VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

KONTINUÁLNÍ LOKALIZACE MOBILNÍHO ROBOTA S VYUŽITÍM AGREGACE DAT Z VÍCE POZIČNÍCH SYSTÉMŮ

CONTINUOUS LOCALIZATION OF A MOBILE ROBOT BASED ON AN AGGREGATION OF DATA FROM DIFFERENT POSITIONING SYSTEMS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ZDENĚK BRHEL**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2020

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2019/2020

Zadání bakalářské práce

Student: **Brhel Zdeněk**
Program: Informační technologie
Název: **Kontinuální lokalizace mobilního robota s využitím agregace dat z více pozičních systémů**
Continuous Localization of a Mobile Robot Based on an Aggregation of Data from Different Positioning Systems
Kategorie: Umělá inteligence

Zadání:

1. Prostudujte problematiku agregace zašuměných dat z více zdrojů za účelem získání přesnější informace. Zaměřte se na typická řešení aplikovaná v mobilní robotice (např. Kalmanova filtrace, MCL, fuzzy regulace apod.).
2. Navrhněte vlastní obecný systém pro agregaci poziční informace z více zdrojů pro kolové roboty (režim řízení Ackermann - natáčecí přední náprava), který bude zohledňovat rozdílnou chybovou charakteristiku získaných měření, a to za účelem dosažení přesnějšího výsledku v kontinuální lokalizaci.
3. Navržený systém implementujte jako ROS uzel a testuje na simulovaných i reálných záznamech z jízd robotu (např. datová sada KITTI dataset), a to alespoň v pěti situacích.
4. Zhodnoťte dosažené výsledky a navrhněte možnosti dalšího pokračování práce. Vytvořte plakátek a krátké video prezentující výsledky vaší práce.

Literatura:

- Bayesian Filtering: From Kalman Filter to Particle filters, and Beyond [online]. Toronto: Chen, Zhe, 2013 [cit. 2018-09-19]
- Pose Estimation of a Mobile Robot Based on Fusion of IMU Data and Vision Data Using an Extended Kalman Filter [online]. Basel: MDPI, 2017 [cit. 2018-09-19]

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2019
Datum odevzdání: 28. května 2020
Datum schválení: 15. června 2020

Abstrakt

Cílem práce je vytvořit aplikaci, která bude přijímat a agregovat data z pozičních systémů (senzorů) a následně pak odhadovat nejpravděpodobnější pozici robota.

Řešením vznikne aplikace, která bude implementována v jazyce C++ s použitím robotického operačního systému (ROS). Aplikace bude konat funkci, která je popsána výše.

Pro filtrování dat od šumu bude použit rozšířený Kalmanův filtr.

Abstract

The main goal of this thesis is to create an application, which will receive and aggregate data from position systems (sensors) and then estimate the most likely position of the robot.

The solution will be an application that will be implemented in language C++ and will be using robot operating system (ROS). This application will perform whole process of aggregation and estimation.

The Extended Kalman filter will be used for filtering noise from data.

Klíčová slova

Lokalizace, agregace dat z více pozičních zdrojů, orientace ve venkovním prostředím, Robotický operační systém, ROS, C++, filtrování dat, odometrie, senzory, IMU, LiDARy, GPS, odhadování stavu robota.

Keywords

Localization, agregation of data from different position systems, orientation in outdoor environment, Robotic operating system, ROS, C++, odometry, sensors, IMUs, LiDARs, GPS, robot state estimation.

Citace

BRHEL, Zdeněk. *Kontinuální lokalizace mobilního robota s využitím agregace dat z více pozičních systémů*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Kontinuální lokalizace mobilního robota s využitím agregace dat z více pozičních systémů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Jaroslava Rozmana, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Zdeněk Brhel
16. června 2020

Poděkování

Děkuji mému vedoucímů Ing. Jaroslavu Rozmanovi Ph.D., který vynaložil svůj čas a úsilí, aby mě navedl na správný směr a poskytl cenné informace, díky kterým jsem porozuměl problematice, kterou se tato práce zabývá.

Dále bych chtěl poděkovat mému zadavateli, Ing. Davidu Hermanovi, a Ing. Jiřímu Župkovi za mnohé přínosné diskuze, zpětnou vazbu a poskytnutí hardwaru, na kterém jsem mohl výslednou práci testovat.

Obsah

Kapitola 1

Úvod

Lokalizace mobilního robota v předem nezmapovaném prostředí představuje jeden ze základních problémů mobilní robotiky. Řešení tohoto problému je podmínkou pro autonomní chování robotů a předpokladem pro řešení dalších úkolů (navigace, mapování).

Na otázku, proč řešit problematiku lokalizace robota, existuje více odpovědí. Robot, jenž se umí orientovat v prostoru, může nahradit člověka v životu nebezpečných prostředích (například přírodní katastrofy, radioaktivní oblast, vojenské aplikace), v člověku nepřístupných prostředích (činnost ve vesmíru, vysokých či nízkých teplotách, stísněné prostory), v dopravě (autopilot, automatické řízení tramvají, automaticky řízená vozidla), v domácnosti (inteligentní vysavače, asistenční roboti), v logistice (třídění a řazení zboží ve skladu) a v mnohých dalších situacích.

Víme, že na to abychom mohli zjistit polohu robota je potřeba měření. V našem případě je takové měření tvořeno více zdroji (které jsou uvedeny v kapitole 3). Každý z těchto zdrojů kontinuálně měří a zasílá data pro zpracování.

Předpokládáme, že žádné měření není přesné. Vlivem vnějšího rušení (robot je v uzavřeném objektu, jede pro nerovné ploše) vznikají při měření nepřesnosti (šum). Problém nastává, pokud máme určovat pozici právě ve chvíli, kdy je aktuální měření nepřesné, a pak se tedy zdaleka neblíží aktuální pozici.

V kapitole 4 probereme některé aktuálně používané metody pro řešení nepřesnosti měření.

Kapitola Robotický operační systém (5) popisuje principy a využití tohoto frameworku.

Různé návrhy takových systémů včetně vlastního návrhu jsou popsány v kapitole 6. Jsou uvedeny hlavní aspekty aplikace a dále testování a experimenty (6.3) v aplikaci a jejich výstupy.

V závěrečné kapitole 7 je pak uveden závěr a návrh na pokračování této práce.

Kapitola 2

Nejistota v robotice

Co je to vlastně robotika? Robotika je věda, která se zabývá vnímáním a manipulací fyzického světa přes počítačem řízené přístroje. Jako úspěšné příklady robotických systémů lze uvést mobilní platformy pro průzkum vesmíru a planet, robotické ruce sloužící v průmyslu na montážních linkách a manipulátory, které jsou schopny pomáhat chirurgům při operacích. Robotické systémy, které se vyskytují v reálném fyzickém světě, jsou schopny vnímat informace ze svého prostředí přes vestavěné senzory. Navíc díky svým komponentám (mechanickým pažím, servám, ...) jsou schopny svým okolím manipulovat. I když robotika je teprve ve svém raném věku, myšlenka inteligentních manipulujících zařízení má obrovský potenciál pro změnu společnosti. Přesto, že by bylo skvělé, kdyby naše domovy byly plné inteligentních asistentů, kteří by si vzali na starost všechny opravy a udržovali dům, robotika má před sebou ještě dlouho cestu a nyní vyvíjení roboti musí počítat s obrovskou nejistotou, která existuje ve fyzickém světě.

Robotická prostředí jsou ze své podstaty nepředpovídatelná, což je první a největší faktor, který tuto nejistotu ovlivňuje. Přestože stupeň nejistoty v dobře strukturovaných prostředích, jako jsou výše zmíněné montážní linky, je malý, v prostředích jako jsou dálnice a domovy, je velmi dynamický a rychle se měnící. To tuto nejistotu dělá téměř neurčitelnou. Nejistota je obzvlášť vysoká pro roboty, kteří pracují v blízkosti lidí. Roboti pro vnímání svého okolí využívají senzorů3. Vnímání senzoru je však limitováno jejich schopnostmi, pro příklad kamery nevidí skrz zdi a rozlišení nemusí být dostatečné. Senzory jsou také pod zátěží nepředvídatelného šumu[1]. Takový senzor je navíc pak limitován použitelností informací, které je schopen poskytnout. Existje zde také možnost, že se senzor pokazí, a detekování takového rozbitého senzoru může být velmi obtížné.

Nejistota může být způsobena jak faktory reálného světa (opotřebením, šumem, mechanickým opotřebením), tak softwarem robota a algoritmickou aproximací.

Do softwaru robota je nutné přidat tzv. model robota. Pojem model je nám schopna vysvětlit opora předmětu *Modelování a simulace* [16], v níž je vysvětlen jako napodobenina systému jiným systémem – v našem případě počítačovým programem. Model systému je abstrakce skutečného sytému a musí napodobovat všechny pro naše účely podstatné vlastnosti daného systému. Příkladem modelu může být soustava diferenciálních rovnic popisující let rakety, nebo její ekvivalent ve tvaru blokového schématu. Tím, že je model pouze napodobenina reálného systému, je do systému vnesena další nejistota.

[1] Znečištění dat vlivem analogového prostředí.

Roboti musejí pracovat v reálném čase. To limituje výpočet, který je hardware schopen vykonat. Většina populárních algoritmů řešící tuto problematiku jsou aproximující, tedy dosahují periodické výsledky za cenu přesnosti.

Řízení nejistoty je tedy nejdůležitějším krokem k fungujícím robotům v reálném světě. Kvůli této problematice vznikl zcela nový pojem – pravděpodobnostní robotika.

2.1 Pravděpodobnostní robotika

Pravděpodobnostní robotika je podle knihy Sebastiana Thruna, Wolframa Burgarda a Dietera Foxe – *Probabilistic Robotics* [18] podskupina robotiky, která se zabývá vnímáním a ovládáním. Rozhoduje se a dělá rozhodnutí podle informací, které vycházejí ze statistických metod. Díky tomu pojme nejistotu, která vznikne ve většině současných aplikací robotiky. V posledních letech se pravděpodobnostní techniky staly jednou z dominantních paradigmatů pro návrhy algoritmů v robotice. Většina algoritmů je založena na stejném základním matematickém základu: Bayesovo pravidlo[2] a jeho rozšíření známé jako Bayesův filtr. Tento matematický zápis je jádrem většiny pravděpodobnostních algoritmů. Pravděpodobnostní robotika předpokládá, že robot, který dbá na svoji vlastní nejistotu a jedná podle ní, bude mít lepší výsledky než ten, který na ni nedbá.

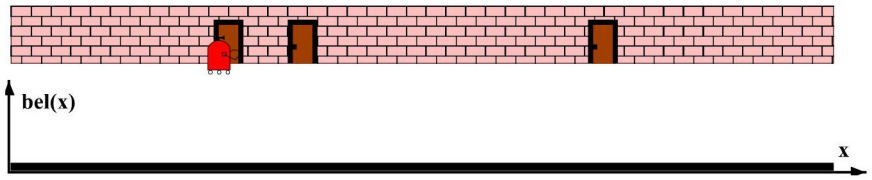
Pravděpodobnostní robotika je relativně nový přístup k robotice, která dbá na nejistotu ve vnímání a akcích robotů. Klíčovou myšlenkou v pravděpodobnostní robotice je reprezentace nejistoty, explicitně používáním teorie pravděpodobnosti. Jinak řečeno, místo spoléhání se na jeden "nejlepší odhad", pravděpodobnostní algoritmy reprezentují informace skrze rozdělení pravděpodobnosti[3] v celém prostoru odhadů. Učiněním tohoto, mohou reprezentovat nejednoznačnost a míru víry (angl. belief) matematicky přívětivým způsobem. Řídící volby mohou být provedeny jako relativní vzhledem k nejistotě a pravděpodobnostní robotika se tak může aktivně rozhodnout snížit svou nejistotu volbou, která se jeví jako nejlepší. V důsledku toho překonávají alternativní techniky v mnoha využívaných aplikacích.

2.2 Lokalizace mobilního robota

Lokalizace mobilního robota je problém odhadování souřadnic robota relativně vzhledem k externímu referenčnímu rámci (například mapy). Při poskytnutí takového rámce prostředí se robot dokáže společně s měřením jeho senzorů lokalizovat. Obrázek 2.1 ilustruje počáteční stav robota, který se snaží nalézt sebe sama ve známém prostředí. O prostředí robota víme, že se v něm nachází troje nerozlišitelné dveře. Úkolem robota je zjistit, kde se nachází skrze vnímání a pohyb.

[2] Bayesova věta je věta teorie pravděpodobnosti, která udává, jak podmíněná pravděpodobnost nějakého jevu souvisí s opačnou podmíněnou pravděpodobností. Více v kapitole 4.1.

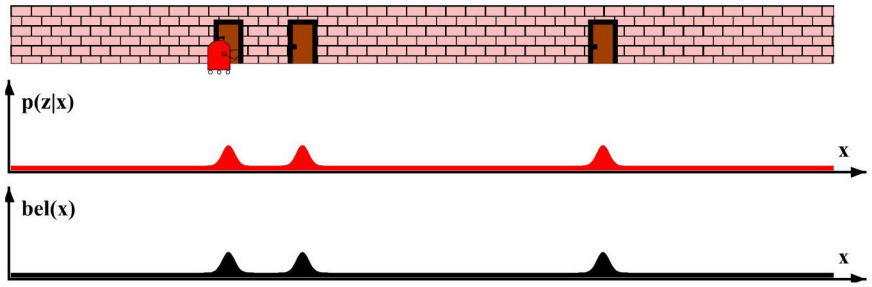
[3] Rozdělení, nebo rozložení pravděpodobnosti náhodné veličiny je pravidlo, které každému jevu popisovanému touto veličinou přiřazuje určitou pravděpodobnost.



Obrázek 2.1: Počáteční stav robota bez uskutečněného měření. [18]

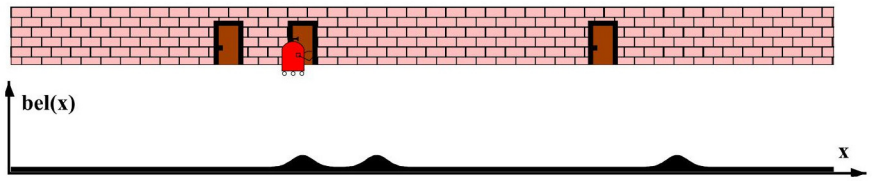
Tento specifický problém je znám jako globální lokalizace. V globální lokalizaci je robot umístěn někde ve známém prostředí a má za úkol se lokalizovat. Pravděpodobnostní paradigma reprezentuje momentální víru robota skrze funkci pravděpodobnostní hustoty $belief$ nad jeho prostorem.

Nyní předpokládejme, že robot provede první senzorové měření a zjistí, že se nachází vedle dveří. Pravděpodobnostní techniky využíjí této informace a aktualizují svoji funkci $belief$. Funkce po aktualizaci je znázorněna v obrázku 2.2. Do funkce $belief$ je umístěna větší pravděpodobnost na místa vedle dveří a menší pravděpodobnost na ostatní místa. Tato distribuce nyní obsahuje tři vrcholy, každý koresponduje právě se dveřmi rozmístěnými po celém prostředí. V tomto měření robot stále neví, kde se nachází. Má však nyní má tři odlišné hypotézy, které jsou stejně přijatelné. Kromě přiřazení pravděpodobnosti oblastem s dveřmi se také přiřazuje pravděpodobnost ostatním oblastem. Tato nezbytná schopnost zajišťuje základ pro uchování robustnosti.



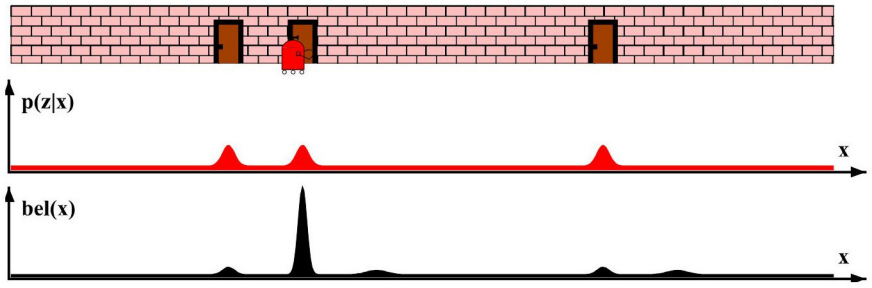
Obrázek 2.2: První měření robota i s aktualizací funkce $belief$. [18]

Ve třetím obrázku 2.3 je zobrazen pohybující se robot. Funkce $belief$ byla posunuta ve směru pohybu. Zároveň nabude většího rozpětí, což se projeví rozprostřením nejistoty na větší plochu a snížením vrcholu.



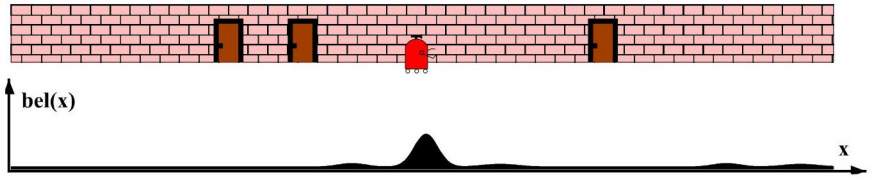
Obrázek 2.3: Robot se pohybuje společně s jeho funkcí $belief$. [18]

Čtvrtá část lokalizace robota (obrázek 2.4 zobrauje *belief* po nalezení dalších dveří. Toto sledování vede náš algoritmus na místo s největší hmotností pravděpodobnosti – vedle druhých dveří. V této chvíli si je robot docela jistý ve své pozici a tedy je schopen se nadále na základě své rychlosti a směru efektivně lokalizovat. Pokaždé, když narazí na dveře, se jeho funkce *belief* aktualizuje a na místa s dveřmi bude přidána patřičná pravděpodobnost.



Obrázek 2.4: Aktualizace funkce *belief* při nalezení dalších dveří. [18]

Na obrázku 2.5 je vidět, jak nyní již lokalizovaný robot pokračuje na cestě chodbou.



Obrázek 2.5: Robot dále pokračuje chodbou společně s funkcí *belief*. [18]

2.2.1 Stav robota

Prostředí robota je charakterizováno jeho stavem.[18] Stav se dá považovat jako sbírka aspektů robota a prostředí, které můžou ovlivnit jeho budoucnost. Stav se časem může měnit, například pozice lidí okolo robota; nebo může zůstat neměnný, jako například zdi budov. Měnící se stav se nazývá *dynamický stav* a neměnící *statický stav*. Stav také zahrnuje proměnné samotného robota, jako jeho pozici, rychlost, informaci o správné funkčnosti senzorů, a podobně. Typické proměnné stavu jsou:

- **Pozice robota**, která se skládá z orientace relativní globálnímu koordinačnímu rámci. Mobilní roboti uchycení k pevnému základu (robotické ramena) mají šest takový proměnných. Tři pro kartézské souřadnice a tři pro jejich úhlové orientace, také známé pod názvem Eulerovy úhly (precese, nutace, rotace). Mobilní roboti, kteří se mohou pohybovat využívají také šest proměnných, přičemž rozdílné jsou však jen první tři. Místo kartézských souřadnic se používá zeměpisná šířka, délka a nadmořská výška. Pozice robota se často uvádí i jako *kinematický stav*.

