VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Miloš Cihlář

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

SROVNÁNI 2D LIDAR SLAM METOD V SIMULACI A V REÁLNÉM SVĚTĚ
COMPARATION OF THE 2D LIDAR SLAM METHODS IN THE SIMULATION AND IN THE REAL WORLD

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Miloš Cihlář

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Adam Ligocki

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Miloš Cihlář ***ID:*** 201303
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Srovnáni 2D LIDAR SLAM metod v simulaci a v reálném světě

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je nasimulovat, vytvořit a otestovat systém, který bude pomocí 2D LIDARu a již existujícího 2D SLAM algoritmu vytvářet 2D mapy vnitřních prostor budovy.

1. Seznamte se s frameworkem Robotic Operation System (ROS), který dále použijte při realizaci práce.
2. Ověřte možností online platformy ROS Development Studio
3. Vyberte nejméně 3 2D SLAM algoritmy v repositářích systému ROS. Po konzultaci s vedoucím algoritmy otestujte v simulačním prostředí Gazebo.
4. Připravte vhodnou mobilní robotickou platformu vybavenou výpočetní jednotkou a 2D LIDARem, kterou bude možné ovládat pomocí použitého SLAM algoritmu.
5. Otestujte nejméně 2 SLAM algoritmy v reálném prostředí. Vyhodnoťte výsledky experimentů.

DOPORUČENÁ LITERATURA:

[1] Montemerlo, Michael, and Sebastian Thrun. FastSLAM: A scalable method for the simultaneous localization and mapping problem in robotics. Vol. 27. Springer, 2007.

Termín zadání: 3.2.2020 ***Termín odevzdání:*** 8.6.2020

Vedoucí práce: Ing. Adam Ligocki

doc. Ing. Václav Jirsík, CSc.
předseda rady studijního programu

ABSTRAKT

Tato práce se zabývá návrhem vhodné mobilní robotické platformy, se schopností vytvořit mapu budovy za použití existujících 2D SLAM algoritmů, a porovnáním SLAM algoritmů v simulaci a reálném světě. Jsou použity tři známé algoritmy, tj. Hector slam, Gmapping a Karto slam. Součástí práce je vytvoření dvoukolového diferenciálně řízeného modelu robota v simulátoru Gazebo a návrh softwaru, který je realizovaný pomocí frameworku ROS. V práci je popsána a implementována základní lokalizace pro mobilní pozemní roboty. Jsou zmíněny metody pro plánování a kontrolu pohybu robota po vypočítané trajektorii. Práce také popisuje vlastní přístup k autonomnímu průzkumu robotem s cílem zmapovat všechny dostupné části budovy. K porovnání a hodnocení vytvořených map je navrženo kritérium kvality určující přesnost mapy.

KLÍČOVÁ SLOVA

SLAM algoritmus, Gazebo, ROS, URDF, Autonomní robot, RRT algoritmus, Bézierova křivka, Pure Pursuit algoritmus

ABSTRACT

This thesis deals with the design of a mobile robotic platform with the ability to build a map with an already existing 2D SLAM algorithm and a comparison of this algorithm in simulation and the real world. It uses three known algorithms i.e. Hector slam, Gmapping, and Karto slam. The part of this work deals with the creation of two wheels differential drive robot models in the Gazebo simulator and with an implementation proposal that is realized with the ROS framework. In thesis is implemented basic localization technique for mobile robots. Planning algorithms and a path controller are mentioned. In thesis is proposed own access to an autonomous exploration of building to build a map of the building. For evaluation of the SLAM algorithm is suggested slam quality criterion.

KEYWORDS

SLAM algorithm, Gazebo, ROS, URDF, Autonomous robot, RRT algorithm, Bézier curve, Pure Puruit Controller

CIHLÁŘ, Miloš. *Srovnání 2D LIDAR SLAM metod v simulaci a v reálném světě*. Brno, 2020, 71 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce: Ing. Adam Ligocki

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Srovnání 2D LIDAR SLAM metod v simulaci a v reálném světě“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno 5. 6. 2020 .....

podpis autora

PODĚKOVÁNÍ

Chci poděkovat vedoucímu bakalářské práce panu Ing. Adamovi Ligockému za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno 5. 6. 2020 .....

podpis autora

Obsah

Úvod **10**

1 Robot Operating System **11**
1.1 Nodes . . . 11
1.2 Messages . . . 12
1.3 Services . . . 12
1.4 Topics . . . 12
1.5 Master . . . 13
1.6 Roslaunch . . . 13
1.7 ROS Development Studio . . . 14

2 Gazebo **15**

3 SLAM **17**
3.1 Landmark extraction . . . 18
3.1.1 Spikes algoritmus . . . 18
3.1.2 Sliding window algoritmus . . . 19
3.2 Data association . . . 19
3.3 SLAM koncept . . . 19
3.3.1 EKF SLAM . . . 19
3.3.2 FastSLAM . . . 19

4 Robot **21**
4.1 Kinematický model . . . 21
4.2 Simulace modelu robota . . . 22
4.2.1 URDF . . . 22
4.2.2 Model robota . . . 23
4.3 Fyzická platforma . . . 25
4.3.1 Struktura robota . . . 25
4.3.2 RPLIDAR A1 . . . 26
4.3.3 Komunikace . . . 26

5 Software **27**
5.1 Transformace . . . 28
5.2 Ovládání . . . 29
5.3 Lokalizace . . . 30
5.3.1 Rozdělení lokalizačních technik . . . 30
5.3.2 Odometrie . . . 31

5.4 Autonomní průzkum . . . 34
5.4.1 Plánování trajektorie . . . 34
5.4.2 RRT algoritmus . . . 37
5.4.3 Implementace RRT algoritmu pro autonomní průzkum . . . 37
5.4.4 Výsledky RRT . . . 40
5.5 Trajektorie . . . 42
5.6 Řízení . . . 44
5.6.1 Manuální mód . . . 44
5.6.2 Autonomní mód . . . 45
5.7 Rviz . . . 47

6 Výsledky **48**
6.1 Testování SLAM algoritmů v simulaci . . . 49
6.1.1 Komplexní interiér . . . 49
6.1.2 Dlouhé chodby . . . 53
6.2 Autonomní vytváření mapy prostředí . . . 55
6.3 Testování SLAM algoritmů v reálném prostředí . . . 56

Závěr **58**

Literatura **60**

Seznam symbolů, veličin a zkratek **63**

A Výsledné mapy získané ze simulace **64**
A.1 Výsledná mapa . . . 64
A.2 Trajektorie . . . 67

B Výsledné mapy získané ze reálného světa **69**

C Adresářová struktura zdrojových souborů **71**

Seznam obrázků

1.1 Struktura ROSu . . . 11
1.2 Detail topiku a zpráv . . . 13
2.1 Fyzikální vlastnosti v simulátoru Gazebo . . . 15
2.2 Nastavení vlastností senzorů . . . 16
3.1 Schéma řešení SLAMu . . . 17
3.2 Landmark extraction . . . 18
4.1 Model robota . . . 24
4.2 Struktura robota . . . 25
5.1 Schéma systému . . . 27
5.2 Vývoj chyby způsobený rotací kolem své osy . . . 33
5.3 Vývoj chyby způsobený přímočarým pohybem . . . 34
5.4 Buněčná dekompozice [13] . . . 35
5.5 Road map algoritmy [13] . . . 36
5.6 RRT algoritmus . . . 37
5.7 Algoritmus RRT v různých prostředích . . . 41
5.8 RRT v úzkém prostoru . . . 41
5.9 Ukázka získávání trajektorie ze stromu . . . 42
5.10 Pure pursuit . . . 46
6.1 Výsledky testovaných SLAM algoritmů v komplexním prostředí se 102 stavovou mapou . . . 49
6.2 KartoSLAM (3 stavová mapa) . . . 50
6.3 Histogramy obsazenosti . . . 51
6.4 Chyby a rozostřenost SLAM algoritmů . . . 52
6.5 Výpočetní náročnost SLAM algoritmů . . . 53
6.6 Hector SLAM v dlouhých chodbách . . . 54
6.7 SLAM algoritmy v dlouhých chodbách s odometrií . . . 55
6.8 Mapa vzniklá z autonomního průzkumu . . . 56
6.9 Histogramy SLAM algoritmů v reálném prostředí . . . 57
6.10 SLAM algoritmy v reálném prostředí . . . 57
A.1 Mapa vzniklá pomocí Karto SLAM algoritmu . . . 64
A.2 Mapa vzniklá pomocí Gmapping algoritmu . . . 65
A.3 Mapa vzniklá pomocí Hector SLAM algoritmu . . . 66
A.4 Karto SLAM trajektorie . . . 67
A.5 Gmapping trajektorie . . . 68
A.6 Hector SLAM trajektorie . . . 68
B.1 Mapa vzniklá pomocí Hector SLAM algoritmu v reálném světě . . . 69
B.2 Mapa vzniklá pomocí Gmapping algoritmu v reálném světě . . . 70

Seznam tabulek

4.1 Parametry senzoru RPLIDAR A1 26
6.1 RMSE srovnání trajektorií SLAM algoritmů 51
6.2 RMSE srovnání trajektorií SLAM algoritmů v uzavřené smyčce . . . 55
6.3 RMSE srovnání trajektorií SLAM algoritmů - autonomní průzkum . . 56

Úvod

Simultánní lokalizace a mapování neboli SLAM řeší problém lokalizace robota v neznámém prostředí. Robot nemá na začátku svého pohybu žádné informace o svém okolí a pozici v něm, a proto musí na základě dat ze senzorů vytvořit mapu prostředí a získat svou polohu. Mapa a poloha robota je neustále aktualizována na základě předešlé mapy a nových informací ze senzorů. Ačkoli se jedná o velice usilovně zkoumanou oblast robotiky, stále neexistuje dostatečně univerzální a přesné řešení.

