je tudíž možné s jeho pomocí navigovat ve stísněných prostorách bez signálu jako při výše zmíněném mapování budov nebo dolů. Oproti GPS ale neobsáhne celou planetu a při chybě senzorů nebo při špatném vyhodnocení výsledků měření může dojít k selhání systému.

Řešení SLAM problému přináší další pokrok v oblasti vývoje autonomních robotů, kteří by v budoucnu mohli asistovat lidem, nebo je i nahradit, v plnění úkolů v nebezpečném prostředí. Povaha těchto úkolů může být různá, od prohledávání sesunutých budov po zemětřesení, přes důlní operace až po asistenční činnosti v pořádkových složkách či armádě (např. průzkum, deaktivace bomb apod.). V mnoha vyjmenovaných odvětvích pracují již dnes, ale většinou se jedná pouze o roboty dálkově ovládané člověkem bez jakékoli formy autonomie.

Nejedná se o triviální problém, protože SLAM z podstaty umožňuje robotu pohybovat se skrz neznámé prostředí, ale taková navigace je možná pouze za předpokladu znalosti nějaké mapy okolního terénu. Mapování tohoto terénu může provést, jen pokud zná svou momentální polohu v prostoru a k určení své polohy potřebuje opět mapu, do které by ji zasadil. Řešení tohoto paradoxu bude vysvětleno v Kapitole 1.

Tato práce se zaměřuje na porovnání tří implementací řešení SLAM problému. Kapitola 1 popisuje SLAM problém a základní přístupy k jeho řešení. Probíranými způsoby řešení jsou Hector SLAM, GMapping a Google Cartographer. Kapitola 2 nabízí bližší rozbor těchto tří

implementací řešení. V Kapitole 3 se nacházejí výsledky experimentů provedených na těchto třech implementacích a jejich vyhodnocení.

1 Formulace SLAM

Jak bylo zmíněno, SLAM je úlohou mapování neznámého prostředí, při kterém robot využívá získávanou mapu k navigaci. Nejprve je vymezen pojem landmark potřebný pro následnou definici SLAM problému za pomoci pravděpodobnosti. Dále jsou popsány možné přístupy k řešení SLAM problému.

1.1 Landmark

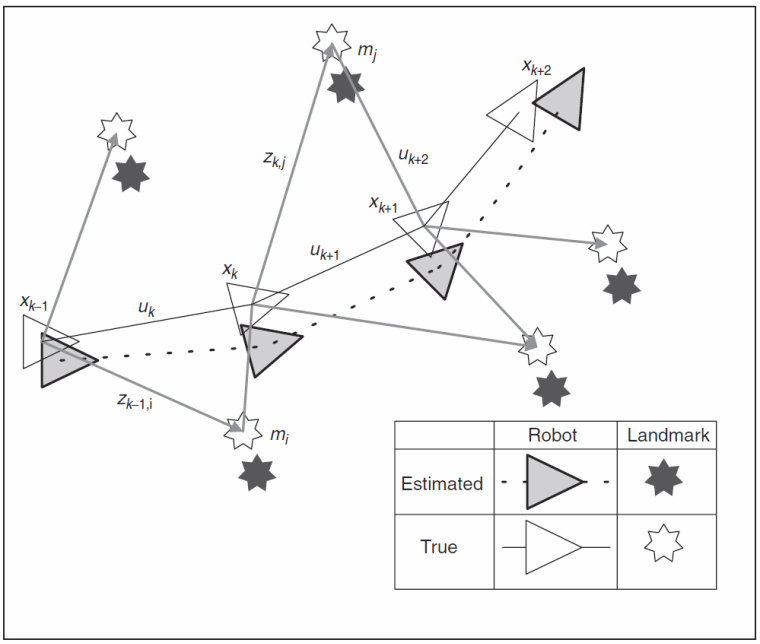
Landmarky jsou charakteristické body v prostředí, ve kterém se robot nachází. Landmark by měl být pro robota snadno rozpoznatelný z různých úhlů i vzdáleností. Podstatnou vlastností je též nezaměnitelnost s jiným landmarkem při jeho znovuobjevení. Ta lze zaručit např. umístěním nebo tvarem landmarku. Právě znovuobjevení landmarku slouží v úloze k určení aktuální polohy robota.

Prostředí by mělo mít dostatek landmarků, aby bylo možno lépe upřesnit polohu robota. Pokud se robot nalezne např. v prázdném rovném koridoru, který je delší než dosah jeho senzorů na každou stranu (aby se nemohl zachytit zdí na jeho koncích), pak výsledná mapa velmi pravděpodobně nebude správná. Robot se v takový moment nemá čeho zachytit, protože každý úsek koridoru pro něj vypadá stejně. V praxi to znamená, že délka koridoru zanesená v mapě bude výrazně kratší než délka skutečného koridoru.

Landmarky by měly zůstat statické po celou dobu mapování. Většina řešení SLAM není robustní vůči pohyblivým landmarkům a pohyb landmarků v takovém případě zapříčiní špatnou asociaci dat, kdy nalezený landmark bude zaměněn s jiným landmarkem z bezprostředního okolí. To může způsobit špatný odhad polohy robota a celý průběh algoritmu znehodnotit [13]. V obrázku 1.1 je blíže znázorněna funkce landmarků. Bílé landmarky značí skutečnou polohu landmarků a bílé polohy robota značí skutečné polohy robota v časech od k-1 do k+2. Tmavé landmarky a polohy robota značí polohy odhadnuté robotem. Šipky k landmarkům znázorňují měření provedené robotem v určitém časovém okamžiku.

1.2 Definice SLAM problému

Při reálných aplikacích je třeba počítat s nepřesností použitých měřících zařízení, a proto nejběžnější přístup popisující SLAM problém využívá pravděpodobnosti [4], např. Bayesovské filtrace. Řešením problému je nalezení sdružené aposteriorní hustoty pravděpodobnosti pozic



Obrázek 1.1: Rozdíl mezi odhadovanou a skutečnou polohou landmarků. Převzato z [1]

jednotlivých landmarků m a stavu robota x_k ve všech časových okamžicích k

$$P(\boldsymbol{x}_k, \boldsymbol{m} | \boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0). \tag{1.1}$$

Je dán počáteční stav robota $\boldsymbol{x}_0$ a je známé pozorování $\boldsymbol{Z}_{0:k}$ a řízení $\boldsymbol{U}_{0:k}$ od počátku až do časového okamžiku k. K výpočtu lze použít rekurentní algoritmus, kterým je možné získat hustotu pravděpodobnosti v časovém okamžiku k pomocí hustoty vypočtené v časovém okamžiku k-1, jejíž tvar je

$$P(\boldsymbol{x}_{k-1}, \boldsymbol{m} | \boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k-1}) \tag{1.2}$$

V této fázi je vhodné definovat model pozorování a pohybový model.

- **Model pozorování** popisuje, s jakou pravděpodobností může robot zaznamenat pozorování z_k za předpokladu, že polohy landmarků a robota jsou známé.

$$P(\boldsymbol{z}_k | \boldsymbol{x}_k, \boldsymbol{m}) \tag{1.3}$$

- **Pohybový model** popisuje, s jakou pravděpodobností se v časovém okamžiku k robot nachází ve stavu $\boldsymbol{x}_k$ za předpokladu, že předchozí stav $\boldsymbol{x}_{k-1}$ je známý a bylo aplikováno řízeni $\boldsymbol{u}_k$.

$$P(\boldsymbol{x}_k | \boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \tag{1.4}$$

  Pohybový model závisí pouze na předchozím stavu robota $\boldsymbol{x}_{k-1}$ a aplikovaném řízení $\boldsymbol{u}_k$. Jedná se tedy o Markovský proces.

Dvoukrokový rekurentní algoritmus je definován:

1) Predikce

$$P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) = \int P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \times P(\boldsymbol{x}_{k-1}, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k-1}, \boldsymbol{x}_0) d\boldsymbol{x}_{k-1} \quad (1.5)$$

2) Korekce

$$P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) = \frac{P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m}) P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)}{P(\boldsymbol{z}_k|\boldsymbol{Z}_{0:k-1}, \boldsymbol{U}_{0:k})} \quad (1.6)$$

Rovnice (1.5) a (1.6) poskytují rekurzivní postup pro výpočet hustoty pravděpodobnosti $P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)$ stavu robota x_k a mapy m v čase k za použití všech předchozích pozorování $\boldsymbol{Z}_{0:k}$ a řízení $\boldsymbol{U}_{0:k}$.

Pro zjednodušení následujícího textu bude sdružená aposteriorní hustota pozic landmarků a robota $P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0)$ zapsána ve tvaru $P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{z}_k)$.

Problém mapování lze formulovat též pomocí podmíněné hustoty pravděpodobnosti ve tvaru $P(\boldsymbol{m}|\boldsymbol{z}_k)$ v případě, že pozice robota je známá. Analogicky lze vyjádřit i problém lokalizace jako $P(\boldsymbol{x}_k|\boldsymbol{z}_k)$ za předpokladu, že mapa prostředí je známa.

Avšak model pozorování $P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{m})$ je závislý jak na stavu robota $\boldsymbol{x}_k$, tak na mapě $\boldsymbol{m}$, z čehož vyplývá, že pravděpodobnostní hustota $P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{z}_k)$ nelze v tomto případě rozdělit takovýmto způsobem

$$P(\boldsymbol{x}_k, \boldsymbol{m}|\boldsymbol{z}_k) \neq P(\boldsymbol{x}_k|\boldsymbol{z}_k) P(\boldsymbol{m}|\boldsymbol{z}_k),$$

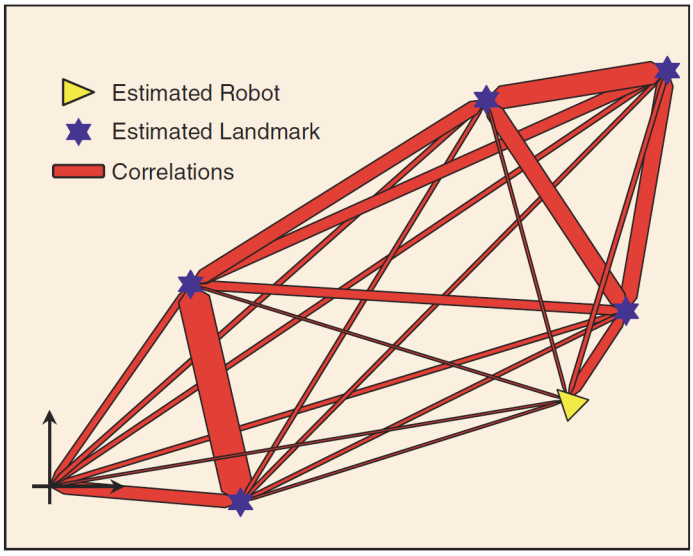
což svědčí o neoddělitelnosti lokalizace a mapování při řešení SLAM problému.

Z obrázku 1.1 vyplývá, že skutečná a předpokládaná pozice landmarků se liší. Příčinou tohoto jevu je nepřesná znalost polohy robota v momentě, kdy provádí měření. Poloha robota se v takovém okamžiku odhaduje na základě předchozí polohy $\boldsymbol{x}_{k-1}$ a dat z odometrie, která je většinou méně přesná, než je přesnost použitého senzoru. Vzdálenosti mezi odhadovanou polohou robota a landmarky tedy bývají naměřeny s mnohem menší chybou, než jaká je chyba odhadu polohy robota. Velkou část chyby odhadu polohy landmarku tedy tvoří chyba odhadu polohy robota a takové chování naznačuje, že chyby v odhadu polohy landmarků jsou vysoce korelované a že vzájemná poloha dvou landmarků je známa s vysokou přesností, a to nezávisle na tom, jak přesná je znalost jejich skutečné polohy. Z pohledu pravděpodobnosti to znamená, že rozptyl sdružené hustoty pravděpodobnosti $P(m_i, m_j)$ je velmi malý i v případě, kdy je marginální hustota pravděpodobnosti $P(m_i)$ značně rozptýlená.

Zvyšováním korelace se zlepšuje odhad relativní polohy landmarků. Korelace roste s každým novým pozorováním a tím se tedy stále zlepšuje i znalost vzájemných poloh landmarků. Z pravděpodobnostního hlediska se rozptyl sdružené hustoty pravděpodobnosti poloh všech landmarků $P(\mathbf{m})$ s každým novým pozorováním zmenšuje.

Při pohledu na obrázek 1.1 se robot v okamžiku k nachází na pozici x_k a pozoruje landmarky m_i a m_j. Naměřená relativní vzdálenost těchto dvou landmarků je nezávislá na tom, kde se robot v časovém okamžiku k nachází, tedy pokud jsou dané landmarky stále v dosahu měřícího aparátu a robot je může zaznamenat. V časovém okamžiku $k+1$ se robot posune na pozici x_{k+1}. Zde pomocí senzorů zachytí už jen landmark m_j a pomocí naměřených dat upraví odhad své polohy i polohy landmarku m_j. Změna odhadu polohy m_j pak zapříčiní i změnu odhadu polohy nyní nepozorovaného landmarku m_i. Toto neplatí např. pro FastSLAM, který nevyužívá vzájemné závislosti landmarků.

Pro lepší představu lze příslušné korelace zobrazit do mapy jakožto obdobu mechanického systému, ve kterém jsou mezi landmarky umístěny pružiny, viz obrázek 1.2. S každým provedeným měřením korelace mezi pozorovanými landmarky rostou a tím roste i tuhost pružin, které jsou na nich ukotveny. Pokud je pak upraven odhad polohy jednoho landmarku, tak se tato změna projeví na všech ostatních landmarcích, které jsou s ním propojeny.



Obrázek 1.2: Zobrazení korelací mezi landmarky. Převzato z [1]

Přístupy k řešení SLAM problému lze rozdělit do dvou skupin. Online metody (též známé jako filtrační metody) odhadují stav robota a mapy v reálném čase. Lze tak v každém okamžiku zjistit odhad polohy robota a podoby jeho okolí, avšak chyby v určení stavu systému většinou nelze opravit. Offline metody (též známe jako vyhlazovací metody) odhadují celou trajektorii robota za pomoci všech získaných pozic robota a měření od jejich spuštění. Získané pozice robota a měření lze též zobrazovat v reálném čase, ale trajektorie robota a mapa nabývají platnosti až po provedení jejich optimalizace, jejíž součástí je většinou metoda nejmenších čtverců [6, 4]. Dále jsou prezentovány tři online metody a jedna offline metoda řešení SLAM problému.

1.3 EKF SLAM

Nejstarší ze způsobů řešení SLAM problému využívá rozšířený Kalmanův filtr (Extended Kalman filter - EKF) k odhadu polohy robota na základě dat získaných z odometrie a pozorování landmarků. Popis EKF SLAMu vychází z [1, 13, 8]. Kalmanův filtr je speciální případ Bayesova filtru pro lineární funkce s Gaussovským rozdělením pravděpodobnosti. Funkce použité pro řešení SLAM problému nejsou lineární, proto je třeba využít EKF, který tyto funkce linearizuje Taylorovým rozvojem prvního stupně. Těmito funkcemi jsou pohybový model a model pozorování.

Pohybový model popisuje translaci robota ve tvaru

$$P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) \iff \boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k, \tag{1.7}$$

kde $\boldsymbol{f}()$ je nelineární funkce modelující pohyb robota. Výsledkem $\boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k)$ je poloha robota v čase k určená ze znalosti jeho polohy v čase k-1 a aplikovaného řízení $\boldsymbol{u}_k$ včetně chyby odometrie $\boldsymbol{w}_k$. Matice $\boldsymbol{w}_k$ je náhodná veličina s normálním rozdělením s nulovou střední hodnotou a kovariancí $\boldsymbol{Q}_k$, která reprezentuje chybu odometrie.

Model pozorování je ve tvaru

$$P(\boldsymbol{z}_k|\boldsymbol{x}_{k|k-1}, \boldsymbol{m}_k) \iff \boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_{k|k-1}, \boldsymbol{m}_k) + \boldsymbol{v}_k, \tag{1.8}$$

kde $\boldsymbol{h}()$ je nelineární funkce modelující pozorování. Výsledkem $\boldsymbol{h}(\boldsymbol{x}_{k|k-1}, \boldsymbol{m}_k)$ je poloha pozorovaného landmarku relativně k odhadované poloze robota, $\boldsymbol{v}_k$ je náhodná veličina s normálním rozdělením s nulovou střední hodnotou a kovariancí $\boldsymbol{R}_k$ zastupující chybu měření.

Pohybový model i model pozorování jsou zde vyjádřeny pomocí nelineární funkce, což znemožňuje použití Kalmanova filtru, který operuje jen s lineárními funkcemi. Rozšířený Kalmanův filtr pracuje i s nelineárními funkcemi, které linearizuje.

1.3.1 Použité matice

Pro výpočet odhadu polohy robota a mapy je využito několik matic. Hlavním úkolem EKF je nalézt střední hodnotu a kovarianci sdružené aposteriorní hustoty pravděpodobnosti (1.1). Rozdělení pravděpodobnosti s touto hustotou je Gaussovského charakteru a jeho střední hodnota udává nejpravděpodobnější stav systému. Tato práce se zabývá algoritmy, které řeší SLAM problém ve dvou dimenzích. Níže popsané matice odpovídají tomuto prostoru a jejich popis vychází z [13]

Stav systému $\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{bmatrix}$

Ve stavu systému je uložen stav robota a všechny zaznamenané landmarky v čase k. Jeho dvě složky mají následující tvar

$$\hat{\boldsymbol{x}}_{k|k} = [x_r, y_r, \theta_r]^T \qquad \hat{\boldsymbol{m}}_k = [x_1, y_1, x_2, y_2, ..., x_n, y_n]^T, \tag{1.9}$$

kde x_r, y_r a θ_r vyjadřují pozici robota v souřadném systému a jeho úhel natočení a každá dvojice x_i, y_i ve vektoru $\hat{\boldsymbol{m}}_k$ popisuje polohu i-tého landmarku. Rozměry stavového vektoru jsou tedy (3+2·n)×1, kde n je počet získaných landmarků od inicializace do časového okamžiku k.

Kovarianční matice $\boldsymbol{P}_{k|k}$

Matice $\boldsymbol{P}_{k|k}$ Uchovává neurčitosti v pozici robota a landmarků a ve vzájemných polohách robota a landmarků i mezi jednotlivými landmarky. Její rozměry jsou (3+2·n)×(3+2·n) pro n nalezených landmarků a lze ji rozdělit na 4 části

$$\boldsymbol{P}_{k|k} = \begin{bmatrix} \boldsymbol{P}_{xx} & \boldsymbol{P}_{xm} \\ \boldsymbol{P}_{xm}^T & \boldsymbol{P}_{mm} \end{bmatrix}_{k|k}, \tag{1.10}$$

kde $\boldsymbol{P}_{xx}$ je matice o rozměrech 3×3 a obsahuje kovarianci pozice robota, $\boldsymbol{P}_{xm}$ je matice o rozměrech 3×2n. $\boldsymbol{P}_{xm}$ se skládá z bloků o rozměrech 3×2 seřazených vedle sebe a každý z nich uchovává informaci o kovarianci mezi robotem a jedním landmarkem. $\boldsymbol{P}_{mm}$ je matice o rozměrech 2n× 2n a skládá se z bloků o rozměrech 2×2, kde každý blok nese kovarianci mezi dvěma landmarky a každý blok na diagonále obsahuje kovarianci jednoho landmarku. $\boldsymbol{P}_{k|k}$ se s každým nalezeným landmarkem rozšíří o dva řádky a dva sloupce, což značně zvyšuje paměťovou a výpočetní náročnost EKF SLAM.

Jakobián pohybového modelu $\nabla \boldsymbol{f}$

Tvar pohybového modelu je

$$\boldsymbol{x}_k = \boldsymbol{f}(\boldsymbol{x}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{w}_k = \begin{bmatrix} x_{r,k-1} + \Delta t \cdot cos\theta_r + w\Delta t \cdot cos\theta_r \\ y_{r,k-1} + \Delta t \cdot sin\theta_r + w\Delta t \cdot sin\theta_r \\ \theta_{r,k-1} + \Delta\theta_r + w\Delta\theta_r \end{bmatrix}, \tag{1.11}$$

kde Δt je změna polohy způsobená aplikovaným řízením, $\Delta t \cdot cos\theta_r$, $\Delta t \cdot sin\theta_r$ a $\Delta\theta_r$ vyjadřují změnu polohy a natočení robota mezi časovými okamžiky k-1 a k a w vyjadřuje chybu odometrie. $\Delta t \cdot cos\theta_r$ a $\Delta t \cdot sin\theta_r$ lze též označit jako Δx_r a Δy_r.

Jakobián je matice složená z parciálních derivací funkce, jejíž vstupem je vektor. Definuje lineární aproximaci dané funkce v blízkém okolí určitého bodu. Pro funkci $\boldsymbol{b} = \boldsymbol{e}(\boldsymbol{z})$ o n vstupech a m výstupech má jakobián následný tvar

$$\boldsymbol{e}(\boldsymbol{z}) = \begin{bmatrix} \boldsymbol{e}_1(\boldsymbol{z}) \\ \boldsymbol{e}_2(\boldsymbol{z}) \\ \vdots \\ \boldsymbol{e}_m(\boldsymbol{z}) \end{bmatrix} \qquad \nabla \boldsymbol{e}(\boldsymbol{z}) = \begin{bmatrix} \frac{\delta e_1}{\delta z_1} & \frac{\delta e_1}{\delta z_2} & \cdots & \frac{\delta e_1}{\delta z_n} \\ \frac{\delta e_2}{\delta z_1} & \frac{\delta e_2}{\delta z_2} & \cdots & \frac{\delta e_2}{\delta z_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\delta e_m}{\delta z_1} & \frac{\delta e_m}{\delta z_2} & \cdots & \frac{\delta e_m}{\delta z_n} \end{bmatrix} \tag{1.12}$$

EKF pracuje s linearizovanými funkcemi a linearizovaný tvar funkce f v daném bodě je reprezentován jakobiánem ∇f

$$\nabla f = \begin{bmatrix} 1 & 0 & -\Delta t \cdot sin\theta_r \\ 0 & 1 & \Delta t \cdot cos\theta_r \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -\Delta y_r \\ 0 & 1 & \Delta x_r \\ 0 & 0 & 1 \end{bmatrix}. \tag{1.13}$$

Jakobián modelu pozorování $\nabla \boldsymbol{h}$

Model pozorování ve dvou dimenzích je ve tvaru

$$\boldsymbol{z}_k = \boldsymbol{h}(\boldsymbol{x}_{k|k-1}, \boldsymbol{m}_k) + \boldsymbol{v}_k = \begin{bmatrix} \sqrt{(\lambda_x - x_r)^2 + (\lambda_y + y_r)^2} + v_v \\ tan^{-1}(\frac{\lambda_y - y_r}{\lambda_x - x_r}) - \theta_r + v_\theta \end{bmatrix}, \tag{1.14}$$

kde λ_x a λ_y udávají pozici landmarku, pro který je pozorování právě modelováno a x_r, y_r a θ_r je pozice robota z $\boldsymbol{x}_{k|k-1}$. $\boldsymbol{v}_k$ je zde rozdělena na dvě složky: v_v, udávající chybu v měření vzdálenosti, a v_θ udávající chybu v měření úhlu. Hodnota v prvním řádku vektoru udává předpokládanou vzdálenost mezi robotem a landmarkem a druhý řádek udává úhel od osy robota k landmarku.

Při použití EKF pro lokalizaci je matice ∇h v podobě

$$\nabla h_{lok} = \begin{bmatrix} \frac{x_r - \lambda_x}{v} & \frac{y_r - \lambda_y}{v} & 0 \\ \frac{x_r - \lambda_x}{v^2} & \frac{y_r - \lambda_y}{v^2} & -1 \end{bmatrix}, \tag{1.15}$$

kde x, y a θ značí polohu a orientaci robota, λ_x a λ_y polohu landmarku, pro který je jakobián právě počítán, a v značí vzdálenost mezi robotem a landmarkem bez započtení chyby. Pro řešení SLAM je třeba rozšířit matici o informace týkající se landmarku, pro který je právě

model pozorování počítán. Matice ∇h je pak ve tvaru

$$\nabla h = \begin{bmatrix} \overbrace{\begin{matrix} \frac{x_r-\lambda_x}{v} & \frac{y_r-\lambda_y}{v} & 0 & 0 & 0 \\ \frac{x_r-\lambda_x}{v^2} & \frac{y_r-\lambda_y}{v^2} & 1 & 0 & 0 \end{matrix}}^{\nabla h_{lok}} & \begin{matrix} & x_1 & y_1 \\ & & \end{matrix} & \begin{matrix} x_2 \\ -\frac{x_r-\lambda_x}{v} \\ -\frac{x_r-\lambda_x}{v^2} \end{matrix} & \begin{matrix} y_2 \\ -\frac{y_r-\lambda_y}{v} \\ -\frac{y_r-\lambda_y}{v^2} \end{matrix} & \begin{matrix} x_3 & y_3 \\ 0 & 0 \\ 0 & 0 \end{matrix} & \begin{matrix} \\ \cdots \\ \cdots \end{matrix} \end{bmatrix} \tag{1.16}$$

a její rozměry jsou $2\times(3+2\cdot n)$ pro n zaznamenaných landmarků, kde první tři sloupce reprezentují předchozí matici ∇h_{lok}, a pro právě počítaný landmark jsou vyplněny dva příslušné sloupce po vzoru (1.16). V tomto případě se jedná o matici ∇h pro druhý landmark. Zbytek hodnot je nulový.

Kovarianční matice chyby pohybového modelu $\boldsymbol{Q}_k$

Každé provedené měření v sobě skrývá nějakou chybu zapříčiněnou nepřesností meřícího zařízení, případně nějakým vnějším vlivem. Jako příklad poslouží robot s kolovým podvozkem, který odhaduje svou polohu podle počtu otočení kol. Pokud se kolo robota protočí, nebo sklouzne stranou, pak může dojít k nárůstu chyby odometrie. Tyto chyby jsou důvodem, proč se v (1.7) vyskytuje $\boldsymbol{w}_k$. Již bylo řečeno, že $\boldsymbol{w}_k$ je chyba odometrie zastoupená normálním rozdělením s nulovou střední hodnotou. Kovarianční matice této chyby je

$$\boldsymbol{Q}_k = \boldsymbol{W}\cdot\boldsymbol{C}\cdot\boldsymbol{W}^T = \begin{bmatrix} c\Delta x^2 & 0 & 0 \\ 0 & c\Delta y^2 & 0 \\ 0 & 0 & c\Delta\theta^2 \end{bmatrix}, \tag{1.17}$$

kde $\boldsymbol{W} = [\Delta x, \Delta y, \Delta\theta]$ a $\boldsymbol{C}$ obsahuje škálovací parametry získané pomocí experimentálních měření.

Kovarianční matice chyby modelu pozorování $\boldsymbol{R}_k$

V modelu pozorování se též počítá s chybou měřícího zařízení $\boldsymbol{v}_k$. Kovarianční matice této chyby má následující tvar

$$R_k = \begin{bmatrix} v\cdot c & 0 \\ 0 & b\cdot d \end{bmatrix}, \tag{1.18}$$

kde v je vzdálenost mezi robotem a landmarkem a c je konstanta určující chybu měření vzdálenosti. Při měření vzdálenosti obyčejně s vyšší vzdáleností roste chyba měření, tudíž je vhodné nastavit c tak, aby odráželo chybu pro měření na jednotku vzdálenosti, která je při výpočtu použita. Pokud je naměřená vzdálenost udávána v metrech a chyba měření je 1 cm na metr, pak by parametr c měl být nastaven na 0.01. Hodnota b udává úhel od osy robota k landmarku a d reprezentuje chybu v měření úhlu. b a d spolu reprezentují chybu v měření úhlu. Při měření úhlu nemá smysl škálovat chybu spolu s velikostí naměřeného úhlu, protože

velikost chyby při naměření úhlu 4° se pohybuje ve stejných hodnotách jako při naměření úhlu 70°. Dává tedy smysl pozici obsahující $b \cdot d$ nahradit konstantou stejnou, jako je maximální chyba v měření úhlu.

Kovarianční matice inovace $\boldsymbol{S}_k$

Matice S_k popisuje kovarianci inovace. Inovace značí rozdíl mezi odhadem polohy robota získaného z dat naměřených senzory a odhadem polohy robota na základě dat z odometrie. Používá se při výpočtu Kalmanova zisku a lze ji vypočítat následnou rovnicí

$$\boldsymbol{S}_k = \nabla \boldsymbol{h} \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T + \boldsymbol{R}_k \tag{1.19}$$

Kalmanův zisk $\boldsymbol{W}_k$

Po predikčním kroku je známý odhad polohy robota z odometrie, na jejímž základu jsou propočteny očekávané polohy landmarků pomocí modelu pozorování. Po provedení pozorování je získán odhad polohy robota z pozorování, který se liší od odhadu polohy z odometrie. Tento rozdíl se nazývá inovace. Jednou z částí korekčního kroku je výpočet Kalmanova zisku, který udává míru důvěry v data naměřená při pozorování. Rozměry matice jsou $(3{+}2{\cdot}n){\times}2$ pro n landmarků a lze ji získat výpočtem

$$\boldsymbol{W}_k = \boldsymbol{P}_{k|k-1} \nabla \boldsymbol{h}^T \boldsymbol{S}_k^{-1} = \begin{bmatrix} x_{r,v} & x_{r,b} \\ y_{r,v} & y_{r,b} \\ \theta_{r,v} & \theta_{r,b} \\ x_{1,v} & x_{1,b} \\ y_{1,v} & y_{1,b} \\ \vdots & \vdots \\ x_{n,v} & x_{n,b} \\ y_{n,v} & y_{n,b} \end{bmatrix}, \tag{1.20}$$

kde první index proměnných udává, zda se jedná o hodnotu ovlivňující stav robota nebo landmarku (r pro robota a číslo pro landmark) a druhý index udává, zda se jedná o hodnotu ovlivňující pozici nebo natočení dané proměnné (v pro pozici a b pro natočení). První a druhý řádek obsahuje koeficienty, kterými bude přenásobena inovace polohy robota, v třetím řádku jsou hodnoty pro přenásobení inovace natočení robota a každý další blok $2{\times}2$ nese koeficienty pro přenásobení inovace jednoho landmarku. První sloupec obsahuje koeficienty ovlivňující vzdálenost, druhý sloupec obsahuje koeficienty ovlivňující natočení. Např. hodnota prvního řádku a prvního sloupce značí kolik se z inovace promítne do x_r ve smyslu polohy a hodnota druhého řádku a druhého sloupce značí kolik se z inovace promítne do y_r ve smyslu natočení.

Velikost hodnot matice Kalmanova zisku závisí na kvalitě odometrie robota a snímacího zařízení, kterým měří polohy landmarků. Pokud bude odometrie robota mnohem přesnější než data z měřícího zařízení, pak budou hodnoty matice malé a naopak.

Jakobián predikce landmarku $\boldsymbol{J}_{xr}$

Matice $\boldsymbol{J}_{xr}$ je jakobián použitý při rozšiřování kovarianční matice $\boldsymbol{P}_{k|k}$ o záznam nově objeveného landmarku. Podobá se jakobiánu pohybového modelu $\nabla \boldsymbol{f}$, konkrétně se jedná o jeho první dva řádky, protože záznam landmarku neuchovává informace o natočení. matice $\boldsymbol{J}_{xr}$ je jakobián predikce landmarku vzhledem k poloze robota v souřadném systému $[x_r, y_r, \theta_r]$.

$$\boldsymbol{J}_{xr} = \begin{bmatrix} 1 & 0 & -\Delta x \\ 0 & 1 & -\Delta y \end{bmatrix} \tag{1.21}$$

Jakobián predikce měření landmarku $\boldsymbol{J}_z$

$\boldsymbol{J}_z$ je jakobián využitý při výpočtu kovariance landmarku do kovarianční matice $\boldsymbol{P}_{k|k}$ rozšířené o nový landmark. Má podobnou funkci jako $\boldsymbol{J}_{xr}$, jen není počítán vzhledem k poloze robota v $[x_r, y_r, \theta_r]$, ale ve smyslu vzdálenosti od robota a úhlu od jeho osy k landmarku.

$$\boldsymbol{J}_z = \begin{bmatrix} cos(\theta_r + \Delta\theta_r) & -\Delta t \cdot sin(\theta_r + \Delta\theta_r) \\ sin(\theta_r + \Delta\theta_r) & \Delta t \cdot cos(\theta_r + \Delta\theta_r) \end{bmatrix}, \tag{1.22}$$

kde Δt je změna polohy způsobená aplikovaným řízením mezi kroky k-1 a k (obdobně jako v (1.11)).

1.3.2 Predikce

V první části algoritmu dochází k odhadu střední hodnoty a kovariance sdružené aposteriorní hustoty pravděpodobnosti (1.1) za pomoci pohybového modelu

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{f}(\hat{\boldsymbol{x}}_{k-1|k-1}, \boldsymbol{u}_k) = \begin{bmatrix} \boldsymbol{x_{r,k-1}} + \Delta t \cdot cos\theta_r \\ \boldsymbol{y_{r,k-1}} + \Delta t \cdot sin\theta_r \\ \boldsymbol{\theta_{r,k-1}} + \Delta\theta_r \end{bmatrix}, \tag{1.23}$$

kde $\hat{\boldsymbol{x}}_{k|k-1}$ je odhad polohy robota pro časový okamžik k vycházející z jeho polohy v čase k-1 a dat z odometrie po aplikaci řízení $\boldsymbol{u}_k = [\Delta t \cdot cos\theta_r, \Delta t \cdot sin\theta_r, \Delta\theta_r]$ a $\hat{\boldsymbol{x}}_{k-1|k-1}$ je poloha robota z časového okamžiku k-1. Dále je vypočten $\nabla \boldsymbol{f}$ dle (1.13) a kovariance chyby pohybového

modelu Q_k

$$\boldsymbol{Q}_k = \begin{bmatrix} c\Delta x^2 & c\Delta x\Delta y & c\Delta x\Delta\theta \\ c\Delta y\Delta x & c\Delta y^2 & c\Delta y\Delta\theta \\ c\Delta\theta\Delta x & c\Delta\theta\Delta y & c\Delta\theta^2 \end{bmatrix}, \tag{1.24}$$

kde c jsou škálovací parametry získané pomocí experimentálních měření. Kovarianční matici $\boldsymbol{P}$ je třeba přepočítat po tom, co robot změnil polohu. Nejprve je přepočítána kovariance robota $\boldsymbol{P}_{xx}$

$$\boldsymbol{P}_{xx,k|k-1} = \nabla \boldsymbol{f} \boldsymbol{P}_{xx,k-1|k-1} \nabla \boldsymbol{f}^T + \boldsymbol{Q}_k, \tag{1.25}$$

kde $\boldsymbol{P}_{xx,k|k-1}$ je kovariance robota pro jeho polohu $\hat{\boldsymbol{x}}_{k|k-1}$ a $\boldsymbol{P}_{xx,k-1|k-1}$ je kovariance robota z časového okamžiku k-1. Poté jsou vypočteny nové kovariance robot-landmark $\boldsymbol{P}_{xm}$

$$\boldsymbol{P}_{xm,k|k-1} = \nabla \boldsymbol{f} \boldsymbol{P}_{xm,k-1|k-1}, \tag{1.26}$$

kde $\boldsymbol{P}_{xm,k|k-1}$ je matice kovariancí mezi robotem na pozici $\hat{\boldsymbol{x}}_{k|k-1}$ a landmarky a $\boldsymbol{P}_{xm,k-1|k-1}$ je kovariance mezi robotem a landmarky z časového okamžiku k-1. V kovarianční matici $\boldsymbol{P}$ se vyskytuje blok $\boldsymbol{P}_{xm}^T$, který je naplněn právě vypočtenou maticí.

1.3.3 Korekce

Druhý krok algoritmu upřesňuje odhad z predikce a zanáší do mapy nové landmarky. Začíná měřením okolí robota a extrakcí landmarků. Poté je pro každý landmark vypočtena předpokládaná vzdálenost a natočení od robota za pomoci modelu pozorování (1.14) a asociací dat je určeno, které landmarky již byly viděny a které jsou objeveny poprvé. Pro každý znovuobjevený landmark je vypočten $\nabla \boldsymbol{h}$ z (1.16) a kovariance chyby měření $\boldsymbol{R}_k$ dle (1.18).

Po nashromáždění těchto informací lze přejít k výpočtu kovarianční matice inovace $\boldsymbol{S}_k$ z (1.19), Kalmanova zisku $\boldsymbol{W}_k$ z (1.20) a výpočtu odhadu pozice robota a landmarků pro časový okamžik k

$$\begin{bmatrix} \hat{\boldsymbol{x}}_{k|k} \\ \hat{\boldsymbol{m}}_k \end{bmatrix} = \begin{bmatrix} \hat{\boldsymbol{x}}_{k|k-1} \\ \hat{\boldsymbol{m}}_{k-1} \end{bmatrix} + \boldsymbol{W}_k \left[\boldsymbol{z}_k - \boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1}) \right], \tag{1.27}$$

kde $\hat{\boldsymbol{x}}_{k|k}$ je odhad polohy robota pro časový okamžik k, $\boldsymbol{z}_k$ je měření v čase k a $\boldsymbol{h}(\hat{\boldsymbol{x}}_{k|k-1}, \hat{\boldsymbol{m}}_{k-1})$ reprezentuje relativní polohu landmarků z času k-1 k poloze robota $\hat{\boldsymbol{x}}_{k|k-1}$. Tento vztah poskytuje optimální výsledek vzhledem k MMSE (Minimum Mean Square Error) [15]. Posledním krokem pro známé landmarky pak je výpočet nové kovarianční matice

$$\boldsymbol{P}_{k|k} = \boldsymbol{P}_{k|k-1} - \boldsymbol{W}_k \boldsymbol{S}_k \boldsymbol{W}_k^T, \tag{1.28}$$

kde $\boldsymbol{P}_{k|k}$ je kovarianční matice pro časový okamžik k. Pokud byly nalezeny nové landmarky, pak pro každou z nich je doplněn vektor $\hat{\boldsymbol{m}}_k$ o jejich souřadnice

$$\hat{\boldsymbol{m}}_k = \begin{bmatrix} \hat{\boldsymbol{m}}_k \\ x_{n+1} \\ y_{n+1} \end{bmatrix}, \tag{1.29}$$

kde x_{n+1} a y_{n+1} jsou souřadnice nového landmarku, a pro tyto landmarky je vypočten $\boldsymbol{J}_{xr}$ z (1.21) a $\boldsymbol{J}_z$ z (1.22). Poté je kovarianční matice $\boldsymbol{P}$ rozšířena o dva řádky a dva sloupce a doplněna o kovariance landmarků

$$\boldsymbol{P}_{k|k,n+1,n+1} = \boldsymbol{J}_{xr}\boldsymbol{P}_{\boldsymbol{xx},\boldsymbol{k}|\boldsymbol{k}}\boldsymbol{J}_{xr}^T + \boldsymbol{J}_z\boldsymbol{R}_k\boldsymbol{J}_z^T, \tag{1.30}$$

kde $\boldsymbol{P}_{k|k,n+1,n+1}$ je nově vzniklý blok o rozměrech 2×2 na diagonále. Dále je přidána kovariance mezi robotem a landmarkem

$$\boldsymbol{P}_{k|k,1,n+1} = \boldsymbol{P}_{\boldsymbol{xx}}\boldsymbol{J}_{xr}^T, \tag{1.31}$$

kde $\boldsymbol{P}_{k|k,1,n+1}$ je blok o rozměrech 3×2, který vznikl na konci prvních dvou řádků matice po jejím rozšíření. Její transpozice je doplněna i na místo průniku sloupců hodnot robota a řádek nového landmarku.

$$\boldsymbol{P}_{k|k,n+1,1} = \boldsymbol{P}_{k|k,1,n+1}^T. \tag{1.32}$$

Poslední dvě operace zahrnují doplnění kovariancí mezi nově vzniklým landmarkem a již známými landmarky

$$\boldsymbol{P}_{k|k,n+1,i} = \boldsymbol{J}_{xr}\boldsymbol{P}_{k|k,1,i}^T, \tag{1.33}$$

kde $\boldsymbol{P}_{k|k,n+1,i}$ je kovariance mezi i-tým landmarkem a nově vzniklým landmarkem a $\boldsymbol{P}_{k|k,1,i}$ je kovariance mezi i-tým landmarkem a robotem. Tímto je vyplněn nově vzniklý řádek. Doplněním transpozice těchto kovariancí do posledních volných míst v nově vzniklém sloupci je kovarianční matice zaplněna

$$\boldsymbol{P}_{k|k,i,n+1} = \boldsymbol{P}_{k|k,n+1,i}^T. \tag{1.34}$$

Zde se k posune na k+1 a proces se opakuje.

1.3.4 Klíčové vlastnosti EKF SLAM

Toto řešení SLAM problému je postaveno na lokalizaci pomocí EKF a sdílí s ní velkou část postupu. Předním pozitivem EKF SLAMu je jeho schopnost s každým měřením upřesnit vzájemnou polohu landmarků v celé mapě. Determinant matice $\boldsymbol{P}_{mm,k}$ má tendenci s přibývajícím časem monotónně konvergovat k nule, což se odráží na neurčitosti v poloze landmarků, která s přibývajícím časem klesá k určité nenulové hranici. Tato hranice je určena neurčitostí v

počáteční poloze robota. Výpočetní složitost roste s přibývajícím počtem landmarků kvadraticky, protože v korekčním kroku je třeba přepočítat celou kovarianční matici. Při aplikaci EKF SLAM na rozsáhlé mapovací problémy může tato složitost představovat problém. Algoritmus je velmi náchylný na špatnou asociaci dat a pokud k pozorovaným datům přiřadí špatný landmark, pak velmi pravděpodobně selže. EKF pracuje s linearizovanou verzí pohybového modelu a modelu pozorování. Výrazné nelinearity v těchto modelech mohou způsobit divergenci algoritmu.

1.4 FastSLAM

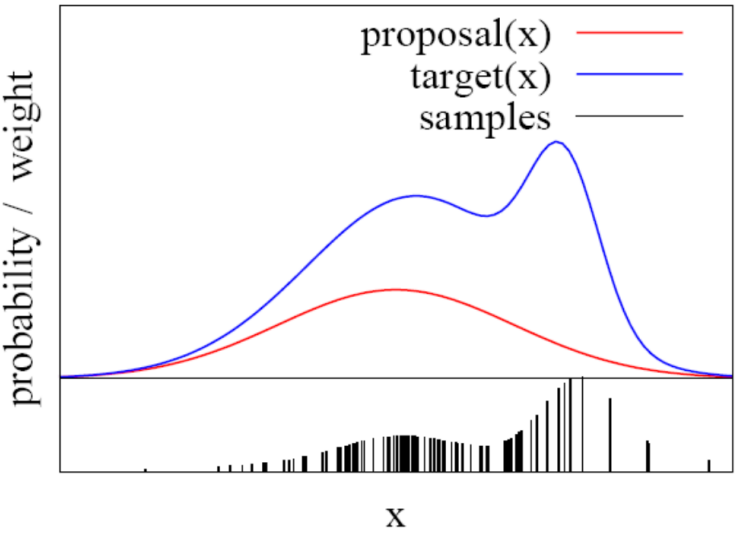
Tento přístup využívá k řešení SLAM problémů částicový filtr. Popis Fast SLAMu vychází z [1, 3]. Částicový filtr dovoluje reprezentovat libovolné pravděpodobnostní rozdělení neparametricky pomocí tzv. vzorků. Obecně je dáno nějaké cílové pravděpodobnostní rozdělení p, jehož vzorky se nedají generovat přímo. Je třeba zavést návrhové rozdělení π, z něhož lze vzorky získat snadno. Návrhové rozdělení je Gaussovo rozdělení.

Návrhové a cílové rozdělení se samozřejmě liší a samotné vzorky návrhového rozdělení nemají žádnou vypovídající hodnotu o podobě cílového rozdělení. Po získání zvoleného počtu vzorků z π je tedy třeba vypočítat jejich váhu w

$$w_k^{[i]} = \frac{P(x_k^{[i]})}{\pi(x_k^{[i]})}, \tag{1.35}$$

kde $w_k^{[i]}$ je váha i-tého vzorku v čase k a $x_k^{[i]}$ je hodnota i-tého vzorku v čase k. Vzorek s váhou se též nazývá částice. Váha částice kompenzuje rozdíl mezi rozložením vzorků z návrhového rozdělení a podobou cílového rozdělení. Od váhy částice se též odvíjí pravděpodobnost, s jakou bude částice vybrána při převzorkování (viz dále). Po vypočtení vah částice reprezentují neparametrickou podobu cílového rozdělení. Čím více částic je použito při odhadu rozdělení, tím přesnější tento odhad je. Znázornění vzorků a jejich vah lze zhlédnout v obrázku 1.3, kde červeně je vyznačena hustota pravděpodobnosti návrhového rozdělení, modře hustota pravděpodobnosti cílového rozdělení a černě jsou vyznačeny jednotlivé vzorky. Váha vzorku je v obrázku vyjádřena jeho výškou.

Ve FastSLAM částicový filtr vypočítává stav systému, jehož dimenze je $(3+2n)$ pro n landmarků. V případě použití částicového filtru pro řešení problému s vysokou dimenzí, kde je nutno vzorkovat každou z nich, jsou výpočetní nároky velmi vysoké. Zmenšení dimenze prostoru, který je třeba vzorkovat, umožňuje v tomto případě Rao-Blackwellizace. Ta udává, že při rozkladu sdružené pravděpodobnosti na $P(A, B) = P(B|A) \cdot P(A)$ a za předpokladu, že $\mathrm{P}(B|A)$ lze vypočítat, pak pouze $\mathrm{P}(A)$ je nutno vzorkovat. V takto Rao-Blackwellizovaném filtru je pak N částic reprezentováno množinou $\{A^{[i]}, \mathrm{P}(B|A^{[i]})\}_i^N$. K aplikaci Rao-Blackwellizace



Obrázek 1.3: Vzorky návrhového rozdělení a jejich váhy. Převzato z [16]

na stav systému při řešení SLAM úlohy je nutno stav systému rozdělit na mapovací část[1] a část trajektorie robota

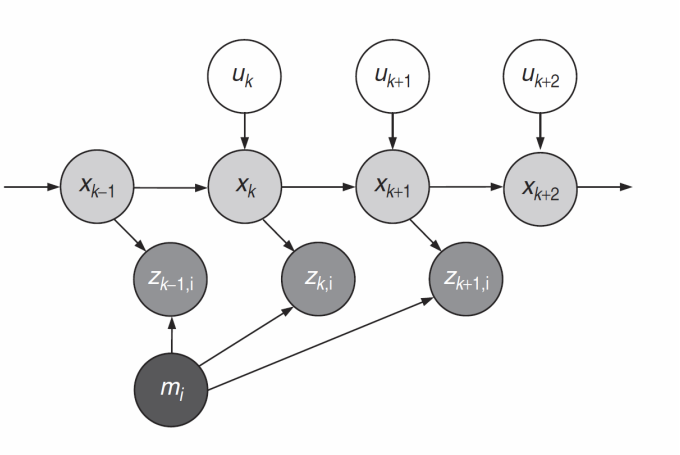
$$P(\boldsymbol{X}_{0:k}, \boldsymbol{m}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) = P(\boldsymbol{m}|\boldsymbol{X}_{0:k}, \boldsymbol{Z}_{0:k})P(\boldsymbol{X}_{0:k}|\boldsymbol{Z}_{0:k}, \boldsymbol{U}_{0:k}, \boldsymbol{x}_0) \tag{1.36}$$

a za předpokladu, že jednotlivé pozice robota v trajektorii jsou přesně známé, pak jsou na sobě landmarky nezávislé, viz obrázek 1.4. Díky této nezávislosti je celková mapa složena z několika na sobě nezávislých normálních rozdělení a lze ji vypočítat pomocí rovnice

$$P(\boldsymbol{m}|\boldsymbol{X}^{[i]}_{0:k}, \boldsymbol{Z}_{0:k}) = \prod_{j}^{M} P(\boldsymbol{m_j}|\boldsymbol{X}^{[i]}_{0:k}, \boldsymbol{Z}_{0:k}), \tag{1.37}$$

kde M je celkový počet landmarků. Rao-Blackwellizovaný stav systému v časovém okamžiku k je pak reprezentován množinou $\{w^{[i]}, \boldsymbol{X}^{[i]}_{0:k}, \mathrm{P}(\boldsymbol{m}|\boldsymbol{X}^{[i]}_{0:k}, \boldsymbol{Z}_{0:k})\}^N_i$ a pro každou částici je třeba vzorkovat pouze pozici robota, která má mnohem menší dimenzi než celý stavový vektor, což vede ke značnému urychlení výpočtů. Každá z N částic nese vlastní váhu, trajektorii robota i mapu. FastSLAM tedy operuje zároveň s N hypotézami stavu systému naráz. Práce s tolika odhady stavu systému a fakt, že většina částic se při správném běhu algoritmu shlukuje kolem reálné polohy robota, mu dovoluje v každé částici prohlásit její odhad polohy robota za jeho přesně známou polohu, i když jde pouze o odhad. Částice s dobrým odhadem polohy si udrží vysokou váhu a budou dále využívány, zatímco částice se špatným odhadem polohy svou váhu ztratí a při převzorkování se vytratí (viz dále). Algoritmus je rozdělen do čtyř kroků.

[1] $\boldsymbol{m}$ by mělo být podmíněno i $\boldsymbol{U}_{0:k}$ a $\boldsymbol{x}_0$, ale jelikož na nich hodnota $\boldsymbol{m}$ nezávisí, tak není třeba je zmiňovat.



Obrázek 1.4: Grafický model SLAM problému. Převzato z [1]

1.4.1 Vzorkování

Pro každou částici z předchozího časového okamžiku k-1 v $\{w_{k-1}^{[i]}, \boldsymbol{X}_{0:k-1}^{[i]}, \mathrm{P}(\boldsymbol{m}|\boldsymbol{X}_{0:k-1}^{[i]}, \boldsymbol{Z}_{0:k-1})\}_i^N$ je vypočteno návrhové rozdělení a z něj je vybrán vzorek

$$\boldsymbol{x}_k^{[i]} \sim \pi(\boldsymbol{x}_k|\boldsymbol{X}_{0:k-1}^{[i]}, \boldsymbol{Z}_{0:k}, \boldsymbol{u}_k) \tag{1.38}$$

a každý takový vzorek je vložen do historie poloh své částice $\boldsymbol{X}_{0:k}^{[i]} = \{\boldsymbol{X}_{0:k-1}^{[i]}, \boldsymbol{x}_k^{[i]}\}$. Tento krok predikuje polohu robota a po jeho dokončení se částice posunou v prostoru do nové pozice závislé na jeho předchozí poloze, aplikovaném řízení a v některých případech i na pozorování okolí z nové pozice. V tomto okamžiku částice ještě nemají váhy odpovídající situaci, ve které se robot právě nachází. Tento krok lze přirovnat ke kroku predikce v EKF SLAM.

1.4.2 Vážení vzorků

Každé částici z předchozího kroku je přiřazena váha

$$w_k^{[i]} = w_{k-1}^{[i]} \frac{P(\boldsymbol{z}_k|\boldsymbol{X}_{0:k}^{[i]}, \boldsymbol{Z}_{0:k-1}) P(\boldsymbol{x}_k^{[i]}|\boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_k)}{\pi(\boldsymbol{x}_k^{[i]}|\boldsymbol{X}_{0:k-1}^{[i]}, \boldsymbol{Z}_{0:k}, \boldsymbol{u}_k)} \tag{1.39}$$

Váha částice vyjadřuje, nakolik se pozice robota a mapa z částice shoduje s pozicí robota a mapou získanou z pozorování.

1.4.3 Převzorkování

Při běhu algoritmu některé částice mohou provést špatnou asociaci landmarků nebo špatně odhadnout polohu robota. V ten moment částice není vhodná pro reprezentaci stavu systému

a ztrácí svoji váhu. Protože algoritmus má k dispozici omezené množství částic, je postupem času velmi pravděpodobné, že k takové chybě dojde u většiny nebo u všech částic, a tím algoritmus ztrácí jakoukoliv vypovídající hodnotu o stavu systému. Takové situaci zamezuje převzorkování.

Při převzorkování je ze stavu systému vybráno N částic, které utvoří nový stav systému. Pravděpodobnost vytažení dané částice se odvíjí od její váhy, čím vyšší váha, tím "lepší"odhad stavu v dané částici, tím větší pravděpodobnost vytažení. Částice je možné vytahovat opakovaně, jinak by tento krok skončil se stejnou množinu jako na jeho začátku. Po utvoření nového stavu systému z vytažených částic je všem částicím udělena stejná váha $w_k^{[i]} = 1/N$.

Tento krok není nutné vykonat při každé iteraci algoritmu, četnost jeho použití závisí čistě na implementaci, dostupném výpočetním výkonu a přesnosti odometrie a použitých senzorů.

1.4.4 Odhad mapy

Pro každou částici je vypočítán nový odhad mapy závislý na datech získaných pozorováním a na trajektorii dané částice. Tento výpočet využívá EKF. EKF update je proveden pro každý landmark zvlášť, protože jeden z předpokladů Fast SLAM je vzájemná nezávislost landmarků. Nejedná se tedy o rozměrné, výpočetně náročné matice jako v EKF SLAM.

1.4.5 Vlastnosti a verze FastSLAM

FastSLAM je podstatně rychlejší a méně náročný na hardware než EKF SLAM. Hlavním důvodem navýšení rychlosti je prohlášení landmarků za vzájemně nezávislé. To zbavuje FastSLAM nutnosti počítat kovariance mezi landmarky, jejichž výpočet v EKF SLAMu zabírá podstatnou dobu. Vzájemná nezávislost landmarků je výsledkem předpokladu, že trajektorie robota je v každé částici přesně známá. Jak již bylo řečeno, může si to dovolit díky vysokému počtu částic a pravidelnému převzorkování. FastSLAM je též robustnější vůči špatné asociaci dat než EKF SLAM. Každá částice si nese vlastní mapu a provádí vlastní asociace landmarků. Pokud v nějaké částici dojde ke špatné asociaci, pak může částice buď představovat nereálnou situaci, přičemž ztratí svou váhu a při převzorkování se vytratí, nebo započne novou hypotézu trajektorie, kterou po několika krocích potká stejný osud.

Díky využití částicového filtru může pracovat i s nelineárními modely a libovolnými rozděleními pravděpodobnosti. Celková složitost algoritmu lze při ukládání dat částic do stromové struktury snížit až na $\mathcal{O}(N \log M)$ pro N částic a M landmarků oproti složitosti $\mathcal{O}(M^2)$ pro EKF SLAM.

FastSLAM je dále možné rozdělit do dvou verzí, konkrétně FastSLAM 1.0 a FastSLAM 2.0. Liší se v prvních dvou krocích algoritmu. Jako návrhové rozložení π volí FastSLAM 1.0

pohybový model

$$\boldsymbol{x}_k^{[i]} \sim P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_k) \tag{1.40}$$

a z (1.39) lze určit výpočet váhy pro částice

$$w_k^{[i]} = w_{k-1}^{[i]} P(\boldsymbol{z}_k|\boldsymbol{X}_{0:k-1}^{[i]}, \boldsymbol{Z}_{0:k-1}) \tag{1.41}$$

FastSLAM 2.0 bere při vzorkování v potaz nejen údaje z odometrie, ale i pozorování ze senzorů

$$\begin{gathered} \boldsymbol{x}_k^{[i]} \sim P(\boldsymbol{x}_k|\boldsymbol{X}_{0:k}^{[i]}, \boldsymbol{Z}_{0:k}, \boldsymbol{u}_k) \\ P(\boldsymbol{x}_k|\boldsymbol{X}_{0:k}^{[i]}, \boldsymbol{Z}_{0:k}, \boldsymbol{u}_k) = \frac{1}{C} P(\boldsymbol{z}_k|\boldsymbol{x}_k, \boldsymbol{X}_{0:k-1}^{[i]}, \boldsymbol{Z}_{0:k}) P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_k), \end{gathered} \tag{1.42}$$

kde C je normalizační konstanta a dle (1.39) je $\boldsymbol{w}_k^{[i]} = C \cdot \boldsymbol{w}_{k-1}^{[i]}$. Ve FastSLAM 2.0 je krok vzorkování složitější než v předchozí verzi, ale díky přesnějšímu určení polohy robota potřebuje pro správný chod mnohem méně částic než FastSLAM 1.0.

1.5 UKF SLAM

Unscented Kalman Filter (UKF) je další varianta Kalmanova filtru. K odhadu stavu systému používá též Gaussovo rozdělení, ale místo linearizace funkce při výpočtu rozdělení pro další iteraci algoritmu využívá unscented transformace. EKF i UKF využívají několik proměnných, jejichž význam je v obou algoritmech stejný. Použité značení z části vychází ze značení deklarovaného v sekci o EKF SLAM. Popis UKF SLAMu vychází z [8, 15]

Unscented transformace umožňuje využít nelineární transformační funkci $\boldsymbol{y}=g(\boldsymbol{x})$ pro transformaci Gaussova rozdělení $\boldsymbol{x}$. Nejprve navzorkuje počáteční rozdělení, tyto vzorky nechá projít nelineární funkcí a na základě výsledných vzorků vypočte transformované Gaussovo rozdělení. Tyto vážené vzorky, tzv. sigma body, jsou rovnoměrně rozloženy po vzorkovaném rozdělení pravděpodobnosti a jsou uspořádány do množiny $\mathcal{X}$, čítající $2n+1$ prvků pro rozdělení o dimenzi n. Těmito prvky jsou

$$\begin{aligned} \boldsymbol{\mathcal{X}}^{[0]} &= \overline{\boldsymbol{x}} = \boldsymbol{\mu} \\ \boldsymbol{\mathcal{X}}^{[i]} &= \overline{\boldsymbol{x}} + (\sqrt{(n+\lambda)\boldsymbol{P}_x})_i \quad \text{pro } i = 1, ..., n \\ \boldsymbol{\mathcal{X}}^{[i]} &= \overline{\boldsymbol{x}} - (\sqrt{(n+\lambda)\boldsymbol{P}_x})_i \quad \text{pro } i = n{+}1, ..., 2n \end{aligned} \tag{1.43}$$

a jejich váhy mají tvar

$$\begin{aligned} w_m^{[0]} &= \frac{\lambda}{n+\lambda} \\ w_c^{[0]} &= w_m^{[0]} + (1-\alpha^2+\beta) \\ w_m^{[i]} = w_c^{[i]} &= \frac{1}{2(n+\lambda)} \quad \text{pro } i = 1, ..., 2n, \end{aligned} \tag{1.44}$$

kde $\overline{\boldsymbol{x}}$ je střední hodnota Gaussova rozdělení, $\lambda = \alpha^2(n+\kappa)$-$n$ je škálovací parametr rozhodující o vzdálenosti sigma bodů od střední hodnoty a jeho parametry α a κ jsou volitelné, $\boldsymbol{P}$ je kovarianční matice rozdělení, i značí i-tý řádek matice, $w_m^{[i]}$ jsou váhy pro výpočet nové střední hodnoty, $w_c^{[i]}$ jsou váhy pro výpočet nové kovariance a β je parametr popisující znalosti o rozdělení pravděpodobnosti. Obvyklé hodnoty použité při volbě volných parametrů jsou $\alpha =< 1 \times 10^{-4}, 1 >$, $\beta = 2$ pro Gaussovo rozdělení a $\kappa = 0$. Parametry α a β určují přesnost aproximace vyšších momentů negaussovských rozlodělení pravděpodobnosti.

Vzorky, které plně vystihují střední hodnotu a kovarianci pro dané Gaussovo rozdělení, jsou následně přepočítány nelineární funkcí a je z nich vypočtena střední hodnota a kovariance transformovaného rozdělení pravděpodobnosti

$$\boldsymbol{\mathcal{Y}}^{[i]} = g(\boldsymbol{\mathcal{X}}^{[i]}) \quad \text{pro } i = 0, ..., 2n, \tag{1.45}$$

$$\overline{\boldsymbol{y}} \approx \sum_{i=0}^{2n} w_m^{[i]} \boldsymbol{\mathcal{Y}}^{[i]} \tag{1.46}$$

$$\boldsymbol{P}_y \approx \sum_{i=0}^{2n} w_c^{[i]} (\boldsymbol{\mathcal{Y}}^{[i]} - \overline{\boldsymbol{y}})(\boldsymbol{\mathcal{Y}}^{[i]} - \overline{\boldsymbol{y}})^T \tag{1.47}$$

1.5.1 Predikce

Algoritmus je rozdělen do dvou kroků, v první části dochází k odhadu stavu robota deterministickým vybráním vzorků z rozdělení pravděpodobnosti předchozího stavu a použitím pohybového modelu $\boldsymbol{f}()$

$$\boldsymbol{\mathcal{X}}_{k|k-1}^{[i]} = \boldsymbol{f}(\boldsymbol{\mathcal{X}}_{k-1|k-1}^{[i]}, \boldsymbol{u}_k) \tag{1.48}$$

$$\hat{\boldsymbol{x}}_{k|k-1} = \sum_{i=0}^{2n} w_m^{[i]} \boldsymbol{\mathcal{X}}_{k|k-1}^{[i]} \tag{1.49}$$

$$\boldsymbol{P}_{k|k-1} = \sum_{i=0}^{2n} w_c^{[i]} (\boldsymbol{\mathcal{X}}_{k|k-1}^{[i]} - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{\mathcal{X}}_{k|k-1}^{[i]} - \hat{\boldsymbol{x}}_{k|k-1})^T + \boldsymbol{Q}_k \tag{1.50}$$

1.5.2 Korekce

V druhém kroku algoritmu je vypočteno předpokládané pozorování pomocí pozorovacího modelu $\boldsymbol{h}()$ a kovarianční matice inovace $\boldsymbol{S}_k$

$$\boldsymbol{Z}^{[i]}_{k|k-1} = \boldsymbol{h}(\boldsymbol{\mathcal{X}}^{[i]}_{k-1|k-1}) \tag{1.51}$$

$$\hat{\boldsymbol{z}}_{k|k-1} = \sum_{i=0}^{2n} w^{[i]}_m \boldsymbol{Z}^{[i]}_{k|k-1} \tag{1.52}$$

$$\boldsymbol{S}_k = \sum_{i=0}^{2n} w^{[i]}_c (\boldsymbol{Z}^{[i]}_{k|k-1} - \hat{\boldsymbol{z}}_{k|k-1})(\boldsymbol{Z}^{[i]}_{k|k-1} - \hat{\boldsymbol{z}}_{k|k-1})^T + \boldsymbol{R}_k. \tag{1.53}$$

Dále je vypočtena matice kovariancí mezi robotem a predikovanou polohou landmarků $\boldsymbol{P}_{xz,k|k-1}$ a Kalmanův zisk $\boldsymbol{W}_k$

$$\boldsymbol{P}_{xz,k|k-1} = \sum_{i=0}^{2n} w^{[i]}_c (\boldsymbol{\mathcal{X}}^{[i]}_{k|k-1} - \hat{\boldsymbol{x}}_{k|k-1})(\boldsymbol{Z}^{[i]}_{k|k-1} - \hat{\boldsymbol{z}}_{k|k-1})^T \tag{1.54}$$

$$\boldsymbol{W}_k = \boldsymbol{P}_{xz,k|k-1} \boldsymbol{S}_k^{-1}. \tag{1.55}$$

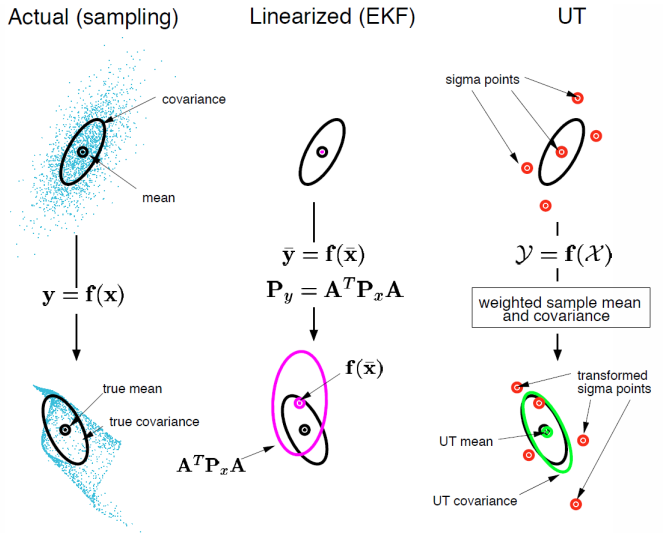
Po vypočtení Kalmanova zisku $\boldsymbol{W}_k$ je stav systému pro časový okamžik k získán z 1.27.

1.5.3 Porovnání UKF a EKF

EKF aproximuje stav systému pomocí linearizace funkcí za použití Taylorova rozvoje 1. stupně. Při aplikaci na silně nelineární funkce může tato linearizace do výpočtu zanést zřetelnou chybu a může vést až k selhání algoritmu. Tyto chyby UKF řeší přístupem, ve kterém lze při výpočtu aplikovat nelineární funkce. Díky tomu lze pomocí UKF přesně vypočítat střední hodnotu a kovarianci transformovaného rozložení do 3. stupně Taylorova rozvoje. UKF dosahuje lepších výsledků při aplikaci na nelineární systémy při zachování stejné výpočetní složitosti jako EKF [15]. Nástin porovnání výsledků linearizované a unscented transformace je zobrazen na obrázku 1.5. V prvním sloupci se nachází reálný výsledek dosažený vzorkováním rozdělení $\boldsymbol{x}$, přepočítáním vzorků pomocí funkce $\boldsymbol{f}()$ a určení Gaussova rozdělení v těchto vzorcích, v druhém sloupci je popsána transformace linearizovanou funkcí, kterou využívá EKF, a ve třetím sloupci je vyobrazena unscented transformace používaná UKF.

1.6 Graph-Based SLAM

Graph-Based SLAM je offline metoda řešení SLAM problému. Pro ukládání dat využívá graf, který po dokončení měření optimalizuje, obvykle pomocí metody nejmenších čtverců. Popis



Obrázek 1.5: Příklad rozdílu mezi linearizovanou a unscented transformací. Převzato z [15]

Graph-Based SLAMu vychází z [4]. Tento přístup je rozdělen do dvou částí.

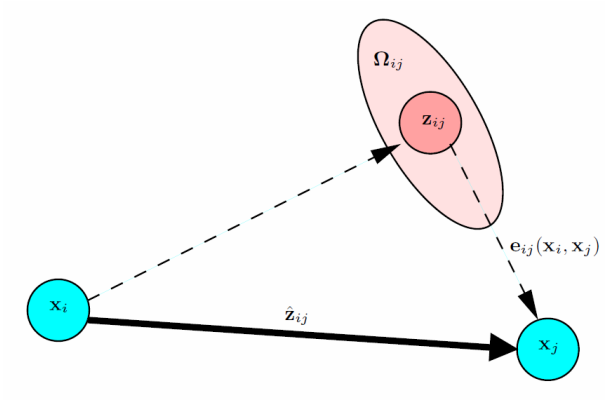
1.6.1 Front-end

První část (tzv. front-end) se stará o tvorbu mapy, kterou reprezentuje grafem. Uzel v grafu nese informace o pozici robota a o měření získaném v této pozici v časovém okamžiku k. Hrany mezi jednotlivými uzly vystihují vzájemnou polohu uzlů, ta je popsána odometrií nebo výsledkem porovnání měření. V případě popisu odometrií se hrana nachází mezi uzly v sousedních časových okamžicích a popisuje posun robota. V případě popisu hrany porovnáváním měření je využíván scan matching. Scan matching porovnává dvě měření z různých časových okamžiků. Pokud je při provádění scan matchingu zjištěna podobnost v naměřených datech, pak je určena relativní poloha lokací, ze kterých byla měření pořízena [12]. Tyto hrany popsané porovnáváním měření nesou informace o tzv. virtuálním měření.

Pokud se robot ocitne ve stavu x_i uprostřed oblasti, kterou již dříve navštívil, pak může porovnat měření z polohy x_i s měřením v poloze x_j, ve které se nacházel při provedení prvního měření v dané oblasti. Pomocí scan matchingu tak získá vzájemnou polohu x_i a x_j a tento vektor mířící z x_i do x_j se nazývá virtuální měření.

Při běhu algoritmu jsou nově získaná měření porovnávaná s předešlými a v případě navštívení již dříve objevené lokace (tzn. naměření podobných dat) je získáno virtuální měření mezi měřením z aktuálního uzlu x_i a měřením z uzlu z předchozího průchodu danou lokací x_j, které je do grafu zaneseno jako nová hrana vycházející z x_i. Virtuální měření však není měření vzájemné x_i a x_j, ale transformace, díky které se měření z uzlu x_i maximálně překrývá s měřením z x_j. Za předpokladu přesné odometrie robota a přesného měřícího zařízení by tedy virtuální měření z uzlu x_i končilo přesně v uzlu x_j, ale ani odometrie, ani měřící zařízení

nejsou naprosto přesné. Reálná podoba virtuálního měření je tedy jiná a lze ji zhlédnout v obrázku 1.6, kde x_i je aktuální uzel, ve kterém robot navštívil již dříve objevenou oblast, x_j je uzel, ve kterém byla daná oblast objevena poprvé, $\hat{z_{ij}}$ je predikce virtuálního měření mezi uzly x_i a x_j odvozená z rozložení grafu, z_{ij} je virtuální měření, Ω_{ij} je informační matice virtuálního měření z_{ij} (viz back-end) a $e_{ij}(x_i, x_j)$ je funkce vyjadřující rozdíl mezi z_{ij} a $\hat{z_{ij}}$ (viz back-end).



Obrázek 1.6: Rozdíl mezi predikcí virtuálního měření a virtuálním měřením . Převzato z [4]

1.6.2 Back-end

V druhé části (tzv. back-end) algoritmus optimalizuje rozložení uzlů v grafu tak, aby získal nejlepší odhad mapy. Nejprve je třeba definovat funkci $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$, která je znázorněna v obrázku 1.6

$$\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{z}_{ij} - \hat{\boldsymbol{z}}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j). \tag{1.56}$$

Funkce $\boldsymbol{e}_{ij}(\boldsymbol{x}_i, \boldsymbol{x}_j)$ vyjadřuje chybu mezi z_{ij} a $\hat{z_{ij}}$. Optimálního rozložení grafu je dosaženo minimalizací chyby $\boldsymbol{F}(\boldsymbol{x})$pro všechna virtuální měření

$$\boldsymbol{F}(\boldsymbol{x}) = \sum_{[i,j]\in\mathcal{C}} \boldsymbol{e}_{ij}^T \boldsymbol{\Omega}_{ij} \boldsymbol{e}_{ij} = \sum_{[i,j]\in\mathcal{C}} \boldsymbol{F}_{ij}, \tag{1.57}$$

kde $\mathcal{C}$ je množina indexů všech virtuálních měření. Cílové rozmístění uzlů $\boldsymbol{x}^*$ je tedy

$$\boldsymbol{x}^* = \underset{\boldsymbol{x}}{\operatorname{argmin}}\, \boldsymbol{F}(\boldsymbol{x}). \tag{1.58}$$

Věrohodnost budovaného grafu se zvyšuje, pokud čtvercová chyba mezi $\boldsymbol{z}_{ij}$ a $\hat{\boldsymbol{z}}_{ij}$ klesá, optimální řešení lze tedy dosáhnout metodou nejmenších čtverců. Front-end poskytl po dokončení měření počáteční odhad rozmístění uzlů $\breve{\boldsymbol{x}}$. Ten umožňuje rovnici (1.58) řešit numericky Gauss-Newtonovým algoritmem, který minimalizuje chybovou funkci $\boldsymbol{e}_{ij}(\boldsymbol{x}_i,\boldsymbol{x}_j)$ a provádí při

tom její linearizaci prvním stupněm Taylorova rozvoje v pracovním bodě $\breve{\boldsymbol{x}}$.

$$\boldsymbol{e}_{ij}(\breve{\boldsymbol{x}}_i + \Delta\boldsymbol{x}_i, \breve{\boldsymbol{x}}_j + \Delta\boldsymbol{x}_j) = \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) \simeq \boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x}, \tag{1.59}$$

kde $\boldsymbol{e}_{ij} = \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}})$ a $\boldsymbol{J}_{ij}$ je jakobián $\boldsymbol{e}_{ij}(\boldsymbol{x})$. Po dosazení linearizovaného tvaru z (1.59) do (1.57) je získán tvar

$$\begin{aligned}
\boldsymbol{F}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &= \boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x})^T\boldsymbol{\Omega}_{ij}\boldsymbol{e}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) \simeq (\boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x})^T\boldsymbol{\Omega}_{ij}(\boldsymbol{e}_{ij} + \boldsymbol{J}_{ij}\Delta\boldsymbol{x}) \\
\boldsymbol{F}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &\simeq \boldsymbol{e}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{e}_{ij} + 2\boldsymbol{e}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{J}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{J}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{J}_{ij}\Delta\boldsymbol{x} \\
\boldsymbol{F}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) &\simeq c_{ij} + 2\boldsymbol{b}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{H}_{ij}\Delta\boldsymbol{x},
\end{aligned} \tag{1.60}$$

kde $c_{ij} = \boldsymbol{e}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{e}_{ij}$, $\boldsymbol{b}_{ij} = \boldsymbol{e}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{J}_{ij}$ a $\boldsymbol{H}_{ij} = \boldsymbol{J}_{ij}^T\boldsymbol{\Omega}_{ij}\boldsymbol{J}_{ij}$. Při dosazení této aproximace do (1.58) vznikne kvadratická forma

$$\boldsymbol{F}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) = \sum_{[i,j]\in\mathcal{C}} \boldsymbol{F}_{ij}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) \simeq \sum_{[i,j]\in\mathcal{C}} c_{ij} + 2\boldsymbol{b}_{ij}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{H}_{ij}\Delta\boldsymbol{x} \tag{1.61}$$

$$\boldsymbol{F}(\breve{\boldsymbol{x}} + \Delta\boldsymbol{x}) = c + 2\boldsymbol{b}\Delta\boldsymbol{x} + \Delta\boldsymbol{x}^T\boldsymbol{H}\Delta\boldsymbol{x}, \tag{1.62}$$

kde $c = \sum_{[i,j]\in\mathcal{C}} c_{ij}$, $\boldsymbol{b} = \sum_{[i,j]\in\mathcal{C}} \boldsymbol{b}_{ij}$ a $\boldsymbol{H} = \sum_{[i,j]\in\mathcal{C}} \boldsymbol{H}_{ij}$. Tato kvadratická forma může být minimalizována vzhledem k $\Delta\boldsymbol{x}$ vyřešením lineárního systému

$$\boldsymbol{H}\Delta\boldsymbol{x}^* = -\boldsymbol{b}, \tag{1.63}$$

kde $\boldsymbol{H}$ je informační matice systému. Linearizované řešení je pak dosaženo pomocí

$$\boldsymbol{x}^* = \breve{\boldsymbol{x}} + \Delta\boldsymbol{x}^*. \tag{1.64}$$

Gauss-Newtonova metoda opakuje kroky (1.62),(1.63) a (1.64), dokud není dosaženo uspokojivého řešení. Jako počáteční odhad $\breve{\boldsymbol{x}}$ v dalším korku používá řešení předchozího kroku $\boldsymbol{x}^*$.

1.6.3 Vlastnosti Graph-Based SLAM

Základní myšelnka Graph-Based SLAMu byla představena v roce 1997, ale uchytila se až v poslední době, protože použitá minimalizace chyby v kroku optimalizace grafu je výpočetně náročná. Nevýhodou proti online přístupům řešení SLAM problému je znalost přesného stavu robota a mapy k dispozici jen po provedení optimalizace grafu, nikoli v každém kroku algoritmu. Graph-Based SLAM ale dokáže snadno opravit jakékoli chyby, které byly do měření zaneseny, pokud je dokáže určit. Tento způsob řešení již v základu řeší problém uzavírání

smyček (tzv. loop closure) pomocí virtuálních měření, který u online přístupů může činit problémy. Metody řešení SLAM problému založené na tomto přístupu v dnešní době patří mezi nejvyspělejší.

2 Porovnávané algoritmy

V Kapitole 2 jsou blíže popsány jednotlivé implementace řešení SLAM problému, na kterých jsou prováděny experimenty. Byly vybrány na základě několika společných znaků. Všechny popisované implementace řeší SLAM problém ve dvou dimenzích, pro měření používají LiDAR, využívají Robot Operating System jako middleware a mapu reprezentují pomocí mřížky (tzv. Occupancy Grid Map).

2.1 Mřížková mapa

Mřížková mapa (Grid map) patří mezi jednu z možností reprezentace okolí robota. Reprezentuje prostředí pomocí mřížky buněk a její přesnost je určena velikostí použitých buněk. Popis mřížkové mapy vychází z [11].

Buňka mapy se může nacházet ve dvou stavech. Obsazené buňky reprezentují objekty v okolí a neobsazené buňky značí volný prostor. Pro určení mapy je třeba vypočítat pravděpodobnost obsazenosti každé buňky. Jednotlivé buňky jsou považovány za nezávislé a pravděpodobnost obsazenosti jedné buňky v jednom časovém okamžiku určuje, zda byla daná buňka v měření zaznamenána jako obsazená nebo neobsazená. Budovaná mapa je závislá na trajektorii robota a na měřeních, která byla provedena v bodech trajektorie. Pravděpodobnost mapy $\boldsymbol{m} = \{\boldsymbol{m_1}, \boldsymbol{m_1}, ..., \boldsymbol{m_k}\}$ má následný tvar

$$P(\boldsymbol{m}|\boldsymbol{x}_{0:k}, \boldsymbol{z}_{0:k}) = \prod_{i=0}^{j} P(\boldsymbol{m}_i|\boldsymbol{x}_{0:k}, \boldsymbol{z}_{0:k}), \tag{2.1}$$

$$P(\boldsymbol{m}_i|\boldsymbol{x}_{0:k}, \boldsymbol{z}_{0:k}) = \frac{1}{1 - e^{l_{k,i}}} \tag{2.2}$$

$$l_{k,i} = l_{k-1,i} + log\frac{P(\boldsymbol{m}_i|\boldsymbol{x}^{0:k}, \boldsymbol{z}^{0:k})}{1 - P(\boldsymbol{m}_i|\boldsymbol{x}^{0:k}, \boldsymbol{z}^{0:k})} - log\frac{P(\boldsymbol{m}_i)}{1 - P(\boldsymbol{m}_i)}, \tag{2.3}$$

kde j je počet jednotlivých buněk v mapě, $P(\boldsymbol{m}_i)$ je konstantní pravděpodobnost obsazenosti a $P(\boldsymbol{m}_i|\boldsymbol{x}^{0:k}, \boldsymbol{z}^{0:k})$ se nazývá inverzní model měření, který vrací vysokou hodnotu, pokud je buňka obsazená, a nízkou hodnotu, pokud není obsazená.

Ve spojení s LiDARem přináší mřížková reprezentace mapy několik výhod oproti popisu pomocí landmarků. Při použití landmarků je třeba do řešení SLAM problému implementovat extrakci landmarků z naměřených dat a správnou asociaci při jejich znovuobjevení. Tyto kroky přinášejí zvýšení výpočetní složitosti a ztrátu většiny informací, které byly při měření LiDARem získány. Přístupy využívající landmarky pro určení polohy robota v mapě musí

být upraveny před použitím spolu s mřížkovou reprezentací mapy, protože každou buňku mřížkové mapy lze v takovém případě chápat jako landmark. V takovém případě jsou nároky na výpočetní techniku příliš vysoké.

2.2 Hector SLAM

Hlavní myšlenkou při vývoji této online implementace řešení SLAM problému byla aplikovatelnost v krizových situacích v obydleném prostředí, tzv. USAR (Urban Search and Rescue). Hector SLAM se soustředí na rychlé řešení, které nevyžaduje příliš velký výpočetní výkon, aby mohl být použit na malých procesorech s nízkou spotřebou. Popis Hector SLAM vychází z [7].

Navzdory zaměření na nízké výpočetní nároky je Hector SLAM schopen dostatečně přesné percepce okolí a odhadu polohy robota tak, aby nepotřeboval implementovat uzavírání menších smyček. Nízké výpočetní nároky umožňují využít rychlé obnovovací frekvence moderních LiDARů, což vede k možnosti relativně rychlého pohybu. Z pohledu mapování generuje mapu vykreslenou ve dvoudimenzionální rovině, ale stav robota odhaduje ve třech dimenzích a šesti stupních volnosti. Díky tomu ho lze využít k mapování různorodých prostředí, ve kterých se robot spolu s LiDARem může naklánět do různých směrů, aniž by tato situace výrazně znehodnotila prováděné měření. Hector SLAM nemá k dispozici žádný back-end, který by optimalizoval již zmapované prostředí a ošetřoval uzavírání smyček.

Určení stavu robota se skládá ze dvou částí. Navigační filtr hromadí data z inerciální měřící jednotky, tzv. IMU (Inertial Measurement Unit) a ostatních volitelných senzorů, mezi které patří GPS, kompas, barometr a další, pro odhad stavu robota. SLAM systém poskytuje stav robota ve dvoudimenzionální rovině. Tyto odhady jsou spolu spřaženy pro udržení konzistetního odhadu stavu robota, ale jsou obnovovány nezávisle na sobě.

Stav systému je popsán vektorem $\boldsymbol{x} = [\boldsymbol{\Omega}^T, \boldsymbol{p}^T, \boldsymbol{v}^T]^T$, kde $\boldsymbol{\Omega} = [\phi, \theta, \psi]^T$ jsou Eulerovy úhly popisující orientaci robota, $\boldsymbol{p} = [p_x, p_y, p_z]^T$ je pozice robota a $\boldsymbol{v} = [v_x, v_y, v_z]^T$ je rychlost robota v daných směrech. IMU zaznamenává řízení v podobě $\boldsymbol{u} = [\boldsymbol{\omega}^T, \boldsymbol{a}^T]$, kde $\boldsymbol{\omega} = [\omega_x, \omega_y, \omega_z]^T$ jsou úhlové rychlosti a $\boldsymbol{a} = [a_x, a_y, a_z]^T$ jsou zrychlení v jednotlivých osách. Záznamy IMU se vztahují k osám robota, nikoliv k souřadnicím systému. Pohybový model robota je definován systémem nelineárních rovnic

$$\begin{aligned} \dot{\boldsymbol{\Omega}} &= \boldsymbol{E}_{\boldsymbol{\Omega}}\boldsymbol{\omega} \\ \dot{\boldsymbol{p}} &= \boldsymbol{v} \\ \dot{\boldsymbol{v}} &= \boldsymbol{R}_{\boldsymbol{\Omega}}\boldsymbol{a} + \boldsymbol{g}, \end{aligned} \tag{2.4}$$

kde $\boldsymbol{R}_{\boldsymbol{\Omega}}$ a $\boldsymbol{E}_{\boldsymbol{\Omega}}$ jsou transformační matice převádějící záznamy IMU ze souřadného systému

robota do souřadnic systému a g je vektor gravitačního zrychlení. Odhad stavu robota z IMU pomocí navigačního filtru je obnovován s frekvencí 100 Hz a je asynchronně upravován odhadem pozice získaného ze scan matchingu měření, případně je ovlivněn i odhady z přídavných senzorů. Rovnice (2.4) popisují systém, na který je aplikovaný EKF pro odhad polohy a orientace robota ve třech dimenzích.

Odhadování stavu robota pouze pomocí IMU vede k chybným výsledkům. Tyto chyby jsou opravovány pomocí scan matchingu a případně doplněním dat z volitelných senzorů. Implementovaný scan matching využívá Gauss-Newtonovy metody pro určení polohy daného měření v mapě.

Měření získaná pomocí LiDARu jsou transformována ze souřadného systému robota do souřadného systému mapy. Měření je formulováno jako sada bodů, jejichž rozmístění kopíruje tvar naskenované překážky. Tyto body mohou být upraveny před jejich dalším využitím. Mezi tyto úpravy patří např. omezení ve smyslu svislé osy z, které prohlašuje data naměřená mimo určité pásmo za neplatná. Toto omezení zabraňuje zanesení stropu a podlahy do generované mapy, pokud je měřící zařízení nakloněno tak, že je registruje.

Mřížková mapa použitá v tomto řešení je diskrétní povahy, což omezuje přesnost mapy a znemožňuje přímý výpočet derivací. Pro jejich odhad je zavedena bilineární filtrace, která umožňuje interpolovat spojitou podobu mapy, odhad derivací a odhad pravděpodobnosti obsazenosti buněk. Díky této interpolaci lze odhadnout funkci pravděpodobnosti obsazení buněk ve spojitém tvaru a tedy odhadnout i její gradient, který využívá scan matching, konkrétně v Gauss-Newtonově metodě. Toto využívání gradientů může vést k ukončení výpočtu v lokálním minimu, tzn. nedosažením optimálního řešení. Hector SLAM tento problém řeší udržováním několika odhadů mapy zároveň, kde jeden odhad mapy používá nejjemnější rozlišení (nejmenší buňky) a každý další odhad mapy má poloviční rozlišení předchozího. Jedná se o nezávisle generované odhady, nikoliv o klony nejjemnější mapy s nižším rozlišením. V každém kroku je v mapách obnoven odhad polohy, což zaručuje konzistenci všech map. Scan matching je prováděn od nejhrubší mapy a výsledný odhad pozice je použit jako počáteční odhad pro každou další jemnější mapu.

Druhá a třetí rovnice v (2.4) jsou získány integrací akcelerace naměřené pomocí IMU, díky čemuž je takový systém nestabilní bez úpravy odhadu stavu robota pomocí měření. Odhad stavu robota získaný pomocí IMU a EKF je promítnut do mapy a je použit jako výchozí bod pro scan matching. Odhad stavu robota ze scan matchingu pak ovlivní odhad polohy následně

$$\boldsymbol{K} = \boldsymbol{P}\boldsymbol{C}^T(\frac{1-\omega}{\omega}\boldsymbol{R} + \boldsymbol{C}^T\boldsymbol{P}\boldsymbol{C})^{-1} \tag{2.5}$$

$$\boldsymbol{P}^+ = \boldsymbol{P} - (1-\omega)^{-1}\boldsymbol{K}\boldsymbol{C}\boldsymbol{P} \tag{2.6}$$

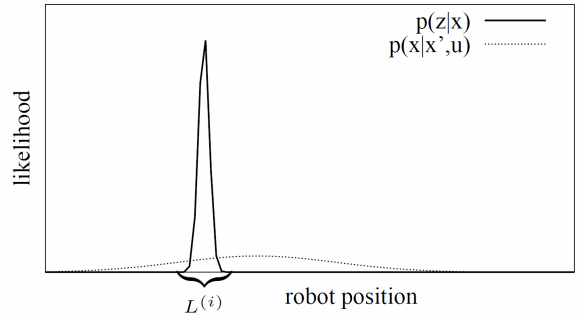
$$\hat{\boldsymbol{x}^+} = \hat{\boldsymbol{x}} + \boldsymbol{K}(\boldsymbol{\xi}^* - \boldsymbol{C}\hat{\boldsymbol{x}}), \tag{2.7}$$

kde $\hat{\boldsymbol{x}}$ je původní odhad stavu z IMU s kovariancí $\boldsymbol{P}$, $\boldsymbol{\xi}^*$ je odhad stavu získaný pomocí scan matchingu, jehož kovariance je $\boldsymbol{R}$, $\boldsymbol{K}$ je Kalmanův zisk, $\boldsymbol{C}$ je matice, která promítá stavový prostor do tří dimenzí, $\omega \in (0, 1)$ je volitelný ladící parametr a $\hat{\boldsymbol{x}}^+$ je výsledný odhad stavu s kovariancí $\boldsymbol{P}^+$.

2.3 GMapping

Tato implementace využívá částicový filtr. Je založena na podobném principu jako Fast SLAM v 1.4 a její popis vychází z [3]. Znakem gMappingu je využití odometrie i pozorování při odhadu návrhového rozdělení, což výrazně snižuje nejistotu odhadu stavu robota v kroku predikce. Dále je v tomto přístupu zahrnut rozhodovací proces, který provádí převzorkování pouze za určitých podmínek.

Při užití částicového filtru je zapotřebí vybrat vzorky z návrhového rozdělení pravděpodobnosti, které pak budou využity pro odhad cílového rozdělení pravděpodobnosti. Čím si jsou návrhové a cílové rozdělení podobnější, tím lepší výsledky bude filtr podávat a tím méně částic bude pro jeho funkci potřeba. Nejjednodušší řešení kroku predikce v částicovém filtru zahrnuje použití pohybového modelu jako návrhového rozložení. Takové řešení však potřebuje velký počet částic, protože, narozdíl od modelu pozorování, pohybový model nedokáže podat přesný odhad stavu robota. Na obrázku 2.1 je vyobrazena věrohodnost pohybového modelu $p(x|x', u)$ a modelu pozorování $p(z|x)$ vhledem k určení polohy robota. Nutnost použít velký počet částic při vzorkování pohybového modelu plyne z toho, že pouze malá část vzorků se nachází v intervalu $L^{(i)}$, ve kterém je stav robota nejpravděpodobnější. Tento problém lze



Obrázek 2.1: Věrohodnost pohybového modelu a modelu pozorování. Převzato z [3]

řešit začleněním nejnovějšího pozorování do návrhového rozdělení

$$P(\boldsymbol{x}_k|\boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{z}_k, \boldsymbol{u}_{k-1}) = \frac{P(\boldsymbol{z}_k|\boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{x}_k)P(\boldsymbol{x}_k|\boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1})}{P(\boldsymbol{z}_k|\boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1})}, \tag{2.8}$$

čímž se výpočet vah změní na

$$w_k^{[i]} = w_{k-1}^{[i]} P(\boldsymbol{z}_k | \boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1}). \tag{2.9}$$

Znalost takového rozdělení není dostupná přímo kvůli nepředvídatelnému tvaru věrohodnostní funkce modelu pozorování. Jeho aproximaci lze získat úpravou částicového filtru. Nejprve je pro každou částici určen počáteční odhad stavu robota $\boldsymbol{x}'^{[i]}_k$ získaný z pohybového modelu. Následně je proveden scan matching pozorování $\boldsymbol{z}_k$ s mapou $\boldsymbol{m}_{k-1}^{[i]}$ v okolí odhadu $\boldsymbol{x}'^{[i]}_k$, jehož výstupem je přesnější odhad polohy $\hat{\boldsymbol{x}}_k^{[i]}$. Dalším korkem je lokální aproximace cílového rozdělení kolem nejpravděpodobnější pozice robota. Té je dosaženo navzorkováním K vzorků $\boldsymbol{a}_j$ v intervalu okolo $\hat{\boldsymbol{x}}_k^{[i]}$ a výpočtem parametrů příslušného Gaussova rozdělení $\mathcal{N}(\boldsymbol{\mu}_k^{[i]}, \boldsymbol{\Sigma}_k^{[i]})$

$$\boldsymbol{\mu}_k^{[i]} = \frac{1}{\eta^{[i]}} \sum_{j=1}^{K} \boldsymbol{a}_j P(\boldsymbol{z}_k | \boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{a}_j) P(\boldsymbol{a}_j | \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1}) \tag{2.10}$$

$$\boldsymbol{\Sigma}_k^{[i]} = \frac{1}{\eta^{[i]}} \sum_{j=1}^{K} \boldsymbol{a}_j P(\boldsymbol{z}_k | \boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{a}_j) P(\boldsymbol{a}_j | \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1}) (\boldsymbol{a}_j - \boldsymbol{\mu}_k^{[i]})(\boldsymbol{a}_j - \boldsymbol{\mu}_k^{[i]})^T, \tag{2.11}$$

kde $\eta^{[i]}$ je normalizační parametr

$$\eta^{[i]} = \sum_{j=1}^{K} P(\boldsymbol{z}_k | \boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{a}_j) P(\boldsymbol{a}_j | \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1}). \tag{2.12}$$

z rozdělení $\mathcal{N}(\boldsymbol{\mu}_k^{[i]}, \boldsymbol{\Sigma}_k^{[i]})$ je pak pro každou částici vybrán nový stav robota $\boldsymbol{x}[i]_k$ a vypočtena váha dle

$$w_k^{[i]} \simeq w_{k-1}^{[i]} \sum_{j=1}^{K} P(\boldsymbol{z}_k | \boldsymbol{m}_{k-1}^{[i]}, \boldsymbol{a}_j) P(\boldsymbol{a}_j | \boldsymbol{x}_{k-1}^{[i]}, \boldsymbol{u}_{k-1}) = w_{k-1}^{[i]} \eta^{[i]}. \tag{2.13}$$

Scan matching umožňuje soustředit vzorkování do pravděpodobnějších oblastí. Výsledná přesnost určení stavu robota je touto úpravou zvýšena v každé částici a pro správný běh algoritmu lze použít méně částic. Problém může nastat při uzavírání smyčky, kdy se získané odhady $\hat{\boldsymbol{x}}_k^{[i]}$ mohou razantně lišit. GMapping ale obnovuje stav systému po každém posunu o 0.5 m nebo otočení o 25°, což by mělo zabránit vzniku problémů v takových situacích. Pokud scan matching selže v určení $\hat{\boldsymbol{x}}_k^{[i]}$, např. kvůli znehodnocenému měření, pak je jako návrhové rozložení použit pohybový model.

Převzorkování je u gMappingu prováděno pouze za určité podmínky. Tzv. efektivní velikost vzorku N_{eff}

$$N_{eff} = \frac{1}{\sum_{i=1}^{N} (\widetilde{w}^{[i]})^2}, \tag{2.14}$$

kde $\widetilde{w}^{[i]}$ je normalizovaná váha částice i, odhaduje jak přesně momentální sada částic vystihuje cílové rozdělení. Čím nižší je hodnota N_{eff}, tím větší rozptyl je mezi váhami částic. Převzorkování je prováděno po poklesu N_{eff} pod $N/2$ pro N částic. Pokud se robot pohybuje ve známém prostředí, pak zůstává hodnota parametru N_{eff} přibližně konstantní, při mapování neznámé oblasti pozvolna klesá a při uzavírání smyčky se výrazně propadá, protože v systému je drženo několik hypotéz stavu robota naráz.

Upřesnění odhadu stavu robota v predikčním kroku algoritmu vede k získání přesnější mapy a snižuje počet potřebných částic pro správný běh algoritmu (pro většinu situací gMapping nepotřebuje více než 80 částic). Převzorkování prováděné pouze při splnění výše zmíněné podmínky výrazně snižuje riziko zahození částice s vyhovujícím odhadem stavu systému.

2.4 Google Cartographer

Poslední testovaná implementace řešení SLAM problému byla vytvořena za účelem urychlení procesu plánování budov. Nabízí řešení uzavírání smyček s využitím submap a metody větví a mezí (Branch and Bound) s nízkými výpočetními nároky. Tato sekce vychází z [5].

V průběhu mapování je budována dvoudimenzionální mřížková mapa. K její tvorbě je přistupováno z lokálního a globálního hlediska. V obou hlediscích je odhadován stav robota $\boldsymbol{\xi} = \{\xi_x, \xi_y, \xi_\theta\}$, který vyjadřuje jeho polohu a natočení v rovině mapy.

Na několika místech Cartographer využívá při výpočtech knihovnu Ceres Solver. Ceres Solver je knihovna v jazyce C++ vyvinutá za účelem modelování a řešení složitých optimalizačních úloh. Mezi takové úlohy patří metoda nejmenších čtvrců pro nelineární modely [17]. Ceres Solver je v algoritmu využit např. při scan matchingu nebo při optimalizaci mapy.

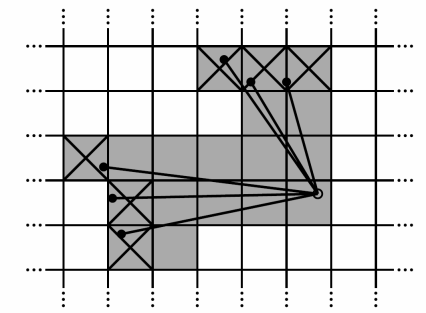
2.4.1 Lokální přístup

V lokálním přístupu je získáván odhad stavu robota pouze v rámci malého úseku prostředí, tzv. submapy M. Každé měření je pomocí scan matchingu porovnáno s právě generovanou submapou a je tak získán stav robota $\boldsymbol{\xi}$. Scan matching je počítán pomocí knihovny Ceres.

Měření jsou před použitím k odhadu stavu transformována do roviny mapy pomocí dat z IMU. To zabraňuje znehodnocení měření z důvodu naklonění skenovacího zařízení. Submapa je tvořena mřížkovou mapou s rozlišením r. Pro každou buňku v mapě je definován pixel, který zabírá čtvercovou plochu o hraně r kolem dané buňky. V obrázku 2.2 jsou tyto pixely v okolí buněk znázorněny.

Po vložení nového měření do submapy jsou dodané informace vykresleny. Koncové hodnoty získané při měření jsou vztaženy k buňce mapy, do jejíž pixelu zasahují. Tato buňka je pak prohlášena za obsazenou. Buňka každého pixelu, který protíná úsečka mezi obsazenou buňkou a polohou robota, je označena jako neobsazená. Jednotlivá měření jsou vyobrazena

v obrázku 2.2 spolu s pixely označenými jako obsazené (šedé pixely s křížkem) a neobsazené (šedé pixely bez křížku).



Obrázek 2.2: Měření vložené do submapy. Převzato z [5]

Měření získaná v lokálním přístupu jsou porovnávána pouze s danou submapou. S každým dalším měřením roste chyba určení stavu robota i okolí. Tento problém adresuje globální přístup.

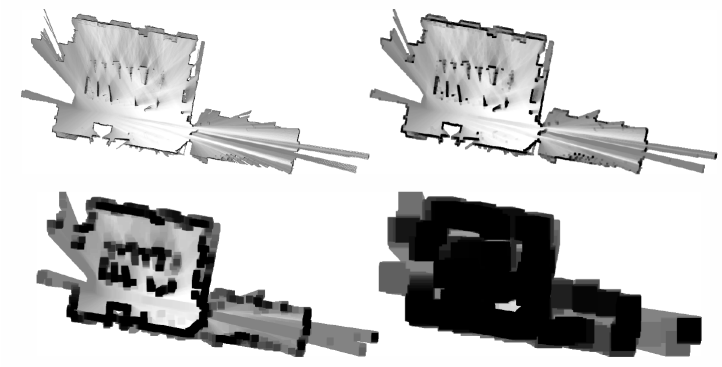
2.4.2 Globální přístup

Mapování větších prostor je řešeno v globálním přístupu. Relativní pozice, ve kterých byla měření vkládána do submap, jsou uchovávány pro využití optimalizačním algoritmem uzavírání smyček. Optimalizace je řešena jako metoda nejmenších čtverců

$$\underset{\boldsymbol{p}_m,\boldsymbol{p}_s}{\operatorname{argmin}} = \frac{1}{2}\sum_{ij}\rho(E^2(\boldsymbol{\xi}_i^m,\boldsymbol{\xi}_j^s,\boldsymbol{\sigma}_{ij},\boldsymbol{\xi}_{ij})), \tag{2.15}$$

kde $\boldsymbol{p}_m = \{\xi_i^m\}_{i=1,\dots,l}$ jsou polohy submap v mapě, $\boldsymbol{p}_s = \{\xi_i^s\}_{i=1,\dots,n}$ jsou polohy měření v mapě, $\boldsymbol{\xi}_{ij}$ jsou relativní polohy submap a měření a $\boldsymbol{\sigma}_{ij}$ je jejich kovariance. Funkce E vyjadřuje chybu v určení poloh submap a skenů. Ve výpočtu je použita Huberova ztrátová funkce ρ, která minimalizuje účinek špatných měření na odhad mapy.

Pro urychlení výpočtu optimalizace je využita metoda větví a mezí. Hypotézy jsou uloženy do stromové struktury, jejíž kořen obsahuje všechny možnosti a obsahy potomků uzlu formují rodičovský uzel. Každý list tohoto stromu obsahuje jednu možnost řešení. Pro procházení stromu je použita metoda prohledávání do hloubky. Efektivita výpočtu je navýšena předpočítáváním mřížkových map pro různé hladiny stromu. Tyto předpočítané mapy mají stejné rozlišení jako výsledná mapa, ale každý blok mapy o velikosti $2^{c_h} \times 2^{c_h}$ je vyplněn maximální hodnotou získanou v daném bloku. Na obrázku 2.3 jsou předpočítané mapy zobrazeny. Optimalizace je prováděna s periodou několika sekund. Uživatel může tedy sledovat vývoj mapy a její úpravy v reálném čase.



Obrázek 2.3: Mapy pro velikosti bloků 1, 4, 16 a 64. Převzato z [5]

3 Experimenty a jejich porovnání

První sekce této kapitoly se zabývá Robot Operating Systemem (ROS), který byl použit jako framework pro testování výše zmíněných implementací. Dále jsou zde popsána testovací data a senzory, které byly použité k jejich zaznamenání. Je zde uveden způsob extrakce trajektorie z jednotlivých implementací řešení SLAM problému a příprava naměřených dat pro porovnání. Následně je popsán způsob výpočtu chyb testovaných implementací. Na závěr je poskytnuto porovnání implementací. Informace o užívání jednotlivých implementací byly získány z dokumentace [20].

3.1 Robot Operating System

> *"The Robot Operating System (ROS) is a framework that is widely used in Robotics. The philosophy is to make a piece of software that could work in other robots with only little changes to the code. What we get with this idea is the ability to create functionalities that can be shared and used in other robots without effort, so we do not need to reinvent the wheel."* [Citováno z 2, str. 2]

Jako framework pro provedení experimentů na testovaných implementacích řešení SLAM problému je v této práci použit ROS. Informace pro tuto sekci byly čerpány z [2, 14]. ROS byl vyvinut roku 2007 v Stanford Artificial Intelligence Laboratory. Při výrobě robota je užito mnoho součástek od různých výrobců. Komunikace s takovými součástkami může probíhat různými způsoby v různých jazycích a po přidání nebo výměně jedné části existuje riziko nutnosti přepsat velkou část robotova programu. Přínosem ROSu je zavedení sjednocené komunikace mezi částmi robota. Poskytuje abstrakci hardwaru, nízkoúrovňové ovládání zařízení a podobné funkcionality operačního systému. ROS byl vydán jako open source software pod BSD licencí.

Data jsou v ROSu uspořádána do balíčků. Balíček je zde nejnižší strukturou schopnou vytvořit program a obsahuje zdrojový kód jednoho nebo více ROS runtime procesů, tzv. uzlů (nodes). Uzel představuje spustitelný kód obecně obsahující jednoduchou funkcionalitu. ROS je postaven jako modulární systém a program je v něm poskládán z mnoha uzlů, které mezi sebou komunikují v rámci peer-to-peer sítě. Komunikace mezi uzly je zajištěna tzv. publish-subscribe modelem pomocí topiců a zpráv. Topic má obdobnou funkci jako datová sběrnice. Uzel může do topicu s určitým jménem publikovat zprávu, kterou obdrží odběratel daného topicu. Do daného topicu může publikovat několik uzlů a stejně tak z něj může několik uzlů

odebírat zprávy. Publikující a odebírající uzly o sobě nemají žádné explicitní informace, pojí je jen název topicu, do kterého publikují nebo z něj odebírají.

Data jsou rozesílána ve formě zpráv s určitým datovým typem. V ROSu je implementováno několik datových typů používaných v běžných programovacích jazycích. Kromě těchto typů lze deklarovat vlastní datové typy. Uzly mohou přijímat zprávy pouze s datovým typem s jehož podobou jsou seznámeny. Vedle běžného uzlu ROS nabízí tzv. nodelet. Tento uzel zpracovává několik uzlů a zajišťuje jejich vzájemnou komunikaci tak, aby se zamezilo přílišnému zatížení sítě, po které uzly komunikují, např při zpracovávání objemných dat.

Topicy jsou využívány pro běžnou komunikaci mezi uzly a v případě, kdy je potřeba získat z uzlu určité informace pro další zpracování mimo ROS, nejdou použít. Pro tuto interakci s uzly jsou v ROSu implementovány služby. Služba je specifikována unikátním názvem a dvěma datovými typy. Jeden udává datový typ požadavku, druhý udává datový typ odpovědi.

ROS byl vyvinut jako jazykově neutrální systém. Pro deklaraci zpráv využívá tzv. interface definition language (IDL). IDL lze pomocí generátorů kódu převést do jazyka vhodného pro koncové použití. Mezi tyto jazyky patří např. C++, Python, OCtave a LISP.

Při běhu programu v ROSu lze veškeré publikované zprávy uložit k pozdějšímu přehrání. Tato uložená data tak umožňují opakovat jistý scénář bez nutnosti znovu provádět experiment. Lze je využít k urychlení ladění algoritmu, pro které tak není nutné opakovat daný experiment. Dalším způsobem využití je porovnávání různých přístupů k řešení úlohy, jako tomu bylo učiněno v této práci. Open-source povaha ROSu umožňuje komukoliv přispívat k jeho rozvoji. Provozuje síť repozitářů, ve které různé instituce vyvíjí vlastní softwarové komponenty, které jsou poté v rámci ROSu zpřístupněny. Software je v repozitářích uložen v podobě výše zmíněných balíčků.

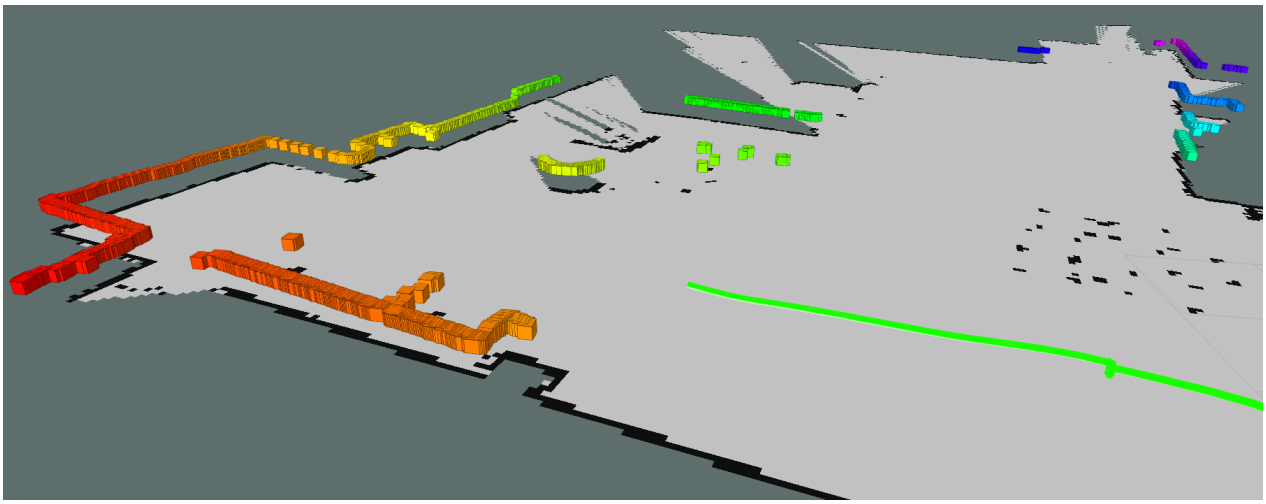
Spolu s ROSem je distribuován vizualizační program rviz. Rviz poskytuje uživateli vizuální zpětnou vazbu o tom, v jakém stavu se momentálně systém nachází. Umožňuje zobrazit polohu robota, jeho způsob vnímání okolí, uraženou trajektorii apod. Další nástroj rxgraph slouží zobrazení uzlů a jejich vzájemné komunikace v grafu. Většina robotických systémů potřebuje sledovat vzájemné pozice určitých modulů. Může se jednat např. o manipulátory a potřebu sledování manipulovaných objektů vzhledem k poloze manipulátoru, nebo o sledování polohy mobilního robota v mapě při lokalizaci. V ROSu je implementován transformační systém s názvem tf právě pro tyto účely. Tf konstruuje strom souřadných systémů různých částí systému, ve kterém jsou uloženy jejich vzájemné polohy. Tento strom transformuje data rozesílaná mezi uzly tak, aby odpovídala souřadnému systému příjemce. Lze tak např. převést data z LiDARu do souřadného systému mapy a následně je zobrazit pomocí rvizu. Pro tuto práci byla použita verze ROS Jade Turtle.

3.2 Testovací data a měřící zařízení

Jako testovací data pro porovnání implementací byly vybrány datasety z MIT Stata Center [19]. V datasetech jsou uložená data získaná v průběhu experimentu. Takový experiment je pak možné simulovat znovu se stejnými výsledky, což umožňuje porovnat různé metody rešení dané úlohy. Důvodem zvolení těchto datasetů bylo přiložení dat o skutečné poloze robota z průběhu měření, tzv. ground truth. Tyto data jsou použity pro porovnání efektivity implementací řešení SLAM problému. Udávaná maximální chyba v určení polohy při měření ground truth je 2 až 3 cm. Vybraná data jsou uložena ve formátu .bag, který je podporován frameworkem Robot Operating System popsaném v 3.1.

V použitých datasetech je uloženo několik typů dat. Těmito typy jsou: měření z LiDARu Hokuyo UTM-30LX Laser, odometrie robota, data z IMU a transformace tf. Transformace byly pro každou testovanou implementaci řešení SLAM problému vybrány tak, aby zaručovaly správnou funkci dané implementace na robotu provádějícím měření. Popsané typy dat jsou v každém MIT datasetu formulovány stejně, tudíž zde prezentované experimenty lze ve stejné formě aplikovat na všechny datasety.

Měřící rozsah použitého LiDARu je 270° a měřitelná vzdálenost se pohybuje v mezích od 0.1 m do 30 m. Mezera mezi jednotlivými naměřenými hodnotami činí 0.25° a udávané přesnost senzoru je ±30 mm při měření do 10 m vzdálenosti a ±50 mm při měření od 10 m do 30 m vzdálenosti [18]. Podobu měření získaných tímto LiDARem lze zhlédnout v obrázku 3.1, kde jsou znázorněny barevnými čtverečky.



Obrázek 3.1: Ukázka průběhu experimentu se zobrazenými daty z měření LiDARem

3.3 Příprava dat

Při provádění experimentu bylo třeba extrahovat odhad trajektorie z každé testované implementace a upravit je pro následné zpracování. Též bylo nutné transformovat data z ground truth, aby s nimi šly výsledky experimentů porovnat.

3.3.1 Extrakce trajektorie

Každá implementace přistupuje k generování trajektorie jiným způsobem. Hector SLAM publikuje topic `slam_out_pose`, který publikuje odhad polohy robota včetně jeho orientace. Zprávy z tohoto topicu byly uloženy pomocí příkazu

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

a použity jako trajektorie pro Hector SLAM. Hector SLAM díky svému online přístupu neimplementuje žádnou formu uzavírání smyček ani optimalizace uražené trajektorie nebo pozorované mapy, tudíž lze použít tento způsob získání trajektorie.

GMapping nepublikuje svoji trajektorii v přímé formě. Pro extrakci trajektorie byl využit `hector_trajectory_server` z Hector SLAMu, který zaznamenává trajektorii robota. Pomocí jeho služby `trajectory`

```
rosservice call /trajectory > gMapping_trajektorie.txt
```

byla získána trajektorie robota zaznamenána gMappingem.

Cartographer nabízí službu `finish_trajectory`

```
rosservice call /finish_trajectory,
```

která provede optimalizaci grafu a vrátí mapu ve formátu .pgm a trajektorii ve formátu .pb. Trajektorii lze zkompilovat a uložit pomocí příkazu

```
protoc --decode cartographer.mapping.proto.Trajectory -I. $(find . -name
'*.proto') < ~/.ros/map.pb
```

Každá implementace interpretuje natočení robota jiným způsobem. Byla provedena úprava úhlu natočení robota tak, aby ve všech trajektoriích se úhel natočení robota pohyboval v intervalu $(-\pi, \pi\rangle$.

3.3.2 Příprava ground truth

Počátek dat poskytnutých v ground truth byl vztažen k souřadnému systému budovy. Data byla orotována a přesunuta tak, aby počáteční stav robota $(x_{r,0}, y_{r,0}, \theta_{r,0})$ byl roven hodnotám $(0, 0, 0)$.

Časové okamžiky, pro které bylo poskytnuto ground truth se lišily od časových okamžiků, ve kterých testované implementace vracely odhad stavu robota. Byla provedena lineární interpolace dat ground truth do časových okamžiků, ve kterých testované implementace poskytují data o stavu robota. Interpolace odhadovaných poloh testovaných implementací do časových okamžiků ground truth byla jako řešení zavržena, protože ground truth poskytuje pro většinu systémů čtyřnásobek dat na stejný časový úsek. Zvolená interpolace je tedy přesnější.

3.4 Výpočet chyby

Pro výpočet chyby testovaných implementací řešení SLAM problému byly použity dva způsoby. První vypočtenou chybou je RMSE (Root Mean Square Error) pro určení polohy robota dle vzorce

$$e(\boldsymbol{p}_{r,k}, \boldsymbol{p}_{gt,k}) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(\|\boldsymbol{p}_{r,k} - \boldsymbol{p}_{gt,k}\|)^2}, \tag{3.1}$$

kde N je počet záznamů v trajektorii testované implementace, $\boldsymbol{p}_{r,k} = [x_{r,k}, y_{r,k}]$ udává odhad polohy robota testovanou implementací v časovém okamžiku k a $\boldsymbol{p}_{gt,k} = [x_{gt,k}, y_{gt,k}]$ je poloha robota v časovém okamžiku k v ground truth.

Tento výpočet chyby ale není dle [9] vhodný pro porovnávání efektivity řešení SLAM problému, protože chyba, která byla robotem provedena na začátku měření má mnohem větší dopad než stejná chyba, která byla provedena až před závěrem měření. Nabízeným řešením v [9] je výpočet chyby založený na relativních polohách jednotlivých odhadů a skutečných poloh

$$\varepsilon(\delta) = \frac{1}{N}\sum_{i,j}^{N} trans(\delta_{i,j} \ominus \delta^*_{i,j})^2 + rot(\delta_{i,j} \ominus \delta^*_{i,j})^2, \tag{3.2}$$

kde N je počet vzájemných relací mezi záznamy, $trans()$ a $rot()$ jsou funkce vyhodnocující chyby v translaci a rotaci robota, $\delta_{i,j}$ je relativní poloha i-tého a j-tého naměřeného záznamu stavu robota, $\delta^*_{i,j}$ je relativní poloha i-tého a j-tého záznamu z ground truth a i a j udávají měření, mezi kterými je počítána relativní poloha.

Zvolený vztah pro relativní polohy zahrnuje do výpočtu chyby i-tý záznam a záznam i+1, který mu v čase bezprostředně následuje. Jako funkce $trans()$ byla zvolena Eukleidovská norma

$$trans(\delta_{i,i+1} \ominus \delta^*_{i,i+1}) = \|\delta_{i,i+1} \ominus \delta^*_{i,i+1}\|. \tag{3.3}$$

Chyba $rot()$ byla zvolena ve tvaru

$$rot(\delta_{i,i+1} \ominus \delta^*_{i,i+1}) = |min(|\delta_{i,i+1} \ominus \delta^*_{i,i+1}|, 2\pi - |\delta_{i,i+1} \ominus \delta^*_{i,i+1}|)|. \tag{3.4}$$

Pro porovnání implementací řešení SLAM problému jsou dále využité tři chyby. Chyba v určení polohy robota $\varepsilon_{trans}(\delta)$

$$\varepsilon_{trans}(\delta) = \frac{1}{N}\sum_{i,i+1}^{N} trans(\delta_{i,i+1} \ominus \delta^*_{i,i+1})^2, \tag{3.5}$$

chyba v určení orientace robota $\varepsilon_{rot}(\delta)$

$$\varepsilon_{rot}(\delta) = \frac{1}{N}\sum_{i,i+1}^{N} rot(\delta_{i,i+1} \ominus \delta^*_{i,i+1})^2, \tag{3.6}$$

a chyba určení stavu robota $\varepsilon(\delta)$

$$\varepsilon(\delta) = \frac{1}{N}\sum_{i,i+1}^{N} trans(\delta_{i,i+1} \ominus \delta^*_{i,i+1})^2 + rot(\delta_{i,i+1} \ominus \delta^*_{i,i+1})^2. \tag{3.7}$$

3.5 Porovnání implementací

V této sekci jsou vyhodnoceny výsledky experimentů provedených na jednotlivých implementacích řešení SLAM problému. Získané mapy, naměřené trajektorie a vývoje chyb v čase jsou ke zhlédntí v dodatku A.1. Zdrojové kódy umožňující replikovat tyto experimenty jsou k dispozici na *https://github.com/struncp/Porovnani-SLAM-implementaci*. Tyto skripty lze po úpravě na použitého robota využít k reálné úloze SLAM.

3.5.1 Vyhodnocení experimentů

Pro jednotlivé implementace byly spočteny chyby dle postupů v sekci 3.4, jejichž výsledky jsou sepsány v tabulce 3.1. Vyhodnocení experimentů vychází z této tabulky, vygenerovaných map v obrázku A.1, odhadů trajektorie robota v obrázku A.2 a grafů v dodatku, které budou blíže specifikovány.

Tabulka 3.1: Chyby implementací řešení SLAM problému

	Hector SLAM	gMapping	Cartographer
Celková chyba určení polohy robota dle (3.1)	0.9612 m	0.6535 m	0.6469 m
Chyba určení polohy robota dle (3.5)	0.0064 m	0.0143 m	0.0123 m
Chyba určení orientace robota dle (3.6)	0.0079 rad	0.0305 rad	0.0351 rad
Chyba určení stavu robota dle (3.7)	0.0144	0.0447	0.0475
Směrodatná odchylka (3.7)	0.0164	0.1484	0.0404

Již bylo řečeno, že způsob výpočtu chyby dle (3.1) je nevyhovující pro porovnávání algoritmů řešících úlohu SLAM. Tato chyba však stále má jistou vypovídající hodnotu o tom, jak přesně daná implementace dokáže určit svoji trajektorii vzhledem k reálnému stavu. Průběh této chyby v čase pro každou implementaci lze zhlédnout v obrázku A.3, případně podrobněji v obrázku A.7. Lze si všimnout, že všechny implementace mají problémy s přesným určením polohy v časech mezi 5. a 8. minutou a mezi 17. a 27. minutou. V prvním časovém úseku je mapován koridor v levé části mapy a v druhém časovém úseku je mapován koridor v pravé části a okruh na jeho konci. Odometrie udává obecně méně přesné měření než LiDAR a testované implementace věří více datům poskytovaným LiDARem. V rovné chodbě s malým počtem záchytných bodů ovšem LiDAR měří data, která si jsou velmi podobná a nelze z nich řádně vyhodnotit stav robota. Koridor pak může být v mapě vyobrazen zkrácený oproti realitě [10]. Dle chyby (3.1) se pro daný scénář jeví Google Cartographer jako nejlepší řešení.

Průběh chyby (3.5) v čase je ke zhlédnutí na obrázcích A.4 a A.8. Zde je problém opět v 17. a 28. minutě, ve kterých jsou mapovány koridory. Průběh chyby (3.5) v čase je k dispozici na obrázcích A.5 a A.9. Zde pravděpodobně došlo ke špatnému způsobu extrakce dat o orientaci robota z experimentu provedeném s gMappingem. Všechny tři implementace jeví v určení orientace vyšší chybu mezi 13. a 17. minutou. V tomto časovém rozmezí mapování probíhá v horní části mapy. Pohyb v tomto užším prostoru vyžaduje od robota velké změny v orientaci. Uprostřed tohoto rozmezí se před 15. minutou nachází oblast, ve které všechny implementace vykazují téměř nulovou chybu. V tento okamžik robot stojí na místě.

Celkovou chybu testovaných implementací dle (3.7) lze zhlédnout v obrázcích A.6 a A.10. V obrázku A.1 je zřetelné zkrácení pravého koridoru u Hector SLAM oproti zbylým implementacím, což je způsobeno absencí back-endu u Hector SLAMu, který by řešil uzavírání smyček. Dle tabulky 3.1 je však za použití tohoto kriteria nejvíce vyhovujícím řešením právě Hector SLAM. Riziko zkrácení koridoru nejlépe zvládl Cartographer, jehož trajektorie se ve výběžku za koridorem kryje s ground truth nejlépe ze všech implementací.

Závěr

Cílem práce bylo bližší představení problému Simultánní Lokalizace a Mapování. Toho bylo dosaženo popsáním funkce několika přístupů k řešení SLAM problému, konkrétně přístupů využívajících Kalmanův filtr, částicový filtr nebo grafovou reprezentaci. Dalším cílem bylo porovnat efektivitu určitých implementací řešení SLAM problému dle správnosti jejich odhadu trajektorie robota. Těmito implementacemi byly Hector SLAM, gMapping a Google Cartographer. Byly vysvětleny přístupy těchto implementací k řešení SLAM problému a pro jejich zprovoznění byl do práce zařazen Robot Operating System, který slouží jako framework pro programování robotických systémů. Po provedení experimentů byly vyhodnoceny jejich výsledky.

Experimenty v praktické části prokázaly, že pro mapování budov se z testovaných implementací nejlépe hodí Hector SLAM. Dále bylo zjištěna slabina Hector SLAMu v jeho absenci back-endu, což mu znemožňuje opravit chybná měření. Tento jev vede především k problémům v dlouhých koridorech bez záchytných bodů, které jsou do mapy zakresleny zkrácené. V případě mapování budovy s dlouhými koridory je z popsaných implementací optimální řešením Google Cartographer. Všechny tři testované implementace se projevily jako použitelné v praxi.

Výsledky dosažené v uskutečněných experimentech vypovídají pouze o schopnosti dané implementace odhadnout stav robota a trajektorii, kterou urazil. Tento přístup k vyhodnocování efektivity daných implementací v řešení SLAM problému jistě lze vylepšit. Rozšířením tohoto přístupu by samozřejmě mohlo být ohodnocení přesnosti vygenerované mapy nebo porovnání hardwarové náročnosti jednotlivých implementací. Mapa vygenerovaná gMappingem v obrázku A.1 též naznačuje možnost vylepšení v nastavení rozeznávání chybných měření. Byla ověřena funkčnost použitých skriptů. Tyto skripty lze po úpravě pro určitého robota použít k reálnému řešení SLAM problému.

Zdroje

[1] DURRANT-WHYTE, Hugh a Tim BAILEY, 2006. Simultaneous Localization and Mapping: Part 1 *IEEE Robotics & Automation Magazine*. June 2006. 99-108. ISSN 1070-9932

[2] FERNÁNDEZ, Enrique, Luis Sánchez Crespo, Anil Mahtani a Aaron Martinez, 2015. *Learning ROS for Robotics Programming*. Vyd. 2. Birmingham: Packt Publishing Ltd. ISBN 978-1-78398-758-0

[3] GRISETTI, Giorgio, Cyrill Stachniss a Wolfram Burgard, 2007. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters *IEEE Transactions on Robotics*. February 2007. 34-46. ISSN 1941-0468

[4] GRISETTI, Giorgio, Rainer Kümmerle, Cyrill Stachniss a Wolfram Burgard, 2010. A Tutorial on Graph-Based SLAM *IEEE Intelligent Transportation Systems Magazine*. č. 4. 31 - 43 ISSN 1939-1390

[5] HESS, Wolfgang, Damon Kohler, Holger Rapp a Daniel Andor, 2016. Real-Time Loop Closure in 2D LIDAR SLAM *2016 IEEE International Conference on Robotics and Automation (ICRA)*. ISBN 978-1-4673-8026-3

[6] HÖGMAN, Virgile, 2012. *Building a 3D map from RGB-D sensors*. Diplomová práce. Stockholm: Royal Institute of Technology, Computer Science. Vedoucí práce Alper Aydemir

[7] KOHLBRECHER, Stefan, Oskar von Stryk, Johannes Meyer a Uwe Klingauf, 2011. A Flexible and Scalable SLAM System with Full 3D Motion Estimation *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*. ISBN 978-1-61284-769-6

[8] KORKMAZ, Mehmet, Nihat Yılmaz a Akif Durdu, 2016. Comparison of the SLAM algorithms: Hangar experiments *MATEC Web of Conferences*. č. 42. ISSN 2261-236X

[9] KÜMMERLE, Rainer, Bastian Steder, Christian Dornhege, Michael Ruhnke, Giorgio Grisetti, Cyrill Stachniss a Alexander Kleiner, 2009. On Measuring the Accuracy of SLAM Algorithms *Autonomous Robots*. č.4. 387-407. ISSN 0929-5593

[10] LEINGARTNER, Max, Johannes Maurer, Gerald Steinbauer, Alexander Ferrein, 2015. Evaluation of Sensors and Mapping Approaches for Disasters in Tunnels *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. ISBN 978-1-4799-0880-6

[11] MILSTEIN, Adam a Xing-Jian Jing, 2008. Occupancy Grid Maps for Localization and Mapping *Motion Planning*. 381-408. INTECH Open Access Publisher. ISBN 978-953-7619-01-5

[12] OLSON, Edwin B., 2009. Real-Time Correlative Scan Matching *2009 IEEE International Conference on Robotics and Automation, ICRA '09*. 4387-4393. ISSN 1050-4729

[13] RIISGARD, Søren a Morten Rufus BLAS, 2003. *SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*

[14] QUIGLEY, Morgan, Brian Gerkey, Ken Conley, Josh Faust, Tully Foote, Jeremy Liebs, Eric Berger, Rob Wheeler a Andrew Ng, 2009. ROS: an open-source Robot Operating System *ICRA workshop on open source software*. č. 3.2

[15] WAN, Eric A. a Rudolph van der Merwe, 2000. The Unscented Kalman Filter for Nonlinear Estimation *Adaptive Systems for Signal Processing, Communications, and Control Symposium 2000. AS-SPCC. The IEEE 2000*. 153-158. ISBN 0-7803-5800-7

[16] Autonomous Intelligent Systems, University of Freiburg ©2013 [online]. http://ais.informatik.uni-freiburg.de/

[17] Ceres Solver - A Large Scale Non-linear Optimization Library ©2016 [online]. http://ceres-solver.org/

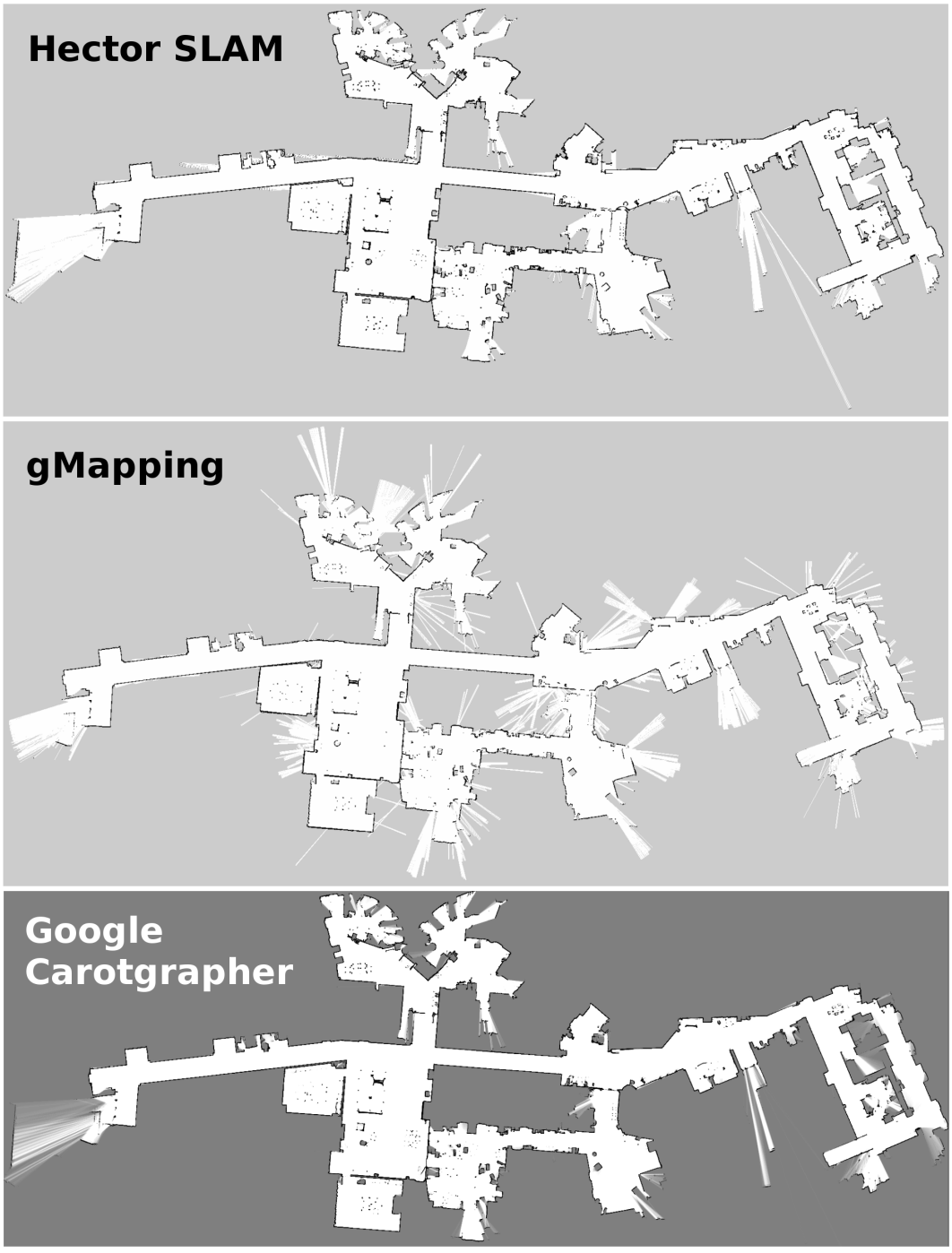
[18] HOKUYO AUTOMATIC CO.,LTD. ©2014 [online]. https://www.hokuyo-aut.jp/

[19] MIT Stata Center Data Set ©2012 [online]. http://projects.csail.mit.edu/stata/index.php

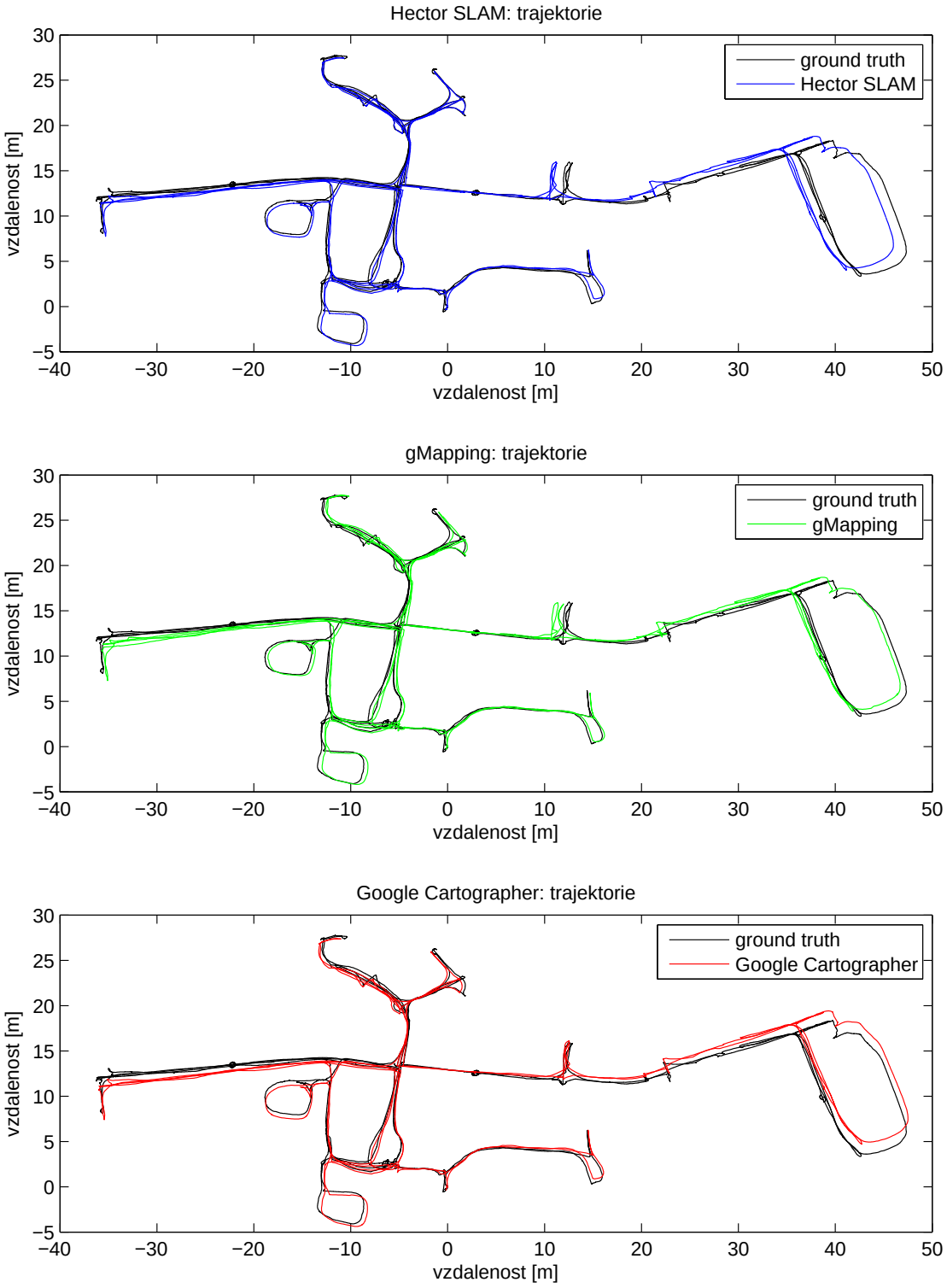
[20] ROS Wiki ©2017 [online]. http://wiki.ros.org/

A Dodatky

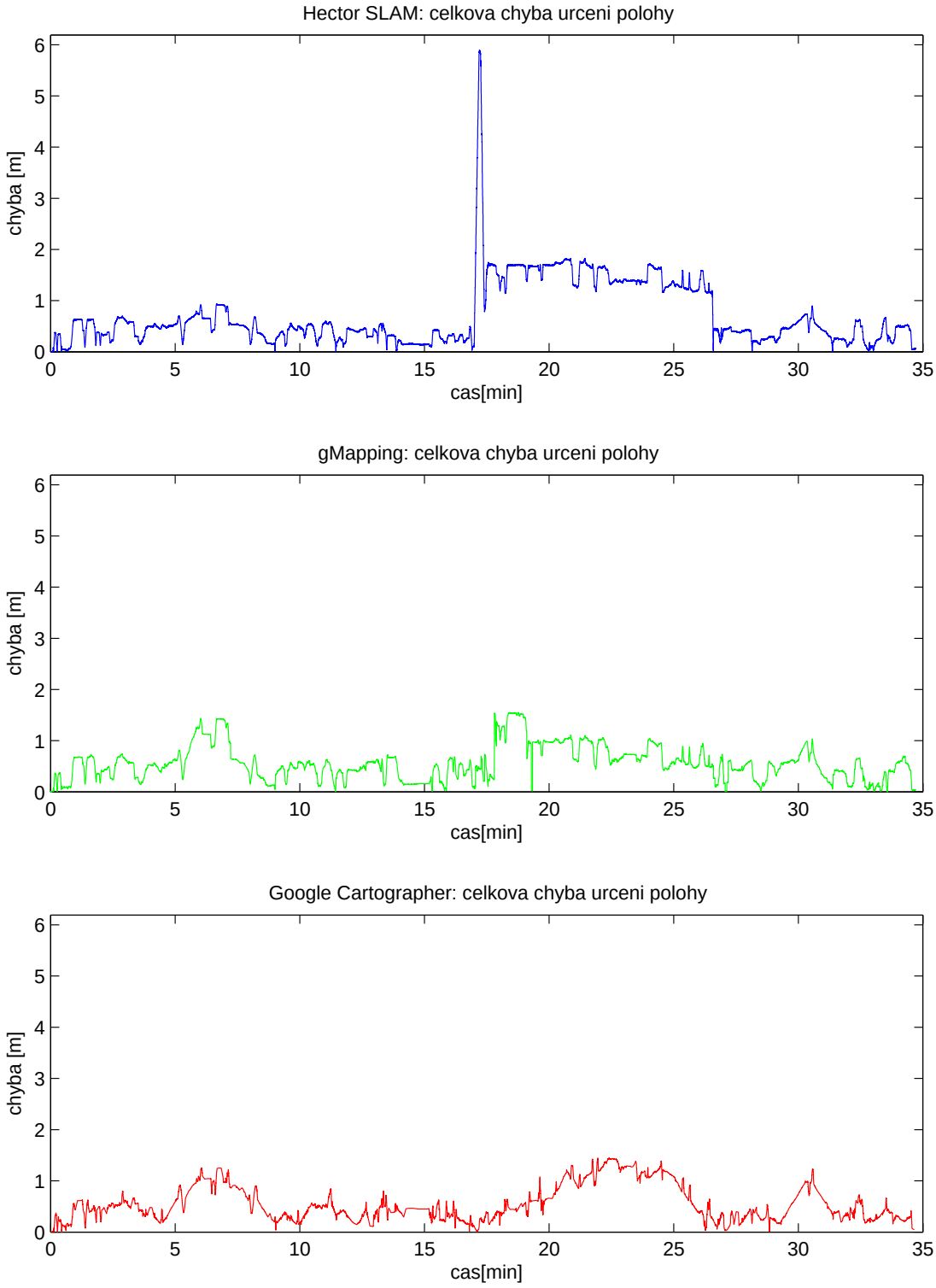
A.1 Výsledky experimentů



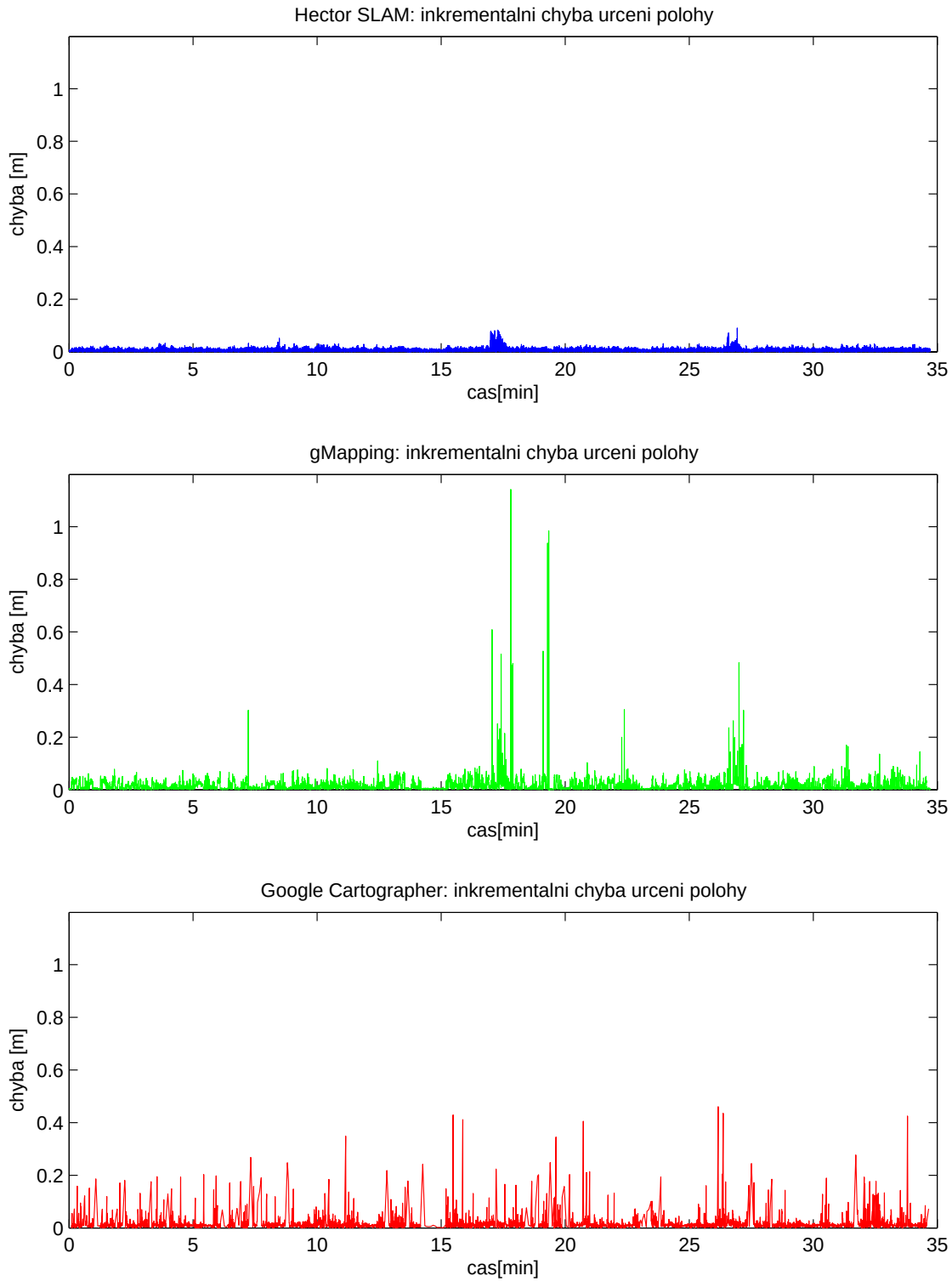
Obrázek A.1: Mapy získané jednotlivými implementacemi



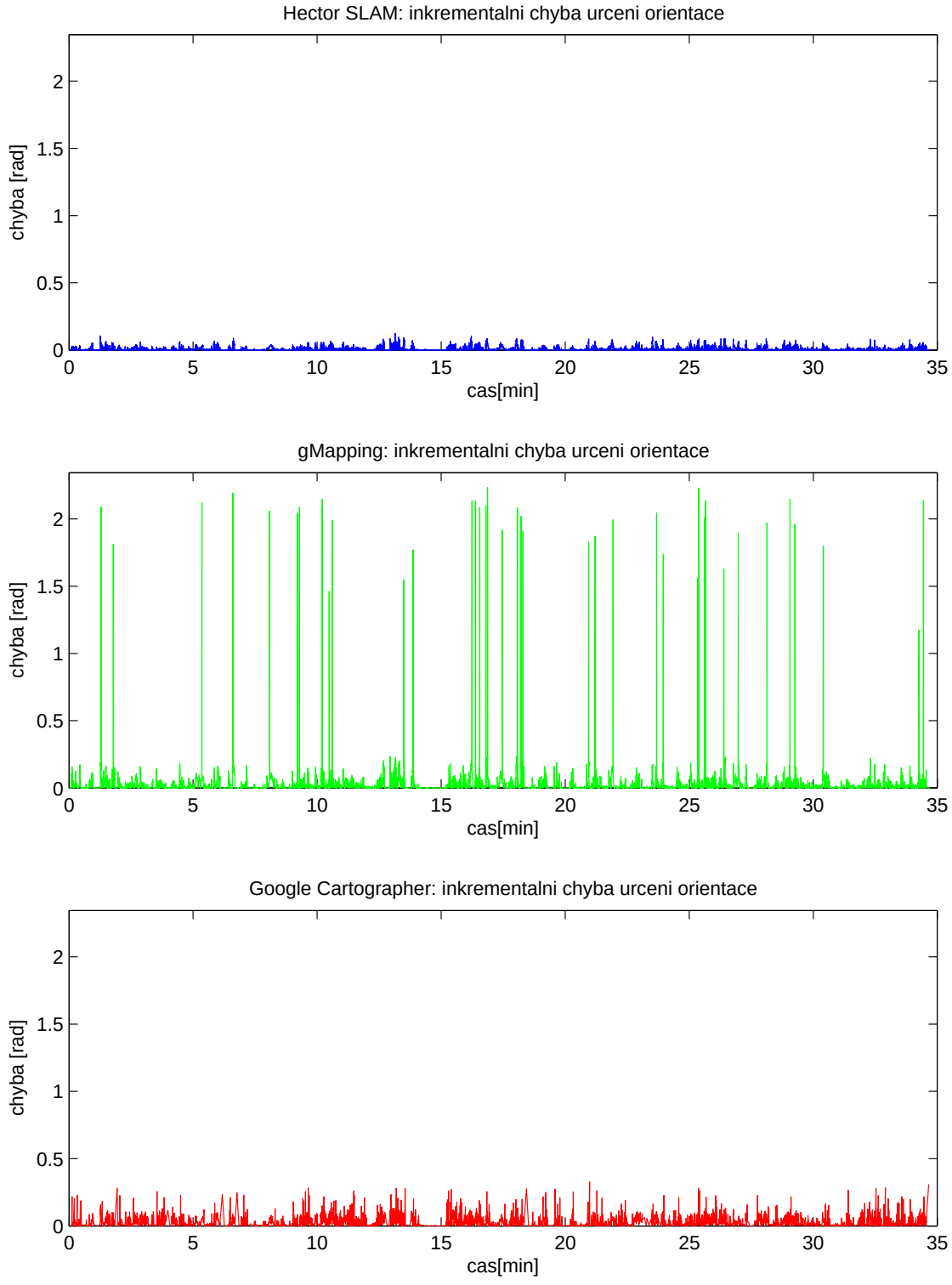
Obrázek A.2: Trajektorie naměřené implementacemi řešení SLAM problému



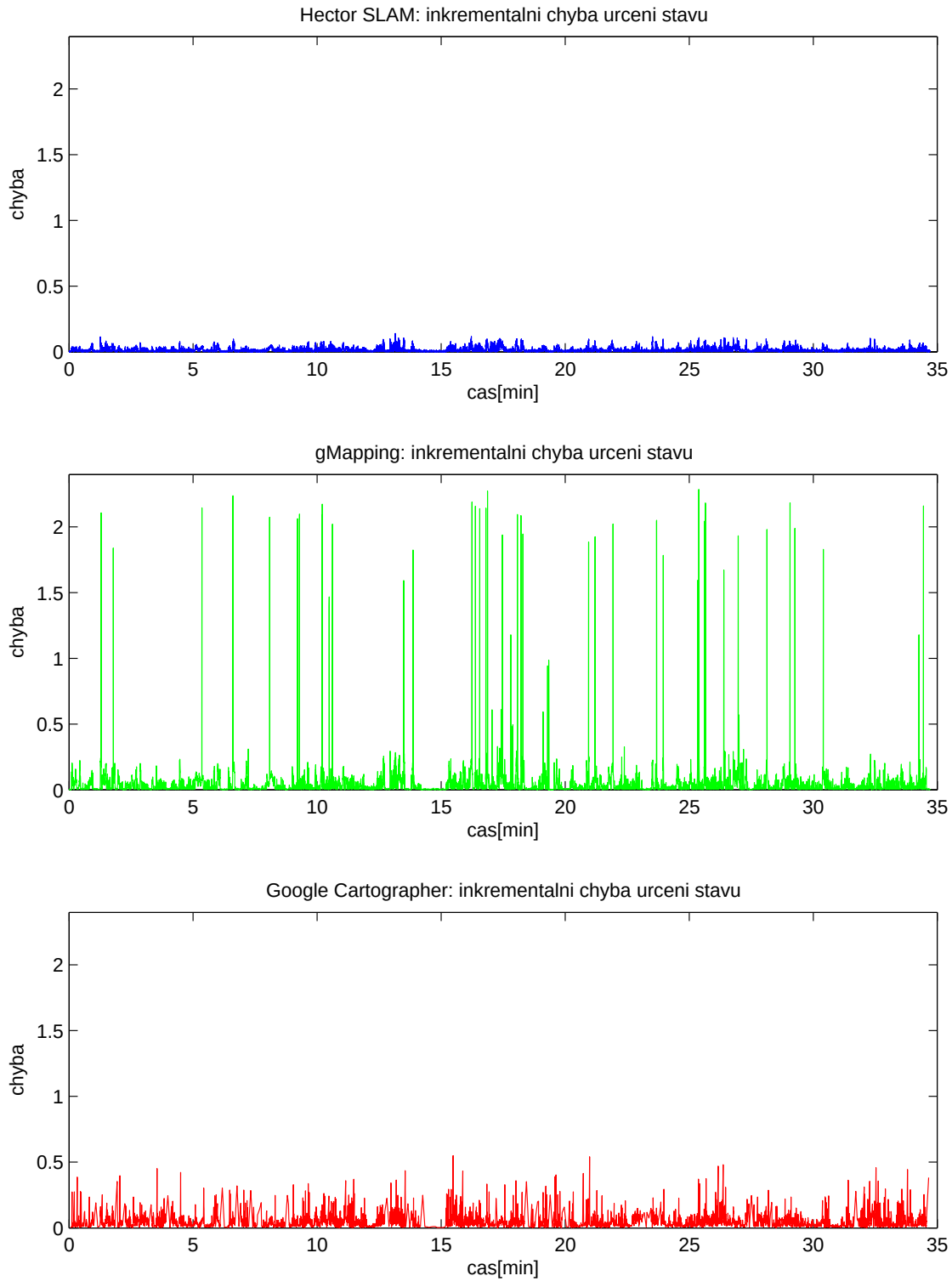
Obrázek A.3: Vývoj celkové chyby určení polohy robota dle (3.1) v čase



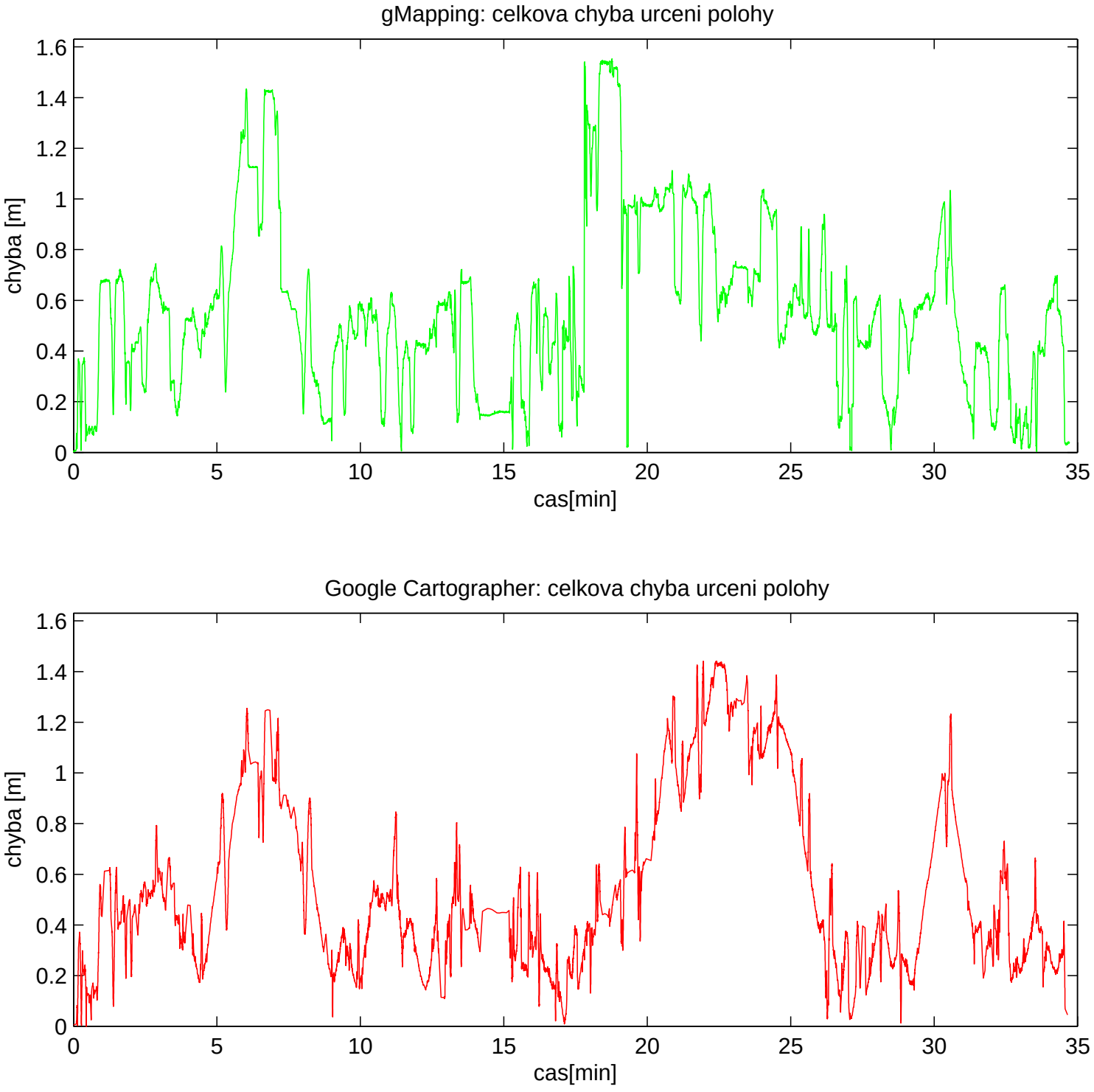
Obrázek A.4: Vývoj chyby určení polohy robota dle (3.5) v čase



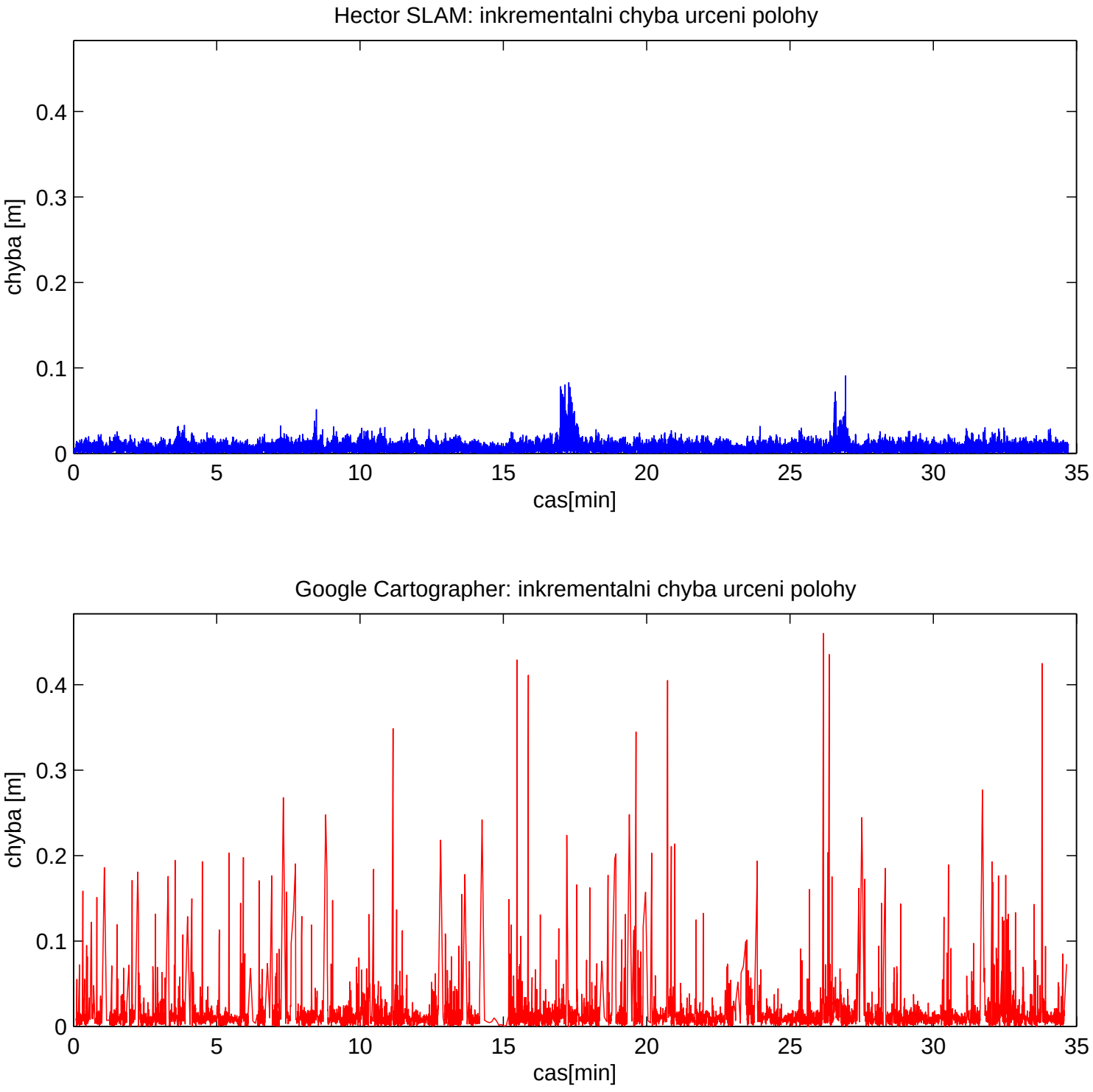
Obrázek A.5: Vývoj chyby určení orientace robota dle (3.6) v čase



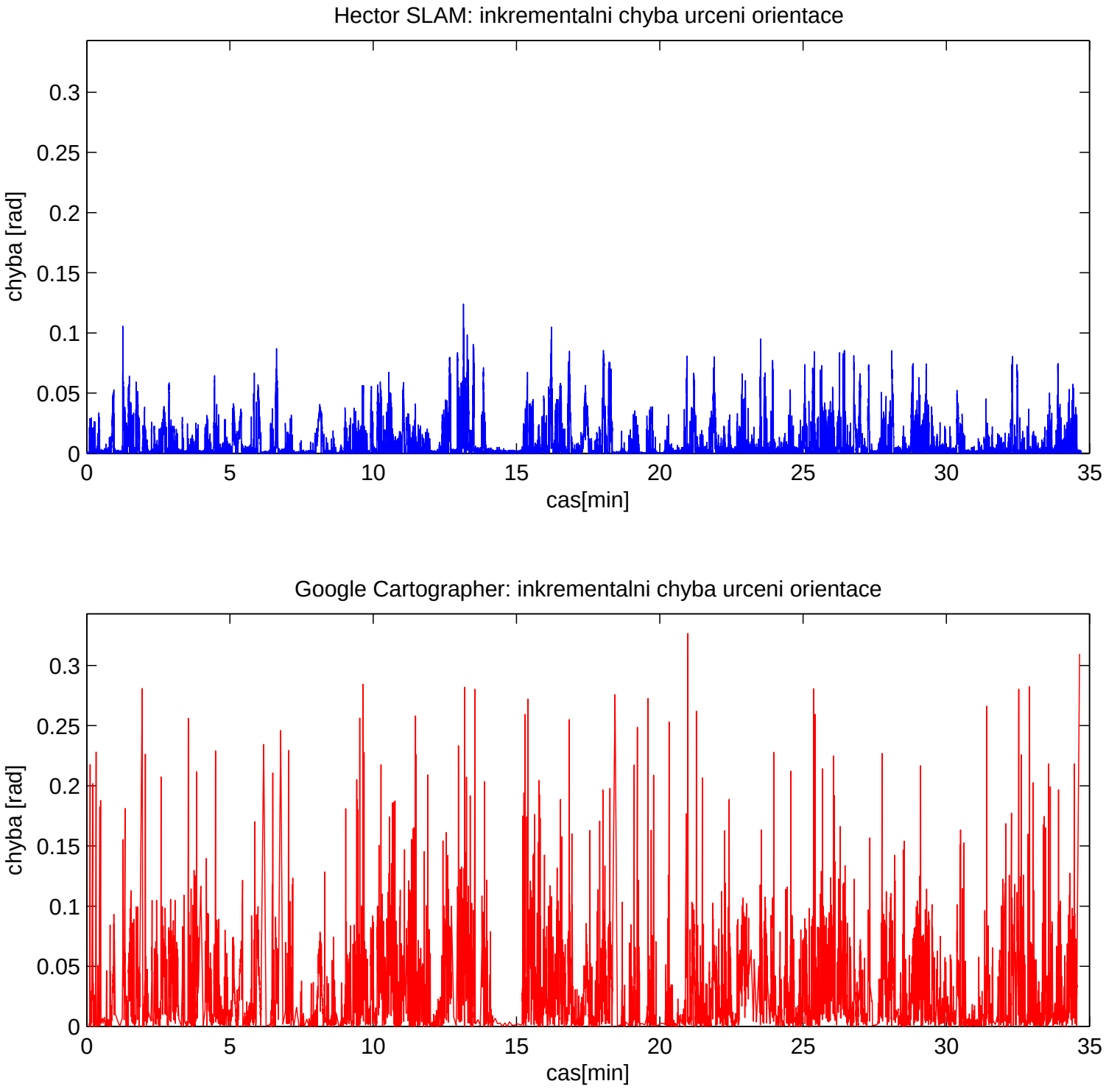
Obrázek A.6: Vývoj chyby určení stavu robota dle (3.7) v čase



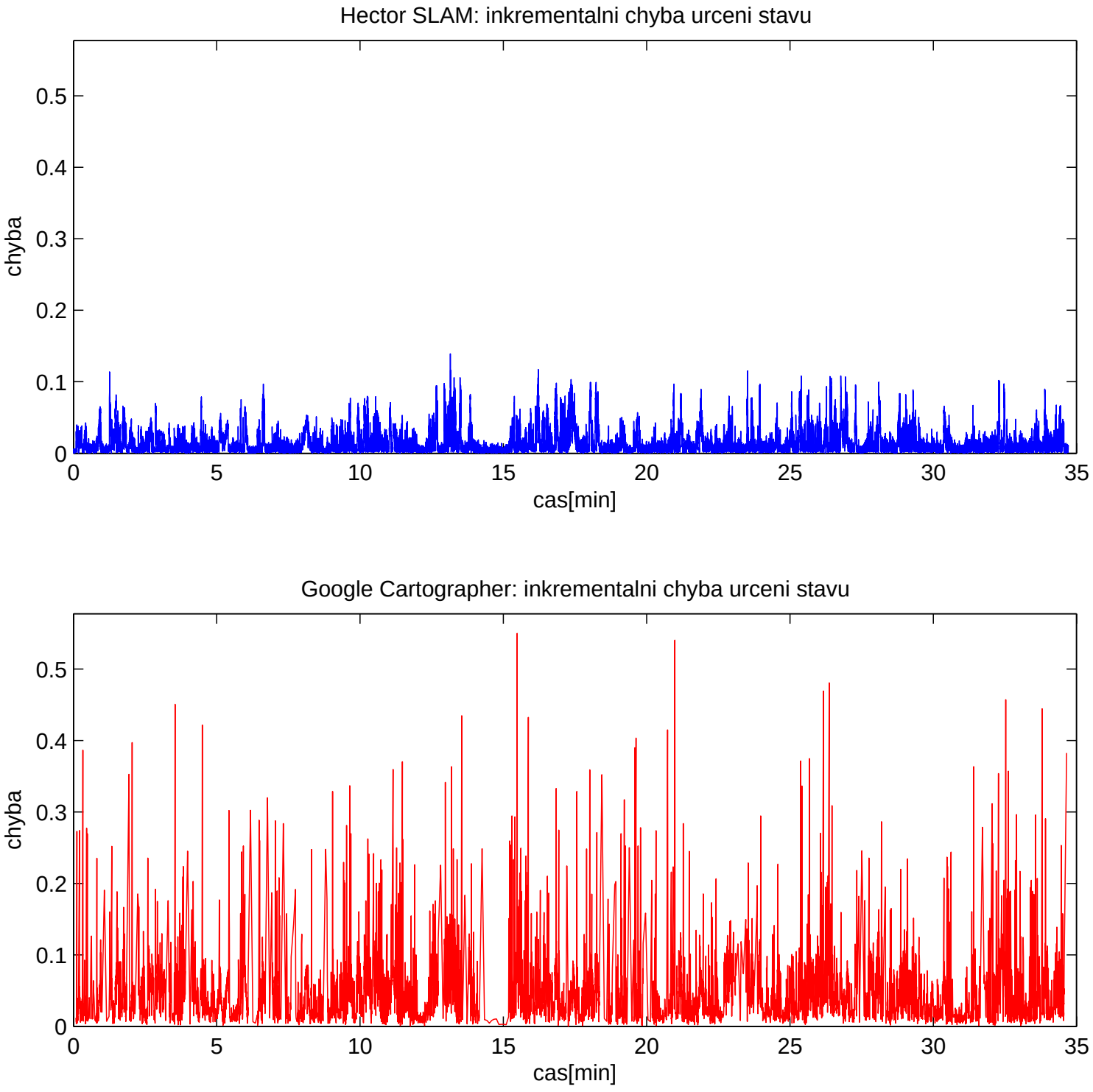
Obrázek A.7: Vývoj celkové chyby určení polohy robota dle (3.1) v čase, detail gMapping a Google Cartographer



Obrázek A.8: Vývoj chyby určení polohy robota dle (3.5) v čase, detail Hector SLAM a Google Cartographer



Obrázek A.9: Vývoj chyby určení orientace robota dle (3.6) v čase, detail Hector SLAM a Google Cartographer



Obrázek A.10: Vývoj chyby určení stavu robota dle (3.7) v čase, detail Hector SLAM a Google Cartographer