- **Konfigurace pohybových částí robota**, jako jsou různé robotické manipulátory. Každá možnost, kterou se robot dokáže libovolně pohnout je součástí kinematického stavu robota.

- **Rychlost robota**, kterou tvoří až šest proměnných, z nichž každá odpovídá rychlosti pohybu ve směrech uvedených v pozici robota. Rychlosti jsou často označovány jako *dynamický stav.*

- **Umístění a vlastnosti okolních objektů v prostředí.** Objektem se rozumí strom, zeď a podobně. Vlastnosti takových objektů mohou být jejich vizuální vzhled (barva, textura). V závislosti na zrnitosti stavu, který je modelován, mají prostředí robotů několik desítek až stovek miliard stavových proměnných.

- **Umístění a rychlost pohybujících se objektů.** Robot často není jediný pohyblivý prvek ve svém prostředí. Ostatní prvky také mají jejich vlastní kinematický a dynamický stav.

- Může existovat velké množství jiných stavových proměnných. Například, informace o funkčnosti senzorů, nabití baterie, ...

Stav bude nazýván úplným, jestliže je nejlepší předpovědí budoucnosti. Jinak řečeno, úplnost znamená, že znalosti minulých stavů, měření nebo kontrol nenesou žádné další informace, které by nám pomohly přesněji předpovědět budoucnost. Je důležité znát, že naše definice úplnosti nevyžaduje, aby budoucnost byla deterministickou funkcí stavu. Budoucnost může být stochastická, tedy žádné proměnné předchozího stavu nemohou ovlivnit vývoj pro budoucí stavy. Časové procesy, které splňují tyto podmínky jsou běžné známé jako Markovovy řetězce.[4]

Pojem úplnost stavu má většinou teoretický význam. V praxi není možné určit úplný stav pro jakýkoliv realistický robotický systém. Úplný stav zahrnuje nejen všechny aspekty životního prostředí, které mohou mít dopad na budoucnost, ale i samotného robota, obsah jeho počítačové paměti, nezávislost okolních pohybujících se objektů a podobně. Získat všechny aspekty úplného stavu je prozatím nemožné. Praktické implementace proto vytyčují malou podmnožinu všech stavových proměnných, jako jsou ty výše uvedené. Stav, který nebere v úvahu tyto aspekty se označuje jako *neúplný stav.*

Ve většině robotických aplikacích je stav spojitý, což znamená, že je definován skrze kontinuum. Jako příklad spojitého stavového prostoru je stav robota. Jeho umístění a orientace vzhledem k vnějšímu souřadnému systému. Někdy je však stav diskrétní. Příklad diskrétního stavového prostoru je stavová proměnná, která modeluje, zda senzor funguje nebo ne. Stavové prostory, které obsahují spojité i diskrétní proměnné, se nazývají *hybridní stavové prostory.*

Ve většině případů zajímavých robotických problémů se stav mění v čase. V této práci se budeme zabývat diskrétním časem, tedy všechny zajímavé události budou probíhat v diskrétních časových krocích. Pokud nějaká operace robota začne v daný čas, bude tento čas označen jako $t = 0$.

[4] Popisují diskrétní náhodné procesy, pro které platí, že pravděpodobnosti přechodu do následujícího stavu závisejí pouze na současném stavu, ne na stavech předchozích.[16]

Kapitola 3

Systémy pro lokalizaci

Na začátek je nezbytné definovat, co se myslí pod pojmem systém pro lokalizaci. Nejprve vyjděme z pojmu systém [2]. Systém (či soustava) je uspořádaný celek složený z částí, které na sebe vzájemně působí. Mezi částmi systému mohou probíhat toky informací, hmoty a energie. Lokalizací rozumíme postup, který určí umístění objektu v prostoru. Systém, který je schopen určit polohu objektu v prostoru nazýváme systémem pro lokalizaci (lokalizační systém).

Nyní uveďme přesnější definici tohoto systému. Systémem pro lokalizaci, neboli senzorem se rozumí ucelená a nezávislá jednotka, která provádí samostatná měření a jejíž naměřené hodnoty jsou specifické pro danou jednotku. Některé jednotky měří globální pozici za pomocí družic, jiné měří lokální polohu za pomocí laserů nebo sil působících na jednotku. Je nutné si však uvědomit, že žádný z těchto systémů není přesný a jejich měření obsahuje nějaký šum. Odstraňování šumu je však záležitostí metod pro lokalizaci v kapitole 4.

V této kapitole se budeme zaměřovat na jednotlivé existující lokalizační systémy a jejich vlastnosti.

3.1 Typy senzorů

Obecně existuje mnoho hledisek, podle nichž lze senzory rozdělit do skupin a kategorií. Nejpoužívanější rozdělení vychází z představy, že senzor lze chápat jako převodník mezi snímanou jednotkou vyjádřenou měřenou veličinou a výstupní naměřenou jednotkou. Nás budou zajímat pouze senzory relevantní ku lokalizaci [1].

Z tohoto tvrzení vyplývá dělení:

- **Dělení podle vstupní veličiny** – rozdělení podle druhu měřené veličiny.
  - Senzory mechanických veličin (rychlost, akcelerace, síly, ...)
  - Senzory geometrických (měření polohy, posunutí, ...)
  - Senzory elektrických a magnetických veličin (měření orientace, ...)
- **Dělení podle výstupní veličiny** – rozdělení podle druhu výstupní veličiny.
- **Dle transformace signálu** – rozdělení podle principu převodu měřené veličiny na veličinu výstupní.

3.1.1 LiDAR

LiDAR (z anglického Light Detection And Ranging), také LADAR [15], je metoda dálkového měření, která využívá světlo ve formě pulzního laseru pro měření vzdáleností k objektům. Tyto lasery kombinované s jinými měřenými daty generují precizní, tří-dimenzionální informace o površích jeho prostředí.

LiDAR se zásadně skládá z

- **Laseru** - optický zdroj elektromagnetického záření, které se může a nemusí vyskytovat v člověku viditelném spektru. Tato technologie nás bude zajímat kvůli lokální lokalizaci [13]. Laser (jako přístroj) se skládá z
  - **Aktivního prostředí**, které je tvořeno látkou, která obsahuje oddělené kvantové energetické hladiny elektronů.
  - **Rezonátoru** sloužícího k zesilování světla. Jsou to dvě vzájemně rovnoběžná zrcadla a zároveň kolmá k ose laseru. Jedno z nich je nepropustné a druhé je polopropustné.
  - **Zdroje záření** k dodávání energie elektronům v aktivním prostředí, aby se mohly přesouvat z nižší energetické hladiny na vyšší energetickou hladinu (většinou elektrický proud, výbojka, nebo chemická reakce).
  - **Laserového paprsku** vycházejícího z aktivního prostředí přes polopropustné zrcadlo. Je koherentní[1] a monochromatický.
- **Skeneru** - zařízení, které je schopné zachytit světelné paprsky a převést je do digitální podoby.
- **Speciálního příjmače GPS** - popsáno v podkapitole 3.1.3.

Samotná lokalizace pomocí LiDARů funguje na podobném principu jako skener čárových kódů, kde hlavní myšlenkou je využití světelného paprsku, který vytváří vzájemné zrcadlo a skenuje paprsek, který se vrátí. Lasery využívané v naší práci ještě navíc počítají s rychlostí vrácení paprsku, z čehož jsou schopny určit vzdálenost překážky pomocí jednoduchého vzorce pro dráhu:

$$s = vt \tag{3.1}$$

LiDARy, které budeme v naší práci používat, fungují jako několik vertikálně nad sebou poskládaných laserů. Kromě toho, že tyto lasery neustále vysílají paprsky, se celé zařízení s lasery se otáčí a je tak schopno vnímat celé své okolí. Vzhledem k tomu, že rychlost světla je mnohonásobně vyšší než rychlost otáčení, systém bez problémů zvládá sbírat informace o odrazech. Z takového systému laserů jsme schopni za dobu jedné otočky poskládat matici prostoru kolem systému, avšak pouze v klidném stavu[2]. Problém ale nastává v případě pohybu robota. Vzhledem k neustálému otáčení laserů může pohyb odchýlit výslednou matici.

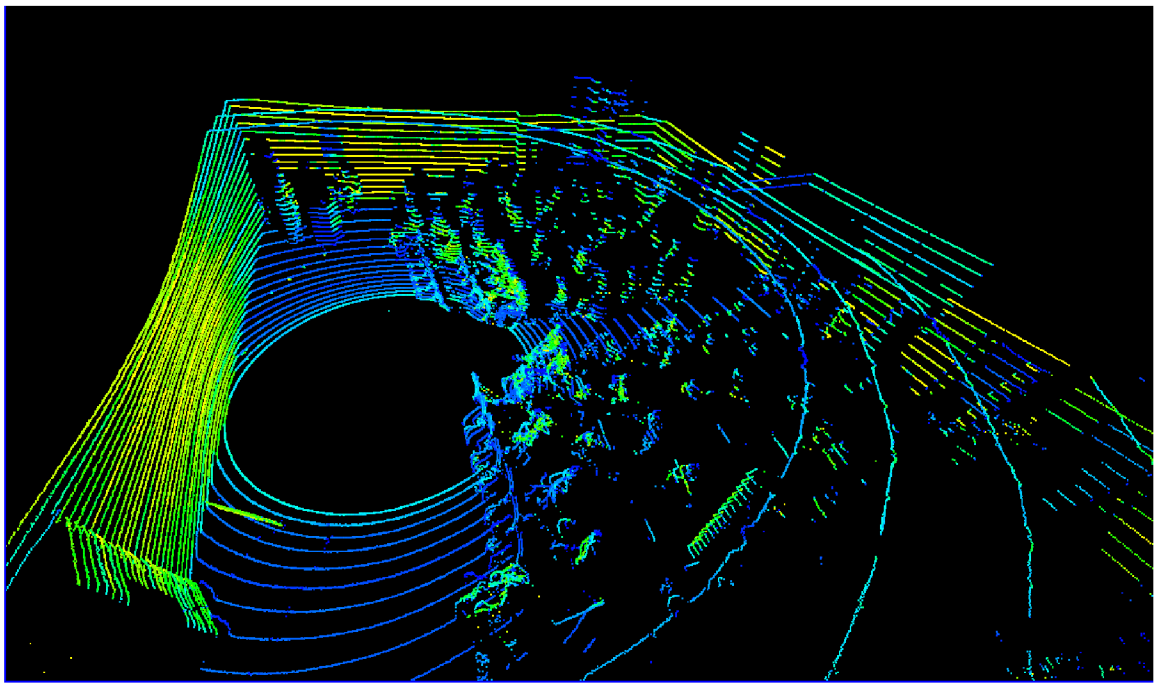
V případě ukládání několika předchozích měření jsme tedy schopni určit rozdíl, a tedy přibližnou rychlost a změnu natočení. Avšak kvůli pomalému otáčení systému není systém schopen změřit kompletní matici ihned, což může být v některých situacích problém. Pro extrémní příklad předpokládejme objekt, který se bude rychle pohybovat okolo robota.

[1] Koherentní vlnění je vlnění o stejné frekvenci, stejného směru kmitání a stejné fázi.

[2] Stav, ve kterém robot stojí na místě

Pokud se bude pohybovat stejným směrem jako rotace laserů, je možné, že ho tento systém nebude schopný detekovat.

Na obrázku 3.1 je znázorněno, jak může vypadat výstup z LiDARu.



Obrázek 3.1: Výstup z LiDARu. [7]

3.1.2 IMU

IMU (z anglické zkratky Inertial measurement unit) je elektronický přistroj, který umožňuje odhadovat orientaci inerciálních sil,[3] kterým je robot vystaven. Princip senzoru je zakládán na měření sil akcelerace a úhlové rychlosti vyvíjené nezávisle na malých vnitřních tělískách.

Inerciální technologie je založena na prvních dvou Newtonových zákonech[14]. První zákon říká, že pohyb těla robota je jednotný a lineární jestliže na něj nepůsobí jiná vnější síla. Druhý zákon definuje, že tato síla vynaložená na hmotu produkuje úměrné zrychlení. Tyto vztahy reprezentují princip měření, ve kterých mohou být vyvinuty snímače schopné měřit pohyb těles, v našem případě robota. Jestliže tedy známe velikost a směr síly působící na tělo robota a jeho hmotnost, můžeme zjistit jeho zrychlení, a tím i jeho rychlost a pozici pomocí první a druhé matematické integrace podle času. [28]

Většina IMU má v sobě tříosé akcelerometry a tříosé gyroskopy, tedy jsou schopné měření ve třech dimenzích (tj. v prostoru).

IMU je tedy schopné zjistit v okamžitém čase jaké síly na robota působí, a tedy zjistit aktuální zrychlení, rotaci a změnu polohy.

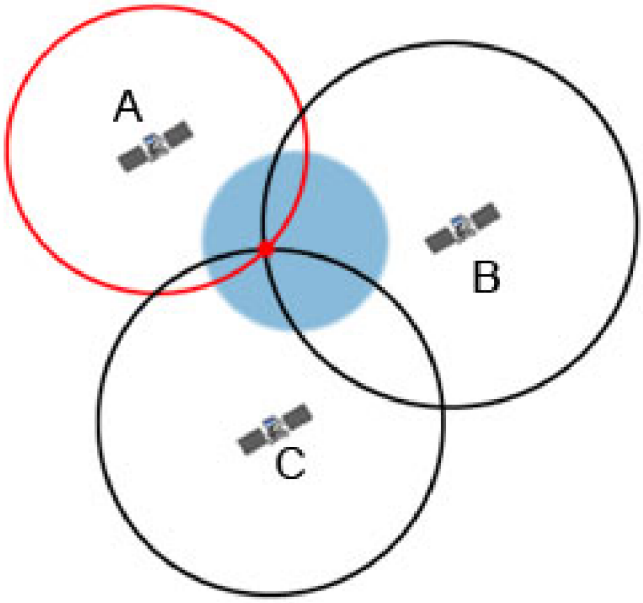
3.1.3 GPS

GPS je síť 31 satelitů na orbitě Země ve výšce 20 000 km nad mořem. Na jakémkoliv místě na zeměkouli jsou viditelné alespoň čtyři z těchto satelitů. Každý z těchto satelitů má v sobě

[3] Vztažná soustava, v níž platí 1. Newtonův pohybový zákon setrvačnosti

atomické hodiny pro uchovávání přesného aktuálního času. Tyto satelity vysílají informace o jejich lokaci a aktuální časové značce a cestují přes signály na frekvencích 1575.42 MHz a 1227.60 MHz. Pokud příjmač zachytí signál, na základě rozdílu aktuálního času, časové značky v signálu a znalosti rychlosti šíření signálu je schopen vypočítat pomocí rovnice dráhy 3.1, jak daleko je každý z těchto viditelných satelitů. Pokud zjistí vzdálenost alespoň 3 satelitů, dokáže za pomocí trilaterace[4] získat naši polohu.

Obrázek 3.2 zobrazuje trilateraci v praxi. Předpokládejme, že satelit A vyšle signál a příjmač ho zachytí. To ho dokáže ujistit, že se nachází někde ve vzdálenosti od vysílače A, tedy po jeho celé opsané kružnici. Následně však zachytí signál od satelitu B, z něhož dokáže zjistit svoji vzdálenost vůči němu. Po této informaci příjmač ví, že se nachází někde na průnicích opsaných kružnic satelitů A a B. K určení, který z těchto dvou průniků to je, potřebuje další zprávu od jiného satelitu. Satelit C vyšle zprávu jako třetí. Poté co ji příjmač zachytí, je schopen zjistit, na kterém z těchto dvou průniků se nachází. Čím více satelitů takovou zprávu odešle, tím přesněji je schopen příjmač určit, kde se nachází.



Obrázek 3.2: Princip GPS trilaterace.[9]

3.1.4 Kinematický stav robota

Kinematický stav robota je další možností jak je možné lokalizovat robota. Samotný robot má většinou data o svých součástech (motoru, natočení podvozku, ...). Jestliže známe rychlost a trajektorii robota z jeho stavu, je možné odhadnout, kudy a kam pojede. Velkou nevýhodou tohoto systému je, že pokud nejde o robota, který se pohybuje na spolehlivém povrchu, tedy povrchu, který neklouže, neprokluzuje a nedrolí se, ale naopak jede po blátě, či jiných površích, je nejistota tohoto systému obrovská a většinou se nevyplatí ji zařadit do celkového systému.

[4]Trilaterace je proces zjišťování absolutní nebo relativní pozice bodů měřením vzdáleností pomocí geometrie kruhů, trojúhelníků, nebo čtverců.

3.1.5 Odměřovací kolečko

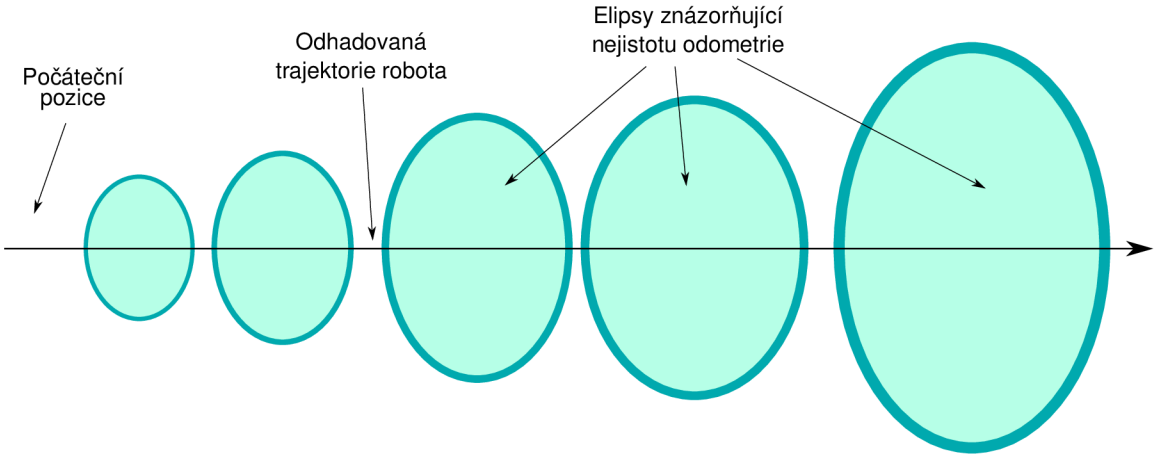
Princip měření odměřovacího kolečka, nebo také kolometru, je velmi podobný tachometru. Kolečko je připevněno k pevnému bodu a při jízdě se otáčí. Na základě znalosti rozměrů kolečka a informace o počtu otáček jsme schopni jednoduše vypočítat ujetou vzdálenost. Aktuální poloha kolečka je reprezentována jako zlomek otáčky. Pokud se kolo otočí o celou otáčku (tedy o 360 stupňů), ujetá vzdálenost se rovná obvodu kola.

Některá kolečka mají ještě přídavnou rotaci celého kolečka. Podobně jako mají nákupní vozíky otočná kolečka, tak i kolometry mohou mít podobnou konstrukci. Tím, že jsme schopni takto otáčet kolečkem, jsme schopni měřit ujetou vzdálenost ve všech směrech, přičemž se kolometr vždy přizpůsobí směru jízdy.