Existuje značné množství implementací SLAM algoritmů, které fungují na různých principech a různých senzorech. Proto cílem této bakalářské práce je vytvoření takové mobilní robotické platformy, která bude schopná ověřit funkčnost různých typů SLAM algoritmů a porovnat je. K vyhodnocení správnosti jednotlivých algoritmů neexistuje jednotné pravidlo, a proto je v práci popsáno kritérium, podle něhož se vyhodnocuje kvalita výsledných map a tudíž i algoritmů.

K vytvoření mobilní platformy potřebné k realizaci práce je nutné vytvořit model robota v simulátoru Gazebo, přidat na něj použité modely snímačů a implementovat základní funkčnost robota jako je řízení a ovládání. Nezbytností pro některé typy SLAM algoritmů je základní lokalizace robota, která je v práci implementovaná pomocí odometrie.

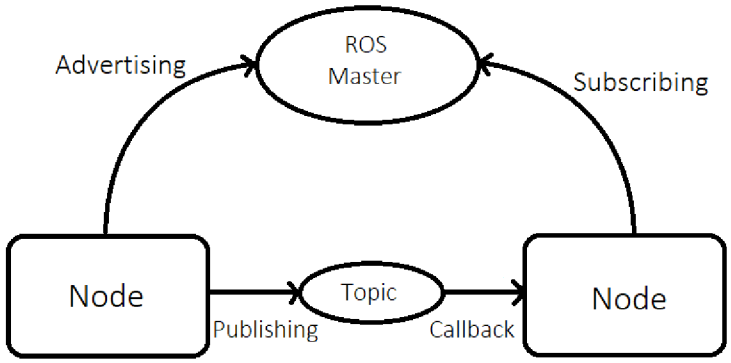
V práci jsou zmíněny možnosti plánování trajektorie robota a je popsán vlastní přístup k autonomnímu průzkumu budov robotem za účelem vytvoření kompletní mapy budovy. Autonomní průzkum je založený na pravděpodobnostním plánování a využití algoritmu RRT neboli Rapidly exploring random tree.

1 Robot Operating System

Robotika je v posledních letech velice perspektivní a rychle rostoucí vědní disciplína, spojující značné množství vědních oborů, jako jsou elektronika, informatika, mechanika, umělá inteligence a další. Proto vznikla potřeba vytvořit nástroj, jenž urychlí a usnadní vývoj a používání robotického softwaru, tj. simulátory a různé vizualizační nástroje. Zrod tohoto nástroje se odehrál v roce 2007 na Stanfordově univerzitě.

ROS (Robot Operating System) je framework běžící na na Linuxové distribuci Ubuntu, který vytváří vysokou míru abstrakce nad hardwarem. Obsahuje sbírku nástrojů, knihoven a konvencí usnadňující vývoj robotů. Filosofie ROSu je v jádru jednoduchá. ROS je open source platforma podobající se peer-to-peer síti, kde se značné množství malých, nezávislých programů může propojit a snadněji mezi sebou komunikovat. Podpora dvou programovacích jazyků nabízí rychlost vývoje v Pythonu a C++ pro vyšší rychlost a stabilitu. [1]

Nejjednodušší struktura ROSu je popsána následujícím obrázkem.



Obr. 1.1: Nejjednodušší struktura ROSu. Nody si pomocí ROS Masteru vymění informace o oznamování a odebírání zpráv na daném topiku a potom už komunikují přímo pomocí UDP komunikace

1.1 Nodes

Jak je vidět na obrázku 1.1, *node*, česky uzel, je jednoduchý program, který může s ostatními komunikovat [2]. Dílčí *nody* mohou vykonávat libovolnou funkci, ať už se jedná o zpracování obrazu, výpočet odometrie nebo plánování trajektorie. Skupina *nodů* může vytvořit komplexní systém zastávající určitou funkci. Výhodou takového přístupu je odolnost vůči pádům jednotlivých programů. Selhání jednoho subsystémů nemusí nutně znamenat pád celého systému.

Hlavním cílem *nodů* je posílat informace prostřednictvím *topicků*, které si ostatní běžící *nody* mohou vytáhnout. *Nodu*, který posílá informace, tedy *messages*, viz 1.2, se říká *Publisher* a ten, který přijímá je *Subscriber*. [2]

1.2 Messages

Messages, česky zprávy, jsou nejmenší jednotkou informace, kterou si má smysl vyměňovat s ostatními *nody* [2]. Jsou využívány standardní primitivní datové typy, které jsou shlukovány do podobného formátu jako je struktura nebo pole v programovacích jazycích.

Zprávy si mohou vývojáři vytvářet sami podle vlastní potřeby, ale v robotice se velmi často používají stejné konstrukce, a proto jsou v ROSu vytvořené takzvané *common_msgs* neboli *common_messages*. Tyto *běžné zprávy* obsahují struktury z různých oblastí robotiky.
Například

- *geometry_msgs* obsahuje základní struktury - *Point, Pose, Quaternion, Twist, Vector, Polygon ...*
- *nav_msgs* se používá při lokalizaci a plánování - *Path, Odometry, OccupancyGrid ...*
- *sensor_msgs* sdružuje *zprávy* z různých senzorů - *Imu, JointState, PointCloud, Image, Range ...*

Zprávy často mají speciální strukturu obsahující informace o počtu odeslaných zpráv nebo čas.[2]

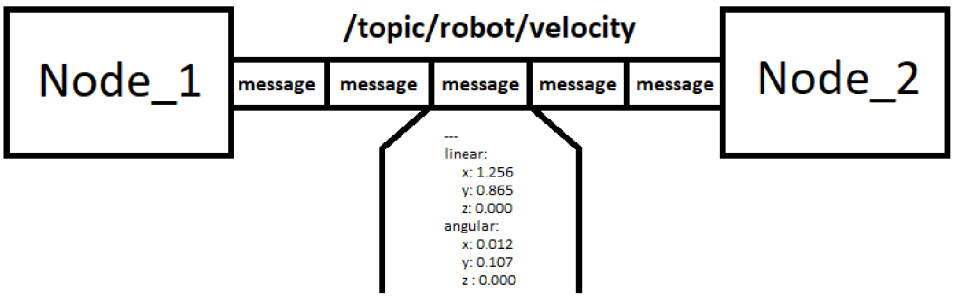
1.3 Services

Další možností, jak posílat data mezi *nody*, jsou *services*. Jsou velmi podobné zprávám, ale obsahují dva typy dat, takzvaně *request*, žádost, a *response* neboli odpověď. *Services* umožňují z jednoho *nodu* zavolat funkci z *nodu* druhého a dostat výsledek. Tuto vlastnost se vyplatí používat za předpokladu, že se data posílají pouze příležitostně. Velmi výhodné použití *services* je při zjišťování dostupnosti dat ze senzorů nebo zpětné vazby při dokončení určité činnosti. [1]

1.4 Topics

Topics, *topiky*, jsou v ROSu poměrně abstraktní konstrukce, které mají za cíl shlukovat data stejného charakteru (tj. *zprávy - messages*) a přenášet je do *nodů*, které si

o ně požádají 1.2. Detailní popis vztahu mezi *zprávami*, *topiky* a *nody* je na obrázku.



Obr. 1.2: Popis zprávy v topiku. Každý topik obsahuje data stejného charakteru a ty jsou shlukovány do zpráv a posílány s předem danou frekvencí.

1.5 Master

Master je jádro celého systému. Stará se o přerozdělování jmen a registraci *nodů* v systému. Pokud chce *publisher* odeslat data na *topik*, oznámí to *masteru*. A stejně tak *subscriber*, pokud chce data přijmout, se musí ohlásit *masteru*. V případě, že příslušný *topik* má svého *publishera* i *subscribera*, potom *master* povolí přenos dat (Obr. 1.1). [2]

1.6 Roslaunch

Za předpokladu, že je *master* aktivní, je možné spouštět prostřednictvím linuxového terminálu jednotlivé *nody*, což se provede následující příkazem.

```
user@hostname$ rosrun ros_package node
```

Vzhledem ke struktuře *ROSu* (tj. snadná komunikace a přenos dat) je nevhodné aby celou funkcionalitu robota vykonával jeden, nebo jen několik málo *nodů*, a proto je vhodné *nody* shromažďovat a spouštět najednou.

Roslaunch je nástroj zabudovaný do ROSu, který umožňuje spouštění více nodů najednou. Velká výhoda tohoto přístupu ke spouštění *nodů* je přehlednost, jelikož jsou všechny spouštěné programy definované v jednom *.launch* souboru. *Launchfile* je psaný v XML formátu. Další výhodou používání *launchfilu* je možnost jednoduché modifikace parametrů. [2]

Launchfile s názvem *file.launch* se spustí následujícím způsobem.

```
user@hostname$ roslaunch ros_package file.launch
```

1.7 ROS Development Studio

Jednou z možností vývoje v *ROSu* je nainstalovat si *ROS* a *Ubuntu* (výchozí operační systém pro práci s *ROSem*), nebo jinou linuxovou distribuci na svůj počítač. Další možností je využít online platformu *ROS Development studio*, kdy *ROS* běží na vzdáleném serveru.

