FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

Bakalářská práce

Simulátory přístupů k úloze Simultánní lokalizace a mapování vytvořené pro systém ROS2

Ondřej Valtr

PLZEŇ

2023

FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

KATEDRA
KYBERNETIKY

Bakalářská práce

Simulátory přístupů k úloze Simultánní lokalizace a mapování vytvořené pro systém ROS2

Ondřej Valtr

Vedoucí práce
Ing. Petr Neduchal, Ph.D.

PLZEŇ **2023**

Citace v seznamu literatury:
VALTR, Ondřej. *Simulátory přístupů k úloze Simultánní lokalizace a mapování vytvořené pro systém ROS2*. Plzeň, 2023. Bakalářská práce. Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Katedra kybernetiky. Vedoucí práce Ing. Petr Neduchal, Ph.D.

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Ondřej VALTR**
Osobní číslo:	**A21B0456P**
Studijní program:	**B0714A150005 Kybernetika a řídicí technika**
Specializace:	**Umělá inteligence a automatizace**
Téma práce:	**Simulátory přístupů k úloze Simultánní lokalizace a mapování vytvořené pro systém ROS2**
Zadávající katedra:	**Katedra kybernetiky**

Zásady pro vypracování

1. Proveďte rešerši v oblasti přístupů k úloze simultánní lokalizace a mapování (SLAM).
2. V rešerši se zaměřte na SLAM ve 2D prostoru vytvářející zejména mapy tvořené jednotlivými významnými body v prostoru (tzv. landmarky).
3. V softwarovém rámci ROS2 vytvořte sadu simulátorů ukazující funkcionalitu přístupů a porovnejte přesnost přístupů.

Rozsah bakalářské práce: **30-40 stránek A4**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

1. Durrant-Whyte, H., & Bailey, T. (2006). Simultaneous localization and mapping: part I. IEEE robotics & automation magazine, 13(2), 99-110.
2. Stachniss, C., Leonard, J. J., & Thrun, S. (2016). Simultaneous localization and mapping. In Springer Handbook of Robotics (pp. 1153-1176). Springer, Cham.

Vedoucí bakalářské práce: **Ing. Petr Neduchal**
Výzkumný program 1

Datum zadání bakalářské práce: **17. října 2022**
Termín odevzdání bakalářské práce: **22. května 2023**

Západočeská univerzita v Plzni
Fakulta aplikovaných věd
L.S.

Doc. Ing. Miloš Železný, Ph.D.
děkan

Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 17. října 2022

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 9. srpna 2023

. . . . . .

Ondřej Valtr

Abstrakt

Tato bakalářská práce se věnuje problematice Simultánní lokalizace a mapování. Úloha je korektně formulována a je uvedena její matematická pravděpodobnostní podoba, jak bývá definována a se kterou je dále pracováno. Rozsáhlá část práce popisuje tři základní přístupy, pomocí nichž lze tuto úlohu řešit. Tyto přístupy byly následně implementovány v jazyce Python. Následně je problém zjednodušen na 2D případ (pozice robota v rovině s úhlem značícím jeho orientaci). Vytvořené zdrojové kódy byly začleněny do struktury systému ROS 2, což vyžadovalo jejich úpravu. V ROSu 2 tak vznikla jednoduchá aplikace, která výsledné simulátory založených na daných přístupech spouští a poskytuje data pro jejich porovnání. Posledním krokem bylo výsledné trajektorie vykreslit pomocí nástroje ROSu 2 pro 3D vizualizaci robotických aplikací: RViz 2. Hlavním přínosem této práce je implementace a otestování systémů řešících Simultánní lokalizaci a mapování v rámci ROS 2.

Abstract

This bachelor thesis deals with the problem of Simultaneous Localization and Mapping. The problem is correctly formulated and its mathematical probabilistic form, as it is usually defined, is given and further worked with. An extensive section of the thesis describes three basic approaches by which the problem can be solved. These approaches were afterwards implemented in Python. The problem is then simplified to the 2D case (the position of the robot in a plane with an angle denoting its orientation). The formed source codes were integrated into the ROS 2 system structure, which required their modifications. This resulted in a simple application in ROS 2 that launches the final simulators based on the given approaches and provides data for their comparison. The final step was to render the resulting trajectories by using ROS 2's 3D visualization tool for robotic applications: RViz 2. The main contribution of this work is the implementation and testing of systems solving Simultaneous Localization and Mapping within ROS 2.

Klíčová slova

landmark • pozice robota • odhadování stavu • hustota pravděpodobnosti • simulace • Python • Robot Operating System 2

Poděkování

Chtěl bych zde poděkovat panu Ing. Petru Neduchalovi, Ph.D. za odborné vedení při tvorbě této bakalářské práce, za vstřícnost, trpělivost a čas, který mi věnoval. Dík také patří tvůrci hezké a perfektně fungující šablony pro jednotný vzhled závěrečných prací FAV ZČU, která byla pro tvorbu této práce použita, panu Ing. Kamilu Ekšteinovi, Ph.D.

Obsah

Úvod 1

Tato bakalářská práce je členěna (kromě tohoto krátkého Úvodu a samozřejmě Závěru) do dvou hlavních kapitol: do Teoretické a Praktické části. Jejich názvy plně vystihují myšlenku tohoto rozdělení. Aby se znovu neopakovaly stejné části textu, jsou předměty jejich náplně vždy uvedeny až na začátku těchto sekcí.

1.1 Simultánní lokalizace a mapování

Simultánní lokalizace a mapování (dále jen SLAM) je dnes již zavedená, samostatná část robotiky. Byla formulována a řešena jako teoretický problém v různých formách. V teoretické a návrhové rovině může dnes být považován za problém vyřešený. Nicméně značné problémy přetrvávají v praktické realizaci obecnějších řešení SLAMu (zejména ve tvoření a používání bohatých map). Může být implementován v řadě odlišných prostředí, např. ve vnitřních a venkovních prostorech, pod vodou, ve vzdušných systémech atd. Jak již bylo zmíněno, v této práci se pojednává o třech základních výpočetních řešeních: Rozšířený Kalmanův filtr (EKF-SLAM), Rao-Blackwellized částicový filtr (FASTSLAM) a Unscented Kalman Filter (UKF-SLAM). Jedná se o principiální přístupy, ze kterých mohou vycházet a stavět na nich pokročilejší algoritmy vhodnějších do prostředí reálného nasazení. Po velice stručné zmínce o historii následuje popsání (formulace a struktura) SLAM problému a jeho výše uvedených základních metod řešení, jsou ukázány konkrétní implementace.

1.2 Historie

Jako počátek pravděpodobnostního SLAMu lze označit rok 1986, kdy se v San Franciscu konala IEEE Robotics and Atomation Conference. Byly zde uvedeny klíčové myšlenky a potenciální problémy. Robot pohybující se v neznámém prostředí vyhledává významné body (landmarky). Bylo představeno, že úplné konzistentní řešení lokalizačního a mapovacího problému by vyžadovalo stav složený z pozice robota a aktuální pozice významných bodů (a s tím spojená aktualizace při každém pozorování), odhady těchto bodů jsou totiž nutně navzájem korelované kvůli chybě

v odhadu pozice robota. Vznikl by tak obrovský stavový vektor (řád roven počtu bodů v mapě). Muselo by se počítat s výpočetní složitostí odhadování v kvadrátu počtu těchto bodů. Dále tedy bylo nutné problém aproximovat (aby byl problém konvergentní). Korelace mezi landmarky byla minimalizována nebo úplně zanedbána, takže byl plnohodnotný filtr zredukován. Do filtru tak vstupovala série oddělených landmarků. Problém tedy byl rozdělen na dvě části: buď probíhalo mapování nebo lokalizace. Za těchto předpokladů dohromady vznikla konvergentní úloha. Posléze však přišel myšlenkový zlom. Návrat k myšlence sloučení mapování a lokalizace zpět do jedné odhadovací úlohy, protože se přišlo na to, že i tento problém je konvergentní. Nejdůležitější bylo zjištění, že čím větší je korelace mezi jednotlivými landmarky (kterou se do této chvíle snažili všichni výzkumníci minimalizovat), tím lepší řešení je nakonec dosahováno.

Teoretická část 2

Tato kapitola přináší teoretické základy k SLAM problematice. Zadefinuje samotný pojem a popisuje matematické pozadí této úlohy. Hned následující podkapitolou je tedy Formulace a struktura SLAM úlohy. Dále je zadefinováno jedno z klíčových slov této práce - landmark a krátká podkapitola se věnuje také nezbytnému úseku tohoto problému a to sice měření vzdálenosti - výčtu základních senzorů v této oblasti. Rozsáhlá část této kapitoly se zabývá popisem základních přístupů k řešení SLAM úlohy. Poslední podkapitola uvádí vybrané metriky umožňující porovnání dosažených výsledků ze SLAM algoritmů.

2.1 Formulace a struktura SLAM úlohy

SLAM je proces, u kterého mobilní robot tvoří mapu prostředí a zároveň tuto mapu používá k určení své pozice v ní. Při tomto odhadování nemá žádnou apriorní informaci. Odhaduje se jak trajektorie platformy, tak umístění všech landmarků.

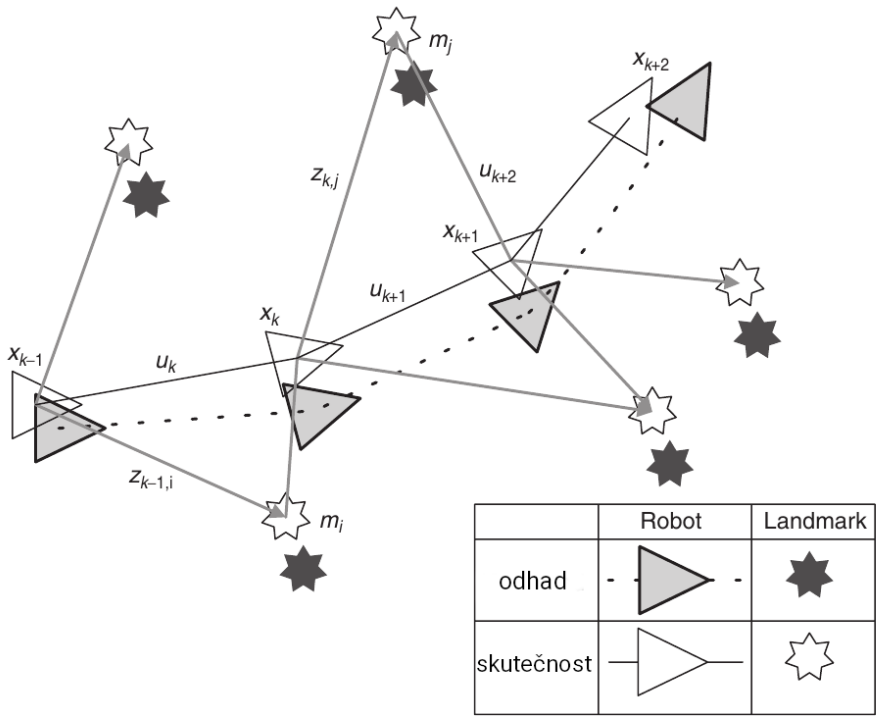
2.1.1 Definice proměnných

v určitém čase k platí:

- $\vec{x_k}$: stavový vektor popisující umístění a orientaci robota
- $\vec{u_k}$: vektor řízení použitý v čase $k-1$ pro dosažení stavu robota $\vec{x_k}$
- $\vec{m_i}$: vektor pozice *i-tého* landmarku, jehož skutečná pozice je předpokládána jako pevná (neměnící se v čase)
- $\vec{z_{ik}}$: vektor pozorování (z robota) *i-tého* landmarku, když během jednoho pozorování zpozorováno více landmarků, psáno do jednoho vektoru $\vec{z_k}$
- $\mathbf{X_{0:k}} = \{\vec{x_0}, \vec{x_1}, ..., \vec{x_k}\} = \{\mathbf{X_{0:k-1}}, \vec{x_k}\}$: množina, historie pozice robota
- $\mathbf{U_{0:k}} = \{\vec{u_1}, \vec{u_2}, ..., \vec{u_k}\} = \{\mathbf{U_{0:k-1}}, \vec{u_k}\}$: množina, historie řízení

- $\mathbf{m} = \{\vec{m}_1, \vec{m}_2, ..., \vec{m}_n\}$: množina všech landmarků
- $\mathbf{Z_{0:k}} = \{\vec{z}_1, \vec{z}_2, ..., \vec{z}_k\} = \{\mathbf{Z_{0:k-1}}, \vec{z}_k\}$: množina všech pozorování (landmarků)

Problém je znázorněn na obrázku 2.1. Původní obrázek z [1]. Je vyžadováno simultánní odhadování jak pozice robota, tak landmarků. Skutečná umístění však nikdy známa nejsou. Objevují se totiž odchylky vzniklé z nepřesností měření.



Obrázek 2.1: Zobrazení základní SLAM problematiky

2.1.2 Pravděpodobnostní SLAM

V této formě vyžaduje počítání hustoty pravděpodobnosti ve všech časových okamžicích k:

$$p(\vec{x}_k, \mathbf{m} | \mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \vec{x}_0) \tag{2.1}$$

Toto pravděpodobnostní rozdělení popisuje sdruženou hustotu pravděpodobnosti umístění landmarků a stavu vozidla v čase k. Je podmíněna získanými pozorováními a řídícími vstupy do (včetně) okamžiku k a počátečním stavem vozidla. Je tedy vyžadováno rekurzivní řešení. Začíná se odhadováním hustoty pravděpodobnosti $p(\vec{x}_{k-1}, \mathbf{m} | \mathbf{Z_{0:k-1}}, \mathbf{U_{0:k-1}})$ pro čas $k-1$, sdružená posteriorní pravděpodobnost

pro vstup $\vec{u_k}$ a pozorování $\vec{z_k}$ je počítána podle Bayesova vztahu. Tento výpočet vyžaduje stavový pohybový model a model pozorování.

Pohybový model *(motion model)*.
hustota pravděpodobnosti

$$p(\vec{x_k}|\vec{x}_{k-1}, \vec{u_k}) \tag{2.2}$$

Jedná se o Markovův proces. To je takový proces, pro který platí:

$$p(\vec{y_k}|\vec{y}_{k-1}, \vec{y}_{k-2}, ..., \vec{y}_0) = p(\vec{y_k}|\vec{y}_{k-1})$$

Čili pro znalost $\vec{y}$ v čase k stačí znát pouze $\vec{y}$ v čase $k-1$ (nezáleží, jak se proces do tohoto stavu dostal, na historii vývoje, postačí jen stav o jeden časový krok starší). V případě pohybového modelu se samozřejmě musí znát ještě řízení.

Model pozorování *(observation model)*.
Popisuje pravděpodobnost vytvoření pozorování $\vec{z_k}$, známa pozice vozidla a umístění landmarků:

$$p(\vec{z_k}|\vec{x_k}, \mathbf{m}) \tag{2.3}$$

SLAM algoritmus je tedy implementován ve standardní dvou-krokové rekurzivní (sekvenční) prediktivní (predikce: tzv. *časová aktualizace: time-update*) korektivní (korekce: tzv. *aktualizace z měření, z pozorování: measurement-update*) formě [1].

Možnost rozdělení úlohy na mapovací a lokalizační problém.

Problém sestavení mapy může být také formulován jako vypočtení podmíněné hustoty pravděpodobnosti:

$$p(\mathbf{m}|\mathbf{X_{0:k}}, \mathbf{Z_{0:k}}, \mathbf{U_{0:k}})$$

Nutný předpoklad tohoto vyjádření však je znalost umístění vozidla robota $\vec{x_k}$ ve všech časových okamžicích k, včetně počáteční pozice. Mapa $\mathbf{m}$ je pak zkonstruována sloučením pozorování z odlišných míst.

Podobně, lokalizační problém by mohl být také formulován:

$$p(\vec{x_k}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \mathbf{m})$$

Tentokrát se předpokládá, že umístění landmarků jsou **přesně** známa.

Struktura pravděpodobnostního SLAMu.

Tato část se zabývá zejména chybami odhadu. Chyba odhadu pozice landmarku je u všech pozorovaných landmarků stejná, podobná (pouze posun ve stejném směru, viz obrázek 2.1): chyby v odhadech umístění jednotlivých landmarků jsou tedy velmi korelované. Z toho plyne, že relativní umístění (vzdálenost) mezi dvěma landmarky i a j, v zavedeném značení: $\vec{m}_i - \vec{m}_j$, je vysoce přesné, i když odhadovaná samostatná absolutní umístění $\vec{m}_i$ a $\vec{m}_j$ jsou docela nepřesná. Znamená to, že sdružená hustota pravděpodobnosti pro pár landmarků $p(\vec{m}_i, \vec{m}_j)$ je velmi "špičatá", i když marginální hustoty $p(\vec{m}_i)$ a $p(\vec{m}_j)$ můžou být docela "rozprostřené, rozložené".

> Důležitý náhled na SLAM: Je třeba si uvědomit, že čím více pozorování je provedeno, tím více roste (monotónně) korelace mezi odhady landmarků. Prakticky to znamená, že odhad relativního umístění landmarků mezi sebou se vždy vylepšuje, nikdy nediverguje (nezáleží na pohybu vozidla). Z toho vyplývá, že čím více pozorování je uděláno, tím více se (monotónně) sdružená hustota pravděpodobnosti pro všechny landmarky $p(\mathbf{m})$ stává "špičatější".

Tato konvergence nastává, protože pozorování robota mohou být považována za "skoro"nezávislá měření relativního umístění (vzdálensoti) mezi landmarky. Jak už bylo zmíněno, relativní umístění pozorovaných landmarků jsou nezávislá na koordinaci vozidla robota.

Vše může být ilustrováno na *ukázkovém příkladě:* Uvažováno umístění robota $\vec{x}_k$. Robot pozoruje dva landmarky (s pozicemi $\vec{m}_i$ a $\vec{m}_j$). Robot se pohne do $\vec{x}_{k+1}$ a znovu pozoruje landmark $\vec{m}_j$. To umožňuje odhadovanou pozici robota a landmarku aktualizovat relativně k předchozí pozici $\vec{x}_k$. Tato změna se propaguje, zpětně aktualizován landmark $\vec{m}_i$ (i když tento landmark není z nové pozice viditelný). Děje se to proto, že tyto dva landmarky jsou vysoce korelované (jejich relativní vzájemné pozice jsou dobře známy) z předchozího měření. Dále fakt, že ta samá naměřená data jsou použita pro aktualizaci těchto dvou landmarků, je dělá více korelovanými. Je tedy na místě připomenout termín "skoro"nezávislých pozorování. Při zaznamenání nových landmarků v pozici $\vec{x}_{k+1}$ jsou tyto landmarky rovnou přiřazeny (tzn. korelovány) do zbytku mapy. Pozdější aktualizace těchto landmarků aktualizuje také landmark na pozici $\vec{m}_j$ a skrz něj landmark $\vec{m}_i$ atd. Kdykoliv je tedy uděláno nové pozorování, přesnost se zvyšuje.

Další intuitivní příklad na pochopení těchto faktů, je možnost představit si mapu (umístění robota a jednotlivých landmarků) jako vzájemně propojenou síť pružin, které představují korelace (jako analogii). Čím více je pružina silnější, tím značí větší korelaci mezi landmarky (případně mezi landmarkem a robotem). Silnější pružinu lze hůře natáhnout: tedy význam to má takový, že nové pozorování relativní umístění změní jen velice málo, zanedbatelně - typicky v případě, kdy pozorováních bylo

již hodně. Nové pozorování se zpětně propaguje do celé sítě propojených pružin (odhad pozic je v průběhu času upravován). Silná pružina se vyskytuje mezi vzájemně blízkými landmarky, naopak slabé pružiny (menší korelace) se často vyskytují mezi landmarky a umístěním robota.

2.2 Landmarky

Jedno z nejvíce frekventovaných slov v této práci je bezpochyby landmark. V této části tedy bude pojem vysvětlen.

> **Landmarky - významné body**, které mohou být v prostředí snadno opakovaně pozorovány a vyznačovány. Jsou použity robotem ke zjištění, kde se nachází (sebe lokalizace). Výběr landmarků závisí na tom, v jakém prostředí robot operuje.

2.2.1 Klíčové body pro výběr vhodných landmarků (dle [2]):

- Mělo by je být možno snadno opakovatelně pozorovat (z jiných pozic, úhlů).
- Měly by být dostatečně unikátní, aby byly v průběhu času snadno identifikovatelní. Dva landmarky by nemělo být možno zaměnit mezi sebou, měly by také být od sebe dostatečně vzdáleni.
- Mělo by jich být dostatek, aby se robot „neztratil“ (mohl se podle nich neustále orientovat).
- Měly by být statické (nehybné).

2.3 Senzory pro měření vzdálenosti

Vzdálenost od landmarků je v reálném prostředí potřeba něčím měřit. Tato práce se primárně věnuje simulačnímu řešení úlohy SLAMu. Nicméně je vhodné zmínit používané senzory alespoň v krátkém přehledu.

2.3.1 LIDAR (laserový skener)

LIDAR (*Light Detection And Ranging - světelná detekce a měření rozsahu*) je velmi přesný a efektivní senzor. Výstup nepotřebuje mnoho výpočetního výkonu. Zároveň je však velmi drahý. Špatné výsledky má v nestandardních prostředí, především pak skrz sklo nebo ve vodě.

2.3.2 Sonar

Už dlouhou historii se jedná o nejlepší řešení do vody (inspirováno přírodou, např. delfíni). Zároveň nepatří mezi nejdražší senzory. Avšak v jiných prostředí než ve vodě dosahuje oproti laserovému senzoru horších výsledků.

2.3.3 Kamera

Tato technologie jako senzor zaznamenala velký rozmach zejména díky zvýšení výpočetního výkonu a implementací pokročilých algoritmů. Mizí tak nevýhoda velké náročnosti zpracování. Jedná se o intuitivní přístup (vidění podobné jako u člověka). Kromě toho, v obraze se vykytuje více informací, dat. Jako nevýhoda se přeci jen ještě uvádí již zmíněná výpočetní složitost zpracování. Především pak ale závislost na světelných podmínkách. Např. chyba způsobená změnou světelných podmínek, nebo přímo tma (v principu nemožnost fungování).

2.4 Řešení SLAM úlohy

Vyžaduje hledání vhodné reprezentace obou modelů (pohybového a modelu pozorování) pro efektivní a konzistentní počítání hustot pravděpodobností:

$$p(\vec{x_k}, \mathbf{m}|\mathbf{Z_{0:k-1}}, \mathbf{U_{0:k}}, \vec{x_0}) \tag{2.4}$$

predikce: tzv. časová aktualizace (time-update) - Na základě pohybového modelu se predikuje v jaké pozici se robot v příštím kroku bude nacházet.

$$p(\vec{x_k}, \mathbf{m}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \vec{x_0}) \tag{2.5}$$

korekce: tzv. aktualizace z měření na základě pozorování $\vec{z_k}$ (measurement update) - Úprava, korekce odhadu dle pozorování. [1]

Jeden způsob je reprezentace stavovým modelem s aditivním Gaussovským šumem. To vede na použití EKF (viz část níže) řešení SLAM problému. Ovšem důležitá je také alternativní reprezentace. Je totiž možné popsat pohybový model (2.2) jako množinu vzorků více obecného (ne-Gaussovského) rozdělení pravděpodobnosti. To vede na použití FASTSLAM algoritmu.

Tři základní způsoby řešení problému jsou: EKF-SLAM, FastSLAM, UKF-SLAM. Následuje teorie k těmto přístupům.

2.4.1 EKF-SLAM

Extended Kalman Filter - Rozšířený Kalmanův filtr

Cílem je popsat pohybový model 2.2 ve formě:

$$p(\vec{x}_k|\vec{x}_{k-1},\vec{u}_k) \Leftrightarrow \vec{x}_k = f(\vec{x}_{k-1},\vec{u}_k) + \vec{w}_k \tag{2.6}$$

kde vektorová funkce *f()* popisuje pohyb vozidla, $\vec{w}_k$ je aditivní Gaussovský pohybový šum (jehož jednotlivé složky jsou nekorelované) s nulovou střední hodnotou a s kovarianční maticí $\mathbf{Q_k}$.

Poté je potřeba popsat i model pozorování 2.3 ve formě:

$$p(\vec{z}_k|\vec{x}_k,\mathbf{m}) \Leftrightarrow \vec{z}_k = h(\vec{x}_k,\mathbf{m}) + \vec{v}_k \tag{2.7}$$

kde vektorová funkce *h()* popisuje, jaké pozorování ze vstupních argumentů (pozice robota a landmarků v čase *k*) vznikne a jednotlivé složky vektoru $\vec{v}_k$ jsou opět aditivní Gaussovské šumy (vzájemně nekorelované) s nulovou střední hodnotou a s kovarianční maticí $\mathbf{R_k}$; mají význam pozorovací chyby (chyby měření).

S těmito rovnicemi (2.6 a 2.7) lze vypočítat požadovaná umístění robota a landmarků jako výsledek EKF metody, která může být aplikována defacto ke spočtení průměru. Použijí se k tomu všechna dosud nasbíraná měření. Výsledek metody tedy bude mít následující matematický význam:

$$\begin{bmatrix} \hat{\vec{x}}_{k|k} \\ \hat{\mathbf{m}}_\mathbf{k} \end{bmatrix} = E\left[\begin{matrix} \vec{x}_k \\ \mathbf{m} \end{matrix}|\mathbf{Z_{0:k}}\right] \tag{2.8}$$

K tomu příslušná kovarianční matice má tvar:

$$\mathbf{P}_{k|k} = \begin{bmatrix} \mathbf{P}_{xx} & \mathbf{P}_{xm} \\ \mathbf{P}_{xm}^T & \mathbf{P}_{mm} \end{bmatrix}_{k|k} = E\left[\begin{pmatrix} \vec{x}_k - \hat{\vec{x}}_k \\ \mathbf{m} - \hat{\mathbf{m}}_\mathbf{k} \end{pmatrix}\begin{pmatrix} \vec{x}_k - \hat{\vec{x}}_k \\ \mathbf{m} - \hat{\mathbf{m}}_\mathbf{k} \end{pmatrix}^T |\mathbf{Z_{0:k}}\right] \tag{2.9}$$

Význam jednotlivých podmatic:
$\mathbf{P}_{xx}$ ... kovarianční matice pro stavový vektor robota (jednotlivých prvků $\vec{x}_k$)
$\mathbf{P}_{xm}$ ... kovarianční matice pro stavový vektor robota s jednotlivými složkami pozic landmarků
$\mathbf{P}_{mx} = \mathbf{P}_{xm}^T$ ... vyplývá z vlastnosti kovarianční matice; stejná jako kovarianční matice $\mathbf{P}_{xm}$, pouze transponovaná
$\mathbf{P}_{mm}$ ... kovarianční matice pro složky pozic landmarků

Tyto pravděpodobnostní ukazatele po spočítání slouží k nahrazení sdružené posteriorní hustoty pravděpodobnosti $p(\vec{x}_k, \mathbf{m}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \vec{x}_0)$: 2.1.

Nyní tedy stačí výše zavedené proměnné 2.8, 2.9 dvěma kroky (predikce 2.4 a korekce 2.5) vypočítat. Postup řešení podle [1].

Predikce (časová aktualizace) - dle toho, jak je pohybový model pro EKF-SLAM 2.6 definován, tak nejlepším odhadem $\vec{x}_k$ je výsledek funkce *f()* bez přidaného šumu (střední hodnota Gaussovského šumu je totiž rovna 0), tedy:

$$\vec{\hat{x}}_{k|k-1} = f(\vec{\hat{x}}_{k-1|k-1}, \vec{u}_k) \tag{2.10}$$

Pozn.: Dolní indexy $k|k-1$ značí fázi výpočtu, kdy se přechází z časového okamžiku $k-1$ do okamžiku k (aplikace predikce - časové aktualizace); tedy $k|k$ značí již finální dopočtenou hodnotu (po korekci z měření).

Obecně se nemusí vyhodnocovat predikce pro nepohybující se landmarky (naopak nutné pro ty, které se pohybovat můžou). V této práci jsou landmarky brány jako nehybné, tudíž pro vypočtení časové aktualizace kovarianční matice $\mathbf{P}$ stačí přepočítat jen podmatici $\mathbf{P}_{xx}$. K vypočtení se použije Jakobián funkce *f()*: $\mathbf{H_f}$ vyhodnocený v předchozím odhadu $\vec{\hat{x}}_{k-1|k-1}$, čili:

$$\mathbf{P}_{xx,k|k-1} = \mathbf{H}_f \mathbf{P}_{xx,k-1|k-1} \mathbf{H}_f^T + \mathbf{Q_k} \tag{2.11}$$

Korekce (aktualizace z měření) - na základě těchto vztahů:

$$\begin{bmatrix} \vec{\hat{x}}_{k|k} \\ \mathbf{\hat{m}_k} \end{bmatrix} = [\vec{\hat{x}}_{k|k-1} \mathbf{\hat{m}_{k-1}}] + \mathbf{W_k}[\vec{z}_k - h(\vec{\hat{x}}_{k|k-1}, \mathbf{\hat{m}_{k-1}})] \tag{2.12}$$

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{W_k} \mathbf{S_k} \mathbf{W_k^T} \tag{2.13}$$

kde matice $\mathbf{S_k}$ a $\mathbf{W_k}$ jsou spočteny pomocí Jakobiánu $\mathbf{H}_h$ funkce *h()*, vyhodnocený v $\vec{\hat{x}}_{k|k-1}$ a $\mathbf{\hat{m}_{k-1}}$:

$$\mathbf{S_k} = \mathbf{H}_h \mathbf{P}_{k|k-1} \mathbf{H}_h^T + \mathbf{R_k} \tag{2.14}$$

$$\mathbf{W_k} = \mathbf{P}_{k|k-1} \mathbf{H}_h^T \mathbf{S_k^{-1}} \tag{2.15}$$

Matice $\mathbf{W_k}$ se nazývá Kalmanovo zesílení a $\mathbf{S_k}$ je inovace (inovační kovarianční matice) [2]. Složky Kalmanova zesílení indikují, jak moc pozici robota a pozice všech landmarků korigovat, aktualizovat (jak velká korekce nakonec bude, něco jako míra důvěry v provedené měření) - v rovnici 2.12 část $\mathbf{W_k}[\vec{z}_k - h(\vec{\hat{x}}_{k|k-1}, \mathbf{\hat{m}_{k-1}})]$: od získaného pozorování $\vec{z}_k$ se odečítá očekávané měření, které je výsledkem funkce *h()* vyhodnocené ve odhadnutém stavu $\vec{x}_{k|k-1}$, získaného z předchozí časové aktualizace 2.10 (předpokládaná pozice robota).

EKF-SLAM fáze.

Přehled iterativního postupu:

1. Prvním krokem je získání dat z okolí robota: dle použitého senzoru, např. laserová data.

2. V reálném nasazení je důležitým aspektem SLAMu tzv. odometrie, odometrická data. Jejich cílem je poskytnout informaci o přibližném pohybu robota např. pomocí měření pohybu kol robota nebo informací přímo použitých k řízení. Slouží jako počáteční odhad místa, kde by se robot mohl nacházet. Je důležité, aby laserová a odometrická data byla vztažena ke stejnému časovému okamžiku.

3. *Asociace dat:* problém správného přiřazení naměřených dat (údajných pozic landmarků) s již naměřenými v minulosti (opětovné pozorování landmarků v jiném čase, z jiného místa, pod jiným úhlem). I když je k dispozici velmi dobrý algoritmus extrakce landmarků, lze se dostat do problémů špatným výběrem landmarků nedodržením bodů zmíněných v seznamu 2.2.1. Je proto vhodné mít nastavenou strategii asociace dat pro minimalizování těchto problémů. Špatná asociace dat může mít za následek dokonce to, že robot bude udávat svojí polohu takovou, ve které ve skutečnosti není. Jedna z technik např. může být přístupem nejbližšího souseda, kdy se přiřazuje landmark nejbližšímu (nejpodobnějšímu) landmarku v databázi určené pro uchovávání již nalezených landmarků (počítáno např. pomocí Euklidovské vzdálenosti).

4. *Aplikace EKF:* používán pro odhad stavu (pozice) robota z odometrických dat a z pozorování landmarků. Poté odhadovány i samotné pozice landmarků.

   Přehled EKF procesu:

   4.1. Aktualizace odhadu aktuálního stavu pomocí údajů odometrie (predikce - časová aktualizace 2.10): sečten původní odhad stavu se složkami, které jsou výsledkem aplikací řízení na dosažení další (dílčí) požadované pozice. Např. pozice robota je dána dvěma souřadnicemi x, y (2D, plocha) a jeho natočením θ (takto bude také uvažován a zadefinován stav robota v praktické části):

   $$\vec{\hat{x}}_{k-1|k-1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

   Výsledek tohoto kroku tedy je nový stav robota (v předchozí části značen jako $\vec{\hat{x}}_{k|k-1}$), lze zapsat jako:

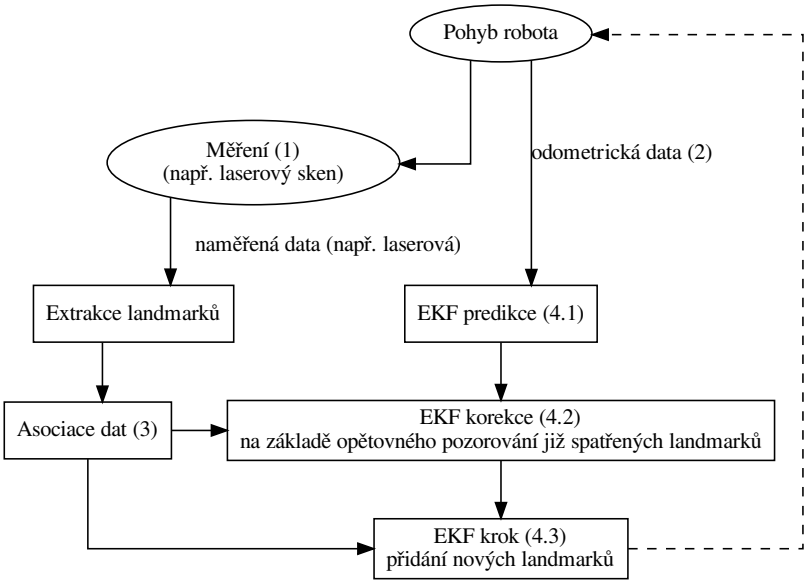
   $$\vec{\hat{x}}_{k|k-1} = \begin{bmatrix} x + dx \\ y + dy \\ \theta + d\theta \end{bmatrix}$$

   4.2. Korekce odhadnutého stavu ($\vec{\hat{x}}_{k|k-1}$) pomocí opětovně pozorovaných landmarků 2.12: odhad současné pozice lze využít k odhadnutí toho,

kde by se daný landmark měl nyní nacházet. Určitě však zde bude nějaký rozdíl oproti naměřeným datům – kvůli tzv. inovaci. Inovace je (stručně řečeno) rozdíl mezi odhadnovanou a skutečnou (spíše naměřenou - chyby v měření) pozicí robota. Je také aktualizována neurčitost každého landmarku: pokud je neurčitost současné naměřené pozice landmarku velmi malá, snižuje tak neurčitost celkovou (každým dalším měřením se rozptyl pozice - lépe řečeno rozptyly jednotlivých složek pozice - landmarku snižuje, tím pádem se zvyšuje jeho „určitost“).

4.3. Do současného celkového stavu jsou přidány nově nalezené (poprvé spatřené) landmarky. Celkový stav – robotova mapa (použitím informace o současné pozici robota a přidáním informace o vztahu mezi novými a starými landmarky).

Pro lepší představu byl vytvořen diagram s výše uvedenými fázemi: 2.2. Pro tvorbu schématu byl využit nástroj *Graphviz*: `https://graphviz.org/`.



Obrázek 2.2: Schéma fází iterativního postupu (v tomto případě EKF-SLAMu)

Zásadní problémy EKF-SLAM řešení (podle [1]).

1. **Konvergence** (pro pozice landmarků): Variance složek pozic jednotlivých landmarků konvergují monotónně pouze k nižší hranici určenou počátečními neurčitostmi v pozici robota a pozorování, tedy nikdy ne k nule (tzv. konvergence neurčitosti landmarku). Nové landmarky jsou totiž získávány během pohybu robota.

2. **Výpočetní složitost**: Krok aktualizace z měření (2.12) vyžaduje, aby všechny landmarky a kovarianční matice **P** (2.13) byly aktualizovány pokaždé, kdy je uděláno nové pozorování. Z toho vyplývá, že výpočetní nároky rostou s kvadrátem počtu landmarků. Tento problém se pak řeší ve vylepšených verzí základního EKF algoritmu.

3. **Asociace dat**: Standardní formulace řešení EKF-SLAMu je speciálně náchylná k nesprávnému propojení naměřeného pozorování s již zaznamenanými landmarky. Zvláště obtížné je, když se robot vrací na stejné místo po dlouhé absolvované trase (tzv. smyčky) a má rozpoznat již jednou viděné landmarky (znovu pozorování landmarků). Tento problém se ještě zhoršuje v reálném prostředí, kde landmarky nejsou pouhé body a tedy z různých pozorovacích míst můžou vypadat odlišně.

4. **Nelinearita**: EKF-SLAM využívá lineární modely nelineárních modelů pohybu (2.2) a pozorování (2.3). To sebou ale přináší velké varování. Nelinearita nevyhnutelně vede k nekonzistentnosti řešení (velký problém v EKF-SLAM přístupu). Konvergence a konzistentnost může být totiž zaručena pouze v lineárním případě.

2.4.2 FastSLAM

FastSLAM algoritmus (nebo Rao-Blackwellizedův částicový filtr) zaznamenal koncepční pokrok v návrhu (rekurzivního) pravděpodobnostního SLAMu. Předchozí snahy se totiž zaměřovaly na vylepšení výsledků EKF-SLAMu. U něj ale přetrvávají jeho fundamentální lineární (Gaussovské) předpoklady. FastSLAM, který má základ v rekurzivním Monte Carlo vzorkování či v částicovém filtrování, byl první, který reprezentoval přímo nelineární procesní model a ne-Gaussovské rozložení pravděpodobnosti pozice. FastSLAM využívá tři techniky: Rao-Blackwellizaci, podmíněnou nezávislost a převzorkování [3]. Stále linearizuje model pozorování (2.3), ale je to typicky rozumná aproximace pro měření s dosahem, když je pozice robota známa (částice). Vysoká dimenze stavového prostoru (pozice robota a landmarků)

SLAM problému ovšem dělá přímou aplikaci částicových filtrů výpočetně neproveditelnou. Jestliže ale historie pozičních stavů robota je známa přesně, potom pokud jsou jednotlivá pozorování podmíněně nezávislá[1], stavy mapy jsou také nezávislé. Odstraní se tak první bod ze seznamu problémů EKF-SLAMu (konvergence pozic landmarků). Každá částice pak definuje odlišnou hypotézu trajektorie vozidla.

Je možné redukovat vzorkovací prostor aplikováním tzv. Rao-Blackwellizací, prostřednictvím níž, když je nějaká obecná sdružená hustota pravděpodobnosti $p(\vec{y}_1, \vec{y}_2)$ vyjádřena jako $p(\vec{y}_1, \vec{y}_2) = p(\vec{y}_2|\vec{y}_1)p(\vec{y}_1)$, tak jestliže $p(\vec{y}_2|\vec{y}_1)$ může být reprezentována analyticky, je potřeba navzorkovat pouze $p(\vec{y}_1)$: $\vec{y}_1^{(i)} \sim p(\vec{y}_1)$, kde horní index i značí *i-tou* částici. Tedy místo pravděpodobnosti pro určitý stav $\vec{y}_1$ se tato pozice bere jako daná (s pravděpodobností rovnou 100 %), pro každou částici jiná. Potom je sdružená hustota pravděpodobnosti reprezentována množinou $\{\vec{y}_1^{(i)}, p(\vec{y}_2|\vec{y}_1^{(i)})\}_i^N$, kde N je počet částic a statistika jako marginální pravděpodobnost $p(\vec{y}_2) \approx \frac{1}{N}\sum_i^N p(\vec{y}_2|\vec{y}_1^{(i)})$ může být dosažena s vyšší přesností, než je možné s vzorkováním přes celý sdružený prostor.

Sdružený SLAM stav (trajektorie pozice robota s pozicemi landmarků) pak může být rozložen na komponenty – podmíněné hustoty pravděpodobnosti trajektorie vozidla robota a mapy:

$$p(\mathbf{X_{0:k}}, \mathbf{m}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \vec{x_0}) = p(\mathbf{m}|\mathbf{X_{0:k}}, \mathbf{Z_{0:k}})p(\mathbf{X_{0:k}}|\mathbf{Z_{0:k}}, \mathbf{U_{0:k}}, \vec{x_0}). \tag{2.16}$$

Na rozdíl od EKF-SLAM řešení, v rovnici 2.16 je uvedeno pravděpodobnostní rozdělení pro trajektorii $\mathbf{X_{0:k}}$ (která je pevně daná pro každou částici) oproti samostatné pozici $\vec{x_k}$ (hlavní myšlenka, když se podmiňuje trajektorie, mapa landmarků se stane podmíněně nezávislá). To je klíčová vlastnost FastSLAMu a důvod jeho rychlosti: mapa je reprezentována jako množina nezávislých Gaussovských rozložení pravděpodobnosti s lineární výpočetní složitostí (oproti kovarianční matici **P** sdružené mapy s kvadratickou složitostí přepočítávání).

Základní struktura FastSLAMu: Rao-Blackwellizův stav, kde trajektorie je reprezentována váženými vzorky a mapa je spočítána analyticky. Proto je sdružená hustota pravděpodobnosti v čase k (2.1) reprezentována množinou

$$\{\vec{w}_k^{(i)}, \mathbf{X_{0:k}^{(i)}}, p(\mathbf{m}|\mathbf{X_{0:k}^{(i)}}, \mathbf{Z_{0:k}})\}_i^N \tag{2.17}$$

kde mapa pro každou částici je složena z (podmíněně) nezávislých Gaussovských hustot pravděpodobností:

$$p(\mathbf{m}|\mathbf{X_{0:k}^{(i)}}, \mathbf{Z_{0:k}}) = \prod_j^M p(\vec{m}_j|\mathbf{X_{0:k}^{(i)}}, \mathbf{Z_{0:k}}) \tag{2.18}$$

[1] Dva jevy A, B jsou podmíněně nezávislé (jevem C), pokud platí: $P(A, B|C) = P(A|C)P(B|C)$; obecně ale pro ně nemusí platit nezávislost: $P(A, B) = P(A)P(B)$

a vektor $\vec{w}_k^{(i)}$ obsahuje historii k vah pro danou částici, čili nejdůležitější je složka $w_k^{(i)}$, která udává aktuální váhu. Tedy např. složka $w_{k-3}^{(i)}$ tak značí váhu trajektorie $\mathbf{X}_{\mathbf{0:k-3}}^{(\mathbf{i})}$, jakou měla částice o tři kroky zpět bez tří nových stavů.

Rekurzivní odhadování je pak nahrazeno nebo spíše vykonáváno částicovým filtrováním pro stavy pozice robota a aplikací EKF na stavy mapy (pozice landmarků). Při aktualizaci mapy pro danou trajektorii $\mathbf{X}_{\mathbf{0:k}}^{(\mathbf{i})}$ je pak každý pozorovaný landmark zpracován individuálně s použitím EKF aktualizace z měření 2.12 ze známé pozice. Z výše uvedeného vyplývá, že nepozorované landmarky zůstávají nezměněny (neplatí tedy analogie s pružinami uvedená pro intuitivní představu struktury pravděpodobnostní SLAM úlohy).

Sekvenční vzorkování podle důležitosti.

SIS (sequential important sampling) - rekurzivní forma vzorkování; Vzorkuje z historie stavu. Ze sdružené hustoty pravděpodobnosti pro jednotlivé navazující stavy robota $\vec{x}_k$ lze dostat rekurzi pomocí[2]:

$$p(\vec{x}_0, \vec{x}_1, ..., \vec{x}_T|\mathbf{Z_{0:T}}) = p(\vec{x}_0|\mathbf{Z_{0:T}})p(\vec{x}_1|\vec{x}_0, \mathbf{Z_{0:T}})...p(\vec{x}_T|\mathbf{X_{0:T-1}}, \mathbf{Z_{0:T}})$$

V každém kroku k jsou částice znovu určeny tentokráte s novým vzorkem $\vec{x}_k$, jenž je získán z tzv. navrhované hustoty pravděpodobnosti $\pi(\vec{x}_k|\mathbf{X_{0:k-1}}, \mathbf{Z_{0:k}})$, která aproximuje skutečnou hustotu $p(\vec{x}_k|\mathbf{X_{0:k-1}}, \mathbf{Z_{0:T}})$ a množině vzorků $\mathbf{X_{0:k}}$ je pak přiřazena váhou tzv. důležitost (kompenzace nesrovnalostí).

Co se týče chyby aproximace, ta roste s časem (spjatá také se sdruženým stavovým prostorem). Zvětšování rozdílů mezi váhami vzorků snižuje statistickou přesnost. Je tedy nutné udělat tzv. *převzorkovací krok*, který obnoví jednotné vážení, ale zapříčiní ztrátu historie částicových informací: klíčová vlastnost *SIS* – s převzorkováním může produkovat seriózní výsledky (statistiky) pouze pro systémy s exponenciálním zapomínáním jejich minulosti. Pro např. šum, aktuální stav v čase k nezávisí na předchozích stavech.

Obecná forma R-B částicového filtru pro SLAM (podle [1]).

1. Pro každou částici spočítána navrhovaná hustota pravděpodobnosti podmíněná danou částicovou minulostí; vzorek $\vec{x}_k^{(i)}$ získán z

$$\vec{x}_k^{(i)} \sim \pi(\vec{x}_k|\mathbf{X}_{\mathbf{0:k-1}}^{(\mathbf{i})}, \mathbf{Z_{0:k}}, \vec{u}_k) \tag{2.19}$$

   a tento nový vzorek je přidán do historie dané částice $\mathbf{X}_{\mathbf{0:k}}^{(\mathbf{i})} = \{\mathbf{X}_{\mathbf{0:k-1}}^{(\mathbf{i})}, \vec{x}_k^{(i)}\}$.

[2] Vychází z možnosti úpravy sdružené hustoty pravděpodobnosti pro více jak dvě proměnné, např. $p(y_0, y_1, y_2) = p(y_0)p(y_1, y_2|y_0) = p(y_0)p(y_1|y_0)p(y_2|y_1, y_0)$

2. Vypočítat novou váhu částice pro trajektorii $\mathbf{X}_{\mathbf{0:k}}^{(\mathbf{i})}$ tvořenou vzorky; podle funkce důležitosti

$$w_k^{(i)} = w_{k-1}^{(i)} \frac{p(\vec{z}_k|\mathbf{X}_{\mathbf{0:k}}^{(\mathbf{i})}, \mathbf{Z_{0:k-1}}) p(\vec{x}_k{}^{(i)}|\vec{x}_{k-1}^{(i)}, \vec{u}_k)}{\pi(\vec{x}_k{}^{(i)}|\mathbf{X}_{\mathbf{0:k-1}}^{(\mathbf{i})}, \mathbf{Z_{0:k}}, \vec{u}_k)} \tag{2.20}$$

v čitateli pohybový model a model pozorování. Ten se liší, protože R-B vyžaduje, aby byla závislost na mapě marginalizována[3].

3. Pokud je potřeba, aplikuje se převzorkování (kdy ho nejlépe udělat je otevřený problém, nelze říci, jaký čas je přesně ten nejvhodnější). Převzorkování se provádí výběrem částic z množiny $\{\mathbf{X}_{\mathbf{0:k}}^{(\mathbf{i})}\}_i^N$, včetně jejich přidružených map, s pravděpodobností výběru úměrnou vahám částic $w_k^{(i)}$. Vybraným částicím pak dána stejná váha $w_k^{(i)} = \frac{1}{N}$.

4. Pro každou částici aplikovat EKF aktualizaci z měření 2.12 na pozorované landmarky ovšem s rozdílem takovým, že je jako při prostém mapování poloha robota považována za známou.

Dvě základní verze FastSLAMu.

- *FastSLAM 1.0*
- *FastSLAM 2.0*

Liší se pouze z hlediska tvaru navrhované hustoty pravděpodobnosti $\pi()$ uvedené v 1. kroku postupu (2.19) a tudíž i ve výpočtech jejich vah - 2. krok (2.20). FastSLAM 2.0 jakožto novější verze je samozřejmě efektivnější řešení. Následuje velice stručný popis obou verzí.

FastSLAM 1.0 - Navrhovaná hustota pravděpodobnosti $\pi()$ je pohybový model (2.2):

$$\vec{x}_k^{(i)} \sim p(\vec{x}_k|\vec{x}_{k-1}^{(i)}, \vec{u}_k)$$

Proto jsou vzorky defacto váženy pouze marginalizovaným modelem pozorování (z rovnice 2.20):

$$w_k^{(i)} = w_{k-1}^{(i)} p(\vec{z}_k|\mathbf{X}_{\mathbf{0:k}}^{(\mathbf{i})}, \mathbf{Z_{0:k-1}})$$

Tato verze je následně uvažována v praktické části.

FastSLAM 2.0 - Navrhovaná hustota $\pi()$ se navíc podmiňuje trajektorií částice známou z předchozího kroku s celou historií pozorování včetně současného:

$$\vec{x}_k^{(i)} \sim p(\vec{x}_k|, \mathbf{X}_{\mathbf{0:k-1}}^{(\mathbf{i})}, \mathbf{Z_{0:k}}, \vec{u}_k)$$

[3] *marginalizace*: pro sdruženou hustotu pravděpodobnosti *p(y,z)* je marginální hustota náhodné veličiny Y $p(y) = \int_{\mathbb{R}} p(y,z)dz$

tato hustota lze vyjádřit pomocí pohybového modelu jako:

$$p(\vec{x}_k|, \mathbf{X}^{(\mathbf{i})}_{\mathbf{0:k-1}}, \mathbf{Z}_{\mathbf{0:k}}, \vec{u}_k) = \frac{1}{C} p(\vec{z}_k|\vec{x}_k, \mathbf{X}^{(\mathbf{i})}_{\mathbf{0:k-1}}, \mathbf{Z}_{\mathbf{0:k-1}}) p(\vec{x}_k|\vec{x}^{(i)}_{k-1}, \vec{u}_k)$$

kde C je normalizační konstanta. Z rovnice 2.20 opět po dosazení $\pi()$ vyplývá vztah pro výpočet váhy: $w^{(i)}_k = w^{(i)}_{k-1} C$. Výhodou této verze je to, že navrhovaná hustota $\pi()$ je lokálně optimální. Z tohoto faktu vyplývá, že rozptyl váhy $w^{(i)}_k$ každé částice je nejmenší možný.

FastSLAM (1.0 a 2.0) není z principu problematiky schopen zapomínat minulost (marginalizováním mapy vzniká závislost na historii polohy a měření a při převzorkování se z této historie čerpá, tudíž se statistická přesnost ztrácí). Ale i tak zkušenosti s výsledky FastSLAMu 2.0 ukazují schopnost v praxi ve venkovním prostředí vytvářet přesné mapy.

2.4.3 UKF-SLAM

Unscented Kalman Filter; do češtiny se Unscented v tomto kontextu nepřekládá (čili v krajním případě jen *"Unscented Kalmanův filtr"*, spíše ale jen zkratka UKF-SLAM).

Opět jako v případě EKF-SLAMu je nutné Gaussovské rozložení pravděpodobnosti. Kalmanův filtr vyžaduje lineární modely: jak pohybového (2.2), tak modelu pozorování (2.3). Pokud je nemá, získá je linearizací Taylorovým rozvojem okolo střední hodnoty odhadu. Aby však bylo docíleno toho, že výsledek není linearizován okolo střední hodnoty (jak to dělá EKF), je třeba udělat tzv. **Unscented Transform** (*"Unscented transformaci"*). Ta může být shrnuta jako postup o čtyřech bodech [4]:

Unscented Transform

1. spočítat množinu tzv. Sigma bodů
2. ke každému Sigma bodu přiřadit váhu
3. transformovat tyto body danou nelineární funkcí
4. z vážených Sigma bodů spočítat Gaussovskou hustotu pravděpodobnosti

UKF-SLAM algoritmus využívá Unscented transformaci v predikčním (2.4) a korekčním (2.5) kroku. Unscented transformace může být brána jako alternativa k linearizaci, dává lepší výsledky než Taylorův rozvoj.

Definice proměnných pro UKF-SLAM.

- $\boldsymbol{\chi^{[i]}}$: množina Sigma bodů; index i pro značení jednotlivých Sigma bodů
- $w^{[i]}$: váha pro *i-tý* Sigma bod
- $\vec{\hat{x}}$: vektor středních hodnot funkce hustoty pravděpodobnosti rekonstruované ze Sigma bodů.
- **P**: kovarianční matice této hustoty

Váhy se volí tak, aby platilo následující:

$$\sum_i w^{[i]} = 1 \tag{2.21}$$

$$\vec{\hat{x}} = \sum_i w^{[i]} \chi^{[i]} \tag{2.22}$$

$$\mathbf{P} = \sum_i w^{[i]} (\chi^{[i]} - \vec{\hat{x}})(\chi^{[i]} - \vec{\hat{x}})^T \tag{2.23}$$

Neexistuje však univerzální postup jak množinu Sigma bodů a jejich váhy zvolit. Dále je tedy uveden pouze základní doporučený algoritmus.

Základní UKF-SLAM algoritmus (podle [4]).

1. První Sigma bod zvolen jako výše uvedená střední hodnota: $\chi^{[0]} = \vec{\hat{x}}$.
2. Další Sigma body jsou situovány kolem této střední hodnoty podle vztahů:

$$\chi^{[i]} = \vec{\hat{x}} + (\sqrt{(n+\lambda)\mathbf{P}})_i, i = 1, ..., n$$

$$\chi^{[i]} = \vec{\hat{x}} - (\sqrt{(n+\lambda)\mathbf{P}})_{i-n}, i = n+1, ..., 2n$$

   kde n je dimenze hustoty pravděpodobnosti, i (nebo $i-n$) je index patřičného sloupce matice $(\sqrt{(n+\lambda)\mathbf{P}})$ a λ je tzv. škálovací parametr (čím větší je jeho hodnota, tím větší je vzdálenost mezi střední hodnotou a jednotlivými Sigma body).

3. Dále je třeba zvolit váhy. Je nutné rozlišit dva druhy $w_m^{[i]}$ a $w_c^{[i]}$. První se použije pro vypočtení střední hodnoty (2.22), druhý zavedený pak pro vypočtení kovarianční matice (2.23).

Pro váhy nultého Sigma bodu pak platí:

$$w_m^{[0]} = \frac{\lambda}{n+\lambda}$$

$$w_c^{[0]} = w_m^{[0]} + (1 - \alpha^2 + \beta)$$

kde α, β jsou parametry Uncented tranformace. Doporučení pro zvolení jejich hodnot je uvedeno níže.

Vztah pro váhy $w_m^{[i]}$ a $w_c^{[i]}$ "*i-tého* až *2n-tého*"$(i = 1, ..., 2n)$ Sigma bodu se však neliší:

$$w_m^{[i]} = w_c^{[i]} = \frac{1}{2(n+\lambda)}$$

Byť, jak už bylo zmíněno před začátkem tohoto postupu, neexistuje univerzální řešení úlohy, je možné použít následující doporučení pro nastavení parametrů použitých výše:

$$\alpha \in (0, 1\rangle$$

$\beta = 2$ (optimální pro Gaussovskou hustotu pravděpodobnosti)

$$\lambda = \alpha^2(n + \kappa) - n$$

$$\kappa \geq 0$$

κ tedy ovlivňuje, jak daleko jsou Sigma body vzdáleny od střední hodnoty $\vec{\hat{x}}$.

4. Potom již lze pro vážené a transformované (transformace vykonána danou nelineární funkcí) body rekonstruovat do nového Gaussovského rozdělení pravděpodobnosti:

$$\vec{\hat{x}}_{k|k-1} = \sum_{i=0}^{2n} w_m^{[i]} g(\chi^{[i]})$$

$$\mathbf{P}_{k|k-1} = \sum_{i=0}^{2n} w_c^{[i]} (g(\chi^{[i]}) - \vec{\hat{x}}_{k|k-1})(g(\chi^{[i]}) - \vec{\hat{x}}_{k|k-1})^T$$

Pozn.: Význam dolních indexů $\vec{\hat{x}}$ a $\mathbf{P}$ je stejný jako u EKF-SLAMu.

Funkce *g()* bude v tomto případě zastupovat pohybový model robota (2.2). Vyjádřeno v pojmech využitých v předešlých přístupech, jedná se tedy o krok *predikce* pohybu (časová aktualizace - 2.4). Postupuje se výše uvedeným způsobem a ke kovarianční matici $\mathbf{P}$ se pouze přičte matice $\mathbf{R}$ vyjadřující pohybový šum.

5. *Korekce* (aktualizace na základě pozorování - 2.5) - algoritmus se vrací opět do prvního kroku a postupuje se stejným způsobem, v tomto případě se však vychází z výsledku předchozího kroku predikce: $\vec{x}_{k|k-1}$ a $\mathbf{P}_{k|k-1}$. Sigma body (množina pro přehlednost značena jako $\boldsymbol{\chi}'$) se tedy vytváří vzhledem k této nové hustotě pravděpodobnosti (reprezentované jejími ukazateli - střední hodnotou a kovarianční maticí). Za funkci *g()* se v kroku čtyři tentokrát zvolí model pozorování (2.3): $\boldsymbol{Z}' = h(\boldsymbol{\chi}')$. Symbolem $\boldsymbol{Z}'$ se značí touto funkcí transformovaná množina Simga bodů $\boldsymbol{\chi}'$. Ke kovarianční matici **P** je poté nutné přičíst šum měření **Q** a vznikne tak matice označená jako **S**:

$$\mathbf{S} = \mathbf{P} + \mathbf{Q}$$

Následuje určení Kalmanova zesílení **K**. K vypočtení Kalmanova zesílení je nejdříve nutné vypočítat tzv. křížovou korelaci stavu a předpokládaného pozorování:

$$\mathbf{P}'^{xz} = \sum_{i=0}^{2n} w_c^{[i]} (\chi'^{[i]} - \vec{x}_{k|k-1})(Z'^{[i]} - \vec{\hat{z}})^T$$

kde $\vec{\hat{z}}$ (střední hodnota, předpokládané pozorování) je výsledek vypočtený v tomto kroku. Kalmanovo zesílení je pak dáno vztahem:

$$\mathbf{K} = \mathbf{P}'^{xz}\mathbf{S}^{-1}$$

Čili pro aktuální krok k je pak nové $\vec{\hat{x}}$ z pozorování získáno:

$$\vec{\hat{x}}_{k|k} = \vec{\hat{x}}_{k|k-1} + \mathbf{K}(\vec{z}_k - \vec{\hat{z}}) \tag{2.24}$$

a k němu příslušná kovarianční matice **P**:

$$\mathbf{P}_{k|k} = \mathbf{P}_{k|k-1} - \mathbf{K}\mathbf{S}\mathbf{K}^T \tag{2.25}$$

Tím je hotový jak celý aktualizační krok přechodu stavu z časového okamžiku $k-1$ do k, tak zároveň celý uváděný postup.

Stojí za povšimnutí, že algoritmus řešení je velice podobný jako v případě EKF-SLAMu, pouze je nahrazen EKF (Rozšířený Kalmanův filtr) a při výpočtech tedy nejsou použity žádné Jakobiány matic. To je principiální vlastnost a jeden z důvodů vytvoření tohoto způsobu řešení SLAM úlohy.

2.5 Metriky pro porovnání přístupů řešící SLAM úlohu

V praktické části byly zhotoveny implementace všech tří přístupů. Poslední podkapitola teoretické části se tedy věnuje popsání ukazatelů, které umožňují jejich porovnání (lépe řečeno jejich výsledků z experimentů). Je třeba poznamenat, že nejsou jediné, existuje jich více. Jako první bude uvedena tzv. RMSE.

2.5.1 RMSE

Root-Mean-Square Error, Odmocnina ze střední kvadratické chyby je základní statistický ukazatel. Pro dvě časové posloupnosti A, B, které se skládají z vektorů $\vec{a_k}$ a $\vec{b_k}$ se RMSE vypočte podle následujícího vztahu:

$$rmse(A,B) = \sqrt{\frac{1}{N}\sum_{k=1}^{N}(\|\vec{a_k} - \vec{b_k}\|)^2} \tag{2.26}$$

kde N je počet vzorků časové posloupnosti.

RMSE bude využito pro polohy robota, kdy vektory budou mít dvě složky (x, y) jako souřadnice v rovině a jedna časová posloupnost pak bude značit skutečnou trajektorii robota, druhá odhadovanou.

2.5.2 Metrika zavedená v [5]

V článku [5], který se zabývá měření přesnosti SLAM algoritmů, byla vytvořena speciální metrika pro tuto problematiku (nepojmenovaná, proto chybí název v nadpisu). Celková chyba ε je zadefinována takto:

$$\varepsilon(\delta) = \frac{1}{N}\sum_{i,j} trans(\delta_{i,j} \ominus \delta^*_{i,j})^2 + rot(\delta_{i,j} \ominus \delta^*_{i,j})^2 \tag{2.27}$$

Vysvětlení výrazu.

Metrika založená na relativním posunu mezi polohami robota: $\delta_{i,j}$ značí relativní polohu *i-té* a *j-té* odhadované pozice v časové posloupnosti (trajektorii robota), $\delta^*_{i,j}$ je relativní poloha *i-té* a *j-té* skutečné pozice robota. Během vyhodnocení byl zvolen index j jako: $j = i + 1$ (následující časový okamžik po indexu i). Potom např. relativní poloha $\delta_{i,j}$ je v zavedeném značení v této práci: $\vec{\delta_k} = \vec{x}_{k+1} - \vec{x}_k$.

Výraz 2.27 může být rozložen do dvou částí: výpočet chyby v určení polohy robota $\varepsilon_{trans}(\delta)$ a chyby v určení jeho natočení $\varepsilon_{rot}(\delta)$.

$$\varepsilon_{trans}(\delta) = \frac{1}{N}\sum_{i,j} trans(\delta_{i,j} \ominus \delta^*_{i,j})^2 \tag{2.28}$$

trans(): Funkce, která určuje chybu v translaci robota:

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

čili určena jako norma vektoru vzdálenosti mezi $\delta_{i,j}$ a $\delta^*_{i,j}$. N je počet binárních relací *(i, j)*.

$$\varepsilon_{rot}(\delta) = \frac{1}{N}\sum_{i,j} rot(\delta_{i,j} \ominus \delta^*_{i,j})^2 \tag{2.29}$$

rot(): Funkce, která určuje chybu v rotaci robota:

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

$\delta_{i,j}$ a $\delta^*_{i,j}$ jsou v tomto případě pouze čísla (ne vektory) - hodnoty úhlů v radiánech. Funkce *min()* pro dané dva argumenty je zde pro případ, kdyby výraz $|\delta_{i,j} - \delta^*_{i,j}|$ přesáhl 2π (pro spočtení reálné chyby orientace). Toto opatření je dostačující, protože samotné úhly natočení robota jsou v rozmezí $\langle -\pi; \pi \rangle$: výsledek $|\delta_{i,j} - \delta^*_{i,j}|$ tedy nemůže přesáhnou 4π.

Celková chyba $\varepsilon(\delta)$ (2.27) je pak dána součtem těchto dílčích chyb:

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

Tato metrika tedy umožňuje vyhodnotit stav robota jako celek, bez nutnosti oddělit a vyhodnocovat zvlášť odlišnými ukazateli pozici robota v prostoru a jeho orientaci.

Praktická část 3

Během tohoto stručného úvodu této kapitoly bude představena hlavní osa: zadání, dílčí cíle, směr, kterým se zbytek práce bude ubírat. Následně bude v každých podkapitolách popsána každá dílčí část. Všechny dohromady pak utvářejí výsledný celek, jehož smyslem (velice stručně řečeno) je vytvořit simulaci SLAM problému pro průjezd robota v určeném prostředí. Hlavní náplní praktické části této práce byl přepis všech tří základních přístupů k řešení SLAM úlohy (EKF-SLAM, FastSLAM, UKF-SLAM), které již byly implementovány v MATLABu, do programovacího jazyku Python. Uveden bude také i stručný návod, jak udělat pomocí vytvořeného jednoduchého GUI prostředí, ve kterém se robot má pohybovat. Následovat bude popis začlenění přepsaných pythonovských zdrojových kódů do struktury systému ROS2, pro který byly kódy napsány a ve kterém nakonec algoritmy budou spouštěny. Bude zde probíhat i vyhodnocování, které je v této práci uvedeno na konci kapitoly. Data ze simulací pak budou poskytovány nástroji Rviz2, který bude dané experimenty graficky zobrazovat. Uvedeny budou muset být bohužel i chyby, či nedostatky, které se během vypracování vyskytly a které kvůli velké rozsáhlosti projektu již z časových důvodu bylo velmi těžké opravit, odstranit.

3.1 Zkonkretizování SLAM úlohy pro praktickou část

Pro zbytek této práce bude SLAM úloha uvažována pro tyto esenciální faktory: Robot se bude pohybovat v rovině a jeho poloha v ní se bude určovat dvojicí čísel (kartézskými souřadnicemi): x, y. Třetí složkou pak bude úhel natočení robota θ v radiánech (standardní případ: úhel, který bude svírat osa vybíhající z čela vozidla s osou x). Je namístě zdůraznit, že úhel bude vždy převeden do intervalu $\langle -\pi; \pi \rangle$, což je pro potřeby SLAMu vyhovující a žádoucí. Čili stav robota bude mít v konkrétním

časovém okamžiku k následující tvar:

$$\vec{x}_k = \begin{bmatrix} x_1^k \\ x_2^k \\ x_3^k \end{bmatrix} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

Časová aktualizace (predikční krok) může být zjednodušeně zapsán jako

$$\vec{x}_{k|k-1} = \begin{bmatrix} x + dx \\ y + dy \\ \theta + d\theta \end{bmatrix}$$

kde dx, dy a $d\theta$ je výsledek aplikace řízení ($d\theta$ je změna v orientaci robota). Hodnoty jsou tedy sečteny s posledním odhadem stavu. Výsledky jsou obdrženy z pohybového modelu.

Pozice landmarku v prostředí má samozřejmě také dvě poziční, kartézské souřadnice $[x_i, y_i]$. Jedno konkrétní pozorování *i-tého* landmarku pak bude přicházet ve tvaru tzv. *vzdálenosti-se-směrem (range-bearing)*. Význam tohoto tvaru je prostý: je to vektor o dvou sloužkách, kde první z nich značí vzdálenost robota od tohoto landmarku a druhá směr, ve kterém se nachází (úhel mezi osou čelního pohledu robota a spojnicí mezi robotem a landmarkem):

$$\begin{bmatrix} range_i \\ bearing_i \end{bmatrix} = \begin{bmatrix} \sqrt{(x_i - x)^2 + (y_i - y)^2} \\ \tan^{-1}(\frac{y_i - y}{x_i - x}) - \theta \end{bmatrix} \tag{3.1}$$

V reálném prostředí se samozřejmě přesně tohoto tvaru nikdy nedosáhne, v obou složkách bude vždy přítomen nějaký šum měření. Šum je tedy v simulacích také připočítáván.

Posledním termínem, který je na úvod této části potřeba zmínit, je tzv. *waypoint (bod cesty)*. Waypointy jsou body, které jsou určeny na začátku úlohy a značí cestu, kudy robot má jet. Je nutné, aby robot projel každým tímto bodem a to v pořadí, které jim bylo přiřazeno. Zároveň by se měl pokusit o to, aby mezi dvěma waypointy byla projeta co nejmenší možná vzdálenost (nejlépe tedy přímá cesta, záleží na překážkách na trase, na podmínkách, na šumu v měření atd.). Poslední waypoint je zároveň cíl cesty. Místo, kam se má robot dostat. V této práci tedy budou mít waypointy dvě souřadnice $[x_w, y_w]$.

3.2 Přepis kódů z MATLABu do Pythonu

Během let 2004 - 2006 vytvořil australský vědec Tim Bailey implementace nejpoužívanějších přístupů k řešení SLAM problematiky uvedených v teoretické části této práce. Jsou naprogramované pro prostředí MATLAB, repozitář je dostupný

na platformě GitHub[1]. Nejtěžší a časově nejnáročnější z praktické části této bakalářské práce tedy bylo přepsat tento projekt z programovacího jazyka MATLAB do Pythonu. Z edukativních důvodů tak vlastně dochází k oživení téměř dvacet let starých kódů. Výsledek je dostupný taktéž na GitHubu[2].

Byla snaha se co nejvíce držet předlohy. Samozřejmě některé části kódu vyžadovaly pro Python větší změnu oproti originálu, ale nějaké zásadní (myšlenkové) se v přepsaných verzí nevyskytují. I tak téměř každá řádka vyžadovala úpravu. Kromě kosmetických změn typu posun indexování o jeden zpět (MATLAB indexuje od 1, Python od 0), nahrazení znaků pro operace lineární algebry (např. * není v Pythonu pro násobení matic, ale pouze pro násobení mezi jejich prvky), importování potřebných modulů (zejména NumPy) či změna závorek (např. z kulatých na hranaté), bylo nutné si dát pozor zejména na klíčový rozdíl mezi MATLABem a Pythonem. Pro reprezentaci matic byl zvolen modul NumPy a je pole (array).

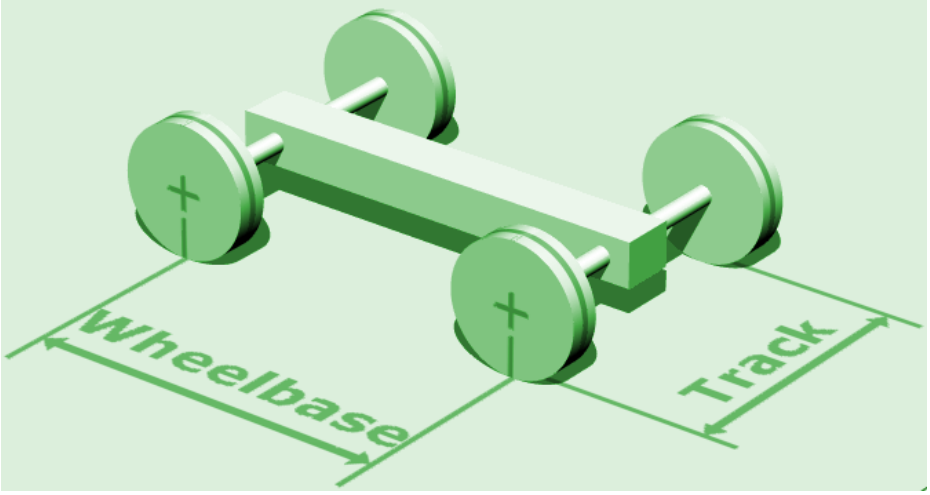
> Klíčovým rozdílem je fakt, že hodnota proměnné matice v MATLABu je reprezentována přímo jejími číselnými hodnotami, kdežto matice v Pythonu (*numpy.array()*) je referenční proměnná, jejíž hodnota je reference (velice zjednodušeně řečeno odkaz do paměti).

Případné neošetření této skutečnosti by mělo vliv na správnost výsledků simulací. Řešit se tato situace dá např. pomocí metody *numpy.copy()*, která vytvoří v paměti kopii příslušné matice (pole) a předá referenci na toto nové pole (např. do nové referenční proměnné).

3.2.1 Charakteristika projektu

Hlavním cílem projektu Tima Baileyho a spol. bylo vytvoření programu pro simulace SLAM úlohy pomocí všech tří základních přístupů. Tedy průjezd robota (vozidla) skrz prostředí s landmarky po zvolené trajektorii dané waypointy na zvolené místo určení. Landmarky jsou pouze body v prostoru, kterými se nesmí projet, jinak se robot může pohybovat kdekoliv. Počátek (např. roviny: [0, 0]) souřadného systému (mapy) se vždy určí tam, kde robot svojí trasu zahajuje. Ve všech přístupech je uvažováno robotické vozítko řídící se Ackermannovou geometrií. Jeden z volitelných parametrů tak je i rozvor (wheelbase). Aby byl naprosto zřejmý význam tohoto technického termínu, je přiložen obrázek 3.1.

[1] `https://github.com/OpenSLAM-org/openslam_bailey-slam`
[2] `https://github.com/vao25/slam_python`



Obrázek 3.1: Typ vozidla robota uvažovaného v praktické části (obrázek z [6])

3.2.2 Přepsané části

V této sekci budou popsány části kódu, které bylo nutné přepsat do jazyka Python. Pro EKF-SLAM byl využit adresář repozitáře *ekfslam_v1* a to sice všechny potřebné skripty pro fungování simulace. Nebyly tedy přepsány skripty, které slouží k vykreslování výsledků (to se vykonává v nástroji Rviz2; uvedeno na konci praktické části). Dále nebyly přepsány *KF_IEKF_update.m* a *update_iekf.m* vykonávající efektivnější verzi aktualizace pro velké stavové prostory (s mnoho landmarky). Součástí tohoto původního adresáře je i skript vytvářející v MATLABu GUI pro nastavení prostředí robota. To také nebylo přepsáno (tato problematika vyřešena samostatně, viz podkapitola níže). Pro FastSLAM byly využity kódy z adresáře *fastslam*. Vynechány byly stejné skripty pro vizualizaci výsledků experimentu a nastavení mapy jako v případě EKF-SLAMu. Navíc nebyl využit skript *fast_to_ekf_diag.m*. Byla zvolena verze FastSLAM 1.0 (podadresář *fastslam1*). Z adresáře *ukfslam* pro UKF-SLAM řešení byla přepsána všechna skripta. Fungování této části navíc vyžaduje i balíček *Matlab Utilities*[3] od Tima Baileyho. Z tohoto balíku jsou nutná tři skripta, která byla následně také přepsána: *unscented_transform.m*, *unscented_update.m* a *sqrt_posdef.m*.

3.2.3 Výsledná struktura

Kvůli přehlednosti byly vytvořeny stejnojmenné adresáře: *ekfslam_v1*, *fastslam*, *ukfslam*. Pokračuje se stručným popisem obsahu, klíčových souborů těchto adresářů. V každém adresáři se nachází soubor s konkrétním nastaveným prostředím pro robota: *file.json* (počáteční úhel orientace, landmarky, waypointy; přesná struktura

[3] balíček *Matlab Utilities* dostupný na adrese: `https://www-personal.acfr.usyd.edu.au/tbailey/software/matlab_utilities.htm`

popsána v podkapitole níže). Dále je vždy přítomen konfigurační soubor *configfile.py* zajišťující konfiguraci, nastavení celé simulace. Ten už se může lišit podle toho, v jakém adresáři se nachází (různý pro odlišné přístupy), v zásadě si jsou ale podobné. V každém se např. nachází již zmiňovaný parametr WHEELBASE, dále pak V (rychlost pohybu robota), MAXG pro určení maximálního úhlu zatočení atd. Kromě volitelných parametrů zde jsou i tzv. přepínače řídící průběh simulace (nabývají hodnoty 1, 0; např. SWITCH_SENSOR_NOISE pro to, zda-li se bude přičítat šum měření). Veškeré parametry, přepínače jsou v souborech podrobně okomentovány, není tedy třeba zde každý rozepisovat.

Hlavní zdrojový soubor pro průběh celé simulace je ve všech případech pojmenován tak, že je zakončen *_sim.py*: *ekfslam_sim.py*, *fastslam1_sim.py*, *ukfslam_sim.py*. Na začátku simulace si hlavní zdrojový kód načte rozložení mapy z *file.json* a parametry z *configfile.py* a poté provede simulaci dle nastavených přepínačů. Výsledky jsou jinak strukturovány pro EKF-SLAM a UKF-SLAM případ než pro FastSLAM. V prvním případě je výsledek vložen do python slovníku *data*. Ten má čtyři klíče: *"true", "path", "i", "state"*. Hodnota pro klíč *"i"* je číslo, celkový počet časových okamžiků v dané simulaci. Hodnota *data["true"]* a *data["path"]* je 2D numpy.array reprezentující matici. Každý sloupec určuje stav robota v časovém okamžiku rovnému indexu tohoto sloupce (matice tedy mají 3 řádky). Ve slovníku *data["true"]* je zaznamenána skutečná cesta, v *data["path"]* pak odhadovaná (kudy si robot "myslí", že projížděl). Hodnota *data["state"]* je seznam tak dlouhý, kolik bylo časových kroků v simulaci. Na každý krok na daném indexu pak připadá další pythnovský slovník s klíči *"x", "P"*. V *data["state"][j]["x"]* je potom uložen celkový odhadovaný stav úlohy (stav robota + pozice landmarků) v daném kroku. V *data["state"][j]["P"]* je uložena diagonála kovarianční matice **P**. Tedy pouze rozptyly pro jednotlivé složky celkového stavu (zjednodušení z důvodu zvýšení přehlednosti a ušetření paměti). V případě *fastslam1_sim.py* je do návratové hodnoty *data* uložen seznam částic. Je to všech N částic ve stavu, kterého dosáhli v posledním časovém okamžiku simulace. Atributy jednotlivých částic jsou: váha částice *w* (2.20), *xv* - vektor se třemi složkami udávající, v jakém stavu robot skončil, *xf* - matice udávající výsledné odhadované pozice landmarků: má dvě řádky, každý sloupec pro jeden landmark a *Pf* - kovarianční matice $\mathbf{P_{mm}}$ (2.9) landmarků (pro vektor vytvořený z matice *xf*, pozice landmarků dány pouze za sebe). Pouze tyto data by byla v dalších fázích nedostatečná, proto byly návratové hodnoty metody posléze v přípravě kódů pro ROS 2 rozšířeny (viz níže).

V každém adresáři potom jsou veškeré moduly, které hlavní soubor využívá pro vykonání simulace, např. metody pro aktualizaci stavu (predikci, korekci), pohybový model atd. Jsou to implementace pojmů vysvětlovaných v teoretické části, jejich název většinou vystihuje předmět jejich činnosti a jsou také podrobně okomentovány. Není tedy nutné zde každý modul rozebírat. Celá simulace již může být spuštěna takto mimo ROS 2, spuštění hlavního zdrojového souboru je dosaženo

pomocí souboru *launch.py*: v příslušném adresáři stačí otevřít terminál a zadat příkaz `python3 launch.py` . Nebo samozřejmě může být využito jakékoliv vývojové prostředí pro Python.

3.3 GUI pro nastavení robotova prostředí

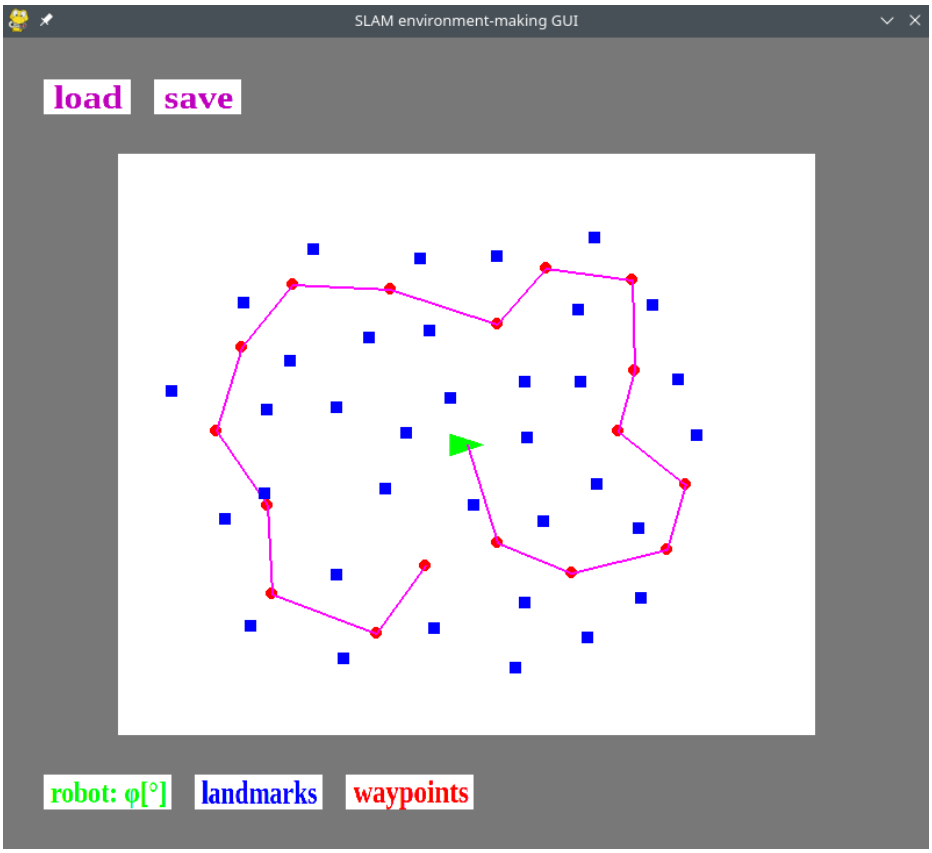
V rámci bakalářské práce vzniklo jednoduché grafické uživatelské rozhraní (GUI) nastavující prostředí robota. V již zmíněném repozitáři se nachází v adresáři *gui*[4]. Smyslem jeho vytvoření bylo to, aby se nemusely souřadnicové hodnoty zadávat ručně přímo do souboru *file.json* a také aby se celá mapa dala s pomocí vykreslení představit. Program byl vytvořen pomocí nástroje *Pygame*[5] určeného zejména pro tvorbu plošinových her. GUI nastavující prostředí lze spustit v adresáři *gui* s použitím terminálu a zadáním příkazu `python3 setup.py` .

3.3.1 Stručný návod k použití

Pokud již v adresáři *gui* existuje patřičný *file.json*, příslušná mapa se z něj načte pomocí tlačítka *load*. Robot je vyznačen pomocí zeleného trojúhelníčku. Jeden z požadavků byl, aby robot byl napevno usazen ve středu plátna a nastala tak požadovaná situace pro výsledné implementace přístupů SLAM úlohy, které zahajují simulaci s pozicí robota v počátku soustavy souřadnic [0, 0]. Všechny pozice landmarků a waypointů jsou pak dány vzhledem k této poloze. Navíc oproti originálním implementacím lze zadat i počáteční orientaci robota (ta je v originálních verzích napevno zvolena jako 0 rad), která je také nastavitelná z tohoto GUI. Stačí kliknout na tlačítko *robot:* $\varphi[^\circ]$ a objeví se prostor pro zadání úhlu[6] ve stupních (uživatelsky přívětivější než radiány, konečná hodnota se před uložením do *file.json* na radiány, které implementace vyžadují, převede automaticky). Číslo je možné upravovat mazáním číslic pomocí klávesy Backspace. Zadaný úhel pak stačí potvrdit stiskem klávesy Enter, nebo kliknutím na jiné tlačítko. Landmarky, které lze přidávat a odebírat po kliknutí na tlačítko *landmarks*, jsou vyobrazeny pomocí modrých čtverečků. Přidání se vykoná kliknutím na požadované místo na bílém plátně, odebrání kliknutím na odebíraný landmark. Waypionty, značené červeným kroužkem, lze utvářet po kliknutí na tlačítko *waypoints*. Přidání a odebrání funguje stejně jako v případě landmarků. Je třeba mít na paměti že waypointy jakožto požadovaná trajektorie, se záměrně přidávají v pořadí daném jejich plánovaným průjezdem, tedy poslední vložený waypiont je cílem cesty.

[4] `https://github.com/vao25/slam_python/tree/main/gui`
[5] `https://www.pygame.org`
[6] možnost zadat hodnotu i s desetinnými čísly, pro tento účel se používá desetinná tečka

Po vytvoření požadované mapy se výsledek uloží do souboru *file.json* kliknutím na tlačítko *save*. Tento soubor pak lze umístit k modulu *launch.py* do požadovaného adresáře vykonávající daný typ simulace SLAM úlohy. Výsledná struktura tohoto souboru pak je: pro klíč *"x3"* float hodnota úhlu v radiánech; pro klíč *"lm"* je hodnota pole polí souřadnic [x, y] landmarků; pro klíč *"wp"* stejná situace jako v případě lanmdarků, akorát pro waypointy.



Obrázek 3.2: Výsledné GUI pro nastavení prostředí robota

3.4 Začlenění přepsaných kódů do struktury systému ROS 2

ROS 2 (Robot Operating System) [7] je open source projekt pro tvorbu profesionálních aplikací určených pro robotiku. Jedná se v podstatě o systém založený na Linuxu a jako takový má mnoho svých distribucí. Obsahuje početné množství softwarových knihoven a nástrojů. Spoléhá na něj velká část průmyslu v robotickém odvětví. Dnes je nasazen v robotech zajišťující produkci po celém svět. Je také základem pro většinu robotických výzkumů. Lze ho využít téměř v jakémkoliv pro-

středí - ve vnitřních nebo venkovních prostorech, ve vodě, ve vzduchu, ve vesmíru... Mezi jeho výhody patří také to, že je multi-platformní - je podporován na Linuxu, macOS, Windows či na různých vestavěných platformách (micro-ROS). V této bakalářské práci byla využita distribuce Foxy Fitzroy. Je založená Linuxové distribuci Ubuntu, využívá tedy kromě svého ROS 2 repozitáře i jeho balíčkovací systém (jeho repozitáře). Dále z tohoto faktu plyne i např. syntaxe terminálu, či aktualizační příkazy.

3.4.1 Vytvoření aplikace

Ve zvoleném adresáři je nejprve vhodné vytvořit workspace adresář pojmenovaný např. *ros2_ws*. Workspace má v ROS 2 patřičnou strukturu, obsahuje tyto podadresáře: *build, install, log, src*. V posledním jmenovaném adresáři jsou obsaženy zdrojové kódy dané aplikace ve formě tzv. ROS packages - balíčků. ROS package je jednotka, kde jsou umístěny a organizovány kódy pro ROS. Adresář *src* může obsahovat libovolné množství těchto balíčků (každý je umístěn ve svém adresáři). Adresář *build* je určený pro ukládání mezilehlých souborů vytvořených ROSem. Adresář *install* je místo, kam se každý balíček bude instalovat a kde jsou nastavovací soubory daného workspace, které jsou užívány pro tzv. "ozdrojování"(source). A adresář *log* je určen pro logovací informace. V adresáři *src* tedy byly vytvořeny dva ROS packages pro fungující implementace SLAM problému přepsaných do Pythonu: *ekfSLAM* a *fastslam* . ROS 2 totiž umožňuje (poskytuje oficiální podporu) psát zdrojové kódy v Pythonu nebo v C++. Důležité soubory, které je potřeba patřičně nastavit, daného package (v případě Pythonu) jsou: *package.xml* - validní (testovaný validity oproti danému schematu) XML dokument obsahující meta informace o daném package, *setup.cfg* - využíván k uvedení vytvořených spustitelných souborů aplikace tak, aby je příkaz `ros2 run` měl k dispozici a *setup.py* pro instrukce, jak se package bude instalovat. V případě Pythonu se uvnitř daného package (v adresáři) nachází další stejně pojmenovaný podadresář. To je prostor pro umístění zdrojových kódů chtěné aplikace, tedy pro tuto práci místo pro nahrání upravených verzí přepsaných kódů.

Změny přepsaných implementací pro ROS 2.

Aby mohly být přepsané kódy začleněny do ROS 2, bylo provedeno několik změn. Pokud se v daném modulu importuje nějaký jiný vytvořený modul, je potřeba před název tohoto modulu umístiti tečku: `import .example` . Je to z důvodu toho, že když ROS 2 sestavuje soubory ze *src*, dává je do adresáře `ros2_ws/install/lib/python3.8/site-packages/<package_name>` . Tento adresář je považován za kořenový pro Python balíčky daného ROS 2 package.

V případě FastSLAMu byly kvůli vyhodnocování a vykreslování rozšířeny návratové hodnoty hlavního modulu *fastslam1_sim.py*. Kromě již popsaného seznamu *data*

jsou navráceny 2D numpy.array *xtrue* a *xpath*, reprezentující matici o třech řádcích (stav robota) a s počtem sloupců odpovídající počtu časových okamžiků simulace (na daném indexu sloupce se nachází stav robota v patřičném časovém okamžiku). Proměnná *xtrue* je pro skutečné stavy, *xpath* pak pro odhadované. Byla také vytvořena matice pro odhadované pozice landmarků *lms* se všemi časovými kroky. Posléze z důvodu toho, že byla použita pouze jedna verze FastSLAMu, byly všechny tři moduly z adresáře *fastslam 1* začleněny mezi ostatní do adresáře o stupeň výše a adresář *fastslam 1* byl zrušen.

Vytvoření tzv. node.

Další změnou je to, že hlavní moduly pro simulace (*..._sim.py*) nejsou spouštěny modulem *launch.py* . Soubory *launch.py* nebyly ani do ROSu nahrány. Spuštění zde obstarává *node.py*. Node (uzel) je v ROSu něco jako základní stavební jednotka. Aplikace jich může mít neomezený počet. Každý node může zasílat a dostávat data od ostatních uzlů. Existuje více schémat komunikace např. zprávy skrz topiky nebo pomocí služeb, akcí atd. V této práci byl v každém package vytvořen jeden node (*node.py*) publikující zprávy do topiků. Dále je uvedeno, z jakých částí jsou tyto moduly této aplikace složeny.

Po importování všech potřebných modulů je zadefinována třída (buď EKF nebo FAST) dědící od již implementované třídy Node z knihovny *rclpy*. V konstruktoru této třídy se nejprve zavolá konstruktor třídy Node a nastaví se jméno uzlu. Následuje vytvoření publisherů pomocí metody ve tvaru `create_publisher(datový typ zprávy, název topiku, velikost fronty)`. Po konstruktoru je zadefinována metoda `run()` spouštějící danou simulaci. Tato metoda vrací výsledky simulace. Již mimo třídu EKF nebo FAST se nachází vstupní bod celé aplikace, hlavní metoda `main()` . V ní je nejprve vytvořena instance výše zmíněných tříd. Poté je spuštěna simulace zavoláním metody `run()` . Potom je z výsledných dat simulace provedeno vyhodnocení dle metrik uvedených v teoretické části (2.26). Metody implementující dané metriky byly vytvořeny za metodou `main()` na konci modulu. Další větší částí tohoto modulu je příprava dat pro Rviz 2. Příprava má zajistit vytvoření zpráv publikovaných do daných topiků, které poté bude Rviz 2 odebírat. Nakonec jsou zprávy do patřičných topiků publikovány pomocí vytvořených publisherů jejich metodou `publish()` . Aby se stihl Rviz 2 spustit a nastavit, opakuje se publikování těch samých zpráv stokrát vždy po jedné vteřině. Poté je daná instance node smazána. Ještě by bylo vhodné doplnit, že během vývoje tohoto modulu byly vytvořeny struktury zpráv, které nejsou určeny pro Rviz 2, pro případ, kdyby např. bylo potřeba tyto data odebírat jinými uzly aplikace (pomocí subscriberů). Jsou připraveny k použití odkomentováním části pro vytvoření jejich publisherů v konstruktoru třídy a příkazů pro zahájení publikování v metodě `main()` .

Finální příprava.

Posledním krokem před spuštěním, je úprava již popsaných potřebných souborů daných package. Do *package.xml* bylo přidáno následující - název balíčku, např. `<name>fastslam</name>` a potřebné závislosti pro spuštění, které je potřeba dodat na základě využitých importovaných nestandardních Pythnovských knihoven (v tomto případě přidány dvě řádky):
`<exec_depend>rclpy</exec_depend>`
`<exec_depend>std_msgs</exec_depend>`
V modulu *setup.py* je potřeba uvést tzv. vstupní bod aplikace (entry_point): to, co pod jakým názvem bude ve výsledku pouštěno, v tomto případě metoda `main()` modulu *node.py*. Do slovníku *entry_points["console_scripts"]* tedy byla vložena řádka
`'ekf_sim = ekfSLAM.node:main',`
v package fastslam obdobně: `'fast_sim = fastslam.node:main',`

Nyní již je aplikace připravena k sestavení. Pro možnost snadné instalace byl opět na Githubu vytvořen samostatný repozitář src:

`https://github.com/vao25/src`

Ten obsahově plně odpovídá adresáři *src* již popsaného workspace *ros2_ws* . Stačí si tedy daný workspace vytvořit a adresář si tam naklonovat. Nachází se zde stručný návod, který může být následován. V něm je pro případné použití obsažen popis již zmíněných topiků a zpráv do nich publikovaných, které mohou být dále využity.

3.4.2 Spuštění simulace

V daném workspace je nejprve nutné aplikaci sestavit (build). K sestavení balíčků je využíván nástroj `colcon`. Je to nástroj pro sestavovací systém ament (v případě Pythonu typ ament_python). Pro sestavení daného package je tedy nezbytné zadat příkaz:

`colcon build --packages-select fastslam` (případně `ekfSLAM`)

Výstup tohoto příkazu je v adresáři *install* (vygenerovaný *.bash* soubor, který nastavuje prostředí). Prostředí je potřeba "ozdrojovat"(source) - před použitím jakéhokoliv nainstalovaného spustitelného souboru je nejdříve nutné nastavit příslušné cesty (*setup.bash* přidá veškeré požadované součásti a cestu nastaví). "Ozdrojování"se vykoná příkazem:

`source install/setup.bash`

S "ozdrojovaným"prostředím lze spustit spustitelné soubory, které colcon sestavil. Požadovaná simulace se pak spustí pomocí přidaného vstupního bodu příkazem:

`ros2 run <package> <entry_point>`

např. `ros2 run fastslam fast_sim`

3.5 Experimenty

Na závěr praktické části dojde na již avizovaný experiment a výsledky simulací budou sloužit k porovnání uvedených přístupů k řešení SLAM problematiky. Všechny experimenty se vykonají s mapou již uvedenou na Obrázku 3.2. Příslušející soubor *file.json* je přiložen i v daných repozitářích na patřičných místech. Simulace tedy budou vykonány pro toto prostředí:

Zdrojový kód 3.1: Prostředí (file.json) pro vykonané experimenty

```
{
"x3": 0,
"lm": [[3,-26], [33,-33], [25,-68], [75,-66], [74,-36],
    [99,4], [49,27], [80,60], [55,89], [13,81], [-16,49],
    [-66,84], [-96,61], [-76,36], [-87,-21], [-104,-32],
    [-93,-78], [-56,-56], [-35,-19], [-56,16], [-7,20],
    [26,3], [-20,80], [-42,46], [25,27], [91,28], [-14,-79],
    [-53,-92], [-86,15], [-127,23], [-26,5], [56,-17],
    [52,-83], [21,-96], [48,58]],
"wp": [[13,-42], [45,-55], [86,-45], [94,-17], [65,6],
    [72,32], [71,71], [34,76], [13,52], [-33,67], [-75,69],
    [-97,42], [-108,6], [-86,-26], [-84,-64], [-39,-81],
    [-18,-52]]
}
```

Dále je nutné uvést, s jakými hodnotami parametrů a přepínačů byly simulace vykonány. Pro zvýšení přehlednosti jsou jednotlivé obsahy konfiguračních souborů *configfile.py* uvedeny až v příloze A. Je také vhodné uvést upozornění pro FastSLAM simulaci, kde byl parametr NPARTICLES (počet částic) nastaven 100, takže simulace trvá trochu déle.

3.5.1 Výsledky experimentů

Jednotlivé přístupy řešící SLAM úlohu byly porovnány dle metrik uvedených v teoretické části. Poznámka k UKF-SLAMu: vyhodnocení bylo provedeno s výsledky z originálních skriptů v MATLABu. Zaokrouhlené výsledky na příslušný počet desetinných míst jsou k vidění v následující tabulce (3.1):

Tabulka 3.1: Výsledky simulací

Metrika pro porovnání	EKF-SLAM	FastSLAM	UKF-SLAM
RMSE (2.26)	0,88	6,27	2,33
ε_{trans} (2.28)	$5{,}934 \cdot 10^{-4}$	$5{,}868 \cdot 10^{-1}$	$1{,}415 \cdot 10^{-3}$
ε_{rot} (2.29)	$1{,}77 \cdot 10^{-6}$	$6{,}34 \cdot 10^{-5}$	$2{,}02 \cdot 10^{-6}$
ε (2.27)	$5{,}951 \cdot 10^{-4}$	$5{,}869 \cdot 10^{-1}$	$1{,}417 \cdot 10^{-3}$

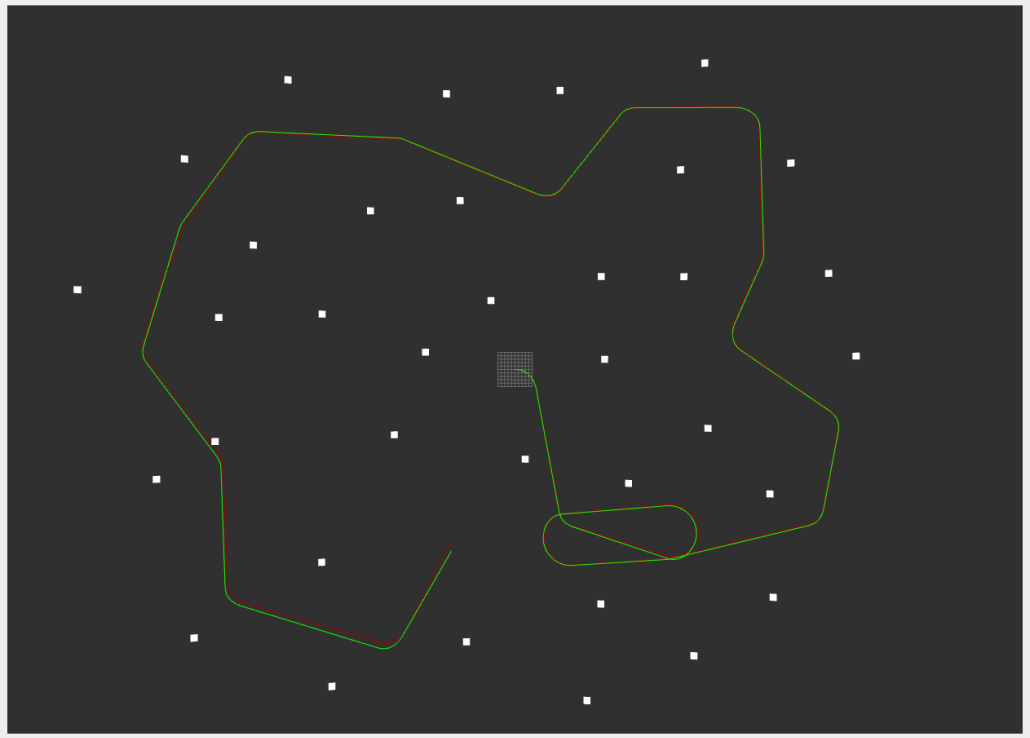
V provedených experimentech tedy ve všech dílčích aspektech nejlépe vychází EKF-SLAM, nejhůře (a to dost výrazně) pak FastSLAM. Výsledný rozdíl je markantní a je viditelný i z vizualizací na Obrázcích 3.3 a 3.4 (popis vykreslení je uveden v následujícím paragrafu). Na Obrázku 3.3 červená křivka celou dobu až do samého závěru kopíruje zelenou, kdežto na Obrázku 3.4 s postupem času mezi těmito dvěma křivkami vzniká stále větší rozestup. Navíc v případě FastSLAMU je vidět i "zubatost". Nebylo by asi správné práci uzavřít s tvrzením, že by se měl používat EKF-SLAM a v žádném případě nikdy FastSLAM (to by se nikde neuváděl v přehledech základních přístupů řešící SLAM problém). Lepší je to asi okomentovat takto: pro dané verze algoritmů (základní verze EKF-SLAMu a FastSLAM 1.0) vzniklých přepsáním originálních verzí a pro originálně implementovaný UKF-SLAM, pro dané nastavení konfiguračních souborů a pro uvažovaný typ mapy lze z výsledků konstatovat, že nejlepším řešením je EKF-SLAM, poté UKF-SLAM a nejhorším FastSLAM. Toto tvrzení lze pronést s ohledem na to, že snad kromě porovnání ukazatele ε_{rot} mezi EKF-SLAMem a UKF-SLAMem je mezi všemi přístupy rozdíl v řádu dekád (jejich mocnin). Jeden z postřehů je také to, že tak jak byly zadefinovány metriky ε_{trans} a ε_{rot}, hodnotu výsledného ε určuje především ε_{trans}. Zároveň však nelze říci, že algoritmy určují orientaci (oproti určovaní polohy) skoro bezchybně, úhly a pozice jsou totiž počítány úplně pro jiné číselné hodnoty a s jinak zadefinovanou metrikou.

Samozřejmě více profesionálnější by byl statistický přístup, udělat experimenty vícekrát pro určité nastavení parametrů v konfiguračních souborech a pro určitou mapu. Ze získaných dat potom udělat např. průměr a směrodatnou odchylku či využít jiný statistický ukazatel. A poté třeba mapu a některé parametry změnit a zase udělat více simulací a pro tyto dané podmínky znovu vyhodnotit.

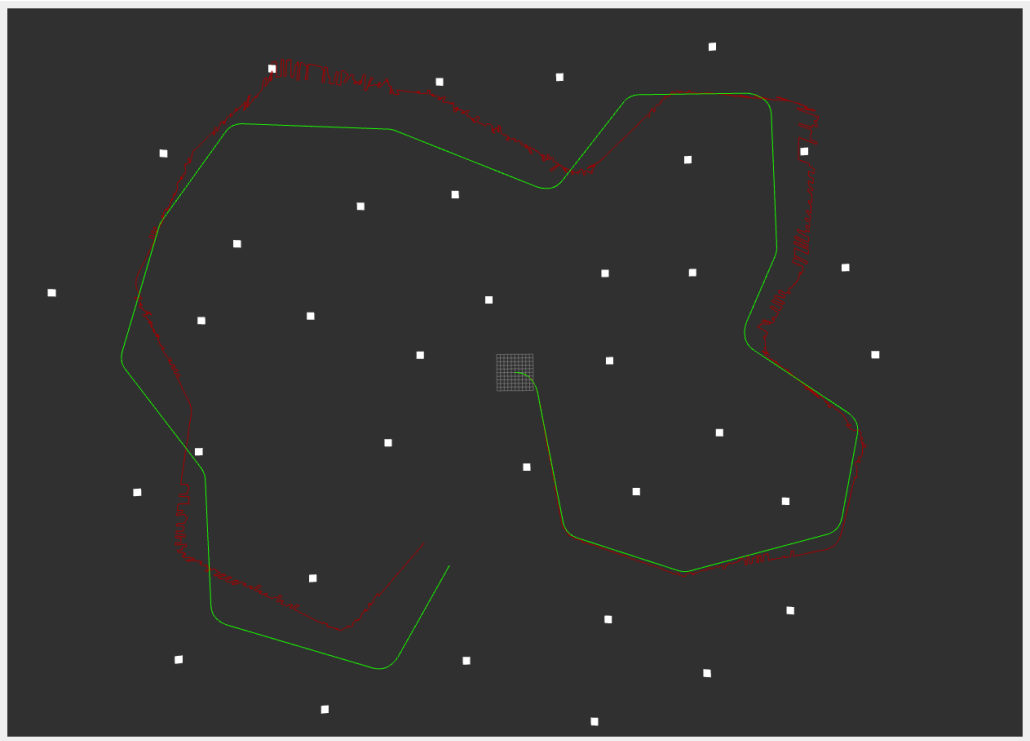
Vizualizace výsledků.

Zobrazení je uděláno pomocí nástroje Rviz 2. Je dostupný ve formě ROS 2 package - **rviz2**. Je to grafické uživatelské rozhraní ROSu, nástroj pro 3D vizualizaci v problematice robotiky. Po jeho spuštění je potřeba přihlásit odebírání příslušných topiků. To se učiní v levém dolní rohu kliknutím na tlačítko *Add* a následně výběrem typu přijímaných zpráv, které se mají vizualizovat. V tomto případě dvakrát přidat

typ *Path* pro skutečnou a odhadovanou trajektorii a *PointCloud2* pro zobrazení landmarků. Po jejich přidání je třeba každý z nich rozkliknout a vybrat příslušný název topiku - v případě typu *Path* zvolit `/true_path` a `/estimated_path`, pro *PointCloud2* `/landmarks`. Aby byly landmarky rozeznatelné, je ještě potřeba zvětšit jejich velikost: atribut Size nastavit např. na 2. Pro rozeznatelnost cest je dobré každou trajektorii nastavit na jinou barvu. V následujících snímcích je vždy zelená barva pro skutečnou trajektorii, červená pro odhadovanou (landmarky jsou vyznačeny bílými čtverečky).



Obrázek 3.3: Vizualizace EKF-SLAM simulace v grafickém prostředí Rviz



Obrázek 3.4: Vizualizace FastSLAM simulace

Závěr 4

Tato bakalářská práce se věnovala problematice Simultánní lokalizace a mapování. Jejím cílem byly implementace simulátorů známých přístupů k problému SLAMu, které bude možné dále využít k výukovým účelům. Na začátku teoretické části bylo zadefinováno matematické pozadí pravděpodobnostního SLAMu. Nejdelší úsek této části byl věnován popisu třech základních způsobů řešení SLAM úlohy. Bylo také uvedeno, pomocí jakých metrik lze SLAM algoritmy mezi sebou porovnávat. Náplní Praktické části bylo přepsání těchto tří algoritmů, implementovaných Timem Baileyem před téměř dvaceti lety v MATLABu, do Pythonu (tedy implementace teoretických poznatků), připravit je pro ROS 2 a data z (v něm) provedených simulací okamžitě poskytnout vizualizačnímu nástroji Rviz 2 a zároveň z těchto dat provést porovnání přesnosti fungování těchto simulátorů (SLAM algoritmů). Došlo na realizaci všech výše uvedených kroků. Co se týče samotného přepisu, ač se to nezdá, tak je poměrně náročný vzhledem k odlišnosti výše zmíněných jazyků. Zároveň byla vyzkoušena činnost práce s profesionálními nástroji jako GitHub a ROS 2 užívaných hojně v praxi. Během práce vznikl i jednoduchý nástroj umožňující nakonfigurování zvoleného prostředí (mapy) robota pro získání vstupních dat pro simulátory.

Jako nejlepší z pohledu všech použitých metrik v této práci vyšlo EKF-SLAM řešení, poté UKF-SLAM a nejhůře s velkým odstupem od těchto dvou potom FastSLAM. Ač byly implementovány nejznámější přístupy k řešení Simultánní lokalizace a mapování ve 2D, tak je zde dále prostor pro další budoucí vylepšení. Jedním směrem je možná implementace 3D simulátoru pro tyto přístupy. Druhou možností je implementace pokročilejších, efektivnějších verzí těchto algoritmů nebo dalších přístupů. Zejména pak metody založené na řešení metodou nejmenších čtverců označované například jako Lazy SLAM, který řeší úlohu pomocí metod optimalizace.

Nastavení konfiguračních souborů pro provedené simulace

A

Simulace byly provedeny s následujícími nastaveními konfiguračních souborů *configfile.py*:

Zdrojový kód A.1: Nastavení configfile.py pro provedenou EKF-SLAM simulaci

```
import numpy as np

"""
Configuration file
Permits various adjustments to parameters of the SLAM
    algorithm.
See ekfslam_sim.py for more information
"""

# control parameters
V = 3 # m/s
MAXG = 30*np.pi/180 # radians, maximum steering angle (-MAXG
    < g < MAXG)
RATEG = 20*np.pi/180 # rad/s, maximum rate of change in steer
     angle
WHEELBASE = 4 # metres, vehicle wheel-base
DT_CONTROLS = 0.025 # seconds, time interval between control
    signals

# control noises
sigmaV = 0.3 # m/s
sigmaG = (3.0*np.pi/180) # radians
Q = np.array([[sigmaV**2, 0], [0, sigmaG**2]])

# observation parameters
MAX_RANGE = 30.0 # metres
```

```
23 DT_OBSERVE = 8*DT_CONTROLS # seconds, time interval between
       observations
24
25 # observation noises
26 sigmaR = 0.1 # metres
27 sigmaB = (1.0*np.pi/180) # radians
28 R = np.array([[sigmaR**2, 0], [0, sigmaB**2]])
29
30 # data association innovation gates (Mahalanobis distances)
31 GATE_REJECT = 4.0 # maximum distance for association
32 GATE_AUGMENT = 25.0 # minimum distance for creation of new
       feature
33 # For 2-D observation:
34 #    - common gates are: 1-sigma (1.0), 2-sigma (4.0), 3-sigma
       (9.0), 4-sigma (16.0)
35 #    - percent probability mass is: 1-sigma bounds 40%, 2-
       sigma 86%, 3-sigma 99%, 4-sigma 99.9%.
36
37 # waypoint proximity
38 AT_WAYPOINT = 1.0 # metres, distance from current waypoint at
        which to switch to next waypoint
39 NUMBER_LOOPS = 2 # number of loops through the waypoint list
40
41 # switches
42 SWITCH_CONTROL_NOISE = 1 # if 0, velocity and gamma are
       perfect
43 SWITCH_SENSOR_NOISE = 1 # if 0, measurements are perfect
44 SWITCH_INFLATE_NOISE = 0 # if 1, the estimated Q and R are
       inflated (ie, add stabilising noise)
45 SWITCH_HEADING_KNOWN = 0 # if 1, the vehicle heading is
       observed directly at each iteration
46 SWITCH_ASSOCIATION_KNOWN= 0 # if 1, associations are given,
       if 0, they are estimated using gates
47 SWITCH_BATCH_UPDATE= 1 # if 1, process scan in batch, if 0,
       process sequentially
48 SWITCH_SEED_RANDOM= 0 # if not 0, seed the randn() with its
       value at beginning of simulation (for repeatability)
```

Zdrojový kód A.2: Nastavení configfile.py pro provedenou FastSLAM simulaci

```
1 import numpy as np
2
3 """
4 Configuration file
5 Permits various adjustments to parameters of the FastSLAM
      algorithm.
6 See fastslam_sim.py for more information
7 """
```

```
# control parameters
V = 3 # m/s
MAXG = 30*np.pi/180 # radians, maximum steering angle (-MAXG < g < MAXG)
RATEG = 20*np.pi/180 # rad/s, maximum rate of change in steer angle
WHEELBASE = 4 # metres, vehicle wheel-base
DT_CONTROLS = 0.025 # seconds, time interval between control signals

# control noises
sigmaV = 0.3 # m/s
sigmaG = (3.0*np.pi/180) # radians
Q = np.array([[sigmaV**2, 0], [0, sigmaG**2]])

# observation parameters
MAX_RANGE = 30.0 # metres
DT_OBSERVE = 8*DT_CONTROLS # seconds, time interval between observations

# observation noises
sigmaR = 0.1 # metres
sigmaB = (1.0*np.pi/180) # radians
R = np.array([[sigmaR**2, 0], [0, sigmaB**2]])

# waypoint proximity
AT_WAYPOINT = 1.0 # metres, distance from current waypoint at which to switch to next waypoint
NUMBER_LOOPS = 1 # number of loops through the waypoint list

# resampling
NPARTICLES= 100;
NEFFECTIVE= 0.75*NPARTICLES; # minimum number of effective particles before resampling

# switches
SWITCH_CONTROL_NOISE = 1 # if 0, velocity and gamma are perfect
SWITCH_SENSOR_NOISE = 1 # if 0, measurements are perfect
SWITCH_INFLATE_NOISE = 0 # if 1, the estimated Q and R are inflated (ie, add stabilising noise)
SWITCH_PREDICT_NOISE = 1; # sample noise from predict (usually 1 for fastslam1.0 and 0 for fastslam2.0)
SWITCH_HEADING_KNOWN = 0 # if 1, the vehicle heading is observed directly at each iteration
SWITCH_RESAMPLE= 1;
SWITCH_SEED_RANDOM= 0 # if not 0, seed the randn() with its value at beginning of simulation (for repeatability)
```

Zdrojový kód A.3: Nastavení configfile.m pro provedenou UKF-SLAM simulaci

```
%%% Configuration file
%%% Permits various adjustments to parameters of the SLAM
    algorithm.
%%% See ukfslam_sim.m for more information

% control parameters
V= 3; % m/s
MAXG= 30*pi/180; % radians, maximum steering angle (-MAXG < g
    < MAXG)
RATEG= 20*pi/180; % rad/s, maximum rate of change in steer
    angle
WHEELBASE= 4; % metres, vehicle wheel-base
DT_CONTROLS= 0.025; % seconds, time interval between control
    signals

% control noises
sigmaV= 0.3; % m/s
sigmaG= (3.0*pi/180); % radians
Q= [sigmaV^2 0; 0 sigmaG^2];
% observation parameters
MAX_RANGE= 30.0; % metres
DT_OBSERVE= 8*DT_CONTROLS; % seconds, time interval between
    observations
% observation noises
sigmaR= 0.1; % metres
sigmaB= (1.0*pi/180); % radians
R= [sigmaR^2 0; 0 sigmaB^2];
% waypoint proximity
AT_WAYPOINT= 1.0; % metres, distance from current waypoint at
    which to switch to next waypoint
NUMBER_LOOPS= 1; % number of loops through the waypoint list

% switches
SWITCH_CONTROL_NOISE= 1; % if 0, velocity and gamma are
    perfect
SWITCH_SENSOR_NOISE = 1; % if 0, measurements are perfect
SWITCH_INFLATE_NOISE= 0; % if 1, the estimated Q and R are
    inflated (ie, add stabilising noise)
SWITCH_HEADING_KNOWN= 0; % if 1, the vehicle heading is
    observed directly at each iteration
SWITCH_SEED_RANDOM= 0; % if not 0, seed the randn() with its
    value at beginning of simulation (for repeatability)
SWITCH_PROFILE= 0; % if 1, turn on MatLab profiling to
    measure time consumed by simulator functions
SWITCH_GRAPHICS= 0; % if 0, avoids plotting most animation
    data to maximise simulation speed
```

Bibliografie

1. DURRANT-WHYTE, Hugh; BAILEY, Tim. Simultaneous Localization and Mapping: Part I. *IEEE Robotics & Automation Magazine*. 2006, s. 99–108.
2. RIISGAARD, Søren; BLAS, Morten Rufus. *SLAM for Dummies: A Tutorial Approach to Simultaneous Localization and Mapping*. mit.edu, 2005. Dostupné také z: `https://dspace.mit.edu/bitstream/handle/1721.1/119149/16-412j-spring-2005/contents/projects/1aslam_blas_repo.pdf`.
3. STACHNISS, Cyrill; LEONARD, John J.; THRUN, Sebastian. Simultaneous Localization and Mapping. In: *Springer Handbook of Robotics*. Ed. SICILIANO, Bruno; KHATIB, Oussama. Cham: Springer International Publishing, 2016, s. 1153–1176. ISBN 978-3-319-32552-1. Dostupné z DOI: `10.1007/978-3-319-32552-1_46`.
4. STACHNISS, Cyrill. *Robot Mapping - Unscented Kalman Filter*. UNI FREIBURG, 2013. Dostupné také z: `http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam06-ukf.pdf`.
5. KÜMMERLE, Rainer et al. On measuring the accuracy of SLAM algorithms. *Autonomous Robots*. 2009, č. 27, s. 387–407.
6. WIKIPEDIE. *Rozchod kol*. 2023. Dostupné také z: `https://commons.wikimedia.org/w/index.php?curid=22533266`.
7. MACENSKI, Steven; FOOTE, Tully; GERKEY, Brian; LALANCETTE, Chris; WOODALL, William. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics*. 2022, roč. 7, č. 66, eabm6074. Dostupné z DOI: `10.1126/scirobotics.abm6074`.

Seznam obrázků

Seznam tabulek

Seznam výpisů