3.2 Odometrie

Slovo odometrie je složeno ze dvou řeckých slov *hodos* (cesta, cestovat) a *metron* (měřit), z čehož se dá odhadnout, k čemu odometrie slouží. Víme, že ze senzorů si náš robot může zjistit informace o sobě a svém prostředí. Tyto informace se dají využít k získání představy o pohybu. Proces, který popisuje transformaci dat poskytnutých senzory na změnu pozice a orientace robota se nazývá odometrie. Zjednodušeně řečeno, je to metoda lokalizace, založená na odhadu změny pozice a orientace robota prostřednictvím informací o otáčení jeho hnacích nebo běžných kol pomocí rotačních enkodérů. [19]

Odometrie vznikla z myšlenky, že naměřenou rotaci kola je možné převést na posun robota. Bohužel, kvůli nedokonalostem zmíněným v předchozí podkapitole 3.1.4, tedy podklouznutí, či protočení kola, se da posun velmi těžko odhadnout.



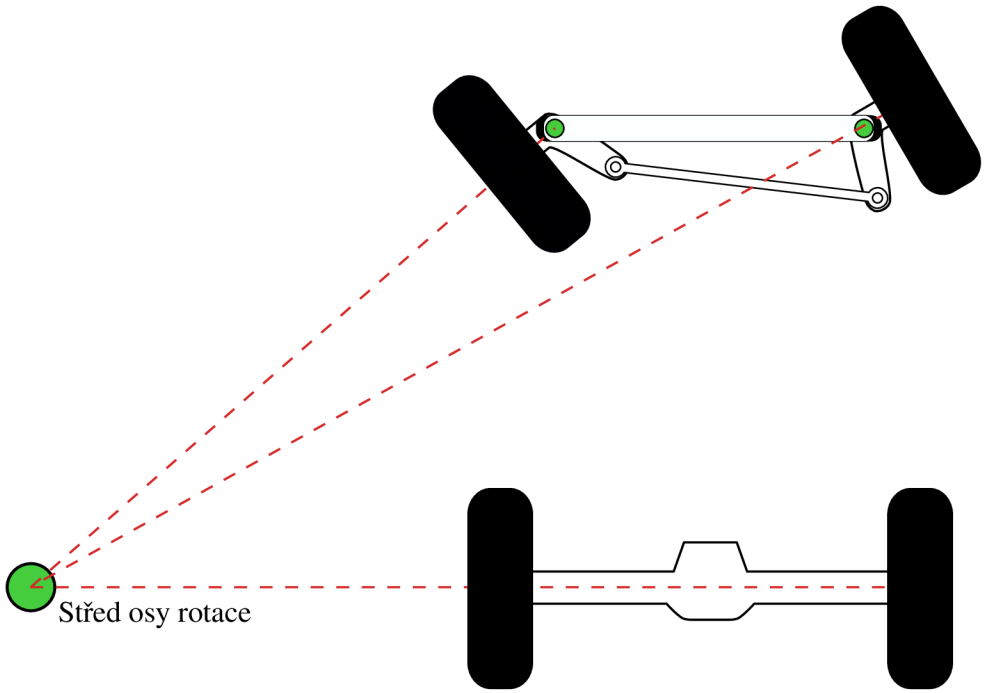
Obrázek 3.3: Rostoucí elipsy znázorňují rostoucí nejistotu pozice získanou odometrií.

Základem odometrie je znalost geometrického modelu robota. Tyto modely se liší zejména tím, jakých druhů pohybu jsou roboti schopni. V této práci se budeme zabývat Ackermanovým řízením.

3.2.1 Ackermanovo řízení

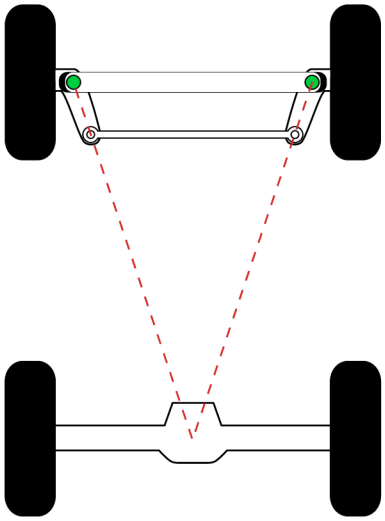
Jako zdroj informací k Ackermanovu řízení využiji bakalářskou práci inženýra Jakuba Hůlky [8], která říká, že Ackermanův podvozek se používá hlavně v automobilovém průmyslu, jeli-

kož je vhodný pro vysoké rychlosti i na středně těžkém terénu. Výhodou je též malá spotřeba enrgie. Nevýhodou jsou holonomní omezení, díky kterým není možné se s podvzokem otočit na místě. Opustit tak stísněné prostory může být velkým problémem.



Obrázek 3.4: Zobrazení Ackermanova podvozku při otáčení se kolem centra rotace. [33]

Obrázek 3.4 zobrazuje, jak se Ackermanův podvozek chová při otáčení. Střed otáčení je tedy možné určit jako průnik kolmých os na trajektorii kol. Pokud má být střed otáčení dobře definován, je nutné, aby vnitřní kolo zatáčelo více než vnější.

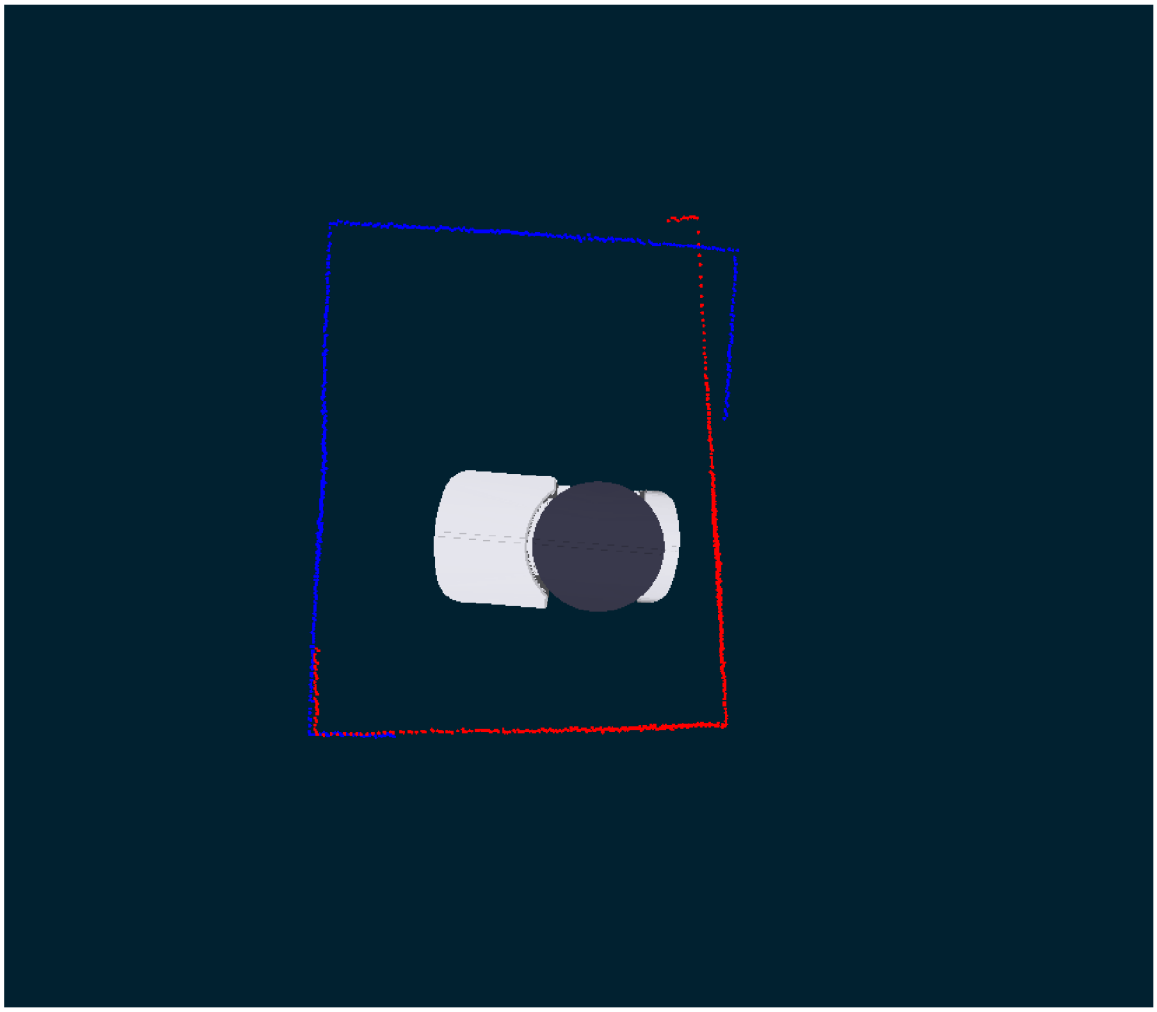


Obrázek 3.5: Jednoduchý design Ackermanovy geometrie. [33]

3.3 Kalibrace senzorů

Kalibrace senzorů je metoda zvyšování kvality senzorů odstraňováním strukturálních chyb v jejich výstupech. Poskytuje prostředky pro zvýšení výkonu zlepšením celkové přesnosti podložních senzorů. Strukturální chyby jsou rozdíly mezi očekávaným výstupem senzoru a jeho skutečným naměřeným výstupem. Jakákoliv z těchto chyb, pokud je opakovaná, může být spočítána v průběhu kalibrace. Díky tomu je možné digitálně vykompenzovat tyto chyby.

Při kalibračním procesu se senzor dává do situace, kdy je znám výstup senzoru. Díky tomu jsme schopni zjistit a odstranit chybu, která vznikla při měření.



Obrázek 3.6: Chyba kalibrace více LiDARů v jednom systému. [17]

Kapitola 4

Metody pro lokalizaci a agregaci dat

V této kapitole se budeme zabývat agregací dat získaných z těchto systémů, odstraňování nepřesností a šumu z naměřeného signálu a s tím spojené zpřesňování měření a dále pak samotnou globální lokalizací.

4.1 Bayesův filtr

Jak je popsáno v knize *Probabilistic robotics*[18], jde o rekurzivní algoritmus, který se skládá ze dvou částí: předpověď a inovace. Pokud jsou proměnné rozděleny Gaussovou (normální) distribucí a přechody stavů jsou lineární, Bayesův filtr je totožý s Kalmanovým filtrem, o kterém budeme mluvit v kapitole 4.2.

Bayesův filtr, nebo také Rekurzivní Bayesovský odhad, využívá Bayesovu větu, která říká, že pokud existují dva náhodné jevy A a B s pravděpodobnostmi $P(A)$ a $P(B)$, přičemž $P(B) > 0$, potom platí

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}, \tag{4.1}$$

kde $P(A|B)$ je podmíněná pravděpodobnost jevu A za předpokladu, že nastal jev B, a naopak $P(B|A)$ je pravděpodobnost jevu B podmíněná výskytem jevu A.

Předpokládejme stav x, který reprezentuje neobjevený Markovův proces,[1] a z je objevený stav skrytého Markovského modelu. Na obrázku 4.1 je znázorněna Bayesova síť takového modelu.

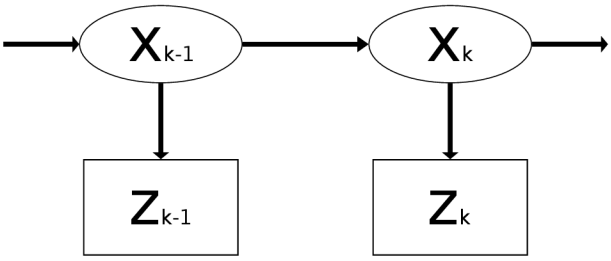
Pravděpodobnost aktuálního skutečného stavu je dána pouze bezprostředně předcházejícím stavem a není závislá na ostatních předešlých stavech, což se dá vyjádřit následující rovnicí:

$$p(x_k|x_{k-1}, x_{k-2}, ..., x_0) = p(x_k|x_{k-1}) \tag{4.2}$$

Obdobně měření v čase k je závislé pouze na současném stavu a nikoliv na ostatních předešlých stavech, tedy platí:

$$p(z_k|x_{k-1}, x_{k-2}, ..., x_0) = p(z_k|x_k) \tag{4.3}$$

[1] Stochastický proces, který splňuje Markovovu vlastnost (viz. [16].), tedy je schopen předpovědět následující stav pouze na základě aktuálního stavu stejně tak dobře, jak by ho dokázal předpovědět za předpokladu znalosti celé historie procesu



Obrázek 4.1: Bayesova síť skrytého Markovského modelu. [34]

Algoritmus

Bakalář Ondrej Čakloš ve své bakalářské práci [37] algoritmus vystihl takto: Nejobecnější algoritmus pro výpočet funkce *belief* udává právě algorimus Bayesova filtru. Tabulka 4.1 znázorňuje pseudo-algoritmus Bayesova filtru. Bayesův filtr je rekurzivní, což znamená, že funkce *belief* v čase t je vypočítaná ze stejné funkce v čase $t-1$. Jeho vstup je *belief* v čase $t-1$, současně s nejnovějším řídícím vektorem u_t a nejnovějším měřením z_t. Jeho výstup je *belief* v čase t.

Algoritmus Bayesova filtru obsahuje dva kroky:

- *předpověď* - Aktualizace $\overline{bel}(x_t)$ (řídící vektor u_t a předchozí stav x_{t-1} indikuje přechod z x_{t-1} do x_t)
- *aktualizace* - Výpočet nového $bel(x_t)$

Aby mohl být algoritmus rekurzivní, je třeba znát počáteční $bel(x_0)$. V praxi se používají metody plné vědomosti, tedy případy, kdy známe počáteční stav a nebo metody úplné ignorace, kdy počateční stav je rovnoměrným rozložením na x_0.

1 Algoritmus $Bayes\_filter(bel(x_{t-1}), u_t, z_t)$:
2 **for** t = 1 to n **do**
3 $\quad \overline{bel}(x_t) = \int p(x_t|u_t, x_{t-1}) bel(x_{t-1}) dx$
4 $\quad bel(x_t) = \eta p(z_t|x_t) \overline{bel}(x_t)$
5 **endfor**
6 return $bel(x_t)$

Tabulka 4.1: Krok algoritmu pro Bayesův filtr

4.2 Kalmanův filtr

Kalmanův filtr [3][18] je matematický proces, který využívá soubor několika rovnic a po sobě jdoucích vstupů dat, pomocí kterých rychle odhaduje pravou hodnotu, pozici, rychlost a další parametry objektu, který měříme. Z dat zatížených nepřestnostmi a šumem dokáže tyto nepřesnosti do jisté míry odstraňovat. Odstraňuje je za pomoci hodnot naměřených v aktuálním měření, modelu systému a vektoru hodnot obsahujícím předchozí stav systému. Jde pravděpodobně o nejlepší známou implementaci Bayesového filtru z kapitoly 4.1. Jde o techniku pro filtrování a předpověď v lineárních Gaussových systémech.

V čase t Kalmanův filtr reprezentuje *belief* střední hodnotou μ_t a kovariancí Σ_t. Funkce pro pravděpodobnost přechodu stavu $p(x_t|u_t, x_{t-1})$ musí být lineární funkce s přidaným Gaussovým šumem. Toto je reprezentováno následující rovnicí:

$$x_t = A_t x_{t-1} + B_t u_t + \epsilon_t \tag{4.4}$$

Vektory x_t a x_{t-1} reprezentují aktuální stav systému a vektor u_t je kontrolní vetor. Oba tyto vektory jsou vertikální. Jejich forma vypadá takto:

$$x_t = \begin{pmatrix} x_{1,t} \\ x_{2,t} \\ \vdots \\ x_{n,t} \end{pmatrix} \quad \text{a} \quad u_t = \begin{pmatrix} u_{1,t} \\ u_{2,t} \\ \vdots \\ u_{m,t} \end{pmatrix}$$

A_t a B_t z rovnice 4.4 jsou matice. A_t je čtvercová matice o rozměrech $n \times n$, kde n je dimenze stavového vektoru x_t. Rozměr matice B_t je $n \times m$, kde m je dimenze kontrolního vektoru u_t. Při násobení stavového a kontrolního vektoru maticemi A_t a B_t, funkce přechodu stavu se stává lineární.

Náhodná proměnná ϵ_t je Gaussův náhodný vektor a modeluje nejistotu při přechodu stavu. Má stejný rozměr jako stavový vektor x_t. Jeho průměr je nula a jeho kovarianci označme jako R_t.

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\} \tag{4.5}$$

Rovnice 4.4 definuje pravděpodobnost přechodu stavu $p(x_t|u_t, x_{t-1})$. Tato pravděpodobnost je získána dosazením této rovnice do vícerozměrného normálního rozdělení 4.5. Střední hodnota odhadovaného stavu je dána $A_t x_{t-1} + B_t u_t$ a kovariancí R_t:

$$p(x_t|u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x - A_t x_{t-1} - B_t u_t)^T R_t^{-1}(x - A_t x_{t-1} - B_t u_t)\} \tag{4.6}$$

Pravděpodobnost měření $p(z_t|x_t)$ musí také být lineární s přidaným Gaussovým šumem:

$$z_t = C_t x_t + \delta_t \tag{4.7}$$

C_t je matice o rozměrech $k \times n$, kde k je rozměr vektoru měření z_t. Vektor δ_t popisuje šum měření. Distribuce δ_t je vícerozměrná Gaussova s nulovou střední hodnotou a kovarianční maticí Q_t. Pravděpodobnost měření je dána následující vícerozměrnou normální distribucí:

$$p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1}(z_t - C_t x_t)\} \tag{4.8}$$

Počáteční funkce $bel(x_0)$ musí být normální distribucí. Střední hodnotu této funkce budeme označovat μ_0 a kovarianci Σ_0:

$$bel(x_0) = p(x_0) = \det(2\pi\Sigma_0)^{-\frac{1}{2}} \exp\{-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1}(x_0 - \mu_0)\} \tag{4.9}$$

Tyto tři předpoklady jsou dostatečné k tomu, aby zajistili, že další $bel(x_t)$ bude vždy Gaussovou funkcí pro jakýkoliv bod v čase t.

Algoritmus

1	Algoritmus $Kalman\_filter(u_{t-1}, \Sigma_{t-1}, u_t, z_t)$:
2	$\overline{\mu}_t = A_t \mu_{t-1} + B_t u_t$
3	$\overline{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$
4	
5	$K_t = \overline{\Sigma}_t C_t^T (C_t \overline{\Sigma}_t C_t^T + Q_t)^{-1}$
6	$\mu_t = \overline{\mu_t} + K_t(z_t - C_t \overline{\mu}_t)$
7	$\Sigma_t = (I - K_t C_t)\overline{\Sigma}_t$
8	return μ_t, Σ_t

Tabulka 4.2: Algoritmus Kalmanova filtru pro lineární Gaussové přechody stavů a měření.

Kalmanův filtr reprezentuje funkci víry $bel(x_t)$ v čase t střední hodnotou μ_t a kovariancí Σ_t. Vstupem Kalmanova filtru je $belief$ v čase $t-1$ reprezentován vektory μ_{t-1} a Σ_{t-1}. Pro aktualizaci těchto parametrů, Kalmanův filtr vyžaduje kontrolní vektor u_t a měření z_t. Výstup je $belief$ v čase t, tedy vektory μ_t a Σ_t.

Na řádcích 2 a 3 algoritmu 4.2 je předpověď $\overline{bel}(x_t)$ reprezentována výpočtem $\overline{\mu}$ a $\overline{\Sigma}$. Pro výpočet celkového $\overline{bel}(x_t)$ je třeba začlenit do výpočtu kontrolní vektor u_t. Střední hodnota se aktualizuje pomocí deterministické verze funkce přechodu stavu 4.4, přičemž střední hodnota μ_{t-1} se nahrazuje stavem x_{t-1}. Aktualizace kovariance uvažuje skutečnost, že stavy závisí na předchozích stavech prostřednictvím lineární matice A_t. Tato matice je dvakrát vynásobená do kovariance, protože kovariance je čtvercovou maticí.

Abychom získali $bel(x_t)$, je nutné transformovat $\overline{bel}(x_t)$. To je znázorněno na řádcích čtyři až šest, začleněním měření z_t. Proměnná K_t vypočítaná na řádku čtyři se nazývá *Kalmanův přírůstek*, také Kalmanův zisk.

Kalmanův zisk je relativní váha pro jednotlivá měření a aktuální odhad stavu. S vysokou hodnotou Kalmanova zisku filtr přikládá větší váhu nejnovějším měřením, díky čemu je následně citlivější na změny. Naopak nízký zisk více věří předpovědi modelu.

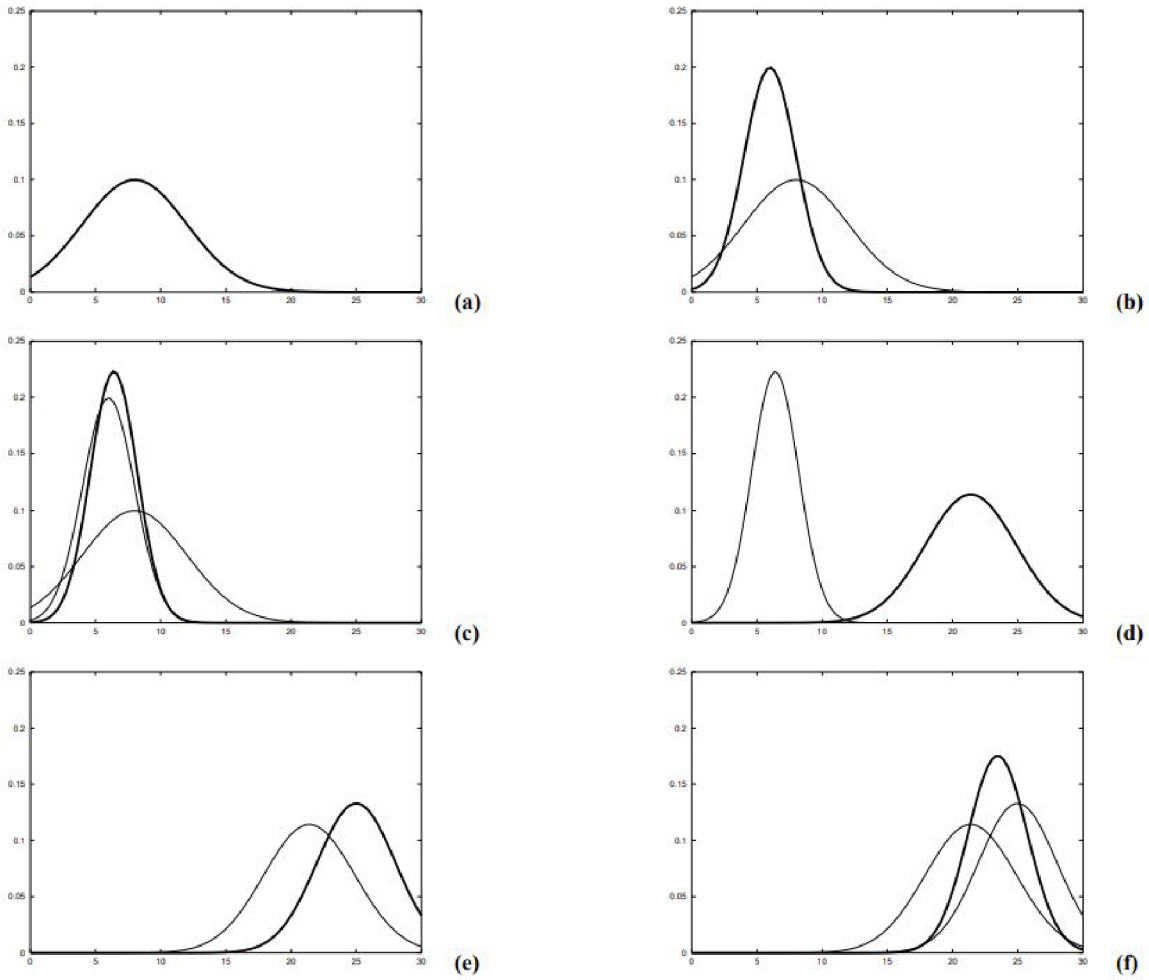
Stejně jako Bayesův filtr i Kalmanův je rekurzivní. To znamená, že odhadovaný stav z předchozího měření a aktuálního měření je nutné k tomu, abychom zjistili odhad pro aktuální stav. Kalmanův filtr může být napsán jako jediná rovnice, ale nejčastěji je koncipován pod dvěma různými frázemi:

Předpověď - Používá předchozí odhad stavu na výpočet předpovědi aktuálního stavu. I když jde o předpověď aktuálního stavu, nepoužívá při výpočtu informace z aktuálního měření.

Aktualizace - Kombinuje aktuální předpověď s aktuálním měřením a vylepší tak odhad aktuálního stavu.

Obrázek 4.2 ilustruje algoritmus Kalmanova filtru pro jednoduchý jednodimenzionální lokalizační scénář. Předpokládejme, že se robot pohybuje spolu s horizontální osou. Část *(a)* zobrazuje počáteční normální distribuci $belief$ - $bel(x_0)$. Robot se ptá svých senzorů na svoji pozici a senzory vrátí měření, které je centrováno na vrcholu tučné Gaussovy křivky

(znázorněno v části *(b)*). Vrchol je předpovězen senzory a jeho šířka koresponduje s nejistotou měření. Zkombinováním předpovědi s měřením (4.2 na řádku čtyři až šest) vznikne Gaussova křivka v *(c)* (tučná). Střední hodnota aktuální funkce *belief* leží mezi dvěmi původními středními hodnotami. Její rozptyl nejistoty je menší, než rozptyl obou původních Gaussových křivek.



Obrázek 4.2: Ilustrace Kalmanova filtru. (a) počáteční *belief*, (b) měření (tučné) s přiřazenou nejistotou, (c) *belief* po integrování měření do funkce *belief* za použití Kalmanova filtru, (d) *belief* po pohybu vpravo (nový šum), (e) nové měření s přiřazenou nejistotou, (f) výsledný *belief*. [18]

Předpokládejme, že se robot pohne vpravo. Její nejistota vzroste kvůli faktu, že přechod do dalšího stavu je stochastický. Řádky dva a tři algoritmu 4.2 nám poskytnou Gaussovu křivku zobrazenou tučně v části *(d)*. Tato křivka je posunuta o stejnou vzdálenost, o kterou se robot pohnul a její rozptyl se zvětšil. Robot nyní získá další měření, to je znázorněno tučnou křivkou v části *(e)*. Po výpočtu získáme křivku zobrazenou v části *(f)*.

Jak tento příklad ilustruje, Kalmanovy filtry střídají krok aktualizace a měření, ve kterém jsou data senzoru integrována do aktuální funkce *belief* s krokem předpovědi, který modifikuje *belief* v souladu s měřením. Krok aktualizace snižuje a krok předpovědi zvyšuje nejistotu ve funkci *belief*.

4.2.1 Rozšířený Kalmanův filtr

Předpoklady lineárních stavových přechodu a lineární měření s přidaným Gaussovým šumem jsou v praxi zřídka splněný.[11][18] Například robot, který se pohybuje s konstantním poziční a rotační rychlostí se typicky pohybuje po kruhové trajektorii, kterou nelze popsat lineárními přechody dalšího stavu. Toto pozorování činí obyčejné Kalmanovy filtry nepoužitelné na všechny netriviální problémy robotiky.

Rozšířený Kalmanův filtr (angl. Extended Kalman filter) překonává předpoklad linearity. Předpokládá, že pravděpodobnost příštího stavu a pravděpodobnosti měření se řídí nelineárními funkcemi g a h.

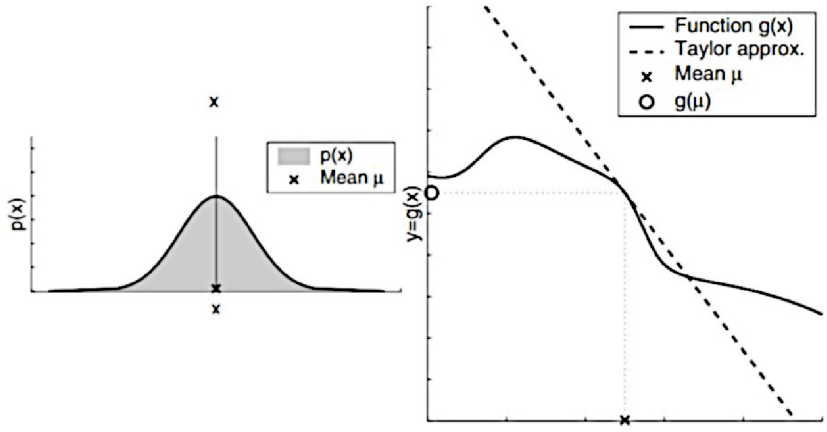
$$x_t = g(u_t, x_{t-1}) + \epsilon_t \tag{4.10}$$

$$z_t = h(x_t) + \delta_t \tag{4.11}$$

Tento model striktně generalizuje lineární Gaussův model základního Kalmanova filtru. Funkce g v rovnici 4.4 nahrazuje matice A_t a B_t. V rovnici 4.7 je matice C_t nahrazena funkcí h. Díky libovolným funkcím g a h však přestává $belief$ být Gaussovou křivkou. Provedení aktualizace funkce $belief$ je obvykle nemožné pro nelineární funkce g a h.

Rozšířený Kalmanův filtr počítá aproximaci skutečné funkce $belief$. Reprezentuje tuto aproximaci Gaussovou křivkou. Funkce $bel(x_t)$ v čase t je reprezentována střední hodnotou μ_t a kovariancí Σ_t. To znamená, že rozšířený Kalmanův filtr dědí ze základního Kalmanova filtru základní reprezentaci funkce $belief$, ale liší se v tom, že tato funkce je pouze aproximací, což v případě Kalmanova fitru nebyla.

Klíčovou myšlenkou rozšířeného Kalmanova filtru je proces zvaný *linearizace*. Předpokládejme, že nám je dána nelineární funkce pro odhad dalšího stavu g. Nelinearity v této funkci zkreslují $belief$, díky čemu ničí její tvar Gaussové křivky. Linearizace aproximuje funkci g lineární funkcí, která vytvoří tečnu k původní funkci. Bod, ve kterém se funkce dotknou pak tvoří střed Gaussové křivky. Promítáním Gaussové křivky skrz tuto lineární aproximaci zaručujeme, že náš algoritmus může v budoucnu s touto křivkou počítat. Pokud můžeme takto linearizovat funkci g, pak funkce pracující s $belief$ jsou stejné, jako u Kalmanova filtru. To stejné platí i pro nelineární funkci h, která se aproximuje stejně a zachovává tak povahu Gaussové křivky. Tento proces je ilustrován v obrázku 4.3.



Obrázek 4.3: Linearizace aplikovaná rozšířeným Kalmanovým filtrem.[18]

Existuje spousta technik pro linearizaci nelineárních funkcí. Filtr využívá metodu zvanou *Taylorův rozvoj*, který vytvoří lineární aproximaci funkci g pomocí hodnoty a náklonu funkce g'. Náklon je dán parciální derivací:

$$g'(u_t, x_{t-1}) = \frac{\nabla g(u, x_{t-1})}{\nabla x_{t-1}} \tag{4.12}$$

Je zřejmé, že hodnota a náklon funkce g' závisí na argumentech funkce g. Logická volba pro výběr argumentů je zvolit stav, který je považovaný za nejpravděpodobnější v době linearizace. Pro Gaussove křivky platí, že nejpravděpodobnější stav je střed předpovědi μ_{t-1}.

Nechť $G_t := g'(u_t, \mu_{t-1})$, pak pravděpodobnost dalšího stavu je pak aproximována jako:

$$\begin{aligned} p(x_t|u_t, x_{t-1}) \approx \det(2\pi R_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T \\ R_t^{-1}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]\} \end{aligned} \tag{4.13}$$

Matice G_t je o velikosti $n \times n$, kde n je rovno dimenzi stavu. Tato matice se často nazývá *Jakobián*. Hodnota Jakobiánu se odvíjí od u_t a μ_{t-1}, protože se liší pro různé body v čase.

Rozšířený Kalmanův filtr využívá stejnou linearizaci pro funkci měření h. Taylorův rozvoj je vytvořen z $\overline{\mu}_t$, tedy stavem, který robot používal za nejpravděpodobnější v době linearizace h:

$$\begin{aligned} h(x_t) &\approx h(\overline{\mu}_t) + h'(\overline{\mu}_t)(x_t - \overline{\mu}_t) \\ &\text{kde } h'(\overline{\mu}_t) =: H_t, \text{ tedy} \\ &= h(\overline{\mu}_t) + H_t(x_t - \overline{\mu}_t) \end{aligned} \tag{4.14}$$

Jako aproximaci pravděpodobnosti dalšího stavu máme:

$$\begin{aligned} p(z_t|x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\{-\frac{1}{2}[z_t - h(\overline{\mu}_t - H_t(x_t - \overline{\mu_t}]^T \\ Q_t^{-1}[z_t - h(\overline{\mu}_t - H_t(x_t - h(\mu_t)]\} \end{aligned} \tag{4.15}$$

Algoritmus

1 Algoritmus $EKF(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$:
2 $\overline{\mu}_t = g(u_t, \mu_{t-1})$
3 $\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$
4
5 $K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$
6 $\mu_t = \overline{\mu}_t + K_t(z_t - h(\overline{\mu}_t))$
7 $\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$
8 return u_t, Σ_t

Tabulka 4.3: Pseudokód rozšířeného Kalmanova filtru

4.3 Částicový filtr

Myšlenka filtru částic [18] je aproximovat domněnku $bel(x_t)$ jako soubor N částic. Soubor částic se dá popsat jako:

$$X_t = \{p_t^{[1]}, p_t^{[2]}, p_t^{[3]}, \ldots, p_t^{[N]}\}. \tag{4.16}$$

Každá z těchto částic je pevný odhad aktuálního stavového vektoru, tedy každá z těchto částic reprezentuje hypotézu, že daná částice je vhodná. V každém kroku jsou částice náhodně navzorkovány, což znamená, že se náhodně rozloží po celém poli. Je zřejmé, že pravděpodobnost, že částice je zahrnuta v X_t, tedy $P(p_t^{[i]} \in X_t)$ je úměrná pravděpodobnosti, že se tato částice nachází na vhodné reprezentaci stavu. Pro aktualizaci stavu je možné použít algoritmus podobný genetickým algoritmům - evoluce nebo přirozenému výběru, které přepokládají, že silné hypotézy (tedy hypotézy s větší pravděpodobností na úspěch) mají větší šanci žít a tvořit další částice, kdežto slabé hypotézy mají větší šanci zahynout. Díky tomuto faktu je možné tvrdit, že většina částic bude centrována okolo silnějších hypotéz.

Algoritmus

Nejprve je nutné vypočítat reprezentaci částic $\overline{bel}(x_{t+1}$ z X_t. Označme ho jako $\overline{X}_{t+1}$. Poté pro každou částici v X_t navzorkujeme novou částici $\overline{p}_{t+1}^{[i]}$ z distribuce $P(x_{t+1}|x_t = p_t^{[i]})$. Tato distribuce je reprezentována přechodovým modelem. Všechny tyto nové částice vložíme do $\overline{X}_{t+1}$. Dále je nutné vypočítat $bel(x_{t+1})$ z $\overline{X}_{t+1}$. Klíčové je zde přiřadit tzv. váhu podle důležitosti pro každou částici v $\overline{X}_{x+1}$. Označme tuto váhu jako $\omega[i]$. Tato váha reprezentuje, jak moc kompatibilní je částice $\overline{p}_{t+1}^{[i]}$ s měřením e_{t+1}. Tato pravděpodobnost je získána z modelu senzorů. X_{t+1} je pak zkonstruována tak, že vybereme náhodných N částic z $\overline{p}_{t+1}^{[i]}$ s pravděpodobností úměrnou jejich váze. Je důležité říct, že každá částice může být vybrána vícekrát. Tento proces se jmenuje převzorkování.

```
1  Algoritmus Particle_filter(X_{t-1}, u_t, z_t) :
2  X̄_t = X_t = ∅
3  for m = 1 to M do
4      Vzorek x_t^[m] ~ p(x_t|u_t, x_{t-1}^[m]]
5      ω_t^[m] = p(_z|x_t^[m])
6      X̄_t = X̄_t+ < x_t^[m], ω_t^[m] >
7  endfor
8
9  for m = 1 to M do
10     Vylosuj i s pravděpodobností α  w_t^[i]
11     Přidej x_t^[i] do X_t
12 endfor
13 returnX_t
```

Tabulka 4.4: Pseudokód algoritmu filtru částic

4.4 Lokalizační algoritmus Monte-Carlo

Monte Carlo [18] je algoritmus využívající filtr částic je použitelný jak pro lokální, tak globální lokalizaci. Může aproximovat téměř všechny distribuce praktického významu. Narozdíl od rozšířeného Kalmanova filtru není vázán na parametrické podskupiny distribuce. Zvýšení vygenerovaných částic zvýší přesnost aproximace. Počet částic M je parametr, který povoluje kompromis mezi potřebným výpočetním výkonem a výpočetní rychlostí.

V lokalizaci Monte Carlo je odhad aktuálního stavu funkcí hustoty pravděpodobnosti distribuován přes celý stavový prostor. Tento odhad je v čase t reprezentován souborem částic M.

$$X_t = \{x_t^{[1]}, x_t^{[2]}, x_t^{[3]}, \ldots, x_t^{[M]}\} \tag{4.17}$$

Algoritmus

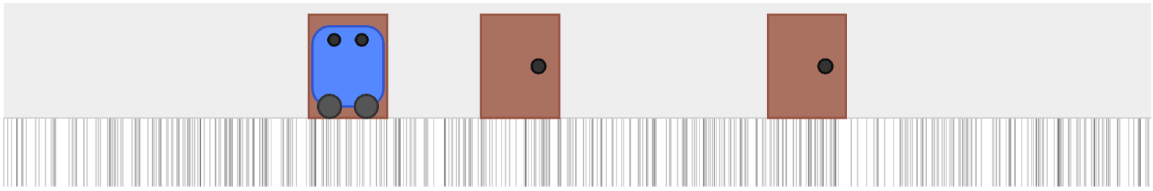
1 Algoritmus $MCL(X_{t-1}, u_t, z_t)$:
2 $\overline{X}_{t+1} = \emptyset$
3 $\Omega = 0$
4 **for** i = 1 to n **do**
5 $\quad \overline{p}_{t+1}^{[i]}$ = náhodný stavový prostorový vzorek podle distribuce $P(x_{t+1}|x_t = p_t^{[i]})$
6 $\quad \overline{X}_{t+1} = \overline{X}_{t+1} \cup \{\overline{p}_{t+1}^{[i]}\}$
7 $\quad \omega_{t+1}^{[i]} = P(e_{t+1}|x_t = \overline{p}_{t+1}^{[i]})$
8 $\quad \Omega = \Omega + \omega_{t+1}^{[i]}$
9 **endfor**
10
11 $X_{t+1} = \emptyset$
12 **for** i = 1 to n **do**
13 $\quad \overline{p}_{t+1}^{[i]}$ = náhodná částice z $\overline{X}_{t+1}$ s pravděpodobnostmi úměrnými jejich hmotnosti
14 $\quad X_{t+1} = X_{t+1} \cup \{p_{t+1}^{[i]}\}$
15 **endfor**
16 **return** X_{t+1}

Tabulka 4.5: Pseudokód algoritmu Monte Carlo localization

Každá částice obsahuje potencionální nový stav. Oblasti s vyšším počtem částic pak znamenají větší pravděpodobnost, že v této oblasti je vhodný nový stav.

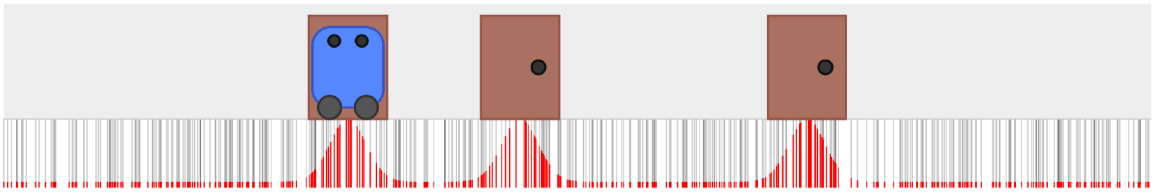
Jeho počáteční globální nejistota je získána tak, že se částice náhodně vygenerují na náhodných místech po celém prostoru.

Podobně jako v kapitole 2.2 částice reprezentují funkci $belief$. Pro příklad si opět uvedeme 1D robota, který se bude pohybovat chodbou s dveřmi. Na obrázku 4.4 je znázorněn robot, který provedl první měření a zjistil, že se nachází vedle dveří. Algoritmus inicializuje částice rovnoměrným rozložením. V této chvíli si robot myslí, že může nacházet kdekoliv, i když je fyzicky vedle prvních dveří.



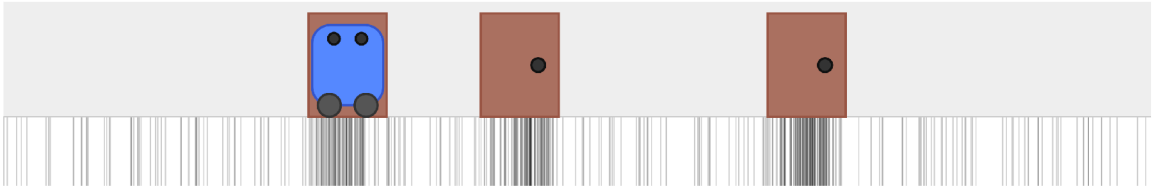
Obrázek 4.4: Inicializace částic v prostoru robota. [18]

Na dalším obrázku 4.5 robot provede aktualizaci měření. Detekuje, že se nachází vedle dveří. Všem částicím přiřadí váhu podle jejich pozice. Všechny částice, které odpovídají měření, získají větší váhu, než ostatní.



Obrázek 4.5: První měření robota. [18]

Robot dále vygeneruje nové částice. Částice jsou přiřazeny dle váhy předchozích částic. Nyní robot věří, že se nachází na jedné z tří pozic - vedle dveří.



Obrázek 4.6: Převzorkování částic. [18]

Pokud se robot pohne, všechny částice se pohnou s ním a zároveň je na ně aplikován šum. Robot takto pokračuje až do doby, než nenalezne další dveře. Po jejich detekci opět provede stejný algoritmus, tedy přiřadí váhu částicím a převzorkuje je.

Kapitola 5

Robotický operační systém

Robotický operační systém (dále jen ROS) [21][23] je flexibilní open-source[1] framework[2] pro psaní softwaru pro roboty. Je to kolekce nástrojů, knihoven a konvencí, které se zaměřují na zjednodušení úkolu vytváření komplexního a robustního chování robota skrz širokou škálu robotických platforem. Poskytuje implementace často používaných funkcí, ovládání nízkoúrovňových zařízení, hardwarovou abstrakci. Mimo jiné také umožňuje správu balíčků (angl. package), která umožňuje instalovat jen potřebné součásti ROSu. ROS je podobný ostatním frameworkům pracující s roboty, jako například Player, YARP, Orcos, CARMEN, Orca a další.

ROS je nyní plně přeložen, vyvíjen a testován na Unixových distribucích. Primárně jsou to Ubuntu a Mac OS X, avšak komunita přispívá do vývoje pro Fedoru, Gentoo, Arch Linux a jiné Linuxové platformy. Je možné jej přeložit i pro Microsoft Windows, avšak na této platformě není podporován.[23]

Systémy ROSu by se daly popsat jako síť mezi sebou komunikujících peer-to-peer[4] volně spřažených procesů, které používají ROS infrastrukturu komunikace. ROS implementuje různé typy komunikace včetně:

- Synchronní RPC komunikace přes služby
- Asynchronní tok dat skrze téma
- Ukládání dat na serveru parametrů

Tyto typy jsou více rozebrány v podkapitole 5.2.

ROS není framework pracující v reálném čase, ale je možné jej integrovat s aplikacemi, které v reálném čase pracují.

5.1 Princip

Distribuované struktury procesů, tzv. uzly (angl. node), mezi sebou komunikují pomocí zpráv (angl. message) skrze témata (angl. topic), nebo služby (angl. services). Tato struktura a komunikace umožňuje individuální tvoření částí celkového systému [2] a následné volné spojení uzlů při jeho spouštění.

Hlavním cílem ROSu je tedy sdílení a spolupráce. Kromě tohoto je však implementován tak, aby plnil cíle, jako jsou:

[1] Open-source je označení programů, jejichž zdrojový kód je poskytnut dalším vývojářům.
[2] Aplikační rámec, neboli framework je platforma pro vývoj softwarových aplikací.

- Jednodušše a prostě navrhnutý - jednoduché integrování pro různé použití
- Nezávislost knihoven - několik knihoven s jasnou funkcionalitou
- Podpora více jazyků - jednoduše implementovatelná struktura
- Jednoduchost testování
- Použitelný jak pro velké, tak malé vývojové systémy

5.2 Koncepce

ROS má 3 úrovně koncepce [20]. Úroveň souborového systému (angl. filesystem level), úroveň výpočetního grafu (angl. computation graph level) a komunitní úroveň (angl. community level).

- Úroveň souborového systému - převážně pokrývá zdroje, které naleznete na disku, jako například:
  - Balíčky (angl. packages) - hlavní jednotka pro organizování software v ROSu. Balíček může obsahovat ROS procesy (uzly), ROS knihovny, soubory dat, konfigurační soubory, nebo jiné užitečné soubory.
  - Meta-balíčky (angl. metapackages) - specializované balíčky, které slouží pro reprezentaci skupiny souvisejících s dalšími balíčky. Obvykle jsou používány k zajištění zpětné kompatibility.
  - Manifesty balíčků (angl. package manifests) - poskytují metadata o balíčku
  - Repozitáře (angl. repositories) - kolekce balíčků, které sdílí společný verzovací systém a můžou být společně přeloženy pomocí automatizovaného překladače *catkin*.
  - Typy zpráv (angl. message types) - detaily a popisy zpráv, definují struktury zpráv posílaných v ROS systémech.
  - Typy služeb (angl. service types) - detaily a popisy služeb, definují struktury dat požadavků a odpovědí pro služby v ROS systémech.
- Úroveň výpočetního grafu - síť peer-to-peer[4] ROS procesů, která společně zpracovává data. Základními koncepcemi výpočetního grafu jsou ROS uzly (angl. nodes), hlavní server (angl. master), server parametrů, služby (angl. services), témata (angl. topics) a pytle (angl. bags). Všechny tyto koncepce poskytují data do grafu.
  - Uzly (angl. nodes) - jednotlivé procesy, které provádí výpočty. ROS je navržen, aby byl modulární a distribuovaný. Systém řízení robota má většinou spoustu uzlů, například jeden provádí řízení kol robota, druhý ovládá kamery, další provádí lokalizaci, ...
  - Hlavní server (angl. master) - poskytuje registraci jmen jednotlivých uzlů a slouží jako vyhledávač pro ostatní uzly výpočetního grafu. Je jakýmsi směrovačem pro ROS systém. Bez hlavního serveru by uzly nebyly schopny najít ostatní, vyměňovat zprávy, nebo volat služby.
  - Server parametrů (angl. parameter server) - umožňuje datům, aby byla uložena skrze klíč. Aktuálně je součástí hlavního serveru.

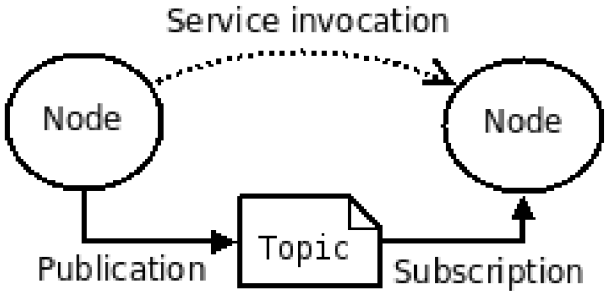
- Zprávy (angl. messages) - uzly mezi sebou komunikují skrze posílání zpráv. Zpráva je jednoduchá struktura dat, která obsahuje typovaná pole. Jsou podporovány standardní primitivní datové typy (integer, float, boolean, ...) stejně jako pole primitivních datových typů. Zprávy mohou obsahovat libovolně vnořené struktury a pole.
- Témata (angl. topics) - zprávy jsou směrovány skrze transportní systém, kde každý uzel může naslouchat, nebo publikovat. Téma je identifikováno jménem, které je použito pro identifikaci obsahu zprávy. Uzel, který chce odeslat zprávu ji publikuje na daném tématu a stejně tak uzel, který přijímá určitý typ dat na odpovídajícím tématu naslouchá. Na jednom tématu může být libovolný počet naslouchajících a publikujících uzlů a jeden uzel může publikovat nebo naslouchat na různých tématech. Naslouchající a publikující uzly si nejsou vědomi vzájemné existence. Cílem je oddělit produkci informací od jejich spotřeby. Téma se tedy dá představit jako silně typovaná sběrnice zpráv, kde každá sběrnice má jméno a každý se na tuto sběrnici může napojit pro odesílání, nebo přijímání zpráv.
- Služby (angl. services) - podobně jako témata, i služby jsou vytvořeny za účelem předávání dat mezi uzly. Model naslouchání/publikování je velmi flexibilní komunikační paradigma, avšak jeho N:N jednosměrný přenos není vhodný pro interakce požadavků a odpovědí, které jsou v distribuovaném systému často žádány. Model požadavek/odpověď zajišťují služby, které jsou definovány jako dvojice zpráv: jedna pro požadavek a jedna pro odpověď. Uzel poskytuje službu pod určitým jménem a klient využívá službu zasláním požadavku a čekáním na odpověď. Tato interakce se dá představit jako volání funkce napříč jednotlivými uzly.
- Pytle (angl. bags) - formát pro ukládání a přehrávání zpráv. Pytle jsou důležitý mechanismus pro ukládání dat, jako jsou například data ze senzorů, které mohou být těžké získat, ale jsou nutné pro vývoj a testování algoritmů.

Hlavní serveru ROSu slouží ve výpočetním grafu jako služba poskytující jména. Ukládá registraci informací témat a služeb pro uzly systému. Uzly komunikují s hlavním serverem, aby o sobě ohlásili a registrovali informace. Při komunikaci uzlů s hlavním serverem obdrží uzly informaci o ostatních uzlech a s odpovídajícími uzly pak navážou spojení. Hlavní server navíc poskytuje uzlům informace o změnách registračních informací, což dovoluje uzlům dynamicky navazovat spojení při vytvoření dalších nových uzlů.

Uzly se připojují k ostatním uzlům přímo, hlavní server jim pouze poskytuje potřebné informace, podobně jako DNS. Uzly, které naslouchají na tématu požádají o spojení s uzly, které na daném tématu publikují a vytvoří spojení na základě dohodnutého protokolu. Nejčastěji používaný protokol v ROS systémech se nazývá TCPROS (viz kapitola 5.2.1).

Tato architektura umožňuje oddělené operace, kde jména jsou primární prostředky, kterými lze stavět větší a složitější systémy. Jména v ROSu mají velmi důležitou roli, neboť všechny uzly, témata, služby a parametry mají jména. Více v kapitole 5.3.

- Úroveň komunitní - zdroje, které umožňují separátním komunitám vyměňovat software a vědomosti.



Obrázek 5.1: Základní koncepce ROSu. [20]

- Distribuce (angl. distributions) - kolekce verzí, které je možné nainstalovat. Distribuce hrají podobnou roli, jako Linuxové distribuce. Distribuce tedy udržují konzistenci verzí skrze kolekci softwaru.
- Repozitáře (angl. repositories) - rozsáhlá síť repozitářů, kde každá instituce může vyvíjet a vydat jejich vlastní software pro komponenty robota.
- ROS Wiki - komunitní Wiki je hlavní fórum pro dokumentaci informací o ROSu. Všichni se mohou přihlásit a přispět do jejich vlastní dokumentace, provádět úpravy a aktualizovat, psát návody, ...
- Systém nahlašování chyb - formou vyplnění formuláře na webových stránkách.
- Seznam kontaktních e-mailových adres - slouží jako hlavní komunikační kanál o nových aktualizacích do ROSu, stejně jako fórum, kde je možné zeptat se na otázky ohledně ROSu.

Navíc k těmto úrovním koncepce, ROS také definuje 2 typy jmen:

- Jména zdrojů balíčku - zprostředkovává hierarchickou jmennou strukturu, která se používá pro všechny zdroje ve výpočetních grafech ROSu.
- Jména grafových zdrojů - jsou používány v koncepční úrovni souborového systému, kde zjednodušují proces odkazování na soubory uložené na disku.

Více k těmto jménům v podkapitole 5.3.

5.2.1 TCPROS

TCPROS[25] slouží jako transportní vrstva pro ROS zprávy a služby. Využívá standardní TCP/IP schránky (angl. socket) pro posílání zpráv. Příchozí připojení jsou zpracovány skrze serverovou schránku s hlavičkou obsahující typ zprávy a informace o směrování. Od naslouchajícího uzlu na TCPROS protokolu je vyžadováno, aby posílal:

- message_definition - Definici zprávy
- callerid - jméno naslouchajícího uzlu
- topic - jméno tématu, na který se naslouchající uzel připojuje
- md5sum[3] - zakódovaný typ zprávy

[3] md5sum je program, který slouží pro kódování a ověřování 128bitové šifry MD5.

- type - typ zprávy

Od publikujícího uzlu v TCPROS protokolu je vyžadováno, aby při úspěšném připojení odpověděl s následujícími údaji:

- md5sum - zakódovaný typ zprávy
- type - typ zprávy

Od klienta služby TCPROS je vyžadováno, aby zasílal následující:

- callerid - jméno uzlu klienta služby
- service - jméno služby, na který naslouchající uzel připojuje
- md5sum - zakódovaný typ zprávy
- type - typ služby

Od služby TCPROS je při úspěšném připojení vyžadováno, aby odpověděl s:

- callerid - jméno uzlu, který službu poskytuje

Pokud se nepovede navázat komunikaci, TCPROS zpráva může obsahovat pole *error*, ve kterém je uložena čitelná chybová hláška.

5.3 Jména

5.3.1 Jména grafových zdrojů

Názvy grafových zdrojů [20] poskytují hierarchickou jmenovací strukturu, která je používaná pro všechny zdroje ve výpočetním grafu ROSu, jako jsou uzly, parametry, témata a služby. Tato jména jsou velmi důležitou součástí mechanismu v ROSu a poskytují zapouzdření. Každý zdroj je definován v jmenném prostoru, který může sdílet s ostatními zdroji. Zdroje mohou tvořit zdroje ve vlastním prostoru a mohou přistupovat ke zdrojům uvnitř, nebo nad vlastním jmenným prostorem. Mezi zdroji v různých jmenných prostorech navázat spojení lze, ale to bývá zajištěno pomocí integračního kódu nad oběma těmito jmennými prostory. Toto zapouzdření izoluje nezávislé části systému od "chybného vniknutí"k jinému zdroji, nebo "krádeži jmen"napříč uzly.

Jména jsou řešeny relativně, takže si jednotlivé zdroje nemusejí být vědomy, ve kterém jmenném prostoru jsou. Uzly, které spolupracují lze takto zapisovat, jako by byly všechny v obou názvů nejvyšší úrovně. Pokud jsou tyto uzly integrovány do většího systému, lze je posunout dolů do jmenného prostoru.

Platná jména

Platná jména musí mít následující charakteristiku:

1. První písmeno musí být písmeno z abecedy (a-z/A-Z), tilda, nebo lomítko.
2. Následující znaky můžou být alfanumerické (a-z/A-Z/0-9), podtržítko, nebo lomítko.

Řešení jmenných prostorů

Existují 4 typy názvy grafových zdrojů: základní (angl. base), relativní (angl. relative), globální (angl. global) a soukromé (angl. private), které mají následující syntax:

- base
- relative/name
- /global/name
- private/name

Standardně je řešení jmenných prostorů prováděno relativně ku jmennému prostoru uzlu, tedy například uzel */pracovni_prostor/uzel1* má jmenný prostor */pracovni_prostor*, a proto druhý uzel *uzel2* bude mít řešení jmenného prostoru */pracovni_prostor/uzel2*.

Jména bez jmenného prostoru jsou základní jména. Základní jména jsou podmnožina relativních jmen a mají stejné pravidla řešení. Jsou také nejvíce používány pro inicializaci jména uzlu.

Jména začínající "/"jsou globální jména a jsou považovány za již plně vyřešená. Užívání globálních jmen je vhodné se vyhýbat, neboť limitují přenositelnost kódu.

Jména, která začínají s " "jsou soukromá jména a konvertují jméno uzlu na jmenný prostor, například *uzel1* v jmenném prostoru */pracovni_prostor/* má soukromý jmenný prostor */pracovni_prostor/uzel1*. Soukromá jména jsou užitečná pro načítání parametrů do specifického uzlu skrze server parametrů.

Další příklady rezoluce jsou uvedeny v tabulce 5.1.

Uzel	**Relativní**	**Globální**	**Soukromý**
/u1	c1 ->/c1	/c1 ->/c1	∼c1 ->/u1/c1
/prostor/u2	c1 ->/prostor/c1	/c1 ->/c1	∼c1 ->/prostor/u2/c1
/prostor/u3	p/c ->/prostor/p/c	/p/c ->/p/c	∼p/c ->/prostor/u3/p/c

Tabulka 5.1: Příklady řešení jmenných prostorů

5.3.2 Jména zdrojů balíčku

Jména zdrojů balíčků [20] jsou používána pro zjednodušení odkazování na soubory a datové typy na disku. Jsou to pouze jména balíčku, ve kterém je žádaný zdroj a jméno samotného zdroje. Například jméno *std_msgs/String* odkazuje na typ zprávy *String*, který se nachází v balíčku *std_msgs*. Některé soubory použité v ROS systémech tento systém odkazování používají. Jsou to:

- Typy zpráv
- Typy služeb
- Typy uzlů

Jména zdrojů balíčků jsou velmi podobná cestám k souborům, avšak díky schopnosti ROSu najít balíčky na disku a dělat rozhodnutí o jejich kontextu jsou mnohem kratší.

Platná jména

Tato jména musí mít striktní pravidla pro pojmenování, protože jsou často používána v automaticky vygenerovaném kódu a z toho důvodu nemůžou mít žádné speciální znaky krom podtržítka. Krom toho musí začínat s písmenem. Platné jméno má následující charakteristiku:

1. Prvním znakem je písmeno (a-z/A-Z)
2. Další znaky můžou být alfanumerické (a-z/A-Z/0-9), podtržítko, nebo lomítko
3. V celém názvu je maximálně jedno lomítko

5.4 Podporované jazyky

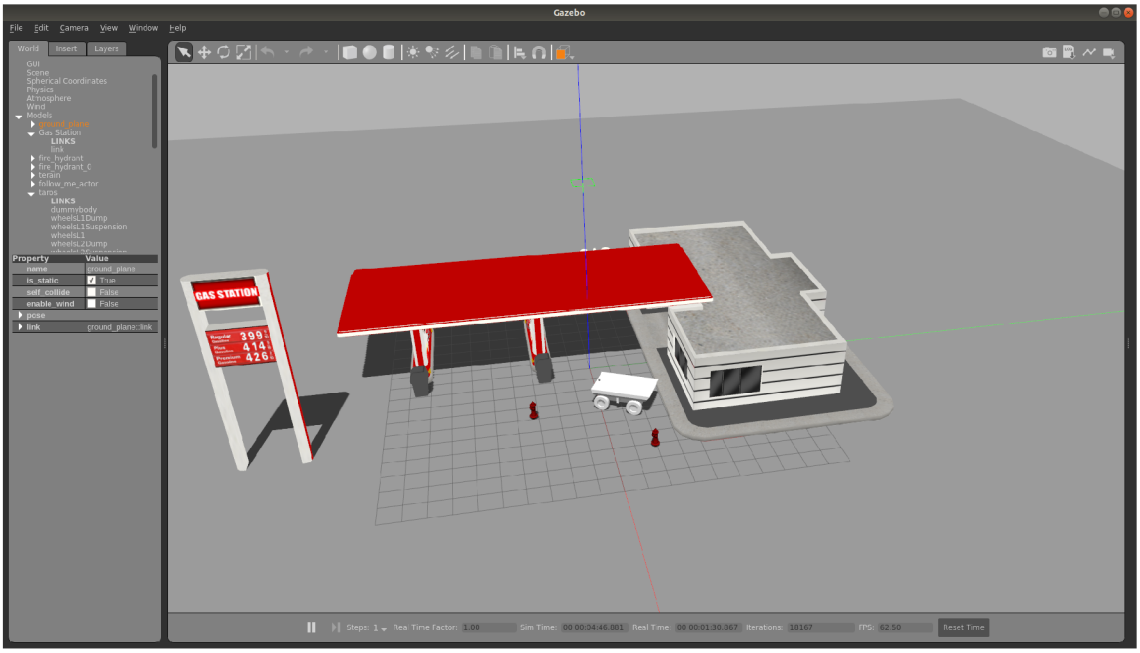
Díky jednoduché implementaci ROSu je možné jej implementovat v libovolném programovacím jazyce, avšak aktuálně jsou podporovány 3 hlavní jazyky:

- **C++** je staticky typovaný, (většinou) kompilovaný, multiparadigmatický, středně pokročilý, obecně užívaný programovací jazyk. C++ je sofistikovaný a efektivní. Je založen na C a byl vytvořen Bjarnem Stroustrupem v roce 1979. Mnoho dnešních operačních systémů, systémových ovladačů, počítačových her a dalších programů využívají C++ jako jejich hlavní jazyk. Jeho všestrannost z něj dělá jeden z nejpopulárnějších jazyků. Důležité koncepty v C++ zahrnují polymorfizmus, předlohy, virtuální metody, ukazatele a jmenné prostory.
- **Python** je interpretovaný, vysokoúrovňový, obecně užívaný programovací jazyk. Byl vytvořen Guidem van Rossumem a byl poprvé vydán v roce 1991. Jeho výhodou je vysoká čitelnost kódu. Interpreti Pythonu jsou dostupní pro mnoho operačních systémů. Podporuje více programovacích paradigmat, včetně objektově orientovaného, imperativních, funkcionálních a procedurálních. Mezi jeho další výhody patří obsáhlá standardní knihovna. Nevýhodou je jeho pomalejší spouštění kódu (avšak v dnešní době zanedbatelná nevýhoda).
- **Lisp** je skupina programovacích jazyků s dlouhou historií s plně rodičovským předponovým zápisem. Originálně vytvořen v roce 1958. Lisp je druhý nejstarší vysokoúrovňový programovací jazyk. Byl originálně vytvořen jako praktická matematická notace pro počítačové programy, ovlivněn notací lambda kalkulu. Stal se rychle oblíbeným programovacím jazykem pro výzkum umělé inteligence. Část jeho hlavních vázaných struktur jsou vázané seznamy.

Existuje mnohem více jazyků (C#, Go, Java, Lua, Ruby, . . .), pro které jsou systémy ROSu implementovány, avšak jsou považovány za experimentální a ne zcela funkční.

5.5 Gazebo pro ROS

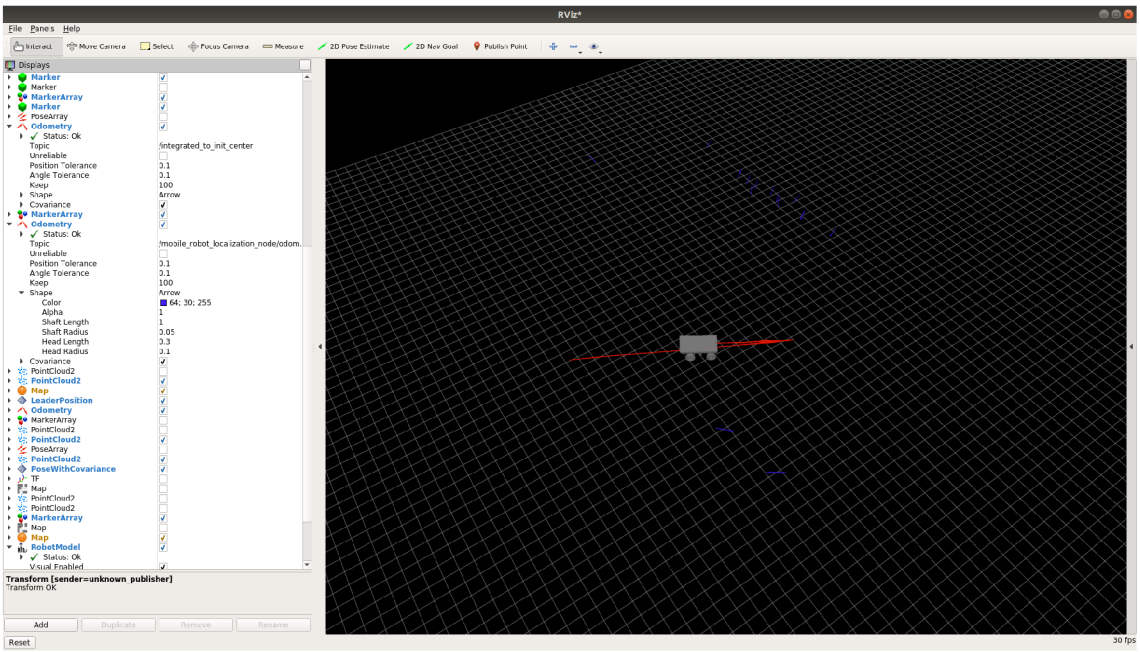
Simulace robotů je nezbytný nástroj pro jejich vývoj. Jedním z nástrojů pro simulaci je simulační nástroj Gazebo (viz obr. 5.2), které spolu s ROSem umožňuje pro robota vytvářet a testovat různé prostředí. K integraci Gazeba je třeba balíček *gazebo_ros_pkgs*, který poskytuje potřebné rozhraní za použití ROS zpráv, služeb a dynamických konfigurací.[22]



Obrázek 5.2: Ukázka simulačního prostředí Gazebo

5.6 RVIZ

RVIZ (ROS Vizualization)[24] je 3D vizualizační nástroj pro ROS. Umožňuje ve zvoleném souřadném systému vizualizovat data, které se přenáší skrze témata. RVIZ lze použít pro vizualizaci skenu z lidarů, pozici robota dle různých senzorů a dalších užitečných informací pro vizualizaci. Na obrázku 5.3 je ukázka nástroje RVIZ.



Obrázek 5.3: Ukázka nástroje RVIZ zobrazující pohyb robota a jeho trajektorii

Kapitola 6

Implementace

6.1 Existující řešení

V této podkapitole se budeme zaměřovat na příklady již existujících implementací lokalizace mobilního robota.

6.1.1 Robot localization ROS

Je to kolekce odhadujících uzlů pro Robot opererating system. Každý z nich je implementací nelineárního odhadovače stavů pro robota pohybujícího se v 3D prostoru. Agreguje data z libovolného množství senzorů. Uzly, které odhadují stav, jsou implementovány pomocí rozšířeného Kalmanova filtru popsaného v kapitole 4.2.1 a nesoustředěného Kalmanova filtru, který je dalším rozšířením. Všechny odhadové uzly sledují patnácti dimenzionální stav robota:
$(X, Y, Z, roll, pitch, yaw, \dot{X}, \dot{Y}, \dot{Z}, \dot{roll}, \dot{pitch}, \dot{yaw}, \ddot{X}, \ddot{Y}, \ddot{Z})$

Všechny stavy v Robot localization sdílejí stejné funkce:

- Splynutí libovolného počtu senzorů.
- Přizpůsobení vstupu pro každý senzor (pokud senzor obsahuje data, která nechceme použít k odhadu stavu, Robot localization umožňuje vyloučení těchto dat).
- Průběžný odhad (každý uzel odhaduje stav vozidla hned, jakmile dostane měření).
- Pokud dlouho nepřichází data z měření, filtr stále pokračuje v odhadování pozici přes vnitřní pohybový model.

6.1.2 Cartographer

Je systém, který poskytuje SLAM (Simultaneous Localization And Mapping) v reálném čase ve 2D a 3D prostoru na několika platformách a konfiguracích senzorů. Poskytuje podporu pro online i offline mód. Je implementován v ROS.

Cartographer se skládá ze dvou podsystémů. První z nich je lokální SLAM, jehož úkolem je vytvořit posloupnost menších map. Každá z těchto map je lokálně konzistentní. Druhý podsystém je globální SLAM, jehož hlavní prací je omezení uzavření smyčky. Dělá to za pomoci hledání vhodných skenů (které jsou seskupeny v uzlích) proti menším mapám. Hledá, pokud možno, co nejkonzistnější globální řešení. Lokální SLAM tedy generuje menší mapy a globální SLAM je na sebe co nejkonzistněji váže.

6.2 Návrh a popis implementace

Tato podkapitola popisuje návrh lokalizačního uzlu pro ROS (viz kapitola 5). Lokalizační uzel je implementován v jazyce C++ a využívá knihovny ROSu. K implementaci byla využita distribuce ROSu Kinetic Kame, avšak díky návrhu ROSu je uzel přeložitelný i na novějších distribucích. Implementace je omezena pouze na operační systémy, na kterých lze ROS nainstalovat (viz kapitola 5). Program byl vyvíjen a testován pouze na systému Ubuntu 16.04 Xenial.

K implementaci byl využit Rozšířený Kalmanův filtr (4.2.1.

6.2.1 Důležité součásti a závislosti

Pro implementaci lokalizačního uzlu byly využity externí soubory a knihovny. V této kapitole budou vyjmenovány a stručně popsány.

K implementaci lokalizačního uzlu byly využity následující balíčky ROSu:

- **roscpp** - je základní knihovna ROSu pro jazyk C++. Obsahuje základní datové struktury a funkce, které umožňují velmi rychlé integrování s rozhraním ROS témat, služeb a parametrů. Jde o nejrozšířenější klientskou knihovnu, která ke navržena pro co největší výkon. [12]
- **eigen_conversions** - je knihovna sloužící pro konverze mezi Eigen strukturami a geometrickými zprávami ROSu. [26]
- **sensor_msgs** - balíček definující zprávy pro běžně používané senzory (lidary, kamery, inerciální jednoty, . . .). [31]
- **geometry_msgs** - poskytuje zprávy pro běžná geometrická primitiva (body, přímky, plochy, vektory, pozice, orientace a další). Tato primitiva jsou navržena tak, aby poskytovala společný datový typ a usnadňovala interoperabilitu v systému. [29]
- **nav_msgs** - balík definuje běžné zprávy používané pro navigaci. [30]
- **tf2** - je druhá generace knihovny transformací, která definuje a sleduje rámce souřadnic (viz kapitola 6.2.2). Udržuje mezi nimi relace pomocí stromové struktury a umožňuje vzájemné transformace bodů, vektorů a jiných struktur mezi dvěma rámci. [32]
- **tf2_geometry_msgs** - umožňuje konverzi mezi zprávami tf2 a geometrickými zprávami. [36]
- **tf2_ros** - knihovna obsahující vazby mezi tf2 a roscpp. Umožňuje rozesílat a přijímat transformace mezi jednotlivými uzly. [6]

Eigen

Jak už bylo zmíněno výše, jeden z balíčků pro ROS je konverze Eigenu a geometrickými zprávami. Z toho důvodu je důležité znát, co je to **Eigen**. Eigen je C++ knihovna šablon pro lineární algebru: vektory, matice a související algoritmy. [5]

6.2.2 Koordinační rámce pro mobilní platformy

Mobilní robot pohybující se v prostoru se pohybuje v několika koordinačních rámcích. Každý z těchto rámců referuje k nějakému bodu. ROS několik základních referenčních rámců definuje a používá:

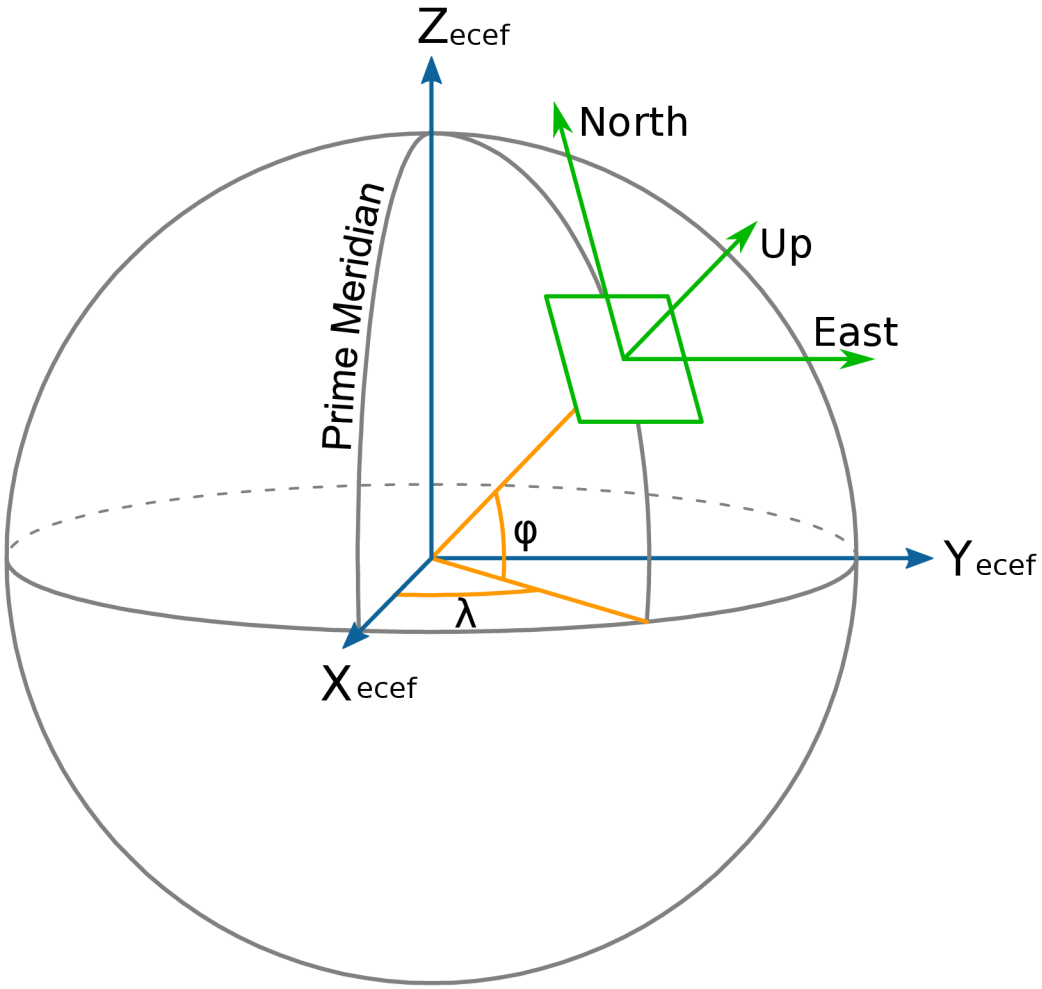
- *base_link (základní)* - tento rámec je přichycen k robotovi a mění se spolu s jeho pozicí. Pro každého robota bude jiná referenční pozice.

- *odom (odometrický)* - jde o rámec, který je fixovaný na určité místo. Pozice jakékoliv mobilní platformy může v tomto rámci v průběhu času driftovat bez jakýchkoliv omezení. Z toho důvodu je tento rámec v případě dlouhodobé globální reference nepoužitelný, nicméně pozice robota v odometrickém rámci je zaručeně spojitá. To znamená, že se pozice mobilní platformy bude vyvíjet hladce a bez diskrétních skoků. V typickém nastavení se odometrický rámec počítá na základě odometrického zdroje, jako například odometrie kol, inerciální jednotky a podobných senzorů.

- *map (mapový)* - jde o další rámec fixovaný na určité místo, jehož osa Z míří vzhůru. Pozice mobilní platformy by relativně ku tomuto rámci neměla v průběhu času výrazně driftovat. Tento rámec však není spojitý a tedy pozice platformy se může v čase měnit s diskrétními skoky. V typickém nastavení lokalizační komponenta konstantě přepočítává pozici v tomto rámci na základě senzorových měření. Díky neustálému přepočítávání je eliminováno driftování, avšak za cenu času mezi jednotlivými měření senzorů a tedy diskrétními skoky.

- *earth (světový)* - tento referenční rámec je navržen tak, aby umožňoval interakci více robotů v jiných mapových rámcích. Pokud aplikace potřebuje pouze jeden mapový rámec, není přítomnost tohoto rámce nutná. V případě, že v systému existuje více mapových rámců, je třeba tomu přizpůsobit i odometrický a základní rámec. V případě, že je mapový rámec globálně odkazován na jedno místo v světovém rámci, je možné mezi nimi libovolně konvertovat pomocí statické transformace. Pokud ne, pak je třeba transformaci vypočítat za pomocí odhadu z aktuální globální pozice a odečtením aktuální odhadované pozice v mapě. V případě, že je mapový rámec odkazován na absolutní pozici, která je však v čase spuštění lokalizace neznámá, může mapový rámec zůstat oddělený až do chvíle, kdy je možné globální pozici adekvátně vyvodit. Na obrázku 6.1 je znázorněna relace ECEF[1] s mapovým rámcem.

[1] ECEF je kartézský souřadnicový systém využívány k určení polohy těles vůči zemi.



Obrázek 6.1: Vizualizace ECEF s tangenciálním mapovým rámcem. [35]

Relace koordinačních rámců je stromovou strukturou, tedy každý rámec má právě jeden rodičovský koordinační rámec a jakýkoliv počet dětských koordinačních rámců. Níže (6.1) je ukázáno, jak jsou na sebe jednotlivé koordinační rámce navázány.

$$earth \longrightarrow map \longrightarrow odom \longrightarrow base\_link \tag{6.1}$$

Mapový rámec je rodičem odometrického a odometrický je rodičem základního i přesto, že by oba rámce měli být logicky připojeny k základnímu rámci. To však není možné z důvodu, že každý rámec může mít právě jednoho rodiče.

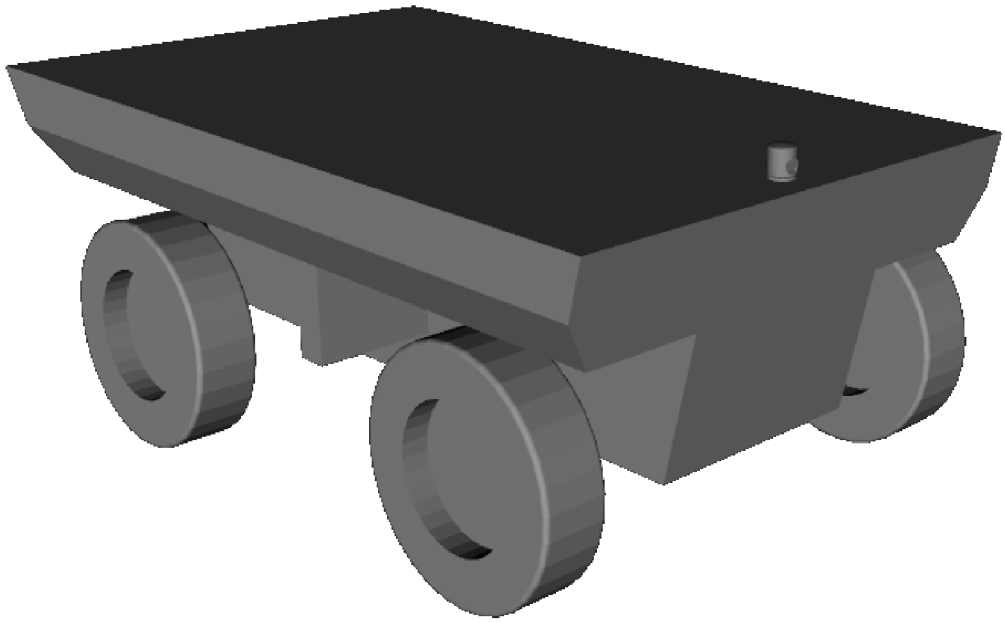
Jak již bylo zmíněno, tyto referenční rámce jsou pouze základní a tvoří minimální reprezentaci tohoto grafu. Ačkoliv by základní topologie měla zůstat stejná, je v pořádku přidat další referenční rámce a napojit je do sytému. Díky tomu jsme schopni poskytnout další funkcionalitu do systému.[35]

6.2.3 Model robota

Model robota byl popsán pomocí jazyku Xacro. Jde o typ jazyku XML který dokáže konstruovat kratší a více čitelné XML soubory za pomoci použití maker (název Xacro je složen z XML Macro). Jazyk se často využívá v balíčku urdf, který slouží k popisu modelu robota

ve formátu, které jsou schopny programy přečíst. Soubory urdf nejsou většinou tvořeny ručně, nýbrž jsou generovány právě ze souborů Xacro. [27] [10] Model čitelný programem je popsán v souboru *robot.urdf*, který se nachází v balíčku *mobile_robot_description*.

Grafická vizualizace robota byla vytvořena v programu Blender a je zobrazena na obrázku 6.2. Kvůli kompatibilitě se simulačním programem Gazebo (viz 5.5) byl model exportován do formátu *dae*[2].



Obrázek 6.2: Grafická vizualizace robota

6.2.4 Zpracování dat ze senzorů

Myšlenka zpracování dat pomocí lokalizačního uzlu se zakládá na konverzi výstupních dat z libovolného senzoru použitelného pro lokalizaci na relevantní data. Program umí zpracovávat 5 různých výstupů ze senzorů:

- **Pózu** (angl. Pose) - představuje pozici a orientaci v prostoru (X, Y, Z, náklon, sklon a rotaci). Zakládá se na zprávě z balíčku *geometry_msgs* - PoseWithCovarianceStamped.
- **Kroucení** (angl. Twist) - představuje lineární a úhlovou rychlost (po ose X, Y, Z a rychlost otáčení se). Zakládá se na zprávě z balíčku *geometry_msgs* - TwistWithCovarianceStamped.

[2]Digital Asset Exchange je formát založený na XML používaný pro výměnu digitálních dat mezi více grafickými programy.

- **Akceleraci** (angl. Acceleration) - představuje akceleraci v prostoru (po ose X, Y a Z). Zakládá se na zprávě z balíčku *geometry_msgs* - AccelWithCovarianceStamped.
- **Odometrii** (angl. Odometry) - je kombinací pózy a kroucení a stejně tak je zpracováván. Zakládá se na zprávě z balíčku *nav_msgs* - Odometry
- **Inerciální jednotka** (angl. IMU) - většinou obsahuje všechna data, tedy pózu, kroucení i akceleraci. Vychází ze zprávy z balíčku *sensor_msgs* - IMU.

Všechny senzory, které byly využity disponují ovladačem pro systém ROS a jsou tak odstíněny jako samostatné uzly. Tyto uzly pak publikují výše zmíněné data.

Senzory se dají libovolně přidávat/odstraňovat v konfiguračním souboru *sensor_config.yaml*, který se nachází ve složce s lokalizačním uzlem. V souboru je několik příkladů přidání senzorů do systému.

Přijaté zprávy jsou konvertovány do vhodného formátu a agregovány pomocí Rozšířeného Kalmanova filtru (viz kapitola 4.2.1).

6.2.5 Konfigurace aplikace

Aplikace se dá do jisté míry nastavit v souboru *mobile_robot_localization.yaml*, která se opět nachází ve složce s lokalizačním uzlem. Soubor je již přednastaven na výchozí hodnoty.

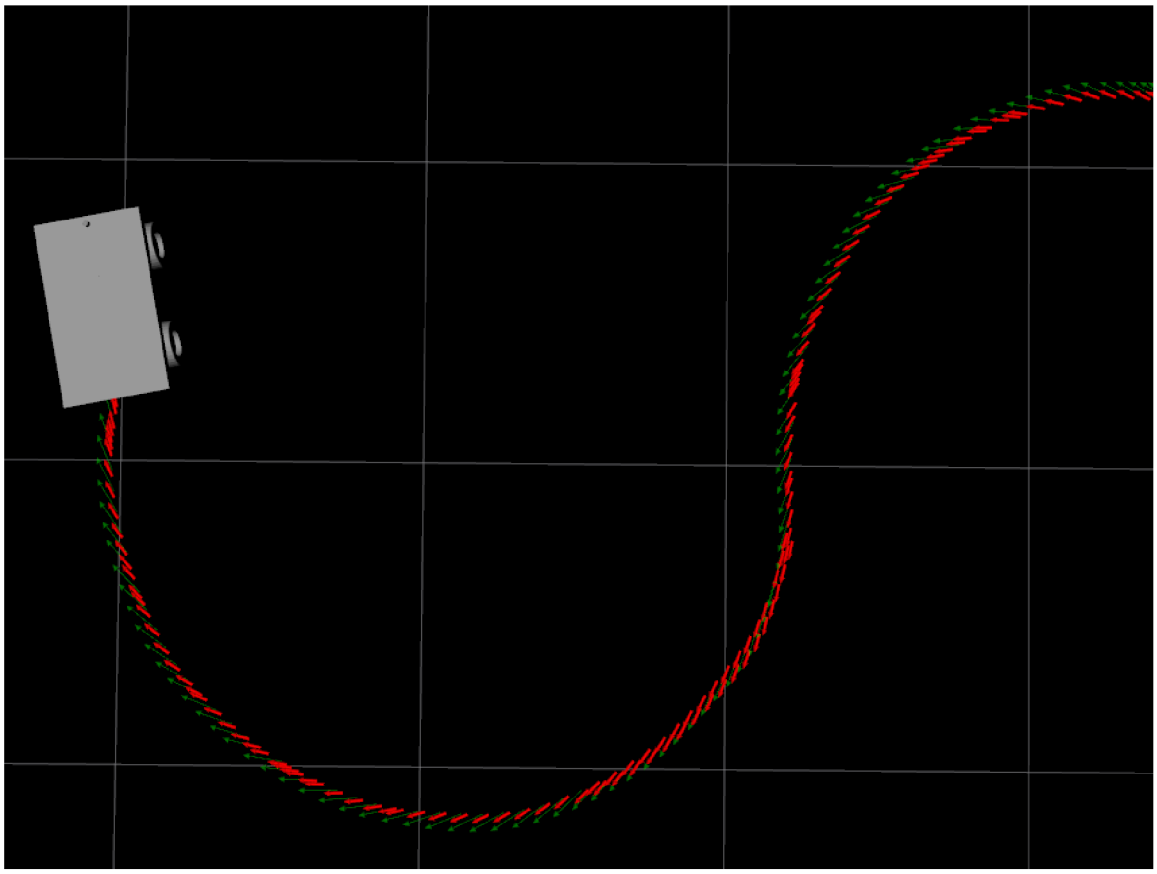
Výstupem aplikace je filtrovaná odometrie robota, tedy jeho pozice, orientace, lineární rychlost a úhlová rychlost. Odometrie tohoto robota je publikována na tématu *mobile_robot_localization/filtered_odometry*. Rámec (viz 6.2.2), ve kterém bude výstup vyjádřen je konfigurovatelný pomocí výše zmíněného konfiguračního souboru.

6.3 Testování a experimenty

Testování probíhalo jak na reálných, tak simulovaných datech. Pro vizualizaci testování byl použit RVIZ (viz 5.6). Simulovaná data byla generována za pomocí simulačního programu Gazebo (viz 5.5). Reálná data byla dodána firmou RCE Systems ve formě pytlů ROSu (5.2).

6.3.1 Základní pohyb v simulačním prostředí

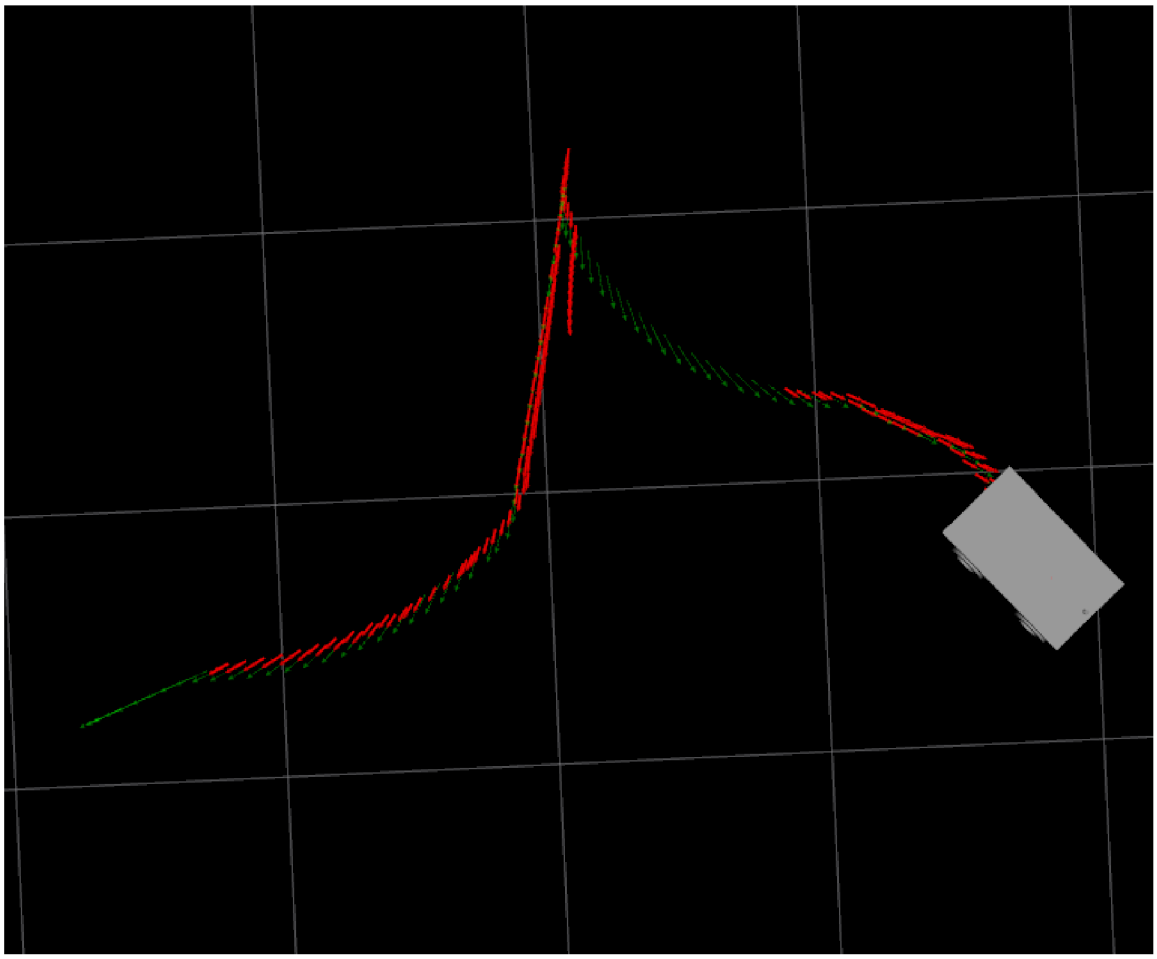
Experiment má za cíl otestovat základní pohyb robota v simulačním prostředí Gazebo. Lokalizační uzel měl nastaven pouze jeden vstup, a to odometrii získanou z podvozku. Odometrie z podvozku je v simulačním prostředí přesnější, než v reálném. Důvodem je obtížné nasimulování všech faktorů, které přesnost ovlivňují (např. podklouznutí, nebo protočení kol). Na obrázku 6.3 lze vidět pozici mobilní platformy při tomto scénáři, kde zelené šipky reprezentují pozici robota (získanou přímo v programu RVIZ) a červené šipky odhad pozice na základě odometrických dat.



Obrázek 6.3: Pohyb v simulačním prostředí

6.3.2 Rychlá změna směru

Robot jedoucí pouze vpřed není příliš reálný. Z toho důvodu je třeba, aby lokalizační uzel reagoval i na rychlé změny. Při použití filtru bez vstupu rychlosti lze rychlou změnu způsobit například prudkým zastavením a couváním v jiném směru. V takovém případě se filtr na nějakou dobu "ztratí". Obrázek 6.4 poukazuje na tento jev a lze na něm vidět následné "nalezení se". Barvy jsou stejné, jako v předchozím experimentu (6.3.1).

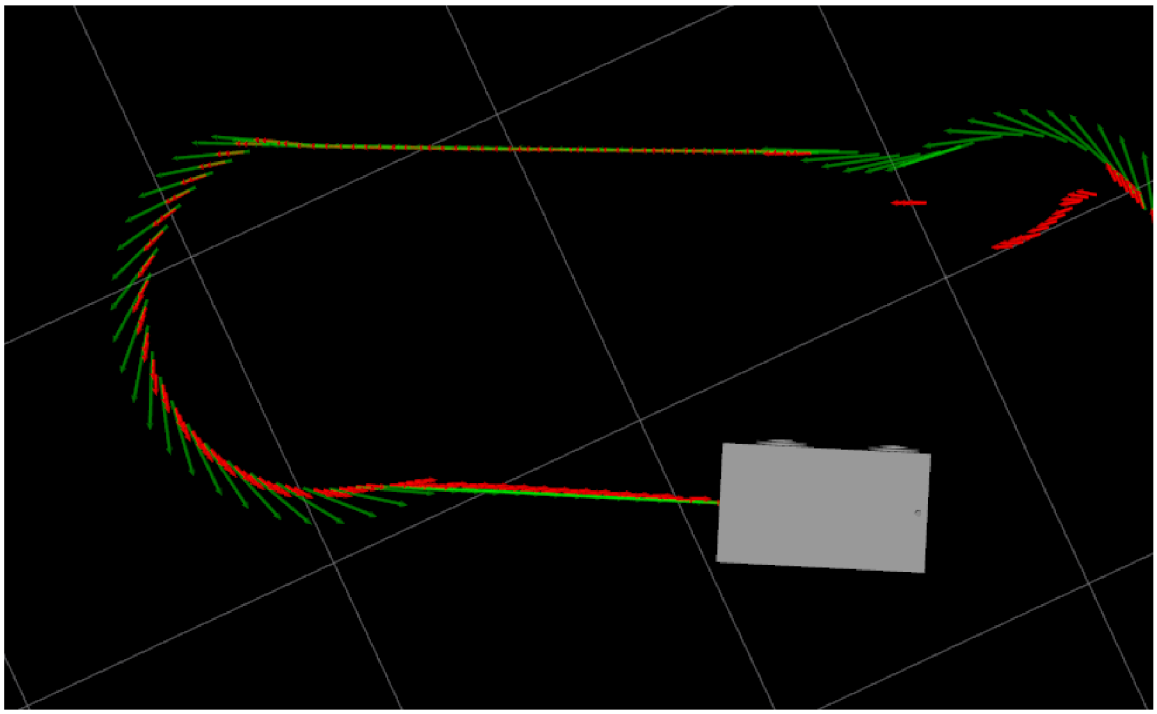


Obrázek 6.4: Chování při prudké změně

6.3.3 Výpadek senzoru

U mobilních robotů se může stát, že robotovi může vypadnout senzor. V takovém případě je třeba si takového problému všimnout. Některé senzory dokážou poznat, že nefungují a tuhle informaci propagovat dál. V našem případě však nemusí výpadek nutně znamenat nefunkčnost senzoru, nýbrž jeho neschopnost změřit přesná, nebo vůbec nějaká data. Příkladem takového chování může být dálkoměr, jehož cíl na odražení je příliš daleko a díky tomu je neschopen vrátit relevantní hodnotu. Jako podobný příklad můžeme uvést GPS, jenž se z důvodu okolního rušení není schopna spojit, nebo inerciální jednotka, která je vlivem magnetického rušení rotována špatným směrem.

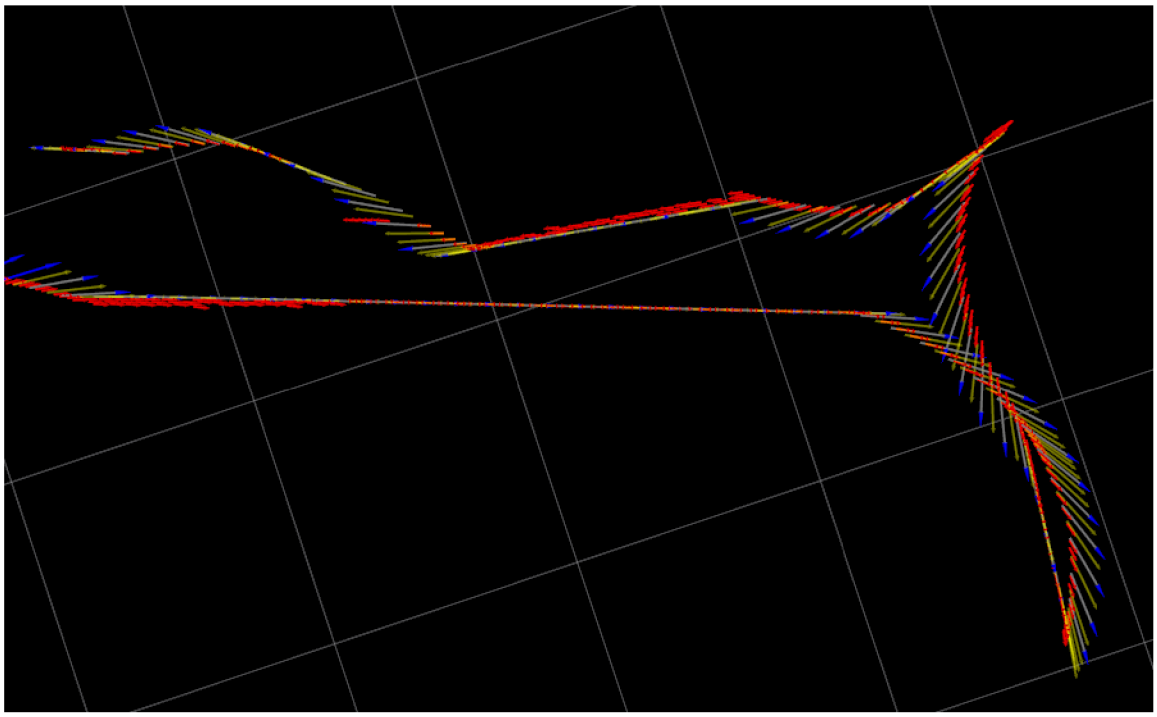
V tomto experimentu byly využity tři senzory. Odometrická zpráva z podvozku, odometrická zpráva z GPS a inerciální jednotka. Na obrázku 6.5 je vidět výpadek inerciální jednotky, která byla hlavní aktualizací pro rotaci. Lze vidět, že hned po počáteční rotaci vlevo se filtr snažil odhadovat z posledních dat a tedy proto stav pokračuje dolů, jako kdyby chtěl robot dále zatáčet. Po několika iteracích však dokáže zjistit, že tento senzor dlouho neaktualizoval svůj stav a snižuje mu váhu na systém.



Obrázek 6.5: Výpadek inerciální jednotky

6.3.4 Filtrování reálných dat

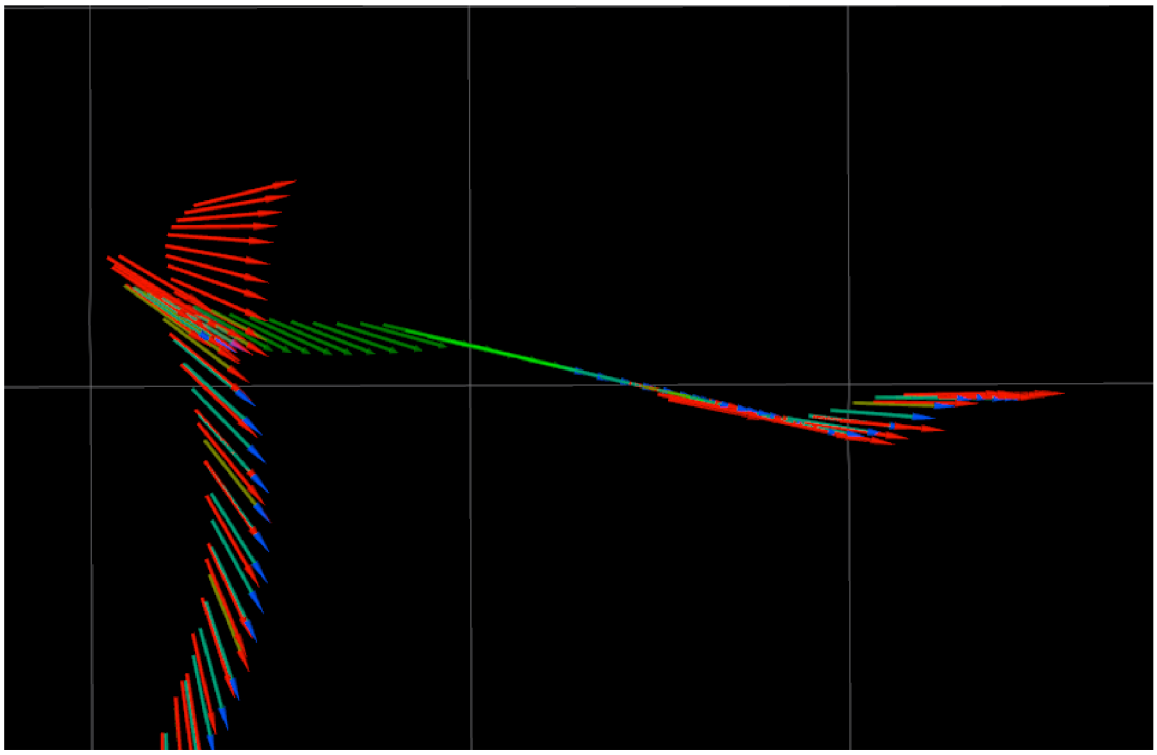
Cílem experimentu bylo otestování lokalizačního uzlu na reálných datech. Jsou agregovány dvě vstupní data. Odometrie z GPS (žlutá) a odometrie z podvozku (modrá), vizualizováno na obrázku 6.6.



Obrázek 6.6: Otočení robota do protějšího směru

6.3.5 Výpadek více senzorů u reálných dat

U reálných dat je obvyklé, že stejně jako u předchozího experimentu senzor na chvíli vypadne. Obrázek 6.7 poukazuje na měření, kde byly agregovány a filtrovány čtyři senzory. Dvě inerciální jednotky (modrá a tyrkysová), odometrie z podvozku (zelená) a odometrie z GPS (žlutá). Při couvání a následném rozjetí se kupředu se dočasně přerušili data z inerciálních jednotek. Odometrie podvozku a GPS je však v pořádku a po několika iteracích se vyfiltrovaná pozice stabilizuje.



Obrázek 6.7: Výpadek více senzorů

Kapitola 7

Závěr

Problematika lokalizace mobilní platformy je v robotice jedna z hlavních odvětví. Je nezbytné, aby každý robot, který potřebuje dosáhnout určité přesnosti (ať již v lokalizaci, na výrobní lince, nebo jinde), znal svůj stav, polohu, pozici, rychlost, nebo jiné klíčové vlastnosti. V rámci této bakalářské práce byly nastudovány jednotlivé senzory, které tento stav zjišťují. Díky různým vnějším šumům, které na tyto senzory působí, je občas obtížné zjistit přesný stav tohoto senzoru. Místo získávání takto zašuměného měření se vyplatí využít lokalizační metody (4, které jsou schopny tato měření zpřesnit.

Pro lokalizaci robota se používá Rozšířený Kalmanův filtr (4.2.1), který byl implementován jako ROS (5) uzel v jazyce C++. Měření dat jsou poslána do filtru, kde se spojí do jednotného formátu. Tento proces se nazývá agregace/fúze dat. Rozšířený Kalmanův filtr pak tyto data rekurzivně periodicky zpracovává a po každé iteraci odešle aktuální stav. Jeho velkou výhodou je jeho rychlost vyhodnocování, která je ve srovnání s jinými lokalizačními metodami několikanásobná.

Navržený systém byl otestován na pěti různých scénářích, tři simulované v programu Gazebo (5.5) a dva postavené na reálných datech uložených v pytlech ROSu. Výstupní data byla porovnávána s vstupními tak, aby byla zřejmá agregace a přibližná pozice robota.

Výstupem navrženého systému je odometrie robota s nastavitelným výstupním koordinačním rámcem.

Jako další rozšíření práce by bylo možné použít více senzorů, jako například využít kamery pro zjištění rychlosti otáčení, využití rychlosti otáčení jednotlivých kol, a dalších. Je možné dále generalizovat program, rozšířit kompatibilitu vstupních zpráv a tak zajistit jeho snadnější používaní. Je možné rozšířit aplikaci o další lokalizační metody a díky tomu zvolit vhodnou metodu pro danou situaci.

Literatura

[1] Adámek, M.: *Obecné rozdělení senzorů*. [Online; navštíveno 17.4.2019].
URL http://www.umel.feec.vutbr.cz/~adamek/uceb/DATA/s_1_2.htm

[2] Backlund, A.: Definition of system. *Kybernetes*, ročník 1, č. 1, 1972: s. 444–451, ISSN 0368-492X, doi:10.1108/j.trit.2016.03.005.
URL https://www.emeraldinsight.com/doi/full/10.1108/03684920010322055

[3] BZARG: *How a Kalman filter works?* [Online; navštíveno 9.1.2019].
URL https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures

[4] Christensson, P.: *P2P Definition*. [Online; navštíveno 18.3.2019].
URL https://techterms.com/definition/p2p

[5] *Eigen*. [Online; navštíveno 15.5.2019].
URL http://eigen.tuxfamily.org/index.php?title=Main_Page

[6] Eitan Marder-Eppstein, Wim Meeussen: *Robot operating system: nav_msgs*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/nav_msgs

[7] GIS, G.: Most popular LiDARs. 2015, online; navštíveno 19.4.2019.
URL
https://i2.wp.com/grindgis.com/wp-content/uploads/2015/01/LiDAR-Image.png

[8] Hůlka, J.: *Senzorický systém pro robotický podvozek*. Bakalářská práce, FIT VUT v Brně, 2012.

[9] Institute of Physics: *Trilateration*. [Online; navštíveno 16.1.2019].
URL http://www.physics.org/article-questions.asp?id=55

[10] Ioan Sucan, Jackie Kay: *Robot operating system: urdf*. [Online; navštíveno 23.5.2020].
URL http://wiki.ros.org/urdf

[11] Levy, S. D.: *Rozšíření Kalmanovy filtrace*. [Online; navštíveno 9.1.2019].
URL https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/#mjx-eqn-matrixgain

[12] Morgan Quigley, Josh Faust, Brian Gerkey, Troy Straszheim: *Robot operating system: roscpp*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/roscpp

[13] Navrátil, L.: *Lasery a pulzní magnety v terapii*. Praha, Alberta, 1994, ISBN 80-85792-09-5.

[14] Newton, I.: *Newtonovy zákony*. [Online; navštíveno 12.12.2018].
URL https://www.grc.nasa.gov/www/k-12/airplane/newton.html

[15] NOAA: *Lidar*. 2013, [Online; navštíveno 16.8.2019].
URL https://oceanservice.noaa.gov/facts/lidar.html

[16] Peringer, P.: *Modelování a simulace - studijní opora*. Brno, VUT, 2012.

[17] ROS: Effects of wrong lidar calibration. https://answers.ros.org/question/229991/effects-of-wrong-lidar-calibration/, 2020, [Online; navštíveno 15.5.2020].

[18] Sebastian THRUN, Wolfram BURGARD a Dieter FOX: *Probabilistic robotics*. London, England: MIT Press, 2006, ISBN 978-0-262-20162-9.

[19] Skalka, M.: *Srovnání lokalizačních technik v robotice*. Diplomová práce, Univerzita Karlova v Praze, 2011.

[20] Stanford Artificial Intelligence: *Robot operating system: Concepts*. [Online; navštíveno 21.4.2019].
URL http://wiki.ros.org/ROS/Concepts

[21] Stanford Artificial Intelligence: *Robot operating system: Documentation*. [Online; navštíveno 18.3.2019].
URL http://wiki.ros.org/Documentation

[22] Stanford Artificial Intelligence: *Robot operating system: Gazebo ROS Pkgs*. [Online; navštíveno 21.4.2019].
URL http://wiki.ros.org/gazebo_ros_pkgs

[23] Stanford Artificial Intelligence: *Robot operating system: Introduction*. [Online; navštíveno 21.4.2019].
URL http://wiki.ros.org/ROS/Introduction

[24] Stanford Artificial Intelligence: *Robot operating system: RVIZ*. [Online; navštíveno 21.4.2019].
URL http://wiki.ros.org/rviz

[25] Stanford Artificial Intelligence: *Robot operating system: TCPROS*. [Online; navštíveno 21.4.2019].
URL http://wiki.ros.org/ROS/TCPROS

[26] Stuart Glaser, Adam Leeper: *Robot operating system: eigen_conversions*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/eigen_conversions

[27] Stuart Glaser, William Woodall, Robert Haschke: *Robot operating system: xacro*. [Online; navštíveno 23.5.2020].
URL http://wiki.ros.org/xacro

[28] TECHNAID: IMU Working Principles. 2015, online; navštíveno 13.2.2019.
URL https://www.technaid.com/support/research/imu-working-principles/

[29] Tully Foote: *Robot operating system: geometry_msgs*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/geometry_msgs

[30] Tully Foote: *Robot operating system: nav_msgs*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/nav_msgs

[31] Tully Foote: *Robot operating system: sensor_msgs*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/sensor_msgs

[32] Tully Foote, Eitan Marder-Eppstein, Wim Meeussen: *Robot operating system: tf2*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/tf2

[33] Wikipedia: Ackermann steering geometry — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Ackermann%20steering%20geometry&oldid=939930842, 2020, online; navštíveno 15.5.2020].

[34] Wikipedia: Recursive Bayesian estimation — Wikipedia, The Free Encyclopedia. http://en.wikipedia.org/w/index.php?title=Recursive%20Bayesian%20estimation&oldid=942105716, 2020, [Online; accessed 15-April-2020].

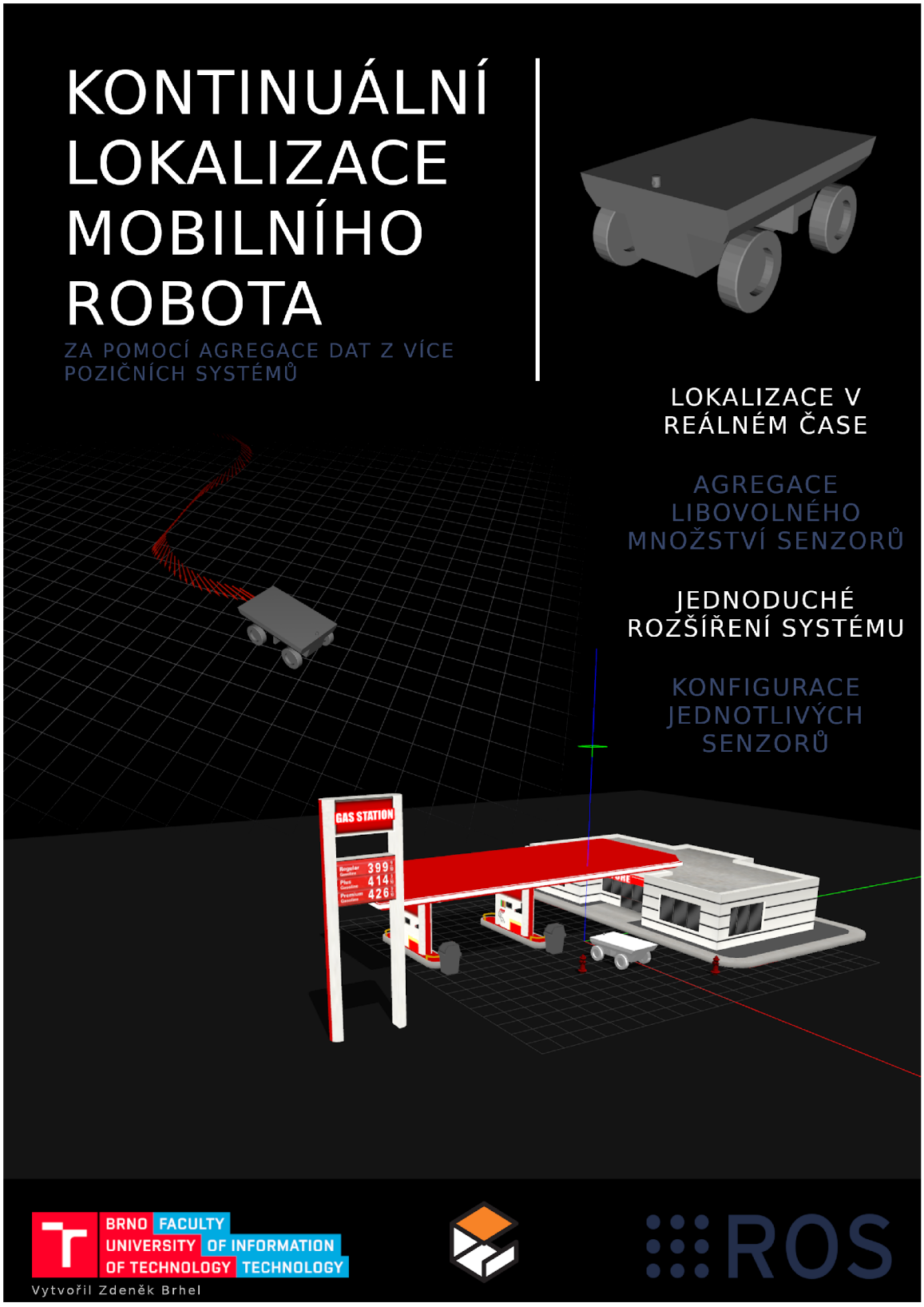
[35] Wim Meeussen: *Coordinate Frames for Mobile Platforms* . [Online; navštíveno 18.5.2020].
URL https://www.ros.org/reps/rep-0105.html

[36] Wim Meeussen: *Robot operating system: tf2_geometry_msgs*. [Online; navštíveno 18.5.2019].
URL http://wiki.ros.org/tf2_geometry_msgs

[37] Čakloš, O.: *Určování polohy robota na základě měření ze senzorů*. Bakalářská práce, FIT VUT v Brně, 2018.

Příloha A

Plakát



Obrázek A.1: Plakát

Příloha B

Obsah DVD

Umístění	Obsah
/README.txt	Popis obsahu DVD, návod na překlad a spuštění aplikace
/bp.pdf	Textová část bakalářské práce
/plakat.png	Plakát
/tex	zdrojové soubory textové části bakalářské práce ve formátu LaTeX
/mobile_robot/src	Zdrojové kódy ROS uzlů
/mobile_robot/bin	Přeložené uzly
/bp.mp4	Prezentační video bakalářské práce