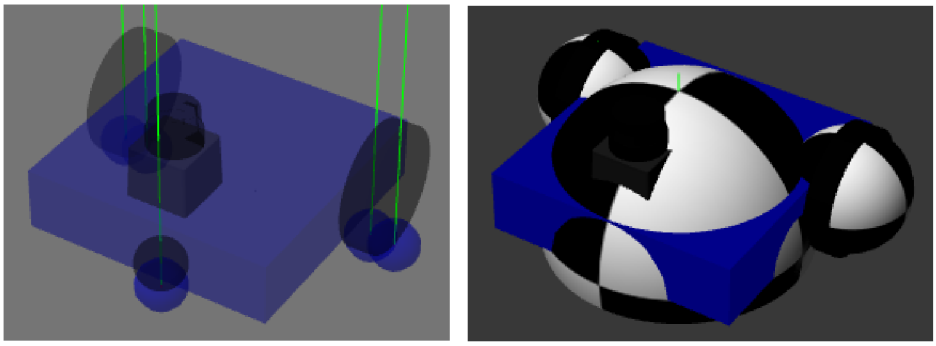
Jedná se o platformu, kde není potřebná žádná instalace a veškerý vývoj a simulace probíhá v internetovém prohlížeči. Obsahuje vestavěný editor, příkazovou řádku, *Rviz* (5.7) a *Gazebo* (2). Výhoda tohoto přístupu je v jednoduchosti. Je připraveno velké množství modelů, které lze bez jakékoli práce simulovat. Při reálném používání moc výhod nezůstane. Přístup do vývojového prostředí trvá řádově jednotky desítek sekund. Simulace, editor i příkazová řádka se musí vejít do okna v prohlížeči a vývoj se zdá pomalejší než na fyzické platformě. Navíc je v *ROS Development studiu* použitý starý software. V roce 2019 se spouští *Gazebo 7*, kdežto na fyzické platformě se používá *Gazebo 10*.

Vzhledem k velkému množství tutoriálů je výhodné *ROS Development studio* použít ke studijním účelům, kdy není potřeba složitého nastavování na fyzické platformě a *ROS Development Studio* umožňuje rychlý a přehledný náhled do problematiky.

2 Gazebo

Gazebo je open-source 3D simulátor reálné fyziky [5]. Je dostupný na operačních systémech založených na linuxových distribucích, Mac OS a Windows.

Simulátory hrají v robotice podstatnou roli, která umožňuje rychlou a efektivní práci v robotice. *Gazebo* nabízí širokou škálu možností a je možné nasimulovat libovolné venkovní nebo vnitřní prostředí, a navíc podporuje vývoj multirobotických aplikací. *Gazebo* je reálný fyzikální simulátor, který se snaží věrně nasimulovat chování modelu ve skutečnosti. Každé simulované součásti robota lze nastavit různé fyzikální prvky jako je tření v různých směrech, hmotnost, momenty setrvačnosti, tuhosti, prokluzy a další. Na obrázcích níže je ukázka fyzikálních vlastností, kde na obrázku vlevo jsou znázorněny kolize robota s prostředím a obrázek vpravo ukazuje rozložení hmotnosti.



(a) Tření (b) Rozložení hmotnosti

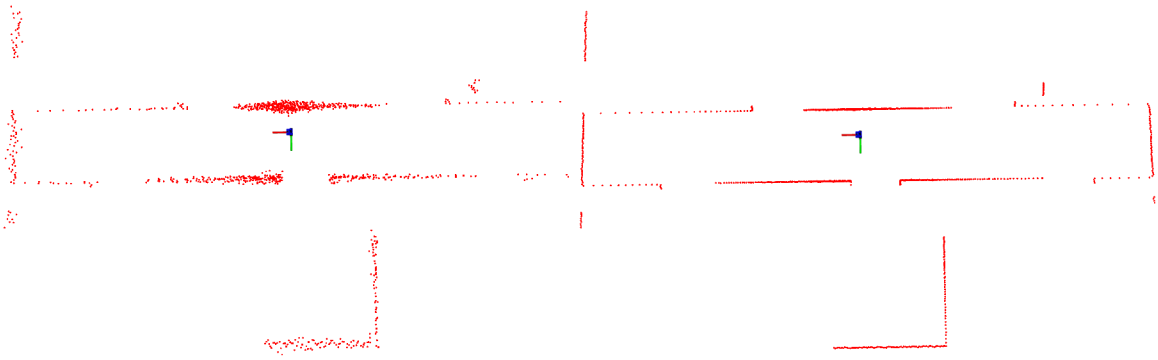
Obr. 2.1: Fyzikální vlastnosti v simulátoru Gazebo

Gazebo je složeno z několika oddělitelných knihoven. První z nich je *Gazebo Master*, který se stará o správu jmen a řídí několik simulátorů fyziky, data ze senzorů a uživatelské rozhraní. Fyzikální simulátor *Gazeba* obsahuje simulaci tuhých těles, kolize a spojení simulovaných těles. Je složen z několika open-source knihoven. Jedná se především o *Open Dynamics Engine (ODE)* a *Dynamic Animation and Robotics Toolkit (DART)*. Dalšími knihovnami používaných při simulaci je knihovna poskytující renderování 3D modelu, knihovna generující data ze senzorů a samozřejmě grafická vizualizace. [5]

Gazebo je při simulaci vnitřně rozděleno na dvě části. První z nich se jmenuje *gzserver* a je to jádro celého systému. Provádí výpočet všech fyzikálních součásti simulace. Je nezávislá na grafickém prostředí *gzklient*, která provádí vizualizaci, takže je možné simulovat bez nutnosti náročné vizualizace. [5]

Jednou z možností *Gazeba*, jak bylo zmíněno, je možnost generovat data ze senzorů. Všem typům senzorů jdou modifikovat jejich základní vlastnosti a parametry.

V této práci je primárně používaný 360° laserový měřič vzdálenosti (*Lidar* - Light Detection And Ranging). Na obrázcích 2.2 je ukázáno nastavení různých vlastností pro senzor typu LIDAR. Možnosti nastavení jsou téměř neomezené. Lze nastavit frekvenci otáčení, počet vzorků za sekundu, minimální rozlišení a maximální vzdálenost. Generace šumu je také zahrnuta v možnostech *Gazeba*, k nimž patří nastavení pravděpodobnostního rozložení šumu, jeho střední hodnota a směrodatná odchylka. [5]



(a) Senzor s velkým šumem

(b) Senzor s malým šumem

Obr. 2.2: Nastavení vlastností senzorů. Nastavuje se přesnost, rozlišení, typ šumu a další vlastnosti.

Součástí simulátoru jsou také vestavěné grafické editory. Editor budov, uživatelsky velice přívětivý a intuitivní, poskytuje velké množství předpřipravených modelů. Jedná se o modely stěn z různých materiálu, okna, dveře, nábytek a také i celé kompletní modely místností a budov. Editor modelů dovoluje stavbu různých komponent především robotů.

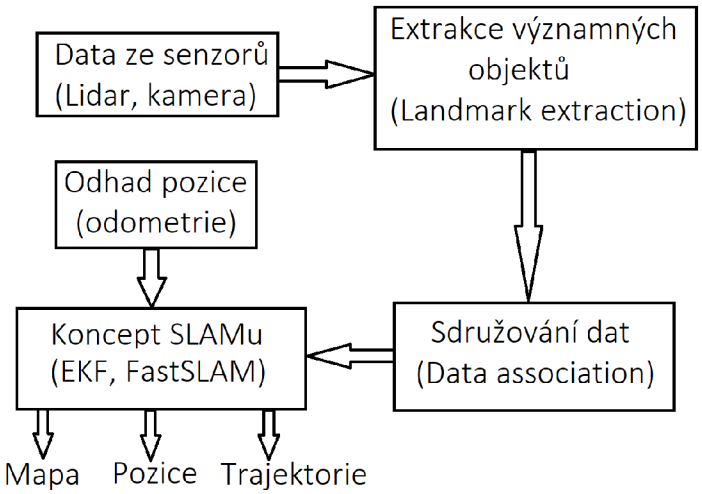
Propojení *Gazeba* a *ROSu* poskytuje množina balíčků s názvem *gezebo_ros_pkgs*. Tyto balíčky vytvoří pomyslnou obálku nad *Gazebem*, které přidávají další prostředky k propojení simulace a *ROSu* prostřednictvím *ROS messages*. Balíčky poskytují značné množství pluginů pro ovládání motorů nebo přidání různých druhů senzorů. K dalším možnostem *gezebo_ros_pkgs* patří dynamické přidání robota (objektu) do prostředí nebo jeho vymazání. [5]

3 SLAM

Jedná se o zkratku slov Simultaneous Localization and Maping. A soustředí se na vytvoření mapy z neznámého prostředí a současnou lokalizaci ve vytvořené mapě [6]. K vytvoření mapy dochází z pozorování okolí různými senzory a z odhadování svého vlastního stavu. *SLAM* je nezbytný pro mobilní roboty pohybující se v neznámém prostředí, ve kterém nejsou k dispozici data o správné aktuální pozici robota, tj. data z GPS [7]. Jedná se o velmi usilovně zkoumanou oblast robotiky.

SLAM může být implementovaný mnoha různými způsoby za použití různých hardwarových a softwarových prostředků. Nejedná se o algoritmus, nýbrž spíše o koncept využívající různé způsoby k dosažení cíle, tj. vytvoření mapy a správné lokalizace. [6]

Proces získávání mapy je ukázán na následujícím obrázku 3.1.



Obr. 3.1: Schéma řešení SLAMu. SLAM algoritmus nejprve získá data ze svého okolí. Ty později zpracovává a porovnává s již naměřenými a zpracovanými daty. Výsledkem je pak mapa, pozice a trajektorie robota.

Prvním krokem k vytvoření mapy je získání dat z okolí robota. Existují převážně dva nejčastěji používané senzory. Prvním z nich je použití kamery. Tento přístup je velice podobný způsobu lokalizace lidí v prostoru. Použití kamery je výpočetně náročné a obtížné na zpracování, ale oproti dalšímu používanému senzoru, tj. laserovému skeneru, nese kamera více informací o svém okolí. Nevýhodou je použití za špatných světelných podmínek nebo tmě. Dalším, již zmíněným, možným typem senzoru, používaného v této práci, je 2D 360° laserový měřič vzdálenosti. [6] Výhodou je jejich přesnost a nenáročné zpracování. Nevýhodou je nemožnost, nebo omezené použití, skeneru v prostředí obsahující větší množství částic ve vzduchu, tj. prach nebo kouř.

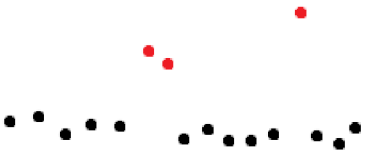
Součástí *SLAMu* je správná lokalizace, a proto dalším vstupem do řešení může být informace o vlastním stavu robota, tj. odometrie (5.3.2). Při pohybu robota se změní data ze senzorů. Z této změny lze vypočítat informace o pohybu. Data z odometrie mohou výsledný odhad pohybu zpřesnit nebo přispět k jeho získání.

3.1 Landmark extraction

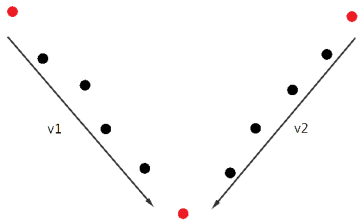
Samotný *SLAM* algoritmus, založený na senzoru *Lidar*, se skládá z několika částí. Jedním z nich je extrakce významných objektů (landmark extraction) z naměřených dat. Jedná se o důležitá místa podle níž se robot později dokáže orientovat. Jsou to rohy místností, geometrické útvary a jiné specifické případy nezapadající do zbytku získaných dat. K vyhledávání význačných bodů je možno použít různé metody. Následující dvě kapitoly popisují základní dva typy význačných bodů a jejich metody.

3.1.1 Spikes algoritmus

Jedná se o algoritmus vyhledávající extrémy, hroty. Vyhledává odlišné hodnoty v jinak spojitém celku (3.2a). Musí se stanovit podmínky pro rozlišení šumu a význačného bodu. [6] Vhodným příkladem může být stůl v místnosti.



(a) Spikes algoritmus



(b) Sliding window algoritmus

Obr. 3.2: Algoritmy na vyhledávání význačných bodů. V prvním případě (a) se jedná o vyhledávání extrémů a v druhém případě (b) algoritmus vyhledává rohy.

Problémem jsou pohybující se objekty (obzvláště osoby), které algoritmus musí rozpoznat a nepřidávat je do význačných rysů. V případě, že se pohybující objekt určí jako významný, může v pozdější fázi dojít k znehodnocení dosavadních výsledků. Aby byl objekt zařazený do význačných rysů, měl by být jednoznačně odlišitelný od ostatních dat, snadno a znovu změřitelný a statický.[6]

3.1.2 Sliding window algoritmus

Algoritmus Sliding window (3.2b), plovoucí okno, slouží k vyhledávání rohů a pracuje se třemi body. Algoritmus si vybere lichý počet bodů, tj. okno, a z nich vybere krajní a prostřední. Obvykle se volí velikost okna mezi 11 až 15 vzorky. Mezi krajními a prostředním bodem se sestrojí dva vektory a vypočte se úhel mezi nimi. [21]

Za roh se obvykle považuje úhel menší než 120°. Problémem mohou být neexistující rohy vznikající tehdy, je li mezi vzorky v plovoucím okně velká vzdálenost.

3.2 Data association

Další částí je asociace dat, ve které dochází k vyhledávání již nalezených význačných bodů v novém pozorování [6].

Na začátku pozorování nejsou k dispozici žádná data, proto se musí vytvořit databáze význačných bodů. Aby se co nejvíce minimalizovaly chyby jsou do databáze zařazeny pouze takové body, které jsou viděny N[1] krát. Při příchodu nových význačných bodů jsou sdruženy s body z databáze. Přiřazení může probíhat na základě různých algoritmů. Nejjednodušší je nearst-neighbor approach [6] nebo ICP [22]. Pokud nový význačný bod odpovídá význačnému bodu z databáze je zvýšen počet pozorování daného bodu, neodpovídá li význačnému bodu je označen za nový a je zvýšen počet pozorování.

3.3 SLAM koncept

3.3.1 EKF SLAM

EKF neboli Extended Kalman Filter je variantou Bayesova filtru [23]. Provádí odhad pozice robota, nebo libovolného systému, pomocí rekurze. Nový odhad stavu robota, v tomto případě polohu, získá na základě odometrie, a tento stav je dále upřesňován z nového pozorování pomoci Kalmanova filtru [24]. Výpočetní náročnost roste s počtem měření [23].

3.3.2 FastSLAM

Je založený na znalosti pozice robota a nezávislém měření význačných bodů v těchto pozicích. Znalost polohy prvního význačného bodu neposkytuje žádné informace o dalších význačných bodech [25]. FastSLAM transformuje SLAM problém na výpočet lokalizace robota z jednotlivých význačných bodů [23] a správná pozice je pak dána

[1] Konstanta větší než nula

nejvyšší pravděpodobností polohy robota, které jsou určeny na základě nezávislého měření význačných bodů. FastSLAM tedy pracuje s množinou různě pravděpodobných trajektorií [25].

4 Robot

Jeden z cílů bakalářské práce je otestovat SLAM algoritmy v simulátoru reálné fyziky. Ke splnění cíle bylo potřeba vybrat a nasimulovat vhodnou mobilní robotickou platformu. V pozemní robotice je nepřeberné množství robotických podvozků, například diferenciální, pásové nebo všesměrové. Pro realizaci byl použit jednoduchý diferenciální dvoukolový robot. V dalších kapitolách je popsána kinematika a model robota.

4.1 Kinematický model

Kinematika se zaobírá geometrií pohybu, především polohu robota v prostoru. Pohyb robota se skládá ze dvou nezávislých kol, které se mohou otáčet v obou směrech. Z tohoto vyplývá, že se střed mezi koly robota pohybuje po kružnici s poloměrem **R**. Mohou nastat tyto případy a nechť v_r značí obvodovou rychlost pravého kola a v_l obvodovou rychlost levého kola a D je rozteč mezi koly.

1. $v_\mathrm{r} = v_\mathrm{l}$, potom se robot pohybuje po kružnici s nekonečným poloměrem (po přímce)
2. $v_\mathrm{r} = \text{-}v_\mathrm{l}$, potom se robot pohybuje po kružnici s nulovým poloměrem (rotace kolem své osy z)
3. $v_\mathrm{r} \neq v_\mathrm{l}$, potom se robot pohybuje po kružnici s poloměrem R (buď po nebo proti směru hodinových ručiček)
4. $v_\mathrm{r} = 0$ nebo $v_\mathrm{l} = 0$, potom se robot pohybuje po kružnici s poloměrem D/2 (rotace kolem osy z pravého, nebo levého kola)

Z této úvahy vyplývá vztah pro poloměr pohybu R. [3]

$$R = \frac{D}{2}\frac{v_\mathrm{l} + v_\mathrm{r}}{v_\mathrm{r} - v_\mathrm{l}} \tag{4.1}$$

Vzhledem k faktu že se jedná o pozemní mobilní robot, pohybující se po rovině, stačí k popisu robota v prostoru 3 souřadnice, kde x a y jsou souřadnice bodu v rovině a θ je úhel natočení.

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} \tag{4.2}$$

Lineární rychlost v robota je dána součtem obvodových rychlostí obou kol děleno dvěma [4].

$$v(t) = \frac{v_{\mathrm{r}}(t) + v_{\mathrm{l}}(t)}{2} \tag{4.3}$$

A úhlová rychlost ω robota [4].

$$\omega(t) = \frac{v_{\mathrm{r}}(t) - v_{\mathrm{l}}(t)}{\mathrm{D}} \tag{4.4}$$

Přepočtené rychlosti do globálního souřadnicového systému (podrobněji popsáno v podkapitole 5.1).

$$\begin{pmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\Theta}(t) \end{pmatrix} = \begin{pmatrix} cos[\theta(t)] & 0 \\ sin[\theta(t)] & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v(t) \\ \omega(t) \end{pmatrix} \tag{4.5}$$

Pozici $\mathbf{p}$ dostaneme integrací rychlostí $\dot{x}$, $\dot{y}$ a $\dot{\theta}$ podle času.

$$x(t) = \int_0^t v(t) cos[\theta(t)] dt \tag{4.6}$$

$$y(t) = \int_0^t v(t) sin[\theta(t)] dt \tag{4.7}$$

$$\theta(t) = \int_0^t \omega(t) dt \tag{4.8}$$

Kinematický model robota je důležitý pro správnou lokalizaci robota, viz 5.3.2.

4.2 Simulace modelu robota

V předchozí kapitole je popsán kinematický model robota. Znalost tohoto modelu je esenciální pro řízení rychlosti a pohybu robota. V této kapitole je popsána struktura a vlastnosti simulovaného modelu.

4.2.1 URDF

Tvorba modelu kompatibilního s *ROSem* je poměrně jednoduchá. Existuje k tomu určený formát URDF (Unified robot description format), což je variace XML pro ROS [2]. *URDF* může být rozšířen o *XACRO* (XML Makro), a jak už název napovídá, jedná se o použití maker pro usnadnění modifikace (obvykle rozměry, hmotnost, nebo limity). Tento formát popisuje všechny součásti robota nebo jakéhokoli modelovaného systému. U každé součásti definuje jeho vzhled a tvar. Popisuje dynamiku prostřednictvím kloubů, hmotností, tření a setrvačnosti.

4.2.2 Model robota

Model simulovaný v této práci je popsaný v *URDF* formátu. Skládá se z těla, dvou rotujících kol, podpěrného statického kola a 2D 360° laserového měřiče vzdálenosti. Všechny prvky lze spojit s dalšími částmi modelu, buďto staticky (pevnou vazbou) nebo pohyblivě (kloubem). Speciálním prvkem v tomto modelu jsou právě klouby, kterými jsou spojeny kola s tělem. Tyto klouby jsou virtuálními motory, kterými lze ovládat robota prostřednictvím *ROSu* v *Gazebu*. Jedná se o plugin, který popisuje jaký typ pohonu kloub bude používat. Tento plugin poskytuje *topic*, přes který lze poslat *message* s polohou kloubu, nebo rychlostí (závisí na typu pohonu). Dále je možnost nastavit maximální rychlost, úsilí, tření nebo prokluz kloubu. Každý prvek robota, v tomhle případě jednoduchý geometrický útvar, se skládá ze tří částí, tj. vizuální, kolizní a fyzikální vlastnosti.

Vizuální a kolizní vlastnosti

Vizuální vlastnosti se v *URDF* formátu zapisují do značek *<visual>*, kde se pro každý prvek popíší geometrické vlastnosti útvaru a pozice vzhledem k základně robota. Kolizní vlastnosti, zapisované do značek *<collision>* jsou v tomto případě stejné jako vizuální, ale u složitějších modelů může být kolizní model robota zjednodušením vizuální součásti robota.

Fyzikální vlastnosti

Fyzikální vlastnosti spolu s kolizním modelem určují chování modelu v simulaci a zapisují se do značek *<inertial>*, kde se primárně definuje hmotnost a momenty setrvačnosti. Ty se pro jednotlivé prvky definují tenzorem setrvačnosti $\mathbf{I}$.

$$\mathbf{I} = \begin{pmatrix} I_{x,x} & I_{x,y} & I_{x,z} \\ I_{y,x} & I_{y,y} & I_{y,z} \\ I_{z,x} & I_{z,y} & I_{z,z} \end{pmatrix} \tag{4.9}$$

Pro kvádr reprezentující tělo robota se šířkou w, výškou h, hloubkou d a hmotností m je tenzor setrvačnosti

$$\mathbf{I_{kvádr}} = \begin{pmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{pmatrix} \tag{4.10}$$

[8]

Pro válec reprezentující kola robota s průměrem r, výškou h a hmotností m je tenzor setrvačnosti

$$\mathbf{I}_{\mathbf{válec}} = \begin{pmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{pmatrix} \tag{4.11}$$

[8]

Pro podpěrné kolo s hmotností m a poloměrem r je tenzor setrvačnosti následující

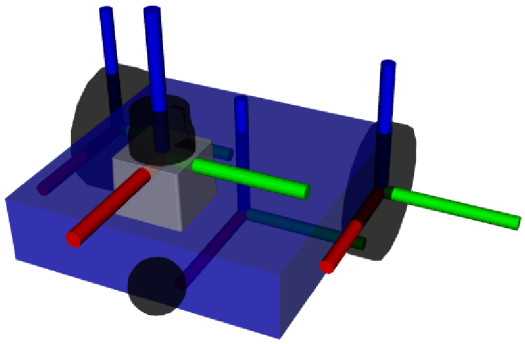
$$\mathbf{I}_{\mathbf{koule}} = \begin{pmatrix} \frac{2}{5}mr^2 & 0 & 0 \\ 0 & \frac{2}{5}mr^2 & 0 \\ 0 & 0 & \frac{2}{5}mr^2 \end{pmatrix} \tag{4.12}$$

[8]

Vzhledem k zanedbatelnému tření oproti kolům má podpěrné kolo v simulaci nastavené nulové tření. Při nesprávně nastavených parametrech dochází k prokluzu kola vzhledem k povrchu, nebo skákání robota při rozjezdu.

Na obrázku 4.1 je zobrazen takto vytvořený model. Aby robot nevypadal tak fádně lze geometrické tvary vizuálně nahradit 3D objekty (například kola) zkonstruovanými v libovolném grafickém editoru.

Orientace hlavních prvku robota je ukázána na obrázku 4.1, kde směr zeleného vektoru odpovídá y-ové ose, červeného x-ové ose a modrého z-ové ose. Nastavení orientace je velice důležité, neboť je potřeba nastavit kola tak, aby se při kladném otočení robot pohyboval dopředu. Ovšem je to jen konvence, kterou není potřeba dodržovat, ale ušetří značné množství práce.



Obr. 4.1: Na obrázku jsou znázorněny jednotlivé komponenty z nichž se model v simulaci skládá a je ukázána jejich trajektorie.

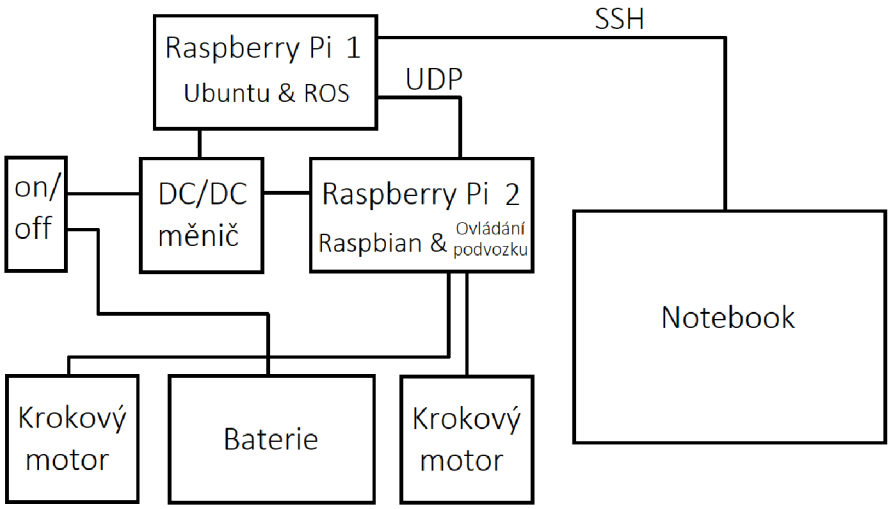
Důležitá je orientace těla robota, zobrazená mezi koly, která určuje směr pohybu.

4.3 Fyzická platforma

Část zadání bakalářské práce se zabývá testováním vybraných *SLAM* algoritmů v reálném prostředí. V této kapitole je popsána schématická struktura robota, vlastnosti senzoru *Lidar* a komunikace s řídicím počítačem.

4.3.1 Struktura robota

Robot se skládá z několika částí, jak je vidět na obrázku 4.2. První z nich je motorická část obsahující dva krokové motory vykonávající pohyb robota. Další částí je senzorická jednotka obsahující *Lidar*, kterému je věnována kapitola RPLIDAR A1 (4.3.2), a virtuální enkodéry, které poskytují informace o natočení kol.



Obr. 4.2: Schématické znázornění jednotlivých komponent při využití reálně robotické platformy.

Robot samozřejmě obsahuje i výpočetní jednotku skládající se ze dvou jednodeskových počítačů Raspberry Pi (dále RPi). Na RPi2 běží operační systém Raspbian a program, který řídí pohyb kol a poskytuje informace o jejich natočení. Tento počítač pracuje v read-only režimu aby se zabránilo zničení paměťové karty při vypínání robota. Na RPi1 je nainstalován operační systém Ubuntu 19.04 a ROS. Všechny implementované algoritmy, které jsou popsány v kapitole Software (5), běží právě na tomhle počítači. RPi1 posílá UDP pakety s informacemi o rychlostech kol počítači RPi2. Viz kapitola Komunikace (4.3.3).

4.3.2 RPLIDAR A1

RPLIDAR A1 je nízkonákladový 2D 360° měřič vzdálenosti. Snímač vysílá infračervené laserové paprsky, které se odrážejí od překážek a následně se vrací k přijímači. Z časového intervalu od vypuštění paprsku po jeho přijetí se vypočítá vzdálenost překážky od které se paprsek odrazil. Snímač se otáčí s frekvencí 5.5Hz a v jednom kole změří 360 bodů. Za sekundu jich změří přibližně 2000. Data jsou přes USB sériově přenášena do počítače. Parametry senzoru ukazuje následující tabulka. Grafická vizualizace dat je znázorněná na obrázku 2.2. [9]

Tab. 4.1: Parametry senzoru RPLIDAR A1

	Min	Typicky	Max
Rozsah vzdáleností	0.15m	-	6m
Úhlový rozsah	0°	-	360°
Přesnost měření	<0.5% ze vzdálenosti do 1.5m	-	<1% ze vzdálenosti
Frekvence vzorkování	-	>= 2000Hz	-
Frekvence otáčení	1Hz	5,5Hz	10Hz
Vlnová délka	775nm	785nm	795nm

4.3.3 Komunikace

Hlavní výpočetní jednotkou robota je RPi1 z obrázku 4.2. Mezi počítačem RPi1 a RPi2 probíhá UDP komunikace. Kdy RPi1 se chová jako UDP klient, který odesílá UDP pakety s informacemi o rychlosti levého a pravého kola, nebo může poslat paket s žádostí o vrácení dat s informacemi z virtuálních enkodérů. RPi2 je UDP server, který přijímá pakety odeslané klientem. V případě žádosti o změnu rychlosti jednotlivých kol server roztočí krokové motory na požadovanou rychlost. Pokud obdrží paket s žádostí o poslání dat z enkodéru, pošle klientu paket s vrácenou odometrií. Paket s informací o rychlostech jednotlivých kol vypadá následovně

```
$echo "spd,leve_kolo,prave_kolo" > /dev/udp/IP_AddServeru/PORT
```

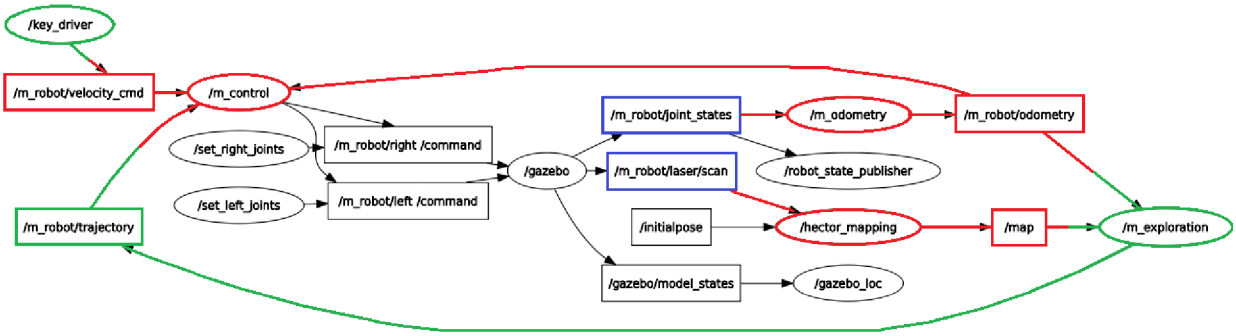
a paket se žádostí o data z enkodérů:

```
$echo "odm" > /dev/udp/IP_AddServeru/PORT
```

IP adresa serveru, tedy počítače RPi2, je nastavená jako statická, aby se nemusel v počítači RPi1 při každém použití zvlášť nastavovat. Komunikace probíhá na portu 8080.

5 Software

V této kapitole je popsán nejnutnější software, který je spolu spolu se *SLAM* algoritmy potřebný k vytvoření mapy budovy. Kapitola se dále věnuje algoritmu k autonomnímu průzkumu budovy a vizualizaci dat. Cílem práce je otestování *SLAM* algoritmů v simulaci a reálném prostředí, proto je celý systém rozdělený tak, aby nebyl problém při přecházení mezi simulací a realitou. Na následujícím obrázku 5.1 jsou schématicky znázorněny všechny komponenty celého systému, běžícího v simulaci, kde elipsa symbolizuje *node*, obdélník je *topik* a směr šipek určuje směr toku dat.



Obr. 5.1: Na obrázku je znázorněn soupis všech nodů. Červenou barvou je znázorněn potřebný software pro tvorbu mapy. Zelená barva znázorňuje způsob pohybu robota (tj. ovládání nebo autonomní pohyb).

Centrem celého systému je *node gazebo*, který vytváří data z *LIDARu* a poskytuje úhel natočení jednotlivých kol. Model v simulátoru *Gazebo* je možné ovládat prostřednictvím *topiků /m_robot/left(right)/command*. V reálném světě by tento *node* nahradil robot a reálná data ze senzorů.

Červenou barvou na obrázku 5.1 je vyznačena část systému potřebná k vytvoření mapy. Z obrázku 5.1 jsou to pouze *nody m_control* zabezpečující pohyb robota, *m_odometry* poskytující základní lokalizaci, což některé *SLAM* algoritmy vyžadují a samotný *"SLAM" node* vytvářející mapu (na obrázku 5.1 *hector_maping*).

Pro průzkumu budovy je možné zvolit manuální mód, kdy uživatel ovládá robota prostřednictvím klávesnice, nebo autonomní mód pro průzkum budovy robotem. Na obrázku 5.1 vyznačeno zeleně. Režimy ovládání lze dynamicky přepínat, což zvyšuje bezpečnost robota pracujícím v autonomním módu.

Následující kapitoly se zabývají dílčími částmi softwaru.

5.1 Transformace

Aby mělo smysl jakkoli pracovat s pozicí robota v prostoru (rovině) je potřeba definovat globální souřadnicovou soustavu. S robotem je možno svázat druhou souřadnicovou soustavu, tj. lokální souřadnicovou soustavu. Její pozice ke globální souřadnicové soustavě je určena jednoznačně, pokud má zadefinované tři souřadnice

$$\mathbf{p} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \tag{5.1}$$

a je definována také orientace bodu **p**. Orientace může být definována několika způsoby. Nejběžnější a nejpřirozenější popis je pomocí Eulerových úhlů [10]. Matice rotace **R** se s využitím Eulerových úhlů vytvoří součinem rotace soustavy kolem osy z,

$$\mathbf{R_z}(\phi) = \begin{pmatrix} cos(\phi) & -sin(\phi) & 0 \\ sin(\phi) & cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.2}$$

rotace kolem nové x'-ové osy a

$$\mathbf{R_{x'}}(\theta) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) \\ 0 & sin(\theta) & cos(\theta) \end{pmatrix} \tag{5.3}$$

rotace kolem nové z"-ové osy lokální souřadnicové soustavy.

$$\mathbf{R_{z''}}(\psi) = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.4}$$

$$\mathbf{R} = \mathbf{R_z}(\phi)\mathbf{R_{x'}}(\theta)\mathbf{R_{z''}}(\psi) \tag{5.5}$$

Násobení matic je výpočetně poměrně náročné, a proto dalším způsobem jak definovat orientaci v prostoru je pomocí kvaternionu, u nichž odpadají náročné maticové operace. V standardních *zprávách ROSu* se pro orientaci používají právě kvaterniony.

Kvaternion [10] je uspořádaná čtveřice čísel ve tvaru $q_0 + q_1 i + q_2 j + q_3 k$, kde q_i jsou reálná čísla a pro symboly i, j, k platí následující identity.

$$i^2 = j^2 = k^2 = -1; \tag{5.6}$$

$$ij = k, ji = -k \tag{5.7}$$

$$jk = i, kj = -i \tag{5.8}$$

$$ki = j, ik = -j \tag{5.9}$$

Jak je vidět násobení symbolů i, j, k není komutativní.

Bod ve třech dimenzích je možno vyjádřit jako tzv. ryzí kvaternion, který nemá reálnou složku $q = 0 + xi + yj + zk$. Rotace v prostoru je možno vyjádřit jako kvaternion q_r s normou $||q_r|| = 1$. Rotaci z jednoho souřadnicového systému P do druhého D vyjadřuje součin, kde q_r^* je sdružený kvaternion $q_r^* = q_0 - q_1 i - q_2 j - q_3 k$

$$q_p = q_r q_d q_r^* \tag{5.10}$$

Geometrická reprezentace kvaternionů je poměrně složitá, ale je uveden příklad rotace pomocí komplexních čísel.

Komplexní číslo $p = x + iy$ může reprezentovat bod v rovině. Komplexní číslo je možno zapsat v exponenciálním tvaru $p = |p|e^{i\theta}$. Podle Eulerova vzorce jde $e^{i\theta}$ přepsat jako:

$$e^{i\theta} = (cos\theta + isin\theta) \tag{5.11}$$

Vynásobením

$$e^{i\theta}(x + iy) = [cos(\theta) + isin(\theta)](x + iy) = [xcos(\theta) - ysin(\theta)] + i[xsin(\theta) + ycos(\theta)] \tag{5.12}$$

vzniknou souřadnice nového bodu zapsány maticově,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} \tag{5.13}$$

což odpovídá známe matici rotace v rovině.

V globální souřadnicové soustavě již lze definovat libovolný objekt. Příkladem může být robot, mapa nebo překážka.

V *ROSu* je možné tyto soustavy vytvářet pomocí knihovny *tf* [2]. Soustavy mohou být vzhledem k sobě statické, příkladem je pozice *Lidaru* a robota, nebo pohyblivé, tj. vztah robot a globálního souřadnicového systému.

5.2 Ovládání

Manuální ovládání je realizováno prostřednictvím klávesnice, kdy uživatel řídí robota klasickými klávesami pro pohyb 'w', 'a', 's', 'd'. Uživatel má možnost zvyšovat a snižovat rychlost robota i v průběhu ovládání.

Program čeká na stisknutí kláves a podle nastavené rychlosti robota se posílají data do *nodu m_control* na */m_robot/velocity_cmd* (obrázek 5.1).

5.3 Lokalizace

Lokalizace robota je jeden ze základních pilířů robotiky, a proto jedna z lokalizačních technik byla implementována i v tomto systému (5.3.2). Jedná se o odhad pozice a orientace robota v rovině. Znalost správné pozice robota je prvním nutným předpokladem k autonomní činnosti robota, neboť bez znalosti přesné pozice není možné plánovat trajektorii a pohybovat se po ní. V následujících kapitolách jsou popsány typy základních lokalizačních technik a jejich výhody a nevýhody.

5.3.1 Rozdělení lokalizačních technik

Mobilní roboti, tj. stroje se schopností pohybu, se dají rozdělit podle stupně autonomie. Existují roboti, kteří jsou dálkově ovládáni (teleoperace) a veškeré jejich vlastnosti jsou určené schopnostmi jejich operatéra. Takoví roboti nepotřebují žádnou lokalizační techniku a jejich cílem je provádět obdržené požadavky. Dalším typem robotů jsou semi-autonomní roboti, kteří mohou být ovládáni operatérem, ale mohou také přejít do autonomního módu. Příkladem práce v autonomním módu může být vyhnutí se překážce, nebo pokračování v úloze při ztrátě signálu s operatérem. Roboti s nejvyšším stupněm autonomie rozhodují o svém chování samostatně. Autonomní a semi-autonomní roboti potřebují ke své správné funkci znát svou aktuální polohu. [11]

Relativní lokalizace

Algoritmy relativní lokalizace jsou založené na odhadu změny polohy a orientace robota tzv. *dead reckoning* [11]. Nová pozice se odhaduje na základě předchozí polohy a informacích o změně. Může se jednat o změnu natočení kol. Takto získaná poloha může být zpřesňována pomocí akcelerometrů a gyroskopů. Pokud je známá počáteční poloha na počátku relativní lokalizace je možné získat absolutní odhad pozice v prostoru. Takovýto odhad je ale zatížen integrační chybou, která znemožňuje používání relativní lokalizace na delší dobu (viz. 5.3.2).

Absolutní lokalizace

Počáteční pozice robota může být neznámá a robot svou pozici odhaduje na základě prostředí ve kterém se pohybuje. Problém absolutní lokalizace je algoritmicky složitější než relativní lokalizace. Její výhodou je ovšem použitelnost v delším časovém intervalu, jelikož není zatížená integrační chybou jako relativní lokalizace.

Pasivní lokalizace

Pasivní lokalizace se používá ve většině autonomních robotů. Jedná se o odhadování pozice pouze na základě informací ze senzoru a nikoli vlastním aktivním pohybem za účelem minimalizace chyby.

Aktivní lokalizace

Jedná se o skupinu algoritmů, které jsou schopny převzít kontrolu nad robotem a vyhýbat se tak místům, kde by mohlo dojít velké chybě v odhadu pozice. Aktivní lokalizace dosahuje obvykle lepších výsledku než lokalizace pasivní.

5.3.2 Odometrie

Odometrie je známá lokalizační technika, která je závislá na znalosti rozměrů robota a typu podvozku. Jedná se především o relativní lokalizační techniku, jelikož je zatížena malou chybou při krátkodobém používání (viz. 5.3.2). V práci je odometrie použita jako absolutní lokalizační technika, jelikož je známá počáteční pozice robota a přesnost je v krátkodobém horizontu dostačující vzhledem ke své jednoduchosti.

Princip výpočtu

Slovo odometrie je složenina dvou řeckých slov *hodos* (cestovat, cesta) a *metron* (měřit) [12]. Jak je patrné z původu slova, celý algoritmus spočívá v odečítání informací o pohybu kol a pomocí kinematického popisu robota (4.1) lze odhadnout jeho orientaci, rychlost a pozici.

Veškerý software v této práci je implementovaný tak, aby se dalo jednoduše přecházet mezi simulací a realitou. V simulaci se informace o pohybu kol získávají přímo z *topiku*, který obsahuje data o všech modelech v simulaci. Tento *topik* publikuje přímo *Gazebo*. Při reálném použití se používají enkodéry ke zjištění pozice hnacích kol. Robot použitý v této práci poskytuje virtuální data plynoucí z požadavku na pohyb robota. Získané informace se posílají na příslušný *topik*, ze kterého se později vypočítá odometrie. Výpočet je ukázán na následujícím algoritmu. Odometrie je výpočetně jednoduchá a nenáročná, proto se velmi často používá.

Chyby odometrie

Chyba odometrie vzniká za předpokladu, že změna pozice kola je nepřesně transformována na pohyb robota. Takové chyby vznikají přirozeně a nejde jim zabránit. Jedná se o tzv. systematické chyby vznikající z nepřesného měření nebo neideálností pohybu. Systematické chyby vznikají hlavně následujícími způsoby.

- Nepřesným měřením úhlu natočení kol (chyba se neprojevuje v simulaci)

Algoritmus 1: Výpočet odometrie

Data: Úhel levého kola L, pravého R, čas příchodu dat T
Result: Rychlost robota V, úhlovou rychlost W, pozice P, orientace Theta
/* -Proměnné s indexem M (LM, WM ...) označují stejnou proměnnou zpožděnou o jeden krok
-P obsahuje x-ovou a y*ovou složku podle (4.6 a 4.7) */

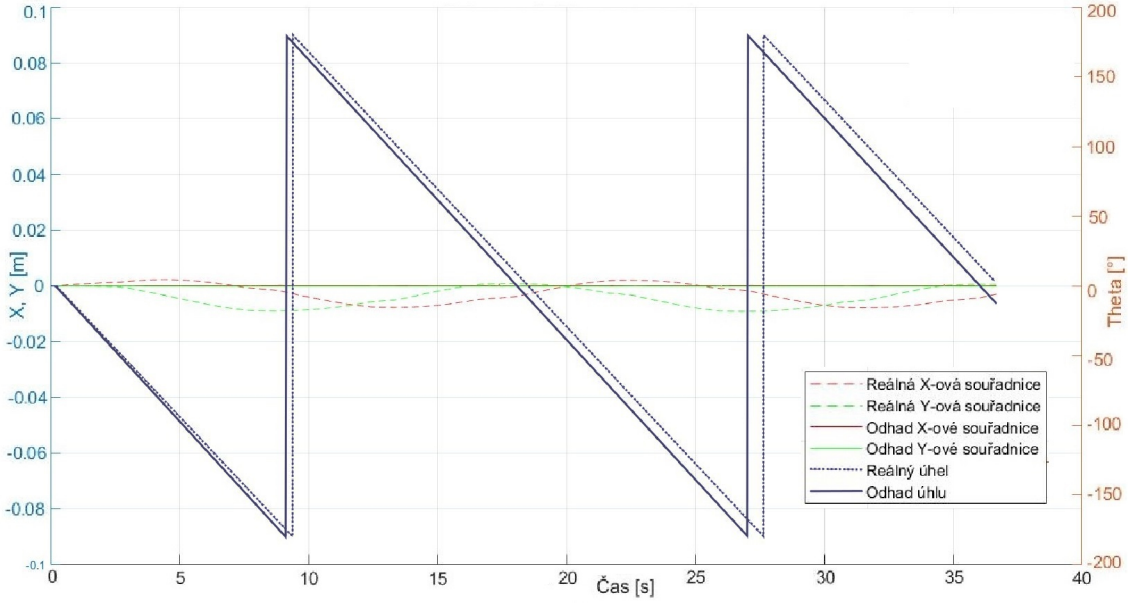
while *true* **do**
```
/*Udržení minulých dat*/
LM, RM, WM, VM, TM ⟵ L, R, W, V,T;

/*Nová data*/
L, R, T ⟵ EnkoderyData();

/*Výpočet nových rychlosti jednotlivých kol, r- poloměr kola*/
rychlostL, rychlostP ⟵ r*(L,R - LM,RM)/(T - TM);

/*Výpočet nové rychlosti robota, D - roztěč robota*/
V ⟵ (rychlostR + rychlostL)/2;
W ⟵ (rychlostR - rychlostL)/D;

Theta ⟵ Theta + KrokNumerickéIntegrace(W - WM);
P ⟵ P + KrokNumerickéIntegrace(V - VM);

P, Theta, V, W ⟵ Topic: /m_robot/odometry;
```
end while

- Nepřesným měřením průměru kola
- Numerickou integrací a derivací (tj. konečná vzorkovací frekvence)
- Nenulovou styčnou plochou kol (obrázek 2.1a)

Systematické chyby rostou lineárně. Velikost jejich směrnice záleží na typu pohybu. Jestliže se robot otáčí pouze kolem své osy, potom systematická chyba způsobená různými nepřesnostmi roste pomaleji, nežli chyba způsobená při rovnoměrně přímočarém pohybu.

Vývoj chyby v čase při rotaci kolem své osy znázorňuje následující obrázek 5.2. Je na něm patrná chyba způsobená nenulovou styčnou plochou kol. Při otáčení kolem své osy robot pouze mění svou orientaci. To je vidět z obrázku 5.2, že souřadnice x a y vzniklé odhadem odometrie jsou rovny nule. Reálné souřadnice robota jsou odečteny ze simulace, takže jsou dokonale přesné a odometrie pracuje s přesnými

hodnotami (narozdíl od reality). Z obrázku je vidět, že pozice robota v simulaci osciluje kolem počátečního bodu. Tento problém plyne z prokluzu kol a nenulové styčné plochy kol s povrchem. A dokazuje existenci systematických chyb.



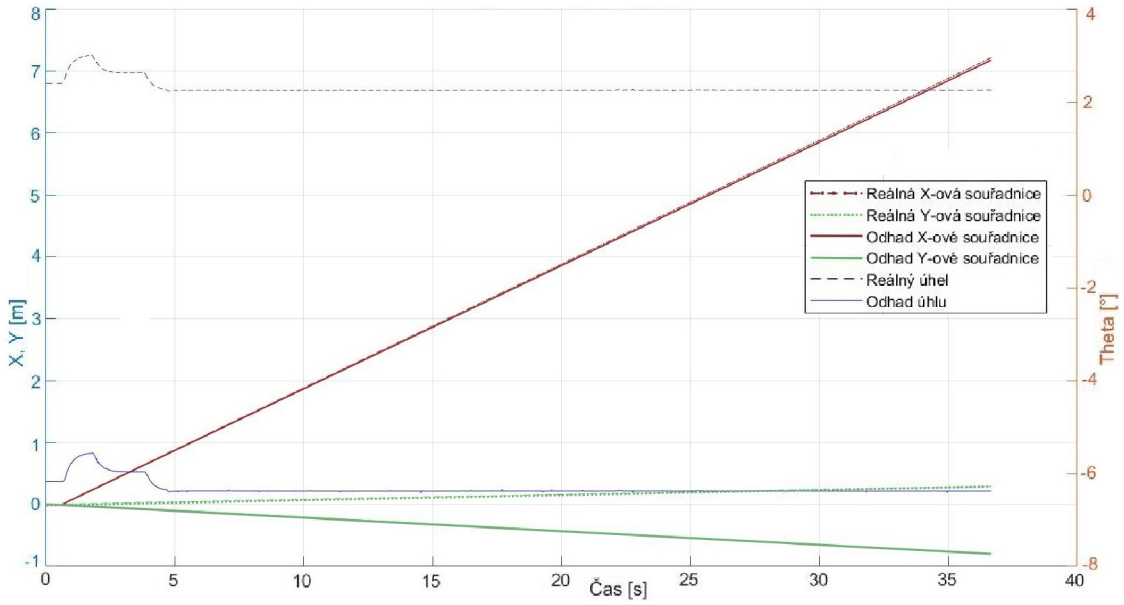
Obr. 5.2: Nárůst chyby způsobený rotací kolem své osy roste pomalu s prodlužující se délkou pohybu. Odhad úhlu (plná čára) a reálný úhel (tečkovaná) je po dlouhou dobu bez jakékoli korekce správný.

Při získáváni dat z následujícího obrázku (5.3) byla úmyslně zavedena chyba odhadu orientace robota, před začátkem pohybu, oproti simulaci přibližně 8.5° a odhad y-ové souřadnice byl stejný jako reálná y-ová souřadnice. Po 36 sekundách se změnila reálná y-ová souřadnice o 0.29m, tj. robot se pohyboval téměř kolmo k y-ové ose. Odhad této souřadnice po 36s pohybu byl -0.79m. Rozdíl reálné a odhadnuté souřadnice byl 1.08m. Chyba byla 372% ze změny y-ové souřadnice. Z grafů je vidět, že i při malé odchylce odhadu od reality je pro větší uražené vzdálenosti odometrie nespolehlivá.

Dalšími chybami jsou chyby náhodné. Systematické chyby lze odhadnou a případně i korigovat, ale náhodné chyby jsou nevypočitatelné. Jedná například o

- nerovnosti v povrchu
- srážka s předmětem
- prokluz kola

Náhodné chyby často vedou k nevratné ztrátě (za použití odometrie) orientace robota.



Obr. 5.3: Nárůst chyby způsobený rovnoměrným přímočarým pohybem se zavedenou chybou odhadu úhlu rychle roste s prodlužující se délkou pohybu. Odhad Y-ové souřadnice (plná, zelená) a reálná Y-ová souřadnice (tečkovaná, zelená) je po delší době bez jakékoli korekce nepoužitelný.

5.4 Autonomní průzkum

Robot pouze za pomoci některého *SLAM* algoritmu a znalosti odometrie (není nutné pro všechny *SLAM* algoritmy) dokáže vytvořit mapu svého okolí. Pohyb, a tedy i schopnost vytvořit mapu daného prostředí, je přenechán na operatéra, který s robotem bude pohybovat tak, aby vytvořil mapu celého prostředí (příkladem může být budova). Tento přístup s sebou přináší řadu výhod a nevýhod. Výhodou může být větší spolehlivost a jednoduchost. Nevýhodou je nutná přítomnost operatéra v blízkosti robota. V některých případech je ovšem nemožné s robotem pohybovat z bezpečného místa, a proto je výhodné použít právě autonomního robota.

V kapitole Lokalizace (5.3) bylo zmíněno, že autonomní robot ke své funkci potřebuje znát svou polohu. Znalost polohy byla zajištěna některými *SLAM* algoritmy, nebo odometrií, nebo jejich kombinací. K autonomnímu pohybu je potřeba rozhodovat kam robot pojede a jakou trajektorii zvolí. Právě plánováním trajektorie se zabývají následující kapitoly.

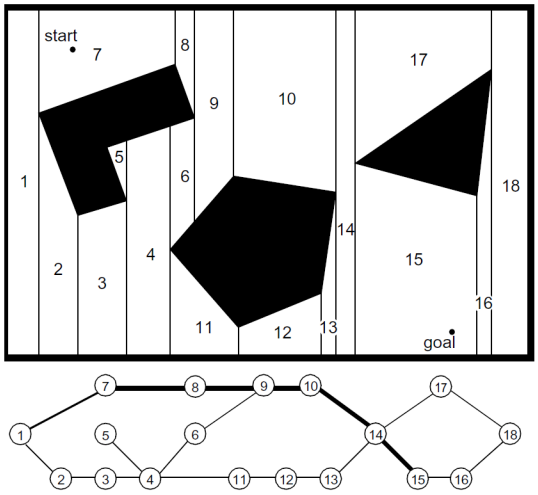
5.4.1 Plánování trajektorie

Aby mělo smysl plánovat trajektorii robota, je potřeba mít alespoň část mapy prostředí, ve které robot zná svou pozici. Mapu vytvoří běžící *SLAM* algoritmus.

Algoritmus na autonomní průzkum prostředí implementovaný v této práci používá mapu typu mřížka obsazenosti. Jedná se o takovou mapu, kde je celá mapa rozdělena na stejně velké buňky (pixely) vytvářející mřížku. Jedná se o aproximaci reálného spojitého prostředí a velikost mřížky určuje přesnost aproximace. Každý pixel $p_{i,j}$ obsahuje hodnotu náležící do intervalu <0,100>, nebo -1 pro neznámý pixel. Hodnota reprezentuje pravděpodobnost obsazenosti, kde 100 znamená obsazeno.

Mapa je rozdělena na množiny $\boldsymbol{O}$ obsahující všechny obsazené buňky, množinu $\boldsymbol{V}$ obsahující všechny průchodné buňky a množinu $\boldsymbol{N}$ obsahující všechny neznámé prvky. Rozdělení pixelů mezi $\boldsymbol{O}$ a $\boldsymbol{V}$ je možné měnit podle velikosti pravděpodobnosti, která je považována za důvěryhodnou.

Za plánování trajektorie je považována taková činnost hledající posloupnost sousedních prvků mřížky obsazenosti náležící do množiny $\boldsymbol{V}$ tak, aby se propojila pozice robota s cílovým bodem. Trajektorii lze získat mnoha způsoby z nichž jsou některé popsané v následujících podkapitolách.



Obr. 5.4: Buněčná dekompozice [13]

Buněčná dekompozice

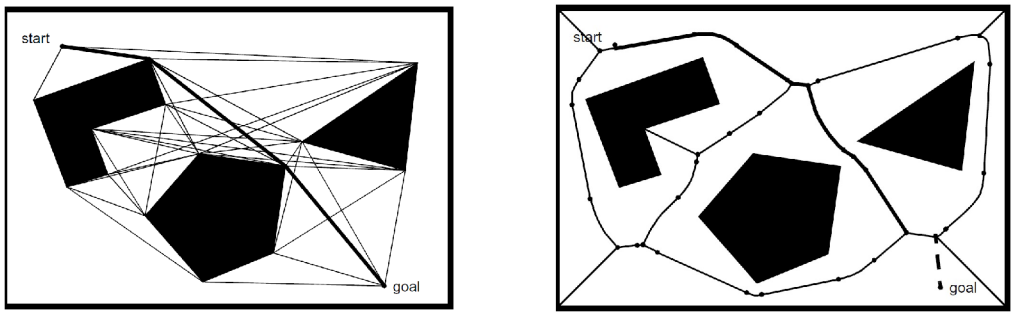
Dekompozice buněk [13] má za cíl rozdělit množinu volných pixelů $\boldsymbol{V}$ do buněk, ve spojitém prostředí obecně do konvexních čtyřúhelníků, tak, aby buňky spolu sousedily (horní část obrázku 5.4). Sousedící buňky vytvoří graf sousednosti (spodní část obrázku 5.4), ve kterém se snadno nacházejí nejefektivnější posloupnosti buněk. Trajektorie spojující počáteční a koncový bod musí procházet právě těmito buňkami.

V grafu sousedností mohou být hrany ohodnocené, nebo neohodnocené. Podle toho jsou vybírány algoritmy na prohledávání. Nejběžnější algoritmy na prohledávání

s neohodnocenými hranami je *prohledávání do hloubky* [14] a *prohledávání do šířky* [15]. S ohodnocenými grafy pracuje *Dijkstrův algoritmus* [16].

Exaktní mapy cest

Mapa cest, nebo známěji *road map* [13], již nepracuje s množinami oblastí, ale s množinami všech cest. Cílem je najít minimální síť tak, aby se jimi pokryla většina volného prostoru (5.5). Z takové sítě se podle potřeby vybírají trajektorie, které spojí počáteční a koncový bod.



(a) Graf viditelnosti (b) Voroného diagram

Obr. 5.5: Road map algoritmy [13]

Jedním z algoritmů je *graf viditelnosti* (5.5a). *Graf viditelnosti* [13] je oblíbený a jednoduchý algoritmus, který pracuje s množinou polygonů. Všechny vrcholy polygonů, počáteční a koncový bod se propojí se všemi vrcholy, mezi nimiž není překážka, tj. vidí na sebe. Algoritmus je velice efektivní a rychlý pro prostředí obsahující malé množství překážek. Nevýhodou je ta vlastnost, že nalezená cesta se pohybuje v blízkosti překážek, což zvyšuje riziko kolize.

Algoritmus hledající cestu tíhnoucí k maximalizaci vzdálenosti překážek od trajektorie je *Voroného diagram* [13]. Každá překážka kolem sebe vytvoří oblast, jejíž body se nacházejí nejblíže k překážce ležící v dané oblasti. Jak ukazuje obrázek 5.5b vzniknou hranice mezi oblastmi, po kterých se robot bude pohybovat.

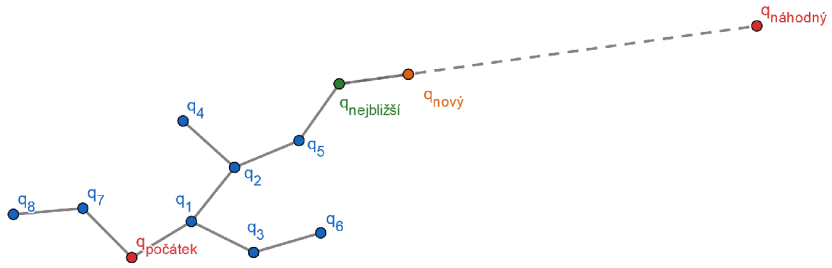
Pravděpodobnostní mapy cest

Dalším přístupem k plánování trajektorie je pravděpodobnostní přístup. Jedná se opět o algoritmus pracující s mapou cest, kdy se nejprve vygenerují cesty a později se hledají trajektorie ve vytvořené síti cest. Cesta se vytváří náhodně vybranými body (splňující dané podmínky), které se postupně přidávají k počátečnímu bodu a tak se cesty průběžně budují. Algoritmy jsou rychlé pro statická a i relativně komplikovaná prostředí (viz 5.4.4).

Algoritmus vybraný pro autonomní průzkum je popsaný v další kapitole.

5.4.2 RRT algoritmus

Jedná se o pravděpodobnostní přístup k plánování trajektorie. *RRT* neboli *Rapidly-exploring Random Tree* [17] používá k prohledávání v množině volného prostoru $\boldsymbol{V}$ rychle rostoucí strom. Jedná se o strukturu, která je zobrazena na obrázku 5.6.



Obr. 5.6: RRT algoritmus. Stromová struktura. Nové větve se přidávají na základě výběru bodu $q_{náhodný}$. Nová větev je přidána do struktury při splnění podmínek.

K vytvoření stromu je nutné definovat počáteční bod, který je většinou umístěn do počátečního bