Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Implementace a nasazení řídícího software pro autonomního robota Turtlebot 2

Bakalářská práce

Autor: Luděk Krůla
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž Ph.D.

Hradec Králové Duben 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 5.5.2020 Jméno a Příjmení Luděk Krůla

Poděkování:
Tímto bych rád poděkoval vedoucímu bakalářské práce Ing. Pavlu Křížovi Ph.D. za metodické vedení práce, také za cenné a odborné rady, které mi během zpracování bakalářské práce poskytoval.

Anotace

Tato bakalářská práce se zabývá implementací programu pro sběr vzorků síly signálů z bezdrátových sítí, kdy je každý vzorek ukládán do datové struktury vytvářející síť bodů s kartézskými koordináty. Program je implementován na robotické open source platformě Turtlebot 2 s RGB-D snímačem Microsoft Kinect, který je využit k rozpoznávání okolního prostředí. Robotická platforma je řízena za pomoci middleware Robot Operating System (ROS) a využití simultaneous localization and mapping (SLAM) algoritmů metody Gmapping. Práce se dále zaměřuje na porovnání dalších metod SLAM a jejich nasazení pro sběr vzorků bezdrátových sítí, jak vyplývá z odborné literatury vyšla, v kombinaci uvedených prostředků, v praktické části této práce metoda Gmapping jako nejvhodnější.

Klíčová slova

lokalizace, mapování, simultální localizace a mapování (SLAM), Robotický Operační Systém (ROS), Gmapping

Annotation

Title: Implementation of control software for autonomous robot Turtlebot 2

This bachelor's thesis pursues the implementation of a program for collecting samples of signal strength from wireless networks where each sample is stored in a data structure creating a network of points with Cartesian coordinates. The program is implemented on a robotic open source platform Turtlebot 2 with RGB-D sensor Microsoft Kinect which is used for the environment recognition. The robotic platform is controlled with middleware Robot Operating System (ROS) and uses simultaneous localization and mapping (SLAM) Gmapping algorithms. The thesis also aims to the comparison of other SLAM methods and their use for the wireless networks samples collection. The practical part of this thesis showed the Gmapping method to be most appropriate with given means, as implied in literature.

Keywords

lokalization, mapping, simultaneous localization and mapping (SLAM), Robot Operating System (ROS), Gmapping

Obsah

1 Úvod ... 1
1.1 Důvod výběru práce ... 1
2 Cíl práce ... 2
3 Rešerše řešení problému SLAM ... 3
3.1 Metody řešení SLAM na bázi filtrování ... 3
3.1.1 Extended Kalman Filter – rozšířený Kalmánův filtr ... 4
3.1.2 Particle filters – částicové filtry ... 6
3.1.3 GMapping ... 7
3.1.4 Hector ... 8
3.2 Optimalizační metody ... 8
3.2.1 Cartographer ... 10
4 Lokalizace a mapování ... 12
4.1 Lokalizace ... 12
4.1.1 Rámce (frames) ... 12
4.1.2 Póza ... 14
4.1.3 AMCL - adaptive Monte Carlo localization ... 14
4.1.4 Transformace ... 15
4.2 Mapování ... 15
4.2.1 Sonar ... 15
4.2.2 LIDAR ... 16
4.2.3 Vision-based RGB-D ... 16
4.2.4 OpenCV a OpenNI ... 16
5 Použité prostředky ... 18
5.1 Robot Operating System (ROS) ... 18
5.2 Architektura ROS ... 18

5.2.1 Nodes 19
5.2.2 Topics - témata 20
5.2.3 Zprávy 20
5.3 Simulační prostředí Gazebo a vizualizace RViz 20
5.3.1 Simulační prostředí Stage robot simulator 20
5.4 Navigace v ROS 21
5.4.1 Navigační zásobník 21
5.4.2 Balíčky costmap_2d a move_base 22
5.4.3 Navigace založené na mapě 22
5.4.4 Reaktivní navigace 23
5.5 Vzdálený přístup 24
5.5.1 Ovládání robota 24
5.6 Mapování do souboru 26
5.6.1 Přehrávání .bag souboru 27
5.7 Hardware 27
5.7.1 TurtleBot 27
5.7.2 Kinect 29
5.7.3 Řídící počítače 29
5.7.4 Ultrazvukový senzor 30
6 Popis řešení 31
6.1 Instalace jednotlivých součástí 31
6.1.1 Instalace snímače Microsoft Kinect 31
6.1.2 Pracovní prostor 32
6.1.3 Instalace ultrazvukového snímače 33
6.2 Postup řešení 33
6.2.1 Řešení nezávislé na SLAM algoritmech 35

6.2.2 Řešení s použitím Gmapping a AMCL 38
6.2.3 Použití Hector SLAM 39
6.2.4 Použití metody Google Cartographer 40
7 Shrnutí výsledků 42
7.1 Nasnímané mapy 43
8 Závěry a doporučení 46
9 Seznam použité literatury 48
10 Přílohy 52

Seznam obrázků

Obrázek 1 Vizualizace EKF 5
Obrázek 2 Vyobrazení funkce částicového filtru 6
Obrázek 3 Užití částicového filrtu v Gmapping 7
Obrázek 4 Vizualizace metod založených na grafech 10
Obrázek 5 Reprezentace rámců 13
Obrázek 6 Rotace v 3D prostoru 14
Obrázek 7 Souborová struktura ROS 19
Obrázek 8 Prostředí simulátoru Stage 21
Obrázek 9 Zobrazení překážek v Rviz 23
Obrázek 10 Ovládání za pomoci interactive markers 25
Obrázek 11 Turtlebot 2 28
Obrázek 12 Rozmístění jednotlivých snímačů na Microsoft Kinect 29
Obrázek 13 Vyobrazení rastru bodů 34
Obrázek 14 Pseudokód dopředného pohybu 35
Obrázek 15 Mapy vytvořené metodou Hector 40
Obrázek 16 Plán 3. patra Budova J UHK FIM 42
Obrázek 17 Mapové vzorky Gmapping 43
Obrázek 18 Mapa s uzavřenou smyčkou Gmapping 44
Obrázek 19 Chyba mapování - paralelní chodby 45

Seznam tabulek

Tabulka 1 Příklad nastavení systémových proměnných. 24
Tabulka 2 Parametry Turtlebot 2. 28
Tabulka 3 Úhly v mapách při různých rychlostech 44

1 Úvod

Roboty jsou v dnešní době již běžným nástrojem, ať už v průmyslu či v každodenním životě. Všední a komerční využití robotů přispívá k agilnějšímu vývoji těchto nástrojů - běžně již využíváme robotické vysavače v domácnosti, nebo k čištění bazénů. SLAM (Simultaneous localization and mapping) je technikou k vytvoření mapy prostředí za současného určování polohy robota. Během pohybu se aktuální měření a lokalizace mění, proto je pro vytvoření mapy nutné sloučit měření s předcházejícími hodnotami.

Široká škála možností aplikací s využitím lokalizace a mapování robotů v prostoru a rychlý vývoj technologií poskytuje možnosti řešení i za použití low tech technologií. SLAM je v dnešní době považován za vyřešený problém, ovšem výzvou stále zůstává realizace řešení za použití technologií s nižší přesností, na druhou stranu o snazší dostupnosti. Značnou podporu těchto technologií a aplikací poskytuje ROS (Robot operating system), který byl k implementaci v této práci použit.

1.1 Důvod výběru práce

Toto téma jsem se rozhodl zpracovat pro můj osobní zájem k robotice jako takové. Zároveň je mou motivací vytvořit řešení, které skloubí současné přístupy k dané problematice a low-tech technologii. Toto spojení je pro realizaci výzvou, ale současně lze takto přiblížit tyto přístupy širší veřejnosti.

SLAM je pro své široké využití užitečným nástrojem a v současnosti čím dál častěji využívaný, ať už jde o roboty v zásobování, domácnostech nebo pod vodní hladinou.

2 Cíl práce

Cílem této práce je navrhnout a implementovat autonomní řešení pro sběr tzv. fingerprintů z bezdrátových sítí mobilním robotem. Sběr fingerprintů by měl probíhat v předem definované síti bodů, ve kterých robot zastaví a provede případné měření.

Mobilní robot, v tomto případě komerční Turtlebot 2, by měl fingerprinty sbírat autonomně např. v budově FIM UHK. Další podstatou práce je porovnat různé aplikace SLAM a otestovat různá nastavení jejich parametrů.

3 Rešerše řešení problému SLAM

SLAM je technikou pro vytváření mapy prostředí za současného odhadování aktuálního umístění robota. Tato technika zprostředkovává mobilnímu robotovi možnost lokalizovat objekty a zároveň sebe sama i bez předcházející znalosti mapy prostředí, což je stěžejní schopnost pro širokou škálu navigačních úkolů.

Základní problémovou situací pro dvourozměrný SLAM je robot pohybující se v prostředí okupovaném dalšími objekty. Robot je vybaven proprioceptivními senzory dávající robotovi schopnost detekce vlastních pohybů a exteroceptivními senzory pro měření poměrného umístění robota a nedalekých objektů v prostředí. Smyslem problému SLAM je odhadnout polohu a orientaci robota a zároveň umístění všech přilehlých objektů.[1]

Objekty neboli features, jsou rozpoznatelné struktury prvků prostředí, které lze získat měřením ze snímačů a matematicky je popsat. Objekt, který je nezaměnitelný, je v prostředí pro svou jedinečnost kotevním bodem pro výpočty pravděpodobnosti pozice robota.[4]

Právě takovéto objekty napomáhají úspěšnému uzavření smyčky. Tato problematika se váže k situaci, kdy robot během navigování prostředím navštíví místo, které již předem navštívil. V takovém případě by mělo být takové místo rozpoznáno a mělo by dojít k překryvu s již vytvořenou mapou. Uzavření smyčky je závislé na snímačích, kvůli šumu snímačů, případně nepřesnosti odometrie, může dojít k reprezentaci stejného místa v mapě dvojmo. [23]

Aby se předešlo problému uzavírání smyčky je, u některých SLAM řešení používán Scan matcher. Jeho úkolem je určit, která data ze senzorů se překrývají (jsou si navzájem podobná) a dále zjistit jejich vztah k případným pózám robota.[24]

Většinu přístupů k řešení problematiky SLAM lze rozdělit do dvou kategorií, a to na bázi filtrování (extended Kalman filter, particle filter) a na Optimalizační (Graph-based). [2]

3.1 Metody řešení SLAM na bázi filtrování

Principem filtračních metod je rekurzivní provádění odhadu následných stavů a aktualizace současného stavu. Metody udržují informaci o prostředí a stavech

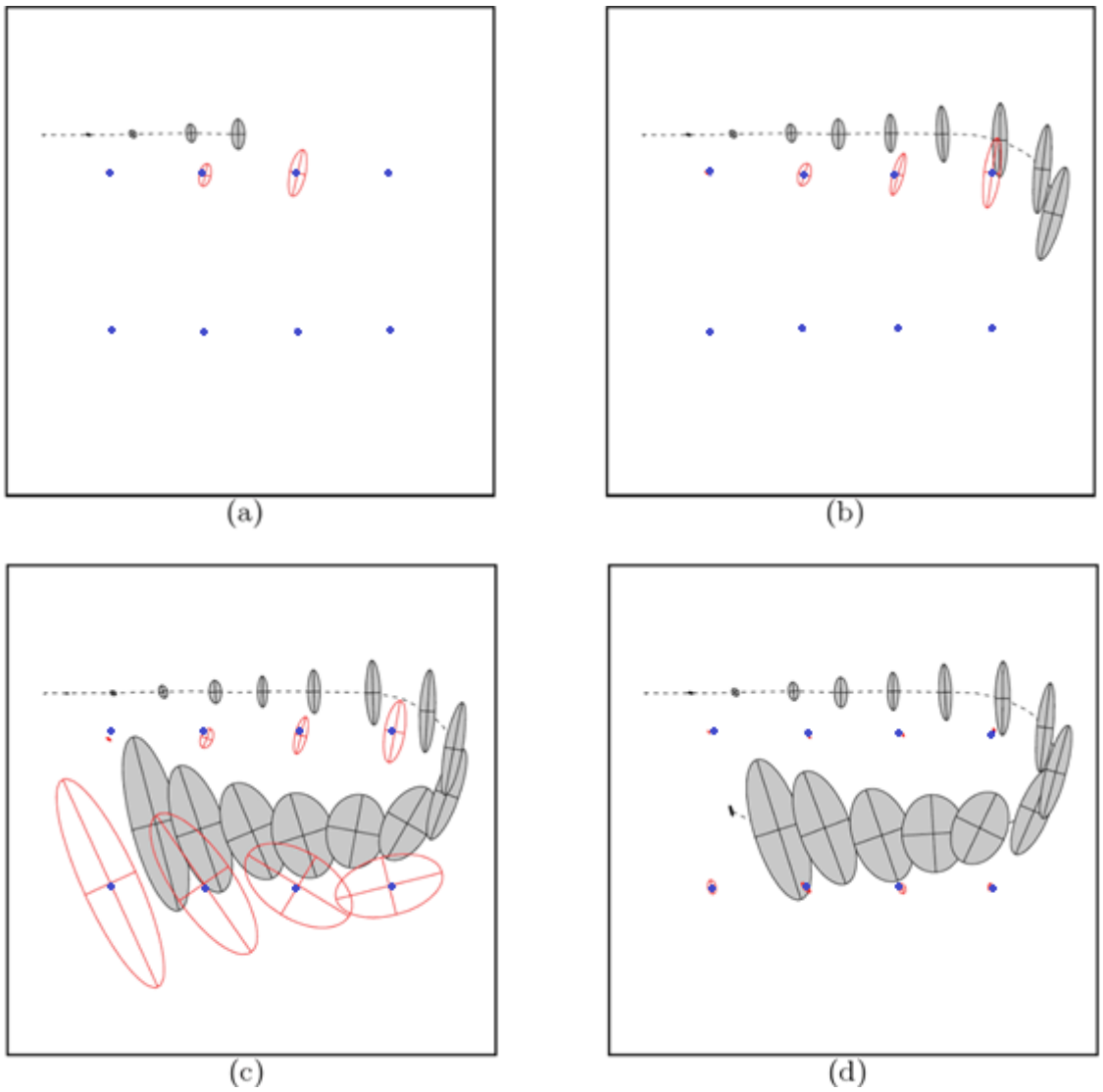
robota jako funkci hustoty pravděpodobnosti. Jedná se o metody neúplné, neřeší problém SLAM úplně. Jednotlivé přístupy se liší právě použitými filtry.[14] Pravděpodobně nejznámější a nejvíce rozšířenou metodou řešení SLAM je EKF (extended Kalman filter). EKF SLAM je založen na rozšířeném Kalmánově filtru. Aplikováním EKF na odhad pozice robota a mapy je řešen SLAM v reálném čase, taktéž lze klasifikovat jako řešení online. [3]

3.1.1 Extended Kalman Filter – rozšířený Kalmánův filtr

Rozšířený Kalmánův filtr je založen na předpokladu, že funkce reprezentující pohyb robota *(g)* a funkce pro měření *(h)* jsou ve svých argumentech lineární a lze tedy použít obecně známé principy Kalmánovy filtrace. EKF SLAM linearizuje funkce *(g)* a *(h)* pomocí rozšíření Taylorovy řady. [3] Právě linearizace je jednou z nevýhod EKF, a to právě pro svou nepřesnost. Navzdory tomu je EKF velice populární metodou řešení.

Mapa v EKF je reprezentována velikým stavovým vektorem, kam se v průběhu času přidávají stavy (informace o poloze, pohybu a čase) robota a objektů prostředí. K těmto stavům je zároveň přidána informace o možné chybě (nejistota) odhadu těchto stavů. Stavový vektor je následně upravován na základě odhadu následného stavu a rozeznání okolního prostředí. Jak se robot pohybuje a provádí měření, stavový vektor a nejistota se aktualizují pomocí standardních rovnic rozšířeného Kalmánova filtru. Vzrůstající chybu odhadu pozice a rostoucí stavový vektor lze vidět na obrázku 1. [6]

Postupným dodáváním informací do stavového vektoru je EKF SLAM schopen si zachovat informaci o celé mapě po celou dobu daného cyklu robota, bez ohledu na celkový počet časových kroků. EKF si tedy za všech okolností udržuje svůj nejlepší odhad mapy a pozice robota, proto lze EKF považovat za proaktivní algoritmus SLAM. [5]



Obrázek 1 Vizualizace EKF
Zdroj: Michael Montemerlo [6]

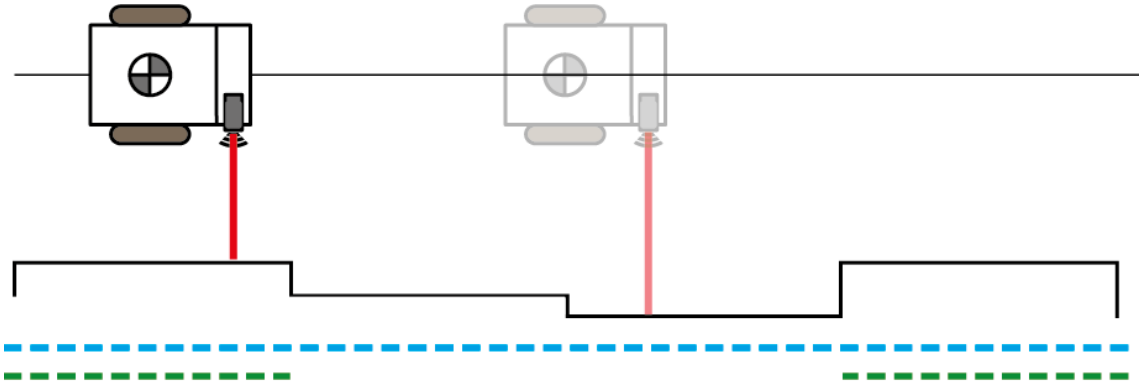
Trajektorie robota je tečkovaná čára, šedivé elipsy reprezentují odhad jeho pozice, malé modré tečky jsou rozeznatelné objekty prostředí, odhad jejich pozice je znázorněn červenými elipsami. Na obrázcích **a-c** je očividný vývoj nejistoty odhadu pozice robota vůči objektům, které potkává. Na obrázku **d** robot zpozoruje objekt, který potkal jako první, čímž nejistota polohy objektů i robota opět klesne.[6]

Mapy v EKF SLAM vycházejí z rysů (tj. bodů, čar, rovin). Jakmile jsou objeveny (detekovány) další objekty, jsou přidávány do stavového vektoru a s nimi i údaj o nejistotě jejich stavů. Nejistota je vyjádřena kovarianční maticí, která je čtvercová, roste tedy exponenciálně. Kvůli výpočtům je proto velikost mapy obvykle omezena na méně než tisíc prvků. Novější přístupy jsou již schopny pojmout větší počet

objektů, využívají rozklad mapy na menší dílčí oblasti, pro které jsou kovariance aktualizovány samostatně. [4]

3.1.2 Particle filters – částicové filtry

EKF metoda pracuje pouze s odhadem prostředí a je limitována na gaussovské modely, oproti tomu částicové filtry jsou neparametrickými implementacemi Bayesových filtrů a jsou nejčastěji využívány k odhadu stavů dynamických systémů. Předpokládejme, že robot vnímá okolní prostředí. Tento vjem lze porovnat s předchozím měřením, a tím tak odhadnout pozici robota. Každý odhad by pak představoval částici. Každá částice nastiňuje pravděpodobnost řešení v dané pozici. Lze pak snížit počet částic, buď vynecháním těch mimo rozsah senzorů nebo odstraněním těch jejichž pravděpodobnost se výrazně liší. Zároveň se zvýší počet částic v oblastech, kde je pravděpodobnost vyšší. [14]



Obrázek 2 Vyobrazení funkce částicového filtru
Zdroj: vlastní zpracování

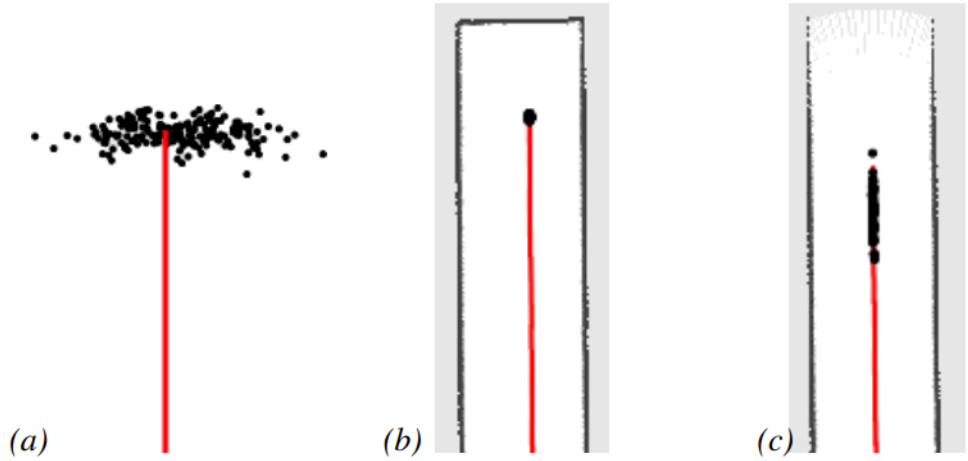
Obrázek vyobrazuje funkci částicového filtru, červeně naznačeno laserové měření vzdálenosti od objektu, modře vyobrazena populace částic, tedy odhad možné pozice robota, zeleně pak zbylé částice po filtraci, jelikož není pravděpodobné na základě vstupních dat, že robot je aktuálně v jiných pozicích.

Klíčovou myšlenkou je reprezentovat pozdější stav skupinou hypotéz na základě vstupních vzorků. Každá hypotéza představuje jeden potenciální stav, ve kterém by systém mohl být. Obecně platí, že čím více vzorků je použito, tím je aproximace lepší.[15]

Schopnost modelovat multimodální distribuce sadou vzorků je výhodou ve srovnání s řadou jiných filtrů. Kalmánův filtr je například omezen na gaussovské distribuce. Podstatou částicových filtrů je reprezentovat distribuce v každém momentu pomocí sady vzorků neboli částic, čímž nám umožňuje rekurzivní odhad sady částic S_t na základě odhadu S_{t-1} předchozího časového kroku. [10]

3.1.3 GMapping

Rozšířeným příkladem využití filtrační metody pro řešení SLAM je GMapping od OpenSLAM. GMapping využívá Rao-Blackwellized particle filter. Přístup pro řešení SLAM od OpenSLAM spočívá v tom, že oproti jiným, kteří používají jako vstupní distribuci pohybový model odometrie, GMapping se tedy zaměřuje na použití přesnějších dat ze sensorů, jako je například LRF (laser range finder). Přístup Gmapping přijímá data z laserového senzoru a odometrii, přičemž je metoda zaměřena na snímače s vyšší přesností. Současná verze Gmapping je optimalizována pro laserové skenery s velkým dosahem, jako je skener SICK LMS nebo PLS. Lasery s krátkým dosahem, jako je skener Hokuyo, nemusí se standardním nastavením parametrů fungovat tak dobře. Přesnější data ze senzorů zvyšují míru pravděpodobnosti, a tudíž algoritmus vykresluje částice daleko efektivněji. Pravděpodobnostní rozdělení je pak o vyšší přesnosti, což umožňuje upravit počet efektivních částic a následně lze rozhodnout, zda je nutné převzorkování. [18]



Obrázek 3 Užití částicového filrtu v Gmapping
Zdroj: Grisetti, Giorgio & Stachniss, Cyrill & Burgard, Wolfram [18]

Zde autoři Gmapping uvádí příklad (obrázek 3), kde je vyobrazena populace částic v různých prostředních a) ve volném prostoru, kde částice reflektují hrubý

pohybový model odometrie, přičemž v zaslepené chodbě b) se částice soustředí v malém bodě a na otevřené chodbě c) rovnoběžně se stěnami chodby. [17]
Za předpokladu použití senzorů o vyšší přesnosti je tato metoda řešení SLAM velice výhodná, jelikož čím lepší jsou vstupní data, tím lepší je výsledná aproximace stavu systému.[11]

3.1.4 Hector

Dalším příkladem využití filtračních metod je Hector SLAM, který využívá rozšířeného Kalmánova filtru. Výsledky měření z EKF následně dává do souvislosti s daty odvislými od inerciální měřící jednotky, chybou odhadu pózy robota ze scanmatcheru, případně do souvislosti s daty z dalších sensorů. Filtrace je založena na modelu pohybů pozemních vozidel. Je nezávislá na datech odometrie a řídících vstupů z důvodu nespolehlivosti těchto dat pro smyky na kluzkých či nerovných površích. [19]
Filtrační a běžná grafová řešení, fungují nejlépe v rovinných prostředích, spoléhají totiž na dostatečně přesnou odometrii a nevyužívají vysokou rychlost aktualizace vstupních dat senzorů, kterou poskytují moderní systémy LIDAR. Hector je zejména vhodný pro nestrukturovaná prostředí, kde dochází k značným pohybům robota i v jiných osách, jako je třeba klopení nebo u implementací leteckých robotů například dronů. Tato open source SLAM metoda využívá scan matcher a je silně závislá na přesných datech ze senzorů.[25]
Za určitých rychlostí a póz vykazují data ze senzorů značný posun. Proto musí být integrovány další informace z jiných senzorů. Ve vnitřních prostorách je vhodné použití scanmatcheru. Hector přistupuje k řešení tohoto problému rozdělením na frontend a backend. Zatímco SLAM frontend používá k odhadu pohybu robota online v reálném čase, backend lze využít k provedení optimalizace grafu póz vzhledem k omezením mezi pozicemi, které byly vygenerovány před použitím frontendu. [20]

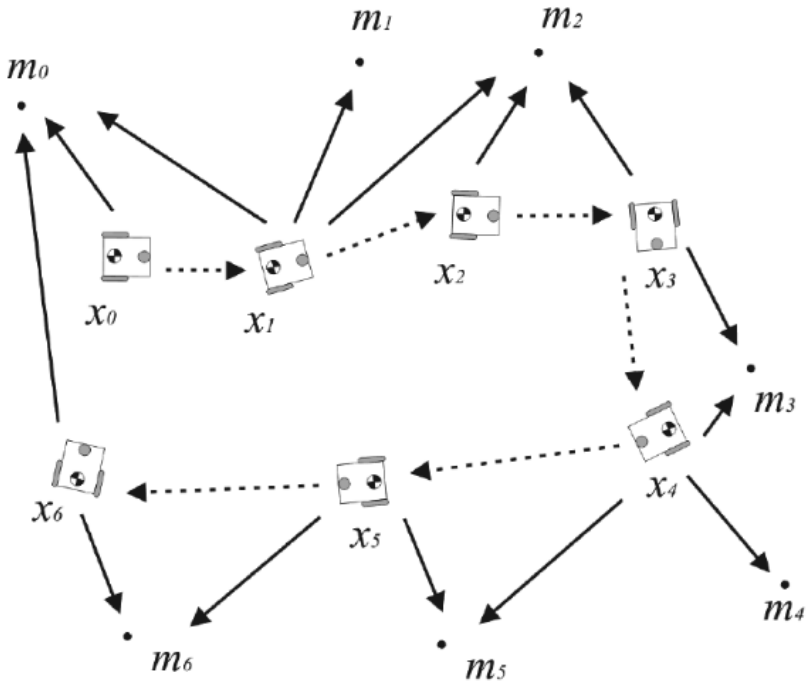
3.2 Optimalizační metody

Optimalizační metody jsou založené na grafech, kdy uzly v grafu reprezentují pozice robota a okolních předmětů a hrany grafu pak vyjadřují vztahy mezi jednotlivými uzly, ať už jde o pohyb robota mezi danými uzly, nebo pouze zpozorování objektů

snímači. [5][12] Vazby mezi jednotlivými uzly nejsou pevné, ale naopak pružné, a právě pružnost těchto vazeb napomáhá k lepším výpočtům trajektorie robota a mapy prostředí pro řešení problematiky SLAM.[4]
Optimalizační metody obsahují dva hlavní úkoly:

- Konstrukce grafu podle hrubých dat ze senzorů.
- Optimalizace grafu na základě nejpravděpodobnějších pozic na základě vztahů mezi nimi (hran v grafu).

Konstrukce grafu je často nazývána front-end a je silně závislá na vlastnostech exteroceptivních snímačů. Front-end probíhá při snímání prostředí, kdy jsou postupně objevovány nové oblasti a do grafu jsou přidávány pózy robota. Při každém přidání nové pózy je snaha nasbíraná data dát do souvislosti, ovšem v průběhu času se hromadí chyba, která ovlivňuje přesnost mapy. Tato chyba se zejména projeví pokud je navštíveno jedno místo dvakrát či vícekrát, může tak být jedno místo v mapě vyobrazeno dvojně. V takovém případě je na snadě optimalizace grafu, která porovnává různě registrovaná místa a snaží se tuto chybu eliminovat. Optimalizace je pak nazývána back-end a je závislá na na abstrakci reprezentovaných dat[2].
Podstatnou výhodou optimalizačních metod oproti filtračním je právě nižší nárok na výpočetní výkon. Jedná se ovšem o tzv. offline metody, tedy neprobíhají v reálném čase, ale jsou vyhodnocovány až následně. Mohou být také implementovány jako online řešení, kdy se pro výpočty používají podmapy. Toto je rozdíl oproti EKF metodám, které pocelou dobu pracovního cyklu robota udržují informace o celé mapě.
Použití podmap značně snižuje výpočetní nároky, zejména v situacích, kdy pracovní cyklus robota trvá dlouho. Pokud jsou rozměry snímaného prostředí větší je i přes fakt že je čas aktualizace grafu konstantní a nároky na paměť jsou lineární ve vztahu k počtu objektů, může být následná optimalizace grafu výpočetně náročná pokud je trasa robota příliš dlouhá, kdy počet hran, které nejsou pevné, značně vzroste.[4]



Obrázek 4 Vizualizace metod založených na grafech
Zdroj: SIEGWART, Roland [4]

Vyobrazení robota znázorňuje pózy (uzly grafu), tečkované čáry naznačuje pohyb robota a plné čáry identifikování objektu senzory robota. (čáry jsou vazbami grafu).

3.2.1 Cartographer

Cartographer je algoritmus vytvořený společností Google a poskytuje řešení SLAM v reálném čase. Cartographer je navržen nezávisle na datech odometrie. Příkladem je použití pro mapování vnitřních prostor, kdy je použito batohu vybaveného laserovým snímačem (LIDAR), přičemž operátor s batohem pouze prochází mapované prostředí.

Během mapování je generována 2D mřížková mapa s rozlišením r = 5cm. Operátor zařízení má možnost sledovat během mapování mapu, která je v reálném čase tvořena během procházení budovou. Jak již bylo zmíněno dříve, pro řešení za pomoci grafu a v reálném čase je v tomto případě využito podmap. Laserové skenování je vkládáno do podmap na odhadovanou pozici s nejvyšší mírou pravděpodobnosti, která je dočasně považována za dostatečně přesnou.

Pro dosažení dobrého výkonu s nižšími požadavky na hardware, tento přístup SLAM nepoužívá částicový filtr. Jelikož se během snímání bere v potaz pouze nejčerstvější

odhad mapy, kumuluje se v průběhu času chyba. Pro zamezení výrazné kumulace chyby je pravidelně spouštěna optimalizace odhadu pózy.

Cartographer se během tvoření submap snaží danou submapu uzavřít, pokud dojde k uzavření smyčky, uzavře i příslušnou submapu. Do této submapy nejsou po uzavření přiřazována žádná data.[26]

4 Lokalizace a mapování

Problém SLAM spočívá v tom, že robot se pohybuje prostředím, ve kterém lokalizuje sám sebe a zároveň provádí další mapování. Pokud bychom měli předem danou mapu a robot by měl určit pouze svou polohu v mapě, jednalo by se o problém lokalizace samotné. [6]

Předpokládáme-li pohyblivého robota se senzorem a jeho polohu v mapě, využívá robot pro odhad polohy v mapě senzorů k rozeznání okolního prostředí a následně porovnává rozpoznané prostředí s danou mapou.

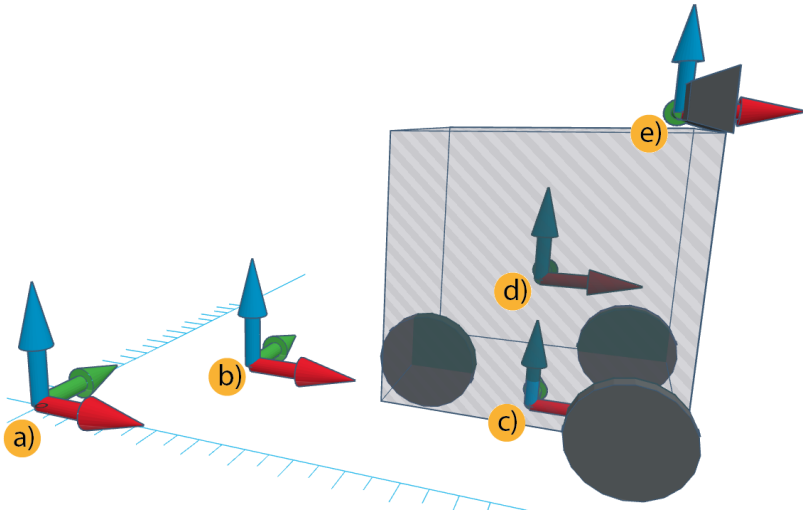
Data ze senzorů jsou vždy poloze dané polohou robota. K rozpoznání polohy dat ze senzorů se užívá transformací, přičemž jednotlivá data mají vlastní rámec. Mezi rámci jsou pak užívány transformace.[7]

4.1 Lokalizace

4.1.1 Rámce (frames)

Hlavními rámci jsou: rámec mapy, který je pevný relativně k reálnému prostředí a je nehybný, a dále rámec robota, který je přímo spojen s robotem a pohybuje se s ním. [8]

Důležitými rámci robota jsou v ROS base_footprint, který se nachází ve půdorysného průmětu robota, dále rámec base_link, který je umístěn ve středu robota a jsou na něj navázány veškerá další příslušenství s vlastními rámci, jako například senzor se svým rámcem. Význačným rámcem je rámec odometrie (odom), jemuž je nadřazený rámec mapy. Rámec odom je pohyblivý, jelikož jeho pozice v mapě je závislá na odhadu pozice robota, jak bylo popsáno u obrázku číslo 1, rámec odom se tedy může posunout v momentě kdy je robot jistě lokalizován v prostředí. [27]



Obrázek 5 Reprezentace rámců
Zdroj: vlastní zpracování

Vyobrazení rámců a) map, b) odom c) base_footprint, d) base_linkd e) base_laser

Algoritmy lokalizace mobilního robota se snaží popsat vztah mezi mapovým rámcem a rámcem robota. Další rámce jsou pak pohyblivé části robota, ať už jde například o kola, nebo úchopová zařízení (paže), přičemž každá jednotlivá hybná část má svůj rámec, který navazuje na rámec nadřazené (předcházející) hybné části.[28]

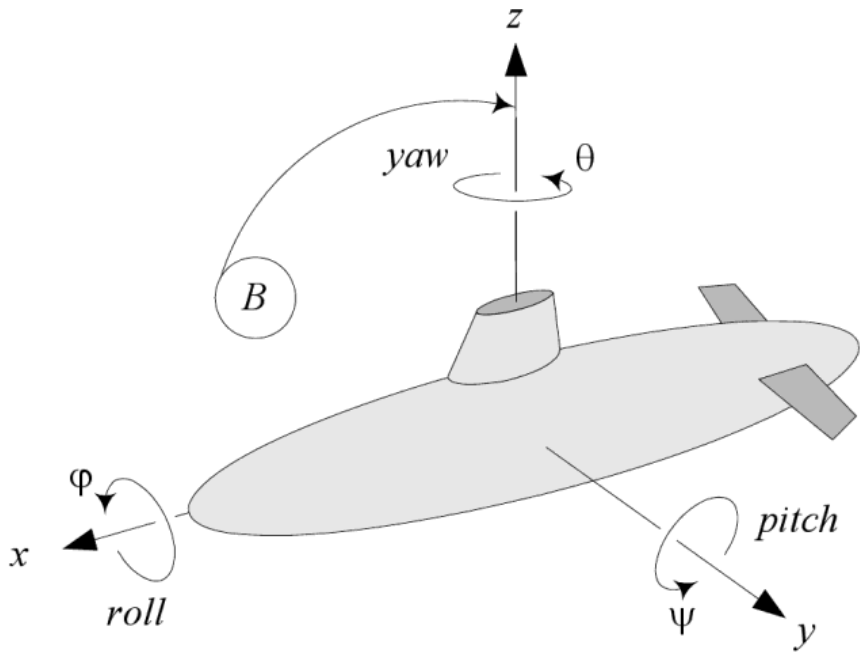
Vazby mezi jednotlivými rámci jsou obvykle známy s vysokou přesností díky enkodérům v jednotlivých vazbách (kloubech), které jsou obvykle namontovány na každém kloubu manipulátoru a přímo měří rotační polohy. Metody zjišťování polohy se liší dle typu enkodéru, mohou být magnetické, optické, odporové nebo kapacitní. Data z enkodérů jsou zpracovávána na nízké úrovni firmware a široká škála manipulátorů je obvykle schopna rozpoznat úhlové polohy jednotlivých kloubů s vysokou přesností. Tento vektor úhlů se nazývá společný stav a je zásadní pro analýzu a řízení manipulátorů robotů.[8]

Pro určení polohy jednotlivých částí robota se matematicky reprezentuje poloha a orientace samotných částí v trojrozměrném prostoru. Za tímto účelem přijímáme metodu obecně používanou v mechanice, kde každému objektu připojujeme ortogonální souřadnicový rámec a jeho polohu a orientaci vyjadřujeme podle polohy počátku a směrů tří os připojeného rámu vzhledem k dané referenci. [11]

V ROS je model robota tvořen předpisem URDF (Unified Robot Description Format), který je založen na formátu XML. V předpisu jsou definovány jednotlivé rámce, spoje mezi rámci a hierarchie rámců. Předpis umožňuje tvorbu modelu různých robotů pomocí základních těles, nebo vkládáním složitějších těles ve formě souborů STL nebo DAE.[29]

4.1.2 Póza

Mobilní robot musí znát svou polohu a směr po celou dobu navigace, čemuž u se říká lokalizace. [9] Póza robota se tedy rozumí údaj o souřadnicích v rámci mapy společně s udaným směřováním robota. U map reprezentovaných dvourozměrně je směr udán pouze rotací kolem osy Z, tedy osy kolmé k osám X a Y. Oproti tomu u mapy tří rozměrů je pro pózu robota nezbytné určení úhlů, které robot zaujímá vůči osám X, Y, Z, tyto úhly jsou nazývány anglickými termíny roll, pitch a yaw, uvedeny v příslušném pořadí, jejich vyobrazení je znázorněno na obrázku 6.[21]



Obrázek 6 Rotace v 3D prostoru
Zdroj: JAZAR, Reza [21]

4.1.3 AMCL - adaptive Monte Carlo localization

Jedná se o metodu lokalizace pro roboty aplikované pro mapování 2D prostředí. AMCL je využíván například v Gmapping k lokalizaci v předem zprostředkované

mapě. Pro určování póz robota vzhledem k dané mapě je touto metodou aplikován částicový filtr.[32]

V ROS je AMCL implementován, ale pracuje pouze s laserovými skenery, do budoucna je možné rozšíření na další typy snímačů. Vstupními daty AMCL jsou mapa založená na laseru, laserové skenování a zprávy transformací zpráv, výstupem jsou pak odhady pózy.[27]

4.1.4 Transformace

K určení polohy robota nebo jeho částí (např. úchopového zařízení) v mapě je využívána vektorová a maticová algebra pro translaci a rotaci mezi jednotlivými rámci. jednotlivé rámce mohou být odvislé od těch předcházejících, například paže robota má rámec počínající v kloubu paže, přičemž kloub samotný je situován v konkrétních souřadnicích v rámci pevné konstrukce robota. Robot samotný se pak nachází v obecném rámci mapy. [22]

Pro transformace je tedy důležitá právě póza robota, u UGV(unnamed ground vehicle) souřadnice X, Y a úhel, u UAV (unnamed aerial vehicle) a například u ponorek pak souřadnice X, Y, Z, a úhly roll, pitch a yaw, přičemž tyto souřadnice jsou udávány za pomoci quaternionu.[21]

4.2 Mapování

Pro účely mapování slouží exteroceptivní snímače, které umožňují robotu vnímat okolní prostředí. Je zřejmé, že povaha a parametry snímače značně ovlivňují kvalitu snímaných dat. Zároveň se liší prostředí, pro která je snímač více či méně vhodný.

4.2.1 Sonar

Sonar je senzor založený na principu vysílání zvukových vln a následném odposlouchávání jejich odrazů od okolí. Samotný princip předesílá jeho nižší rychlost. Jeho dalšími nevýhodami jsou vlastnosti materiálů, pro hladké materiály může být úhel odrazu příliš velký a může docházet k odrazům mimo spektrum senzoru. Mohou se vyskytnout i materiály, které zvukové vlny absorbují. Výhodou sonaru jsou oblasti se sníženou optickou viditelnost, jako jsou mlžná zakouřená či zakalená prostředí, kde také nachází nejširší využití.[5]

4.2.2 LIDAR

Jedná se o zkratku pro Light detection and ranging. Senzor podobně jako sonar vysílá signál a následně snímá odraz signálu, ovšem v tomto případě se jedná o vyzařování laserového signálu.[5]

Výhodou laserového signálu je vzdálenost snímání, která je oproti sonaru větší. Užitím světelného paprsku se také značně zvyšuje rychlost snímání, a jelikož se jedná o koncentrovaný paprsek, je i rozlišení snímače podstatně jemnější. [6]

4.2.3 Vision-based RGB-D

Senzory RGB-D, jako je Microsoft Kinect nebo Asus Xtion, Kamery RGB-D jsou snímací systémy, které snímají obraz v RGB obdobně jako běžný fotoaparát a zároveň obstarávají informace o hloubce jednotlivých pixelů. Jedná se o hloubkovou mapu, obvykle kódovanou jako obraz ve stupních šedi. Hloubkovou mapu lze reprezentovat i jako mračno bodů, tj. uspořádanou množinu 3D bodů. Zařízení kompatibilní s OpenNI jsou podporována komunitou ROS, která poskytuje stack zprostředkovávající data z RGB-D senzorů pro další použití.[30]

RGB-D senzory jsou levnější alternativou objemnějším LIDAR senzoru, ale zároveň komplexnějšími informacemi ztrácejí na přesnosti.[2]

4.2.4 OpenCV a OpenNI

OpenCV je populární open source knihovnou pro počítavé vidění v reálném čase a strojové učení, obsahuje více než 2 500 algoritmů. Knihovna je zdarma jak pro komerční, tak akademické účely. OpenCV je napojeno do ROS pomocí zásobníku vision_opencv. [29]

Knihovna OpenCV obsahuje podporu vizuálních aplikací v reálném čase, jako je sledování pohybujících se objektů nebo detekce a rozpoznávání obličejů. Příkladem dalších podporovaných algoritmů jsou identifikace objektů, sledování lidských gest a výrazů obličeje, modelování scén na základě obrazů z více zdrojů, vytváření 3D modelů objektů a další. [31]

OpenNI je framework pro zařízení pro tzv. Natural Interaction (přirozenou interakci). Jedná se o zařízení známá jako Microsoft Kinect nebo Asus Xtion Pro.

Knihovna ovladačů OpenNI pro Linux pracuje jako middleware mezi zařízením a aplikačním softwarem. [32]

Balíčky? OpenNI pro ROS pak obsahují launch soubory pro transformaci dat ze senzorů pro použití ve formě mračna bodů, hloubkové mapy apod. Dalším modulem OpenNi je Nite, který provádí sledování kostry, gesta a další. [27]

4.2.4.1 Feature detektory

Pro RGB-D senzory a kamery, díky schopnosti poskytovat RGB snímky, je dostupná škála feature detektorů. Feature detektory jsou schopny z poskytnutých snímků extrahovat určité informace, které mohou být nápomocné pro dané výpočetní úkony, ať už jde třeba o rozeznávání objektů pro lokalizaci robota v prostoru. Příklady algoritmů detektorů objektů jsou uvedeny níže. V rámci této práce tyto typy detektorů nebyly použity. [4]

Detektor	Popis
SIFT	Hodnotí k lokalizaci klíčových bodů rozdíl Gaussiánských funkcí aplikovaných pomocí scale-space na řadě obrázků.
SURF	Je založen na součtech výsledků 2D Haar vlnek a efektivně využívá integrální obrazy.
SIFTGPU	Stejné jako SIFT, ovšem implementován na GPU
FAST	Detekuje a analyzuje přilehlé pixely klíčových pixelů.
ORB	Počítá binární řetězce z obrazových patchů a SIFT.

Tabulka 1 Feature detektory
Zdroj: Francisco Ferrer [23]

5 Použité prostředky

V rámci této práce byly pro implementaci SLAM použity mobilní robot Turtlebot 2 společně s počítačem Acer Aspire v5, dále vzdálený počítač HP4530s s operačním systémem Linux Ubuntu s middleware ROS.

5.1 Robot Operating System (ROS)

Robot Operating System (ROS) je open source, flexibilní softwarový framework pro programování robotů. Díky hardwarové abstrakční vrstvě poskytuje vývojářům vytvářet robotické aplikace i za použití základního hardware. ROS také poskytuje různé softwarové nástroje pro vizualizaci a ladění robotických dat, jako jsou Rviz, Gazebo. Architektura ROS spočívá v balíčcích (package), čímž nabízí dobrou modularitu a opakovatelnost. Významnými balíčky jsou právě balíčky pro SLAM a AMCL (Adaptive Monte Carlo Localization). Tento middleware je také vybaven podporou pro širokou škálu senzorů, ať už se jedná o LIDAR, Sonar, či hloubkové snímače, jako je Kinect. Toto jsou hlavní výhody oproti jiným platformám např. Player, YARP, Orocos, MRPT. Ovšem ROS samotný je značně rozsáhlý a je tak složitější na učení, dále je v ROS složitější práce se simulačním software a vytvářením vlastních modelů robotů (URDF).[31]

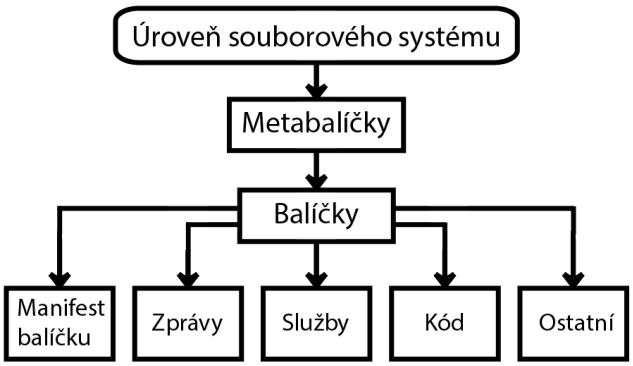
5.2 Architektura ROS

ROS poskytuje krom hardwarové abstrakce, knihoven, balíčky, uzly i systém zasílání zpráv. Ros poskytuje značnou modularitu díky architektuře založené na uzlech. Tato architektura má výhodu v tom, že pokud některý z uzlů nepracuje správně, ostatní uzly mohou dále nezávisle pracovat a nemusí dojít k pádu celého systému. ROS zprostředkovává i robustní metody k obnově funkčnosti v případech, kdy je některá součást robota nefunkční.[29]

Třídění do balíčků umožňuje dělit projekty dle užití, smyslem je udržovat jednotlivé balíčky menšího rozsahu, poskytnout užitečnou funkci a zaměřit se na jejich znovupoužívání. Balíčky mohou obsahovat uzly ROS, knihovny nezávislé na ROS, konfigurační soubory, a jiné užitečné moduly. Na obrázku číslo 7 je vyobrazen souborový systém ROS včetně uspořádání balíčků. Při tvorbě vlastních balíčků je žádoucí strukturu dodržovat, případné odchylky mohou negativně ovlivnit

funkcionalitu balíčku. ROS dokáže v případě spuštění balíčku se špatnou souborovou strukturou ve vážných případech na tento fakt upozornit.

Součástí balíčku je manifest balíčku (package manifest) obsahující informace o balíčku, včetně jeho názvu, verze, popisu, licenčních informací, dependecies apod. [32]



Obrázek 7 Souborová struktura ROS
Zdroj: přepracováno na základě předlohy - LENTIN, Joseph [32]

5.2.1 Nodes

ROS Nodes (uzly) jsou procesy, které provádí výpočty pomocí klientských knihoven ROS, jako jsou například roscpp a rospy (knihovny pro C++ a Python). Jeden uzel může komunikovat s ostatními uzly pomocí témat (topic) ROS, služeb a parametrů ROS.

Robot může využívat vícero uzlů, například jeden uzel zpracovává obrazy z kamer, jiný uzel zpracovává sériová data z robota, další uzel lze použít k výpočtu odometrie atd. Pro chod robota je mnohdy nutný právě provoz několika uzlů zároveň. [27]

Spouštět uzly jeden po druhém za pomocí *rosrun* je vhodné pro testování a ladění, ale většina systémů ROS obsahuje mnoho uzlů a není žádoucí je jednotlivě spouštět a zastavovat. Příkaz *roslaunch* nabízí možnost spouštět více uzlů najednou, a to i s případnými argumenty. *Roslaunch* je nástroj, který načítá XML popis sady uzlů, poté tyto uzly spouští a monitoruje. Toto je výhodné zejména u složitějších systému využívajících větší počet uzlů. Podle konvencí, mají soubory XML Roslaunch příponu *.launch* a nazývají se "launch files:".[8]

5.2.2 Topics - témata

Témata zprostředkovávají komunikaci mezi uzly, které mohou data, jak publikovat, tak zároveň datům naslouchat. Uzel publikující data může být odposloucháván jedním i více uzly a témata slouží k tomu, aby odposlouchávané informace a ty publikované byly daného formátu (typu). Tedy aby uzly byly schopné se dorozumět.[27]

5.2.3 Zprávy

ROS využívá systém zpráv pro zajištění komunikace mezi jednotlivými uzly. Zprávy jsou v ros definovány předpisem, tak aby bylo zajištěno, že odběratelský i vysílací uzel pracovávají data o stejném formátu. K zajištění přenosu zprávy je tedy nutné dodržet souhlasný typ zprávy a typ tématu. ROS obsahuje již předdefinované typy zpráv, ale lze vytvářet zprávy vlastní.
Zpráva by měla mít dvě části, a to pole (nezaměňovat s datovým typem pole – array) a konstanty. Pole definují typ dat, která budou přenášena jako int32, float32 nebo string a konstanty určují jména pro příslušná pole.[33]

5.3 Simulační prostředí Gazebo a vizualizace RViz

ROS je vybaven simulačním 3D prostředím Gazebo, které je určeno k simulaci prostředí, robotů, senzorů i dalších objektů. Pro simulaci lze vytvořit věrnou podobu reálných prostředí, přičemž Gazebo poskytuje nejen zpětnou vazbu snímačů, ale zároveň fyzické interakce s okolními předměty.[33]
Zatímco Gazebo simuluje reálné prostředí, nástroj RViz vizualizuje ROS zprávy (kapitola 5.2.3.). RViz je tedy užitečným nástrojem k ladění, kdy lze vizuálně ověřovat data zprostředkovaný zprávami. Zprávy mohou být jak reálného charakteru, tak i simulované.[34]

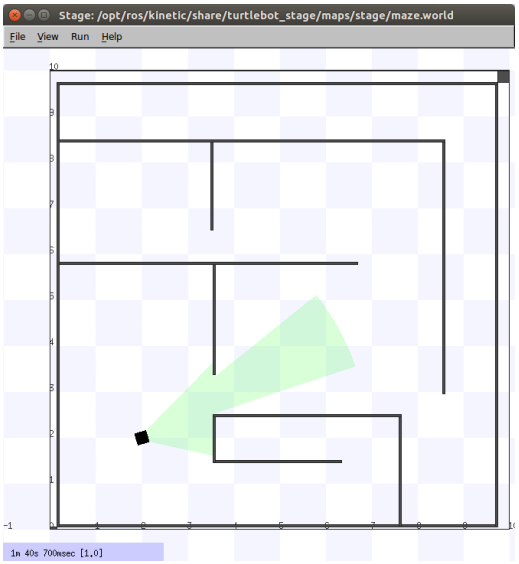
5.3.1 Simulační prostředí Stage robot simulator

Stage simulátor je dalším, jednodušším, simulačním prostředím, které simuluje Turtlebota ve dvourozměrných prostředích definovaných bitmapami. V simulaci lze zobrazit skenovací rozsah RGB-D snímače lze tak snadněji posuzovat reakce robota na dané prostředí.

Simulační prostředí Stage lze otevřít zadáním příkazu:

```
roslaunch turtlebot_stage turtlebot_in_stage.launch
```

Spuštěním tohoto launch souboru se otevře simulační prostředí Stage a současně i prostředí Rviz.[27]



Obrázek 8 Prostředí simulátoru Stage
Zdroj: vlastní zpracování

5.4 Navigace v ROS

Jakmile je robot úspěšně lokalizován, tedy má povědomí o své póze v prostředí a je schopen mapovat okolní prostředí, tedy jej vnímat, jsou splněny podmínky pro možné navigování robota v prostoru. Pro navigaci jsou stěžejní tři hlavní balíčky: move_base, gmapping (použito v této práci) a AMCL. Balíček move_base uvažuje pohyb robota mapou k cílovému bodu. Gmapping vytváří mapu na základě dat ze senzorů a AMCL je zodpovědný za lokalizaci robota v dané mapě.

5.4.1 Navigační zásobník

Autonomní mobilní robot se může pomocí svých mapovacích a lokalizačních algoritmů autonomně pohybovat ze své aktuální polohy do cílové pozice. ROS poskytuje některé výkonné balíčky pro autonomní roboty, jako jsou navigační zásobník ROS, mapování a AMCL. Navigační zásobník neboli *navigation stack*, řeší problematiku, jako vyhýbaní se překážkám, plánování tras nebo případné chování

v případech nemožnosti pohybu pro přílišnou blízkost překážek. To vše díky informací z odometrie, dat ze senzorů a příkazům k pohybu robota.
Dle jednotlivých konstrukčních specifik robota je podstatné určitým způsobem změnit konfiguraci parametrů zásobníku.
Pro správnou funkci navigačního zásobníku je nutné mít přesně nastavený strom transformací a také korektní publikaci dat pomocí zpráv. Nastavení zásobníku dále musí sledovat tvar a dynamiku robota jako takového.[34]

5.4.2 Balíčky costmap_2d a move_base

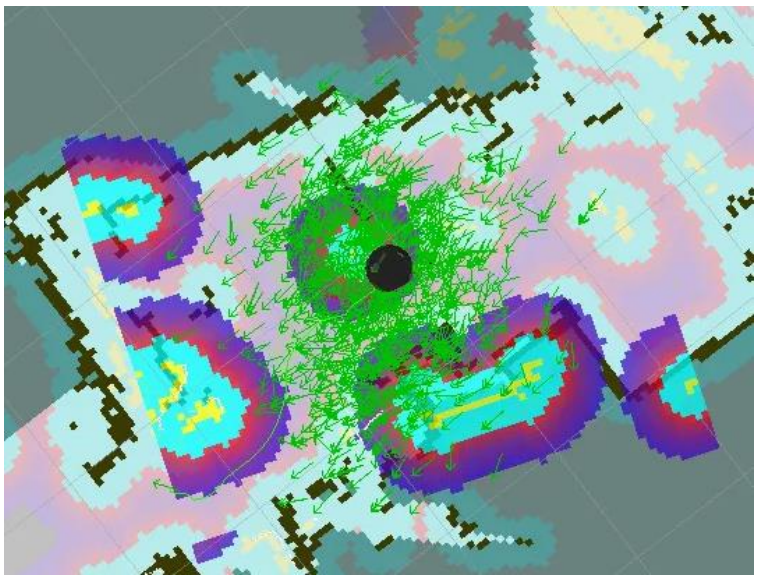
Na základě dat ze snímačů vytváří balíček costmap mapu s překážkami, přičemž překážky v mapě obalí uživatelem definovaným rádiusem, kterým je následně definováno, jak blízko předmětu může robot být, jedná se o definování bezpečné vzdálenosti pro pohyb.
Balíček move_base je jedním z nejstěžejnějších komponent navigation stack a je implementací funkce, kdy je zadáním navigačního cíle proveden pokus o dosažení daného cíle robotickou základnou. Move_base k dosažení cíle udržuje informaci ze dvou costmap, jednu globální k naplánování trasy v celkovém měřítku a druhou lokální pro reakce na přilehlé překážky. Tato funkcionalita je podpořena kooperací nejen těchto dvou balíčků, ale také AMCL a globálních a lokálních plánovačů tras. Takto je dosaženo efektivního navigování prostředím v navigačních aplikaci s předem zprostředkovanou mapou.[8]

5.4.3 Navigace založené na mapě

V případech, kdy máme k dispozici mapu prostředí (YAML a PGM soubory), lze robotu určit souřadnice a směr, tedy pózu, v mapě a díky trasovým plánovačům může robot dosáhnout zadané pozice.
Pro vizualizovaní takovéto navigace spustíme uzel balíčku, následující m příkazem:

```
roslaunch turtlebot_navigation turtlebot_navigation.launch
map_file:=/tmp/my_map.yaml
```

Tímto se spustí navigační balíček společně s vizualizačním programem RViz. Robot se v předložené mapě objeví na inicializační poloze 0,0. Pro navigaci je nutné robota určit přibližnou reálnou polohu v mapě za pomoci 2D pose estimate (určujeme souřadnice a směr).



Obrázek 9 Zobrazení překážek v Rviz
Zdroj: vlastní zpracování

Ve vizualizaci lze vidět reprezentaci dat ze snímačů společně s heatmapou zobrazujícím blízké překážky, o což se stará Local path planner. Modrá barva naznačuje prostor, který je dosažitelný a červená barva oblasti příliš blízko překážek.

Následně lze pomocí 2D Nav Goal zadat pózu v mapě, na kterou se robot pokusí navigovat. V RViz se, pakliže je možné trasu realizovat, vykreslí trasa zelnou barvou, ta je vypočítána za pomoci Global Path planner. Během průjezdu robota prostředím se navrhovaná trasa mění na základě dat z Local Path Planneru, takto se vyznačuje spolupráce Local a Global Path plannerů.[35]

5.4.4 Reaktivní navigace

Local Path banner během pojezdu prostředím neustále sbírá data ze senzorů (informace o prostředí). V případech, kdy se v prostoru nachází překážka, je robot díky navigačnímu zásobníku vrácen o kus zpět a pokusí se přiblížit k místu znovu. Robot se v některých případech může dostat i do situací, kdy sám nemůže přiblížení ihned zopakovat, což uvede robot do tzv. recovery módu. V tomto módu Turtelbot

rotuje kolem své osy a rozhlíží se senzory kolem sebe a snaží se v okolí najít trasu, která by splnila nároky na vyhýbaní se překážkám. Toto chování značně závisí na konstrukci robota a jeho vybavenosti příslušnými senzory.[32]

5.5 Vzdálený přístup

Pro ovládání robota lze využívat vzdáleného přístupu. Počítač řídící robota a vzdálený počítač by pro takovéto využití měli být ve stejné síti. Následně je nutné znát IP adresy obou zařízení a tyto příslušně nastavit v systémových proměnných prostředí ROS v souboru *.bashrc*.

	Turtlebot PC	Vzdálené PC
ROS_MASTER_URI	172.16.254.1:11311	172.16.254.1:11311
ROS_IP	172.16.254.1	172.16.254.2
ROS_HOSTNAME	172.16.254.1	172.16.254.2

Tabulka 1 Příklad nastavení systémových proměnných.

Zdroj: vlastní zpracování

Je dobrou zvyklostí tyto proměnné nechat vypsat při otevření nového okna terminálu.

Při takovémto nastavení lze pak některé aplikace, jako je třeba RViz nebo teleop spouštět na vzdáleném počítači a tím tedy výpočetně nezatěžovat počítač řídící robot.

5.5.1 Ovládání robota

Jak bylo popsáno v kapitole 5.3.2 lze robota ovládat i pomocí zadávání navigačních cílů v RViz. V tomto případě není předem dána trasa, kterou robot bude projíždět, ta je vypočítána až po zadání navigačního cíle a je v průběhu průjezdu plánovanou trasou upravována. RViz dále nabízí možnost ovládání robota pomocí interactive markers, kterými lze ovládat natočení a směr pojezdu robota – obrázek číslo 10. Pro použití interactive markers je nutná předchozí instalace:

```
sudo apt-get install ros-indigo-turtlebot-apps ros-[xxx]-turtlebot-interactions
sudo apt-get install ros-indigo-turtlebot-interactive-markers
```

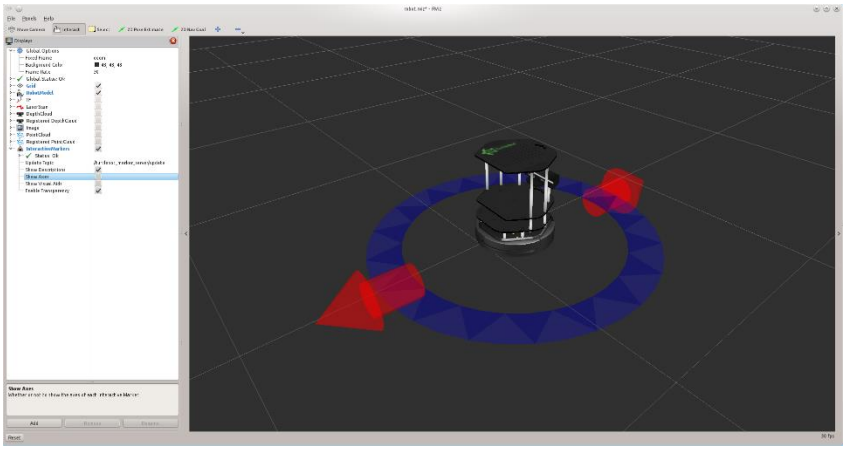
([xxx] je označení distribuce ROS např.“kinetic“)

Následně v jednom okně terminálu spustit turtlebot _markers_server:

```
roslaunch turtlebot_interactive_markers interactive_markers.launch –screen
```

Poté v novém okně terminálu spustit Rviz se zobrazováním interaktivních markerů:

```
roslaunch turtlebot_rviz_launchers view_robot.launch
```



Obrázek 10 Ovládání za pomoci interactive markers

Dalším způsobem je vzdálené řízení robota za pomoci balíčku teleop, který umožňuje provádět ovládání ze vzdáleného počítače prostřednictvím klávesnice,

```
roslaunch turtlebot_teleop keyboard_teleop.launch
```

nebo umožňuje použít k ovládání robota i herních ovladačů různých výrobců. Herní ovladač Playstation 3:

```
roslaunch turtlebot_teleop ps3_teleop.launch
```

Herní ovladač Xbox:

```
roslaunch turtlebot_teleop xbox360_teleop.launch
```

Robot lze ovládat také pomocí aplikace ROS Control pro systém Android, v současné době již není tato aplikace udržována. [35][27]

5.6 Mapování do souboru

ROS umožňuje zaznamenávat zprávy z jednotlivých témat. Takto lze vytvořit i bag soubor, ze kterého lze následně vytvořit mapu offline. Takovýto přístup lze vhodně využít při testování různých nastavení parametrů, kdy lze na stejná data aplikovat různé přístupy, a tak najít nejvhodnější nastavení parametrů. [29]

Při zaznamenávání je věcné přímo určit témata, která chceme zaznamenávat. samozřejmě lze zaznamenat témata všechna, ale nemusí to být pro množství témat vhodné. Při použití senzoru kinect je potřeba spustit uzel, který zprostředkovává data ze senzoru:

```
roslaunch freenect_launch freenect.launch
```

Dále v dalším okně terminálu spustit uzel pro převod hloubkového obrazu na laserový scan:

```
rosrun depthimage_to_laserscan image:=/camera/depth/image_raw
```

Pokud během záznamu do .bag souboru je použit i SLAM Gmapping není nutné spouštět tyto dva předchozí uzly, o jejich spuštění se postará launch soubor GMappingu.

Příklad příkazu pro záznam témat scan a tf s uložením do souboru data.bag:

```
rosbag record /scan /tf -O data.bag
```

Téma tf poskytuje souřadnice několika rámců v průběhu času, přičemž téma scan zprostředkovává převod hloubkového obrazu na laserový scan. Po dokončení mapování stačí ukončit program, kde je nahrávání spuštěno.

Při náběru bag souboru je nutné brát v potaz jaký systém bude následně bag soubor vyhodnocovat, například Google Cartographer nezapočítává do odhadu mapy prostředí odometrii, ale jsou v tomto systému podstatná data třeba z inercial measurement unit. Je tedy podstatné, data z příslušných témat zahrnout do bag souboru.

5.6.1 Přehrávání .bag souboru

Předem vytvořené soubory typu .bag lze následně přehrát, přičemž lze modifikovat parametry jako je rychlost přehrávání nebo odkud přehrávání začít, což je užitečné pro debugging, nebo situace kdy není k dispozici robot.
Pro přehrání je nutné vynulovat čas v ROS, čehož lze dosáhnout příkazem

```
rosparam set use_sim_time true
```

Následně lze spustit požadované uzly, které budou odebírat zprávy z bag souboru, jako například mapování.

```
rosrun gmapping slam_gmapping scan:=scan
```

Poté zprostředkujeme očekávaná data přehráním bag souboru

```
rosbag play --clock data.bag
```

Následně lze pracovat obdobně jako při reálném provozu robota, systém bude publikovat témata zaznamenaná v .bag souboru.[8]

5.7 Hardware

K implementaci řešení byla v této práci použita kombinace mobilní robotické platformy se senzory a řídících počítačů. Implementace je značně závislá na použití těchto prostředků, zejména z hlediska kvality poskytovaných dat senzorů, výpočetního výkonu počítačů a také konstrukční schopnosti robotické platformy, což vyplývá z popisů metod řešení SLAM v kapitole 3.

5.7.1 TurtleBot

TurtleBot je cenově dostupná vývojářská sada robota s open source softwarem. TurtleBot sestává z mobilní základny, 2D/3D senzoru vzdálenosti, přenosného počítače nebo Single board computer (SBC). Turtlebot je navržen tak, aby byl snadno

smontovatelný, dostupný, a zároveň využitelný i s volně dostupnými díly, či díly, které lze snadno vytvořit z běžně dostupných materiálů.

Obrázek 11 Turtlebot 2
Zdroj: www.turtlebot.com [36]

ve své nejzákladnější formě skládá z Kobuki základny, na které je přidělaný systém polic. K plné funkčnosti TurtleBota je potřeba do základny připojit řídící počítač a kameru pro snímání okolí. Pokud není uvedeno, jinak v textu je TurtleBot vždy myšlen jako celkový systém tzn. včetně řídícího počítače a kamery.

5.7.1.1 Kobuki základna

Kobuki základna je hlavní částí Turtlebota, na kterou lze případně připevnit systém polic pro další zařízení. Základna je vybavena aktuátory umožňující pohyb robota, jedná se o dvě profilovaná kola o průměru 76 mm.

5.7.1.2 Senzory

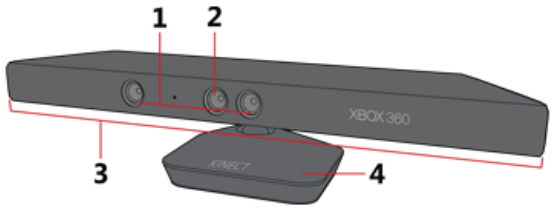
Základna je dále vybavena senzory pro zajištění detekce kolizí a polohy zařízení. Enkodéry v kolech základny poskytují informace o odometrii, gyroskopickým senzorem a dalšími pomocnými senzory.

Enkodéry v kolech	25700 cps 11.5 ticks/mm
Gyrosskopický senor	110 deg/s
Pomocné senzory	3x nárazu, 3x převisu, 2x propadu kola

Tabulka 2 Parametry Turtlebot 2.
Zdroj: www.turtlebot.com [36]

5.7.2 Kinect

Stěžejním prvkem Turtlebot pro SLAM je RGB-D snímač, často používaným je Kinect V1. Kinect obsahuje různé snímací technologie, jako jsou hloubkový senzor, čtyři mikrofony a barevná kamera. Díky těmto snímačům je Kinect schopen rozpoznat pohyb, obličeje, a díky analýze zvukového signálu také rozeznat hlas. Rozmístění jednotlivých senzorů je zřejmé z obrázku níže. Hloubkový senzor se skládá z IR projektoru (na obrázku položka č. 1 vlevo) a IR kamery (na obrázku položka č. 1 vpravo). [37]



Obrázek 12 Rozmístění jednotlivých snímačů na Microsoft Kinect
Zdroj: Ghazalbash Somayeh

1. 3D Hloubkový senzor (IR projector + IR Kamera / Hloubkový senzor)
2. RGB kamera
3. Řada mikrofonů
4. Motor náklonu

5.7.3 Řídící počítače

V rámci práce byl jako vzdálený počítač (Workstation) použit přenosný počítač HP ProBook 4530s
Procesor: Intel(R) Core(TM) i5-2430M CPU @ 2.40GHz 2.40 GHz
Nainstalovaná paměť RAM: 8.00 GB,
Grafická karta: Intel HD Graphics 3000

Jako počítač řídící Turtlebot 2 byl použit přenosný počítač Acer Aspire V5-122P
Procesor: AMD A6-1450M 1GHz
Nainstalovaná paměť RAM: 6.00 GB,

5.7.4 Ultrazvukový senzor

Kvůli jednomu z nedostatků senzoru Kinect, neschopnost snímat lesklé nebo průhledné povrchy, je zároveň implementován ultrazvukový senzor HY-SRF05. Tento senzor je řízen mikrokontrolerem Arduino UNO, který nejen obsluhuje snímač, ale zároveň předává data do ROS. Ultrazvukový senzor je napájen z mikrokontroleru, dále mikrokontroler vysílá opakovaně signál senzoru prostřednictvím pinu číslo 9 (trigger – trigPin) a čte odražený signál, který je snímán senzorem, na pinu číslo 10 (echo – echoPin). Následně je na základě rozdílu času vyslání signálu a času odposlechu jeho odrazu s použitím konstanty pro rychlost zvuku vypočítána vzdálenost od překážky. Komunikace mezi mikrokontrolerem a ROS je provedena přes metapackage ROS Serial. Mikrokontorler je k počítači řídící Turtlebot připojen prostřednictvím USB portu, čímž je zároveň i mikrokontroler napájen. Na mikrokontroleru běží program, který přes ROS seriál zasílá do ROS zprávy typu sensor_msgs/Range Message.

6 Popis řešení

Z rešerše dokumentace ROS vyplývá, že zvolené prostředky jsou pro daný účel v rámci middleware ROS podporovány. K jejich využití byla nutná instalace jednotlivých součástí.

Počítač Acer Aspire v5 byl již předinstalován s verzí ROS Indigo Igloo. Instalace ROS na počítač HP ProBook 4530s byla provedena na základě podporovaného hardware, bohužel nebylo možné použít stejnou verzi ROS nejaktuálnější distribuci ROS (Melodic Morenia - 2018), jelikož ta podporuje pouze procesory architektury AMD a ARM. Verze Indigo Igloo již není dlouhodobě podporována, byla tedy použita distribuce Kinetic Kame 2016.

6.1 Instalace jednotlivých součástí

Instalace ROS je provedena na již předinstalovaný operační systém Ubuntu Linux Xenial Xerus. Během instalace je nutné brát na zřetel kombinaci systému Ubuntu a danou verzi ROS.

Pro pohodlnost ovládání a instalaci robota bylo použito nastavení komunikace mezi řídícím počítačem robota a pracovní stanicí (řádné nastavení proměnných prostředí) a další kroky byly prováděny za pomoci SSH.

Pokud není již v ROS předinstalovaný Turtlebot2 lze snadno nainstalovat nutné balíčky pro jeho provoz příkazem:

```
sudo apt-get install ros-[xxx]-turtlebot-*
```

([xxx] je označení distribuce ROS např.“kinetic“)

6.1.1 Instalace snímače Microsoft Kinect

ROS verze Indigo Igloo a předcházející měly jako výchozí RGB-D snímač určený Microsoft Kinect, pozdější distribuce již měly přednastavený Asus Live Pro. Aby tedy bylo možné použít snímač Microsoft Kinect, bylo nejprve nutné na vzdálený počítač s ROS Kinetic nainstalovat balíčky OpenNi a moduly pro senzor. Balíčky jsou dostupné v rámci repozitáře:

```
git clone https://github.com/OpenNI/OpenNI.git
```

Poté je třeba odstranit hlavičku Unstable, udělit přístupová práva pro spustitelnost souboru RedistMaker a následně soubor spustit

```
cd OpenNI
git checkout Unstable-1.5.4.0
cd Platform/Linux/CreateRedist
chmod +x RedistMaker
./RedistMaker
```

RedistMaker script provede kompilaci, pak je potřeba složku Redist a dle architektury procesoru a verze OpenNI najít a spustit instalační script.

```
cd ../Redist/OpenNI-Bin-Dev-Linux-[xxx]
sudo ./install.sh
```

([xxx] je označení architektury procesoru a číslo vydání OpenNi)

Následně nainstalovat moduly pro Microsoft Kinect senzor. Naklonováním repozitáře.

```
git clone https://github.com/avin2/SensorKinect
cd SensorKinect
```

Po otevření složky SensorKinect, instalace pokračuje podobně jako u OpeNI.

```
cd Platform/Linux/CreateRedist
chmod +x RedistMaker
./RedistMaker
chmod +x install.sh
./RedistMaker
sudo ./install.sh
```

Následovala příprava pracovního prostoru Catkin.

6.1.2 Pracovní prostor

Pro vytváření vlastních programů v rámci ROS je vhodné vytvořit vlastní pracovní prostor Catkin. Nový pracovní prostor byl vytvořen, následujícími příkazy.

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
```

Následně bylo nutné uvést nový pracovní prostor jako zdroj pro spouštění kódu.

```
source devel/setup.bash
```

Aby se zajistilo spouštění kódů z nového pracovního prostoru je dobré uvést zdrojovou složku kódu jako systémovou proměnou v souboru *.bashrc.*

6.1.3 Instalace ultrazvukového snímače

Ultrazvukový snímač je připojen k řídícímu počítači přes mikrokontroler Arduino a komunikace je prováděna prostřednictvím rosserial. Pro použití mikrokontroleru společně s ROS je nutná předchozí instalace IDE pro Arduino (běžná instalace) a instalace metabalíčku rosserial:

```
sudo apt-get install ros-kinetic-rosserial
cd src                                          *předpoklad že stále nacházíme ve složce catkin_ws
git clone https://github.com/ros-drivers/rosserial.git
cd ..
catkin_make
catkin_make install
```

Rossrial nabízí službu pro širokou škálu zařízení, proto je nutné dále nainstalovat klientské knihovny pro Arduino.

```
sudo apt-get install ros-kinetic-rosserial-arduino
```

Následně je nutné nainstalovat Arduino IDE knihovny pro ROS.

```
cd  arduino-[xxx]/libraries
sudo ./install.sh
```

([xxx] je označení verze Arduino IDE)

V tomto stádiu instalace je potřeba aby byl spuštěný ekosystém ROS, toho lze docílit spuštěním příkazu roscore v dalším okně terminálu (nutné ponechat spuštěný).
V původním okně terminálu spustíme skript pro vytvoření knihoven Arduino pro ROS.

```
rosrun rosserial_arduino make_libraries.py .
```

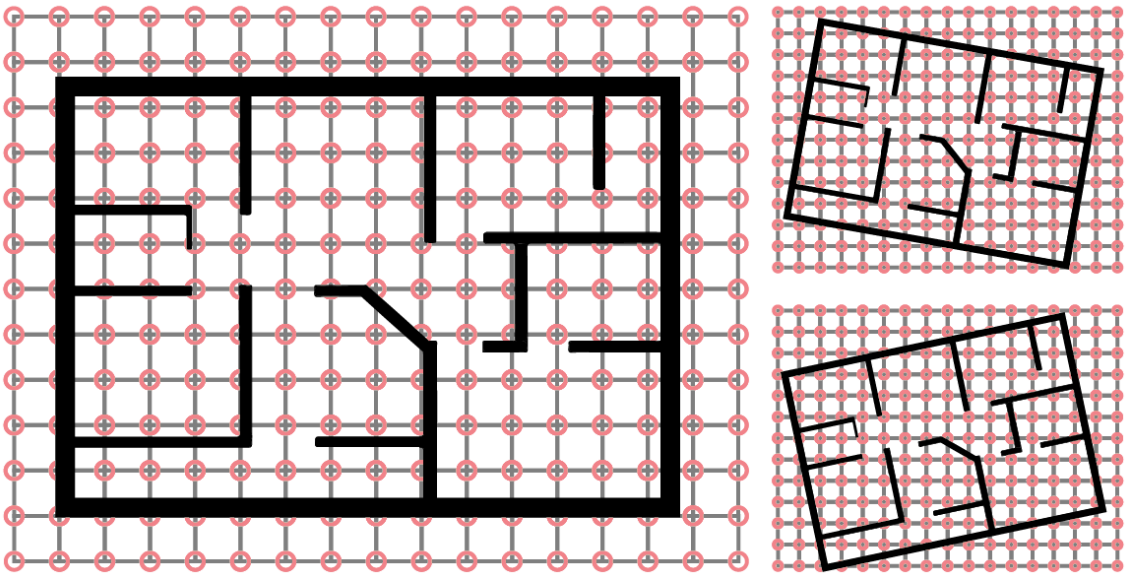
pozn. Na konci příkazu pro spuštění skriptu je symbol tečky specifikující destinaci pro instalaci knihoven (aktuální složka)

Tímto byly nainstalovány potřebné komponenty a sestava připravena pro použití pro SLAM.

6.2 Postup řešení

Uvažujme ortogonální dvourozměrnou síť bodů pro sběr vzorků (dále jen síť). Robot by měl být schopen autonomně projet dosažitelnými body v síti a v nich provézt měření vzorků z bezdrátových síti. Na obrázku číslo 13 je vyobrazena síť

v překryvu s půdorysným plánem budovy (plán), vlevo ideální situace, kdy jsou síť i zdi paralelní, ovšem lze předpokládat, že se situace bude blížit nákresům vpravo.



Obrázek 13 Vyobrazení rastru bodů
Zdroj: vlastní zpracování

Robot se na počátku měření může vyskytovat, kdekoliv v příkladovém plánu (není předem dána jeho pozice). Následný sběr vzorku by měl probíhat, tak že robot navštíví všechny body, jichž lze fyzicky dosáhnout, přičemž se efektivně vyhýbá překážkám. V každém z bodů zastaví a provede měření nebo k němu dá podnět. Měření může probíhat za pomoci externího zařízení na palubě robota.

Aby síť pokryla všechny možné body plánu, předpokládáme síť o neznámé velikosti. Síť je průběžně tvořena pojezdem robota jednotlivými body, které jsou postupně přidávány podobně jako v semínkovém vyplňování v rastrové grafice (floodfill algorithm). Algoritmus je navržen s využitím zásobníku, proti případnému přetečení a osmisměrným přidáváním bodů. Jsou udány podmínky hlídající již navštívené místo v síti a dále místo, které není fyzicky dosažitelné.

Sběr vzorků a procházení bodů v síti je zprostředkováno pomocí hashmapy s klíči definující souřadnice bodu v síti.

6.2.1 Řešení nezávislé na SLAM algoritmech

Aby robot byl co nejvíce autonomní, nabízí se řešení nezávisle na SLAM algoritmech. Byly definovány metody pro pohyb robota, zpracování signálů ze senzoru Kinect a plánování tras.

6.2.1.1Pohyb robota

Nejprve byl pro řešení vytvořen program provádějící pohyb robota, k tomu jsou vytvořeny subscriber pro zjišťování póz robota a Publisher k zadávání příkazů k pohybu.

- Subscriber topic: /odom
- Publisher: cmd_vel_mux/input/teleop

Odom topic poskytuje informaci o póze Turtlebota v Odom_frame (rámec odometrie), souřadnice jsou odvislé pouze od rámce mapy a nulový bod se nachází v místě, kde je robot spuštěn. Pro přesnost pozice robota se použije dále tf topic pro transformaci mezi odom frame a base footprint.
Publikování příkazů k pohybu je prováděno prostřednictvím určování rychlostí zprávami typu geometry_msgs/Twist. Twist message vyjadřuje rychlost rozdělenou na lineární a úhlovou část (Vector3 linear, Vector3 angular).
Metoda pro dopředný pohyb robota spočívá v zjištění aktuální pozice robota z odom topic a následně zasíláním příkazu pro rychlost pohybovat robotem a za pomoci výpočtů Eukleidovské metriky zjišťovat již ujetou vzdálenost. Toto je prováděno opakovaně do té doby, kdy je vzdálenost požadovaná menší nebo rovna té kterou robot ujel (ukázka č. 14).

```
while(True){
  publish(velocity_message);
  distance_moved = 0.5 * math.sqrt(((x-x0) ** 2) + ((y-y0) ** 2));
  if(distance <= distance_moved) {
   break;
  }
```

Obrázek 14 Pseudokód dopředného pohybu
Zdroj: vlastní zpracování

Podobně je řešena rotace robota, ovšem zde je podstatný úhel natočení, který je v póze vyjádřen za pomoci quaternionu. Je tedy nutné napřed quaterninon převézt na hodnoty *roll*, *pitch* a *yaw*. V případě úhlu natočení (směřování robota) ve dvourozměrném mapování bereme v potaz pouze úhel yaw (kapitola 4.1.2).
Zprostředkování dat o póze robota je provedeno za pomoci callback funkce, která zároveň transformuje pózu z odom frame do base footprint.

6.2.1.2 Zpracování dat z tématu scan

Zpracování dat z tématu scan probíhá za pomoci callback funkce. Téma scan je v případě senzoru Kinect převedeno z tématu depth_image (hloubkový obraz) za pomoci uzlu depth_image_to_laser_scan. Callback funkce zpracovává data ze senzoru, tak že na základě informací z tématu scan zjistí rozsahy pozorovacích úhlů, přírůstek úhlu mezi jednotlivými hodnotami, pole hodnot a dále jednotlivé hodnoty zpracovává v rozsahu daném senzorem. Každá hodnota v poli znázorňuje snímanou vzdálenost objektu.
V případě snímače Microsoft Kinect jde o pole v rozsahu 1.02121603489 rad s přírůstkem 0,00159814720973 rad, tedy délce 640 hodnot v zorném úhlu 58°. Tato data lze pak různými způsoby vyhodnocovat a zjišťovat tak přítomnost objektů v celém rozsahu pozorovacího úhlu a zároveň určovat jejich vzdálenost od robota.
V uváděném řešení je celkový rozsah rozdělen na čtyři části - levá, levá středová, pravá středová a pravá. V každé z částí rozsahu je hodnocena nejbližší překážka (nejnižší hodnota v poli) a na základě indexu daného pole se hodnotí blízkost ke středu robota. Levá a pravá část rozsahu slouží k identifikaci překážek při bocích robota a tyto informace jsou vyhodnocovány při dopředném pohybu robota, tak že se robot mírně odvrací od překážky, přičemž si stále zachovává dopředný směr pohybu. V případě identifikace překážky ve středových částích rozsahu robot přestane vykonávat dopředný pohyb a na základě informace o blízkosti ke středu robota je přikročeno k rotaci v opačném směru, než je identifikována překážka. Pokud je překážka v pohledu situována tak že nelze rozhodnout o směru rotace, robot poodjede vzad a otočí se do směru s nejnižší hodnotou ze senzoru. Jednotlivé reakce na překážky byly následně zahrnuty do metody pro pohyb robota vpřed.

6.2.1.3 Vyhýbání se překážkám

V rámci pohybu robota je důležité, aby robot byl schopen rozeznat prostředí a zároveň patřičně reagovat. Při použití Gmapping je nutné robota nejprve ručně lokalizovat za pomoci RViz (kapitola 5.4.3). Poté lze následně využívat schopnost vyhýbání se překážkám. Takovéto řešení vyžaduje mapu prostředí, kterou je nutno předem vytvořit. Takovýmto způsobem jsou pak využívány algoritmy global a local path plannerů.

Bez nutnosti předchozí mapy jsou použity metody vyhodnocování dat z tématu scan v předchozí kapitole a zároveň data z ultrazvukového snímače.

6.2.1.4 Funkce řešení nezávislého na SLAM algoritmech

Po spuštění Turtlebota se spustí uzly pro zprovoznění základní funkcionality robota:

```
roslaunch turtlebot_bringup minimal.launch
```

Úspěšné spuštění je ohlášeno zvukovým signálem Turtlebota. Následně lze spustit libovolnou metodu pro SLAM k vytváření mapy, například Gmapping:

```
roslaunch turtlebot_navigation gmapping_demo.launch
```

A spustit script vytvořený pro sběr vzorků:

```
rosrun uhk_fp fc.py
```

Anebo pomocí vytvořeného launch souboru:

```
roslaunch uhk_fp uhk_free_move.launch
```

Po dokončení sběru vzorků je v terminálu zobrazeno: „Fingerprint collection finished“. V tuto chvíli lze uložit mapu zprostředkovanou zvolenou mapovací metodou (Gmapping):

```
rosrun map_server map_saver -f /tmp/my_map
```

Za parametr -f se zapisuje cesta a název souboru, kam se má mapa uložit. Uložením je vytvořen soubor obsahující vizuální reprezentaci mapy (v podobě obrázku typu pgm) a také souboru typu yaml, který obsahuje metadata o naskenované a uložené mapě, jako je název, rozlišení, úroveň prahování apod.

6.2.2 Řešení s použitím Gmapping a AMCL

Za použití Gmapping s předem vytvořenou mapo lze využívat navigačního zásobníku (navigation stack), který zprostředkovává funkce global a local plannerů. I přesto že se může zdát poskytnutí mapy v některých případech nežádoucí, nebo náročné, jedná se o přístup, který přináší vysokou funkcionalitu právě díky navigačnímu zásobníku. Mapu lze vytvořit ručním ovládáním robota prostředím, které chceme zmapovat, za použití Gmapping a následně použít takto vytvořenou mapu. Dalším způsobem je vytvoření prostředí v simulačním softwaru Gazebo, kde lze v simulaci využít přesnějších senzorů a opět simulovaně projet prostředí. Takto vytvořená mapa bude věrněji reflektovat dané prostředí.

6.2.2.1 Zadávání cílových bodů

V případě zprostředkování předchozí mapy lze v RViz udávat příkazy k pojezdu do vyžádané pózy. Podobně to lze pak řešit za pomoci programu a zadávat tak navigační cíle pro robota s patřičnou pozicí. Ovšem při využití algoritmů AMCL je nutné pro zajištění autonomního pohybu v prostředí robota nejprve ručně lokalizovat v mapě pomocí 2D pose estimate v programu RViz. Po uspěšné lokalizaci robota je pak užito move_base a AMCL k zadávání navigačních cílů.

Robot se, v případě zadání navigačního cíle, díky AMCL snaží dosáhnout zadaného cíle za současného vyhýbání se překážkám. Pokud použijeme v předchozím programu metodu pro zadávání cílů s využitím AMCL, je pak řešeno pouze procházení sítě bodů, o ostatní se stará AMCL a Gmapping. Je nutné pouze uvést v iteraci procházení sítě podmínku pro nedosažení cíle.

6.2.2.2 Funkce řešení s použitím Gmapping a AMCL

Postup pro spuštění je stejný jako v předchozím případě, jen pro spuštění scriptu pro sběr vzorků použijeme příkaz:

```
rosrun uhk_fp fc_move_base.py
```

Anebo pomocí vytvořeného launch souboru:

```
roslaunch uhk_fp uhk_move_base.launch
```

Dokončení sběru vzorků je stejně jako v předchozí metodě oznámeno vypsáním: „Fingerprint collection finished“. Poté lze opět mapu uložit:

```
rosrun map_server map_saver -f /tmp/my_map
```

6.2.3 Použití Hector SLAM

Metoda Hector vyžaduje instalci hector_slam:

```
sudo apt-get install ros-indigo-hector-slam
```

([xxx] je označení distribuce ROS např.“kinetic“)

Hector SLAM byl použit na bag soubor, který byl zaznamenán průjezdem jižní strany budovy FIM Univerzity Hradec Králové ve třetím patře. Hector Slam se spouští launch souborem, který Spuštění Hector SLAM v jednom okně terminálu:

```
roslaunch hector_slam_launch tutorial.launch
```

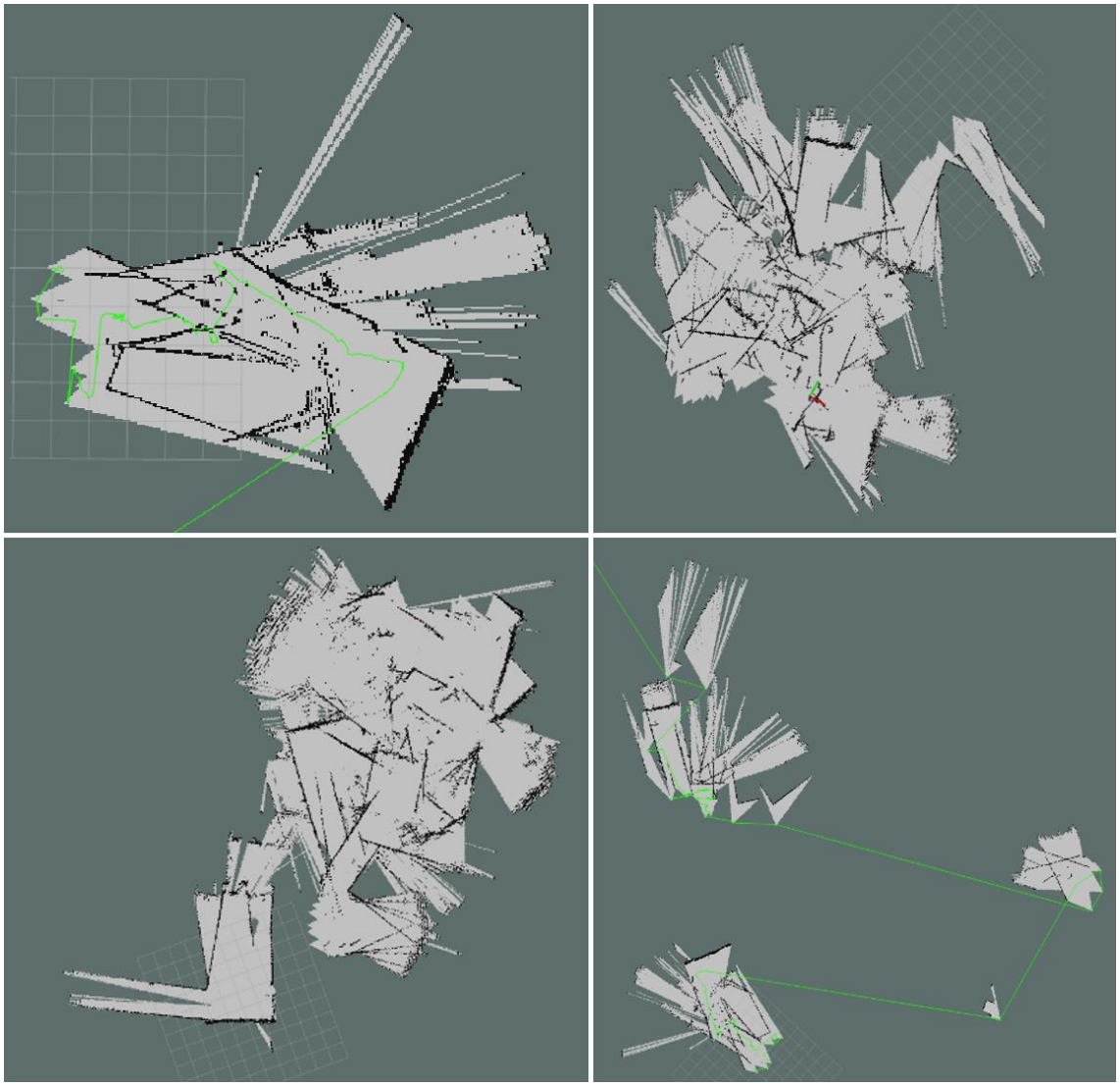
Výše uvedeným launch souborem se spustí uzel hector_mapping společně hector_trajectory server a hector_geotif, které jsou nezbytné pro tvorbu map. Započetím přehrávání požadovaného souboru souboru bag, lze pak v Rviz sledovat mapovací proces:

```
rosbag play uhk200420.bag  --clock
```

Po dokončení mapovacího procesu lze mapu uložit, mapy se při použití Hector SLAM ukládají do složky 'hector_slam/hector_geotiff/maps'. Uložením mapy dojde k vytvoření dvou souborů .tif s obrazovou reprezentací mapy a .twf obsahující informace o georeferencích. Uložení mapy se provede zadáním:

```
rostopic pub syscommand std_msgs/String "savegeotiff"
```

S navzorkovaným bag souborem a použitím Hector SLAM byly vytvořeny mapy viz. obrázek číslo 15. Mapy jsou vytvořeny s různým nastavením, první mapa je vytvořena s launch souborem tutorial.launch, druhá mapa je vytvořená s nastavením, které pro Hector zprostředkovává rámce tak jako pro Gmapping (uhk_hector_as_gmapping.launch), třetí mapa je vytvořena bez transformace z odom frame (uhk_hector_no_odom.launch) a poslední mapa vznikla s nastavením bez transformací (uhk_no_tf.launch).



Obrázek 15 Mapy vytvořené metodou Hector
Zdroj: vlastní zpracování

6.2.4 Použití metody Google Cartographer

Pro Google Cartographer byl vytvořen separátní pracovní prostor (workspace) pojmenovaný carto_ws – viz. postup v kapitole 6.1.2. K instalaci je doporučeno užití nástrojů příkazového řádku wstool, rosdep a build systém ninja pro rychlejší vytvoření build. Jejich instalace:

```
sudo apt-get update
sudo apt-get install -y python-wstool python-rosdep ninja-build
```

Následně do připraveného pracovního prostoru naklonujeme repozitáře z github:

```
cd carto_ws
wstool init src
wstool merge -t src
https://raw.githubusercontent.com/googlecartographer/cartographer_ros/master
/cartographer_ros.rosinstall
wstool update -t src
```

Instalace cartographer_ros dependecy a následný build:

```
src/cartographer/scripts/install_proto3.sh
sudo rosdep init
rosdep update
rosdep install --from-paths src --ignore-src --rosdistro=[xxx] -y
catkin_make_isolated --install --use-ninja
```

([xxx] je označení distribuce ROS např.“kinetic“)

Pro používání nového pracovního prostoru určeného pro Cartographer je nutné jej nejprve uvést jako zdroj pro spouštění balíčků.

```
source install_isolated/setup.bash
```

Google Cartographer využívá k popisu parametrů robota konfigurační soubory lua, které jsou pak uvedeny v launch souborech jednotlivých zprostředkovaných metod. V rámci této práce byly vytvořeny konfigurační soubor lua i launch soubor (v příloze). Spuštění launch souboru:

```
roslaunch cartographer_ros my_robot.launch
bag_filename:=/home/turtlebot/uhk200420.bag
```

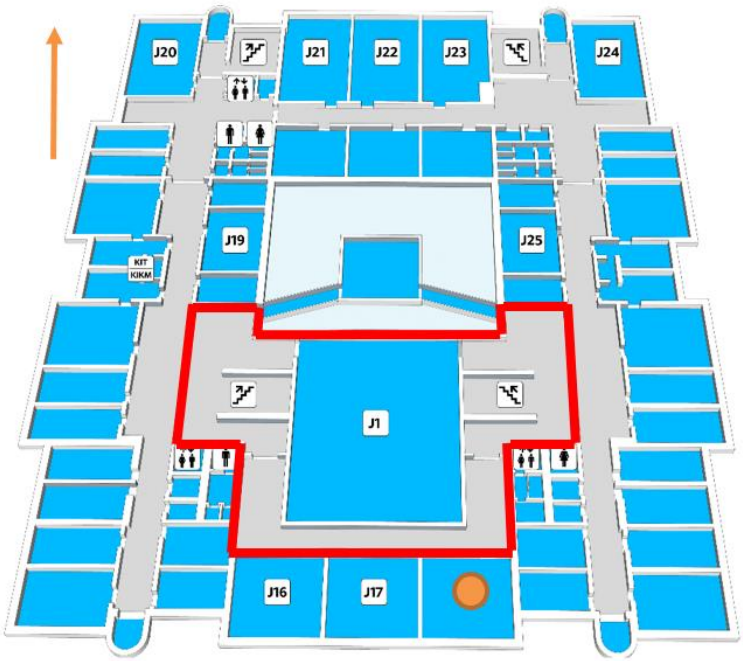
Tímto je přehrán soubor bag a po dokončení přehrávání je vytvořen soubor .pbstream, který popisuje trajektorie robota. Následně lze vytvořit mapu spuštěním následujícího launch souboru s uvedením bag a pbstream souborů:

```
roslaunch cartographer_ros assets_writer_backpack_2d.launch
bag_filenames:=/home/turtlebot/uhk200420.bag
pose_graph_filename:=/home/turtlebot/uhk200420.bag.pbstream
```

Bohužel se se zaznamenaným bag souborem nepodařilo za pomoci Cartographeru vytvořit mapu, přičemž byl předem kladen důraz na odebíraná témata, tak aby byla uplatnitelná pro Cartographer. Byla odebírána témata tf, odom, scan a mobile_base/sensors/imu_data.

7 Shrnutí výsledků

Sběr dat byl prováděn za manuálního pojezdu robota pomocí teleop a probíhal na budově Fakulty informatiky a managementu Univerzity Hradec Králové ve třetím patře. Plán třetího patra je vyobrazen na obrázku č. 16, v plánu je oranžovým bodem vyznačena laboratoř MOVITEK, bude takto označen i v dalších zobrazeních mapy. Oranžovou šipkou je naznačen směr k severní straně budovy (nekoresponduje přesně se směrem dle světových stran). Oblast zaznamenaná ve zmíněném bag souboru je orámována červeně.



Obrázek 16 Plán 3. patra Budova J UHK FIM
Zdroj: https://fim.uhk.cz/kroky/cz/

Testování programů pro sběr vzorků probíhalo za pomoci simulačního prostředí Stage -viz kapitola 5.3.1. Následně byl program otestován přímo na Turtlebotovi 2, kde docházelo k nesrovnalostem se simulovaným prostředím a tím reálným, kdy potřebná transfromace mezi rámcem odometrie (odom frame) a rámcem základny robota (base_footprint frame) v simulaci fungovala správně, při nasazení na Trutlebot bohužel transformace vyvolávala chybová hlášení a bránila spuštění programu. Následně byl program nezávislý na metodách SLAM upraven bez použití transformací. Tento program je spíše vhodný na prostředí menších rozměrů a nemusí fungovat bezchybně v prostředích s úzkými chodbami. Zároveň použití

odom_frame pro výpočty navigačních cílů není zaručeným řešením, protože během následné lokalizace může dojít k deformaci sítě bodů. Program s využitím move_base je velice efektivní, jelikož řeší plánování tras a není nutná další implementace detekce a reakce na překážky.

7.1 Nasnímané mapy

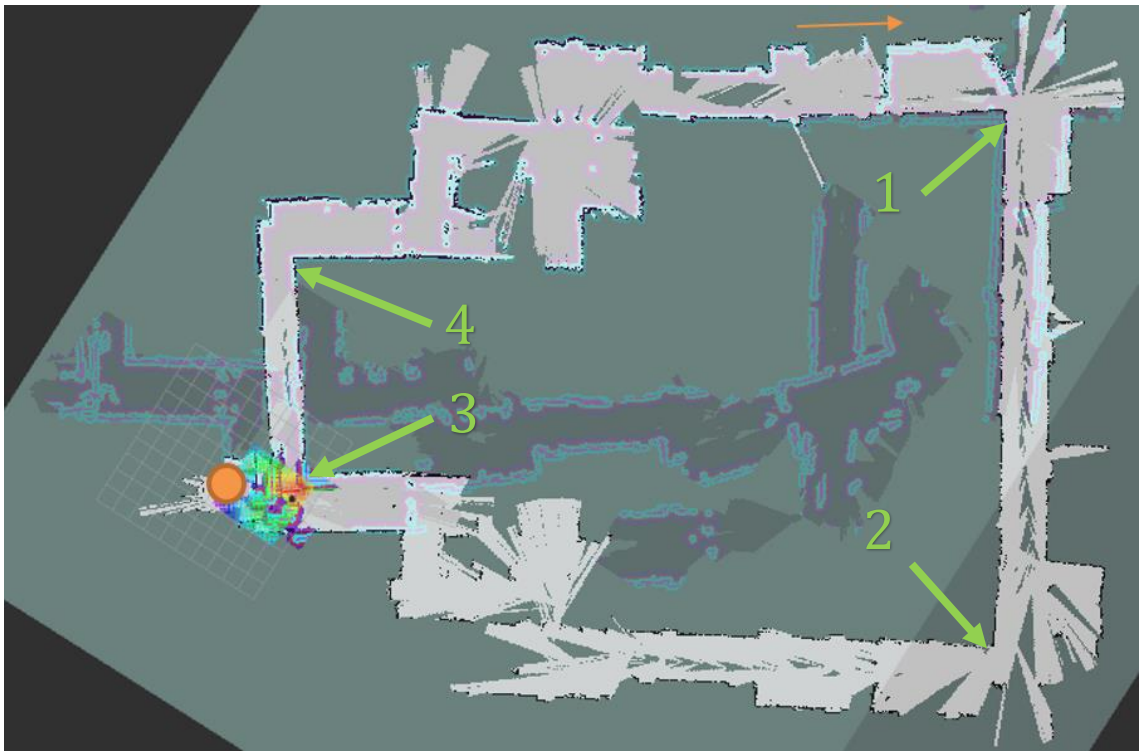
Ke snímání map bylo zejména využíváno Gmapping, jelikož tato metoda do svých výpočtů zahrnuje i model odometrie, který mohl podpořit správné vykreslení mapy. Z rešerše vyplývá, že data, která snímač Microsoft Kinect poskytuje nelze srovnávat s daty z LIDARu, už jen pro úzký pozorovací úhel. Všechny uváděné metody jsou spíše stavěné pro snímače s vyšší přesností, jako je právě LIDAR, proto během náběru mapových vzorků opakovaně docházelo k zádním problémům, které bránili dosažení reprezentativního ztvárnění mapy.

Obrázek 17 Mapové vzorky Gmapping
Zdroj: vlastní zpracování

Nasbírané mapové vzorky 1

Na obrázku č. 17 jsou vyobrazeny mapové vzorky. První obrázek je příkladem problému uzavření smyčky, kdy se chodby díky neuzavřené smyčce překrývají a vytváří tak spirálovitě chodby nové. Na obrázku je souhlasný bod zvýrazněn šipkou a také pozicí laboratoře. Druhá mapa vyznačuje problém uneseného robota, kdy je robot během mapování manuálně přesunut na jinou pozici (bez pomoci motorů) a kvůli žádným datům o trajektorii, kterou robot urazil, je následně nově mapován další prostor, který překrývá již zmapovaný prostor s poslední známou informací

z odometrie. Třetí mapa vyobrazuje úspěšné zavření smyčky, detailnější pohled na tuto mapu je na obrázku číslo 18.



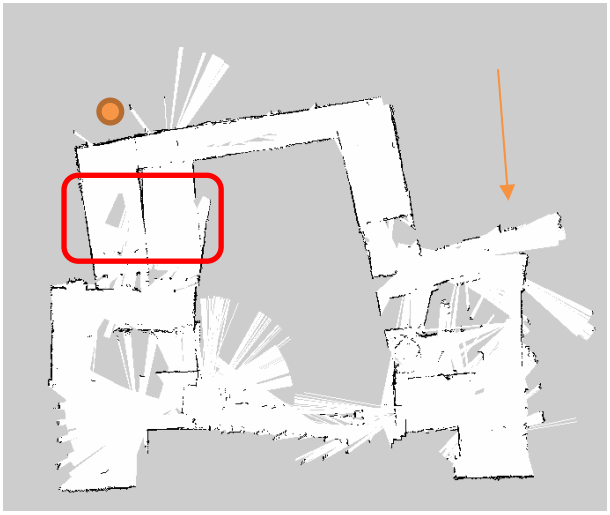
Obrázek 18 Mapa s uzavřenou smyčkou Gmapping
Zdroj: vlastní zpracování

Úspěšného uzavření smyčky bylo dosaženo snížením rychlosti na 0,2m/s a opakovaným navštěvováním již zmapovaných míst, což vedlo ke zvýšení jistoty polohy robota v prostoru. Robotem byla již zmapovaná místa navštívena třikrát. Za použití rychlosti 0,5m/s vyznačovalo měření značné odchylky. Vyšší rychlostí bylo riziku smyku kol robota podstatně vyšší a vyznačovalo se nepřesností nejen v ujeté vzdálenosti, ale i ve snímání úhlů rohů viz. Tabulka číslo 3. Měřené úhly jsou označeny na obrázku 18 zelenou šipkou a číslem.

Rychlost 0,5m/s		Rychlost 0,2m/s	
Naměřený úhel (°)	Odchylka Δ	Naměřený úhel (°)	Odchylka Δ
69°	21°	88°	2°
92°	2°	91°	1°
86°	4°	95°	5°
98°	8°	91°	1°

Tabulka 3 Úhly v mapách při různých rychlostech
Zdroj: vlastní zpracování

Vyšší rychlost dále negativně přispívala k problematice uzavření smyčky tvorbou paralelních chodeb, jak je patrné z obrázku číslo 19 – orámované červeně.



Obrázek 19 Chyba mapování - paralelní chodby
Zdroj: vlastní zpracování

Další metody byly hodnoceny pouze na vzorku z bag souboru jižní části budovy – viz. červeně vyznačená část na plánu obrázek č. 16. Zatímco při Gmapping byla vytvořena mapa, která zdánlivě reprezentuje prostředí (obrázek č. 19), se stejnými vstupními daty metoda Hector vyprodukovala mapy s daleko nižší kontextuální hodnotou. Z těchto map jsou sotva patrné chodby či rohy zdí a jsou také nesouvislé. Tento rozdíl může být ve značné míře způsoben způsobem, kterým byl bag soubor nasnímán. Při sběru dat se robotem pojíždělo tam a zpět tak, aby robot rozpoznal již navštívené prostředí viz kapitola 3.1.1, zatímco pro Hector a Cartographer tento postup nemusí být vhodný.

8 Závěry a doporučení

Tato bakalářská práce teoreticky popisuje a implementuje algoritmy pro řízení mobilního robota Turtlebot 2 pro sběr vzorků z bezdrátových sítí. V teoretické části práce pojednává o úvodu do problematiky, nastiňuje existující řešení SLAM a jejich metody, dále poskytuje přehled základních funkcionalit ROS. Druhá část, praktická, analyzuje možná řešení, popisuje dílčí funkce navrhovaných řešení a popisuje postupy pro různé metody SLAM. Smyslem praktické části práce bylo navrhnout a implementovat řešení pro řízení autonomního robota za podpory prostředků popsaných v páté části této bakalářské práce. Za pomoci popisovaných metod byl navržen a implementován program schopný dosáhnout kladeného cíle. V práci shrnuté výsledky zhodnocují použité metody SLAM a chyby, které se během nasazení SLAM metod vyskytovaly. Podstatnými okolnostmi při návrhu programu bylo efektivní využití do ROS již integrovaných postupů a jejich aplikace s danou technologií, přičemž dosáhnout reprezentativních výsledků. Důležitým faktorem byla také snaha o dosažení cíle s méně nákladnou technologií kompatibilní s ROS. Neposledním aspektem bylo upřednostnit autonomní řízení robota a usnadnit tak případnou práci operátora zařízení.

Pro mapování prostor bylo použito algoritmů Gmapping, toto řešení vyznačovalo pro SLAM obecně známé vady jako je třeba problematika uzavření smyčky. Uzavření smyčky bylo dosaženo až opakovaným průjezdem robota stejnými pasážemi, které napomáhalo jistějšímu lokalizování robota v mapě. Řešení by bylo možné obohatit o snímání vizuální odometrie, která by nebyla nákladným prostředkem pro podpoření již poskytované odometrie samotného Turtlebota. Daná sestava Turtlebota projevovala nedostatky zejména v odometrii, která kvůli materiálu kol značně podléhala smykům, dále úzký pozorovací úhel snímače Microsoft Kinect, který je pro lokalizaci na základě snímaného prostředí hraniční. Jedním z nedostatků byla také neschopnost Kinectu snímat lesklé a průhledné povrchy, která byla odstraněna užitím ultrazvukového snímače, ale jeho použití je pouze ve smyslu odstranění případných kolizí s prostředím. Nicméně není v této sestavě řešení neproveditelné, ovšem bylo by přínosné jej obohatit o případné postupy opakovaného navštěvovaní některých míst, nebo toto zajistit pojížděním tam a zpět.

Konstrukce robota v mnohém negativně přispívala ke kvalitnímu zpracování mapy, a to zejména v případě kol, která se často na dlaždicích smýkala, a tudíž data odometrie, která Gmapping zpracovává, byla často nepřesná. Toto by bylo možné odstranit použitím robota s větším průměrem kol s vyšší adhezí. Významným zlepšením by byla předem připravená mapa prostředí, která by umožnila dále vylepšit problémy lokalizace, použitím tzv. april tagů, nebo RFID čipů.

Middleware ROS poskytuje obsáhlou podporu pro zmiňované řešení, a to i díky dostupným SLAM metodám, ovšem vyžaduje rozsáhlé dohledávání zdrojů a poznatků přesahující rámce studia.

Další práce by se mohla zaměřit na optimální využití balíčků navigačního zásobníků ROS a jejich metody implementovat za použití snímače LIDAR. Jednou z dalších možností by bylo použití vizuální odometrie, nebo užití feature detektorů v prostřední populovaným konkretními featury. Neposlední možnost je návrh vlastních algoritmů pro costmapy a plánovače tras.

9 Seznam použité literatury

[1] WANG, Zhan, Shoudong HUANG a Gamini DISSANAYAKE. Simultaneous localization and mapping: exactly sparse information filters. Hackensack, NJ: World Scientific, c2011. New frontiers in robotics, v. 3. ISBN 9814350311.

[2] M. A.-LATIF ET AL, Doaa. 3D Graph-based Vision-SLAM Registration and Optimalization. INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING. 2014, (8), 124. ISSN 1998-4464.

[3] JEFFERIES, Margaret E. a Wai K. YEAP. Robotics and cognitive approaches to spatial mapping. New York: Springer, c2008. ISBN 3540753869.

[4] SIEGWART, Roland, Illah R. NOURBAKHSH a Davide SCARAMUZZA. Introduction to Autonomous Mobile Robots. 2nd ed. London: The Mit Press, 2011. ISBN 978-0-262-01535-6.

[5] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. Probabilistic robotics. Cambridge, Mass.: MIT Press, c2005. ISBN 0262201623.

[6] MONTEMERLO, Michael, Sebastian THRUN a Bruno SICILIANO. FastSLAM: a scalable method for the simultaneous localization and mapping problem in robotics. 1. Berlin: Springer, c2007. ISBN 9783540463993.

[7] MAHTANI, Anil; Sanchez., FERNÃ¢ANDEZ, Enrique and SÃ¢ANCHEZ, Luis. Effective Robotics Programming with ROS. Birmingham, UK : Packt Publishing, 2016.

[8] QUIGLEY, Morgan, Brian GERKEY a William D. SMART. Programming robots with ROS. Sebastopol: O'Reilly & Associates Incorporated, [2015]. ISBN 1449323898.

[9] KOZŁOWSKI, Krzysztof. Robot motion and control 2007. London: Springer, c2007. ISBN 9781846289743.

[10] STACHNISS, Cyril. Robotic Mapping and Exploration. 55. Berlin: Springer-Verlag Berlin Heidelberg, 2009. ISBN ISBN 978-3-642-01096-5.

[11] YOSHIKAWA, Tsuneo. Foundations of robotics: analysis and control. Cambridge, Mass.: MIT Press, c1990. ISBN 0262240289.

[12] MENDES, Alexandre, Yamin YAN a Shiefeng CHEN. Intelligent robotics and applications: 11th International Conference. 2. Newcastle: Springer Berlin Heidelberg, 2018. ISBN 9783319652887.

[13] NÜCHTER, Andreas. 3D robotic mapping: the simultaneous localization and mapping problem with six degrees of freedom. Berlin: Springer - Verlag, 2010. Springer tracts in advanced robotics. ISBN 978-3-642-10058-1.

[14] Robot intelligence technology and applications. New York, NY: Springer Berlin Heidelberg, 2018. ISBN 9783319784519.

[15] DOUCET, Arnaud, Nando DE FREITAS a Neil GORDON. Sequential Monte Carlo methods in practice. New York: Springer, c2001. ISBN 0387951466.

[16] JIA, Yingmin, DU, Junping and ZHANG, Weicun. Proceedings of 2019 Chinese Intelligent Systems Conference: Volume III. Singapore : Springer Singapore, 2020.

[17] Grisetti, Giorgio & Stachniss, Cyrill & Burgard, Wolfram. (2007). Improved Techniques for Grid Mapping With Rao-Blackwellized Particle Filters. Robotics, IEEE Transactions on. 23. 34 - 46. 10.1109/TRO.2006.889486.

[18] Grisetti, Giorgio & Stachniss, Cyrill & Burgard, Wolfram. (2005). Improving Grid-based SLAM with Rao-Blackwellized Particle Filters By Adaptive Proposals and Selective Resampling. 2432-2437. 10.1109/ROBOT.2005.1570477.

[19] Kohlbrecher, Stefan & Meyer, Johannes & Graber, Thorsten & Petersen, Karen & Klingauf, Uwe & Von Stryk, Oskar. (2014). Hector Open Source Modules for Autonomous Mapping and Navigation with Rescue Robots. 624-631. 10.1007/978-3-662-44468-9_58.

[20] Kohlbrecher, Stefan & Von Stryk, Oskar & Meyer, Johannes & Klingauf, Uwe. (2011). A flexible and scalable SLAM system with full 3D motion estimation. 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR). 10.1109/SSRR.2011.6106777.

[21] JAZAR, Reza N. Theory of applied robotics: kinematics, dynamics, and control. 2nd ed. New York: Springer, c2010. ISBN 978-1-4419-1749-2.

[22] PAUL, Richard P. Robot manipulators: mathematics, programming, and control : the computer control of robot manipulators. Cambridge, Mass.: MIT Press, c1981. ISBN isbn978-0-262-16082-7.

[23] SALES , Francisco Ferrer. SLAM and Localization of People with a Mobile Robot using a RGB-D Sensor. University of Coimbra Faculty of Sciences and Technology Department of Electrical and Computer Engineering [online]. 2014. [Cit. 12 November 2019]. Dostupné z: https://ap.isr.uc.pt/archive/FSales_MSc_thesis_SLAM_people_detect_RGB-D.pdf

[24] NARTINS, Gonçalo dos Santos. A Cooperative SLAM Framework with Efficient Information Sharing over Mobile Ad Hoc Networks. University of Coimbra Faculty of Sciences and Technology Department of Electrical and Computer Engineering [online]. 2014. [Cit. 14 December 2019]. Dostupné z: https://ap.isr.uc.pt/archive/GMartins_dissertation_final.pdf

[25] BADODKAR, D N. and DWARAKANATH, T A. Machines, Mechanism and Robotics: Proceedings of iNaCoMM 2017. Singapore : Springer Singapore, 2019.

[26] HESS, Wolfgang, KOHLER, Damon, RAPP, Holger and ANDOR, Daniel. Real-time loop closure in 2D LIDAR SLAM. 2016 IEEE International Conference on Robotics and Automation (ICRA) [online]. 2016. [Cit. 5 January 2020]. DOI 10.1109/icra.2016.7487258. Dostupné z: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45466.pdf

[27] FOOTE, Tully. ROS Dokumentace. ros.org [online]. [Cit. 1 September 2019]. Dostupné z: http://wiki.ros.org/Documentation

[28] KELLY, Alonzo. Mobile robotics: mathematics, models, and methods. New York : Cambridge University Press, 2013.

[29] LENTIN, Joseph. Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System, 2nd Edition. s.l. : Packt Publishing Limited, 2018.

[30] KOUBÂA, Anis. Robot operating system (ROS): the complete reference. Cham : Springer, 2016.

[31] FAIRCHILD, Carol and HARMAN, Thomas L. ROS robotics by example: bring life to your robot using ROS roboticapplications. Birmingham, UK : Packt Publishing Limited, 2016.

[32] LENTIN, Joseph. ROS robotics projects: build a variety of awesome robots that can see, sense, move, and do a lot more using the powerful Robot Operating System. Birmingham, UK : Packt Publishing Ltd., 2017.

[33] FERNANDEZ, Enrique. Learning ROS for Robotics Programming - Second Edition. Packt Publishing, 2015.

[34] MARTINEZ, Aaron and FERNÁNDEZ Enrique. Learning ROS for robotics programming: a practical, instructive, and comprensive guide to introduce yourself to ROS, the top-notch, leading robotics framework. Birmingham : Packt Publishing, 2013.

[35] YOONSEOK, Pyo, HANCHEOL, Cho, RYUWOON, Jung and TAEHOON, Lim. ROS Robot Programming [online]. Seoul, Republic of Korea : ROBOTIS Co.,Ltd., 2017. [Cit. 15 November 2019]. Dostupné z: https://www.pishrobot.com/wp-content/uploads/2018/02/ROS-robot-programming-book-by-turtlebo3-developers-EN.pdf

[36] What is a TurtleBot? TurtleBot [online]. [Cit. 20 June 2019]. Dostupné z: https://www.turtlebot.com/

[37] ZHANG, Zhengyou. Microsoft Kinect Sensor and Its Effect [online]. [cit. 2020-04-28]. Dostupné z: https://www.researchgate.net/publication/254058710_Microsoft_Kinect_Sensor_and_Its_Effect

10 Přílohy

Příloha 1 - Struktura přiloženého zip souboru - UHK_FIM_SLAM_Ludek_Krula.zip

Příloha č. 1

Struktura přiloženého zip souboru - UHK_FIM_SLAM_Ludek_Krula.zip

Vytvořené konfigurační a launch soubory pro metodu Google Cartographer

\uhk_slam\carto_ws\uhk_cartographer\config\

uhk_carto.lua (573.0B)

\uhk_slam\carto_ws\uhk_cartographer\launch\

uhk_carto.launch (1.2KB)

Vytvořené launch soubory pro metodu Hector

\uhk_slam\hector_ws\src\hector_slam\hector_mapping\launch\

tutorial.launch (769.0B)

uhk_hector_as_gmapping.launch (769.0B)

uhk_hector_no_odom.launch (730.0B)

uhk_no_tf.launch (637.0B)

Vytvořený program pro ultrazvukový senzor na mikrokontroler Arduino

\uhk_slam\other\arduino\uhk_ultrasound\

uhk_ultrasound.ino (2.4KB)

Zaznamenáný bag soubor a Google Cartographerem vytvořený pbstream

\uhk_slam\other\bag\

uhk200420.bag (243.8MB)

uhk200420.bag.pbstream (421.0B)

Scripty a launch soubory pro sběr fingerprintů

\uhk_slam\catkin_ws\src\uhk_fp\launch\

uhk_free_move.launch (1017.0B)

uhk_move_base.launch (1.7KB)

\uhk_slam\catkin_ws\src\uhk_fp\src\

fc.py (15.6KB)

fc_move_base.py (12.2KB)

Oskenované zadání práce

Zadání bakalářské práce

Autor:	**Luděk Krůla**
Studium:	I1700647
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Implementace a nasazení řídícího software pro autonomního robota Turtlebot 2**
Název bakalářské práce AJ:	Implementation of Control Software for Autonomous Turtlebot 2 Robot

Cíl, metody, literatura, předpoklady:

Cíl: Cílém této práce je navrhnout a implementovat autonomní řešení pro sběr tzv. fingerprintů z bezdrátových sítí mobilním robotem. Sběr fingerprintů by měl probíhat v předem definované síti bodů, ve kterých robot zastaví a provede případné měření. Mobilní robot, v tomto případě komerční Turtlebot 2, by měl fingerprinty sbírat autonomně např. v budově FIM UHK. Smyslem práce dále je porovnat různé aplikace SLAM a otestovat různá nastavení jejich parametrů.

Osnova:

Úvod

1. Cíl práce
2. Rešerše řešení
3. Lokalizace a mapování
4. Použité prostředky
    1. Hardware
    2. Software
5. Navržené řešení
6. Dosažené výsledky a jejich zhodnocení
7. Závěr

- LENTIN, Joseph. ROS Robotics Projects: Make your robots see, sense, and interact with cool and engaging projects with Robotic Operating System. 1. Packt Publishing, 2017. ISBN 9781783554720.
- PAUL, Richard P. Robot manipulators: mathematics, programming, and control : the computer control of robot manipulators. Cambridge, Mass.: MIT Press, 1981. ISBN isbn978-0-262-16082-7.

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	Ing. Pavel Kříž, Ph.D.

Datum zadání závěrečné práce: 14.1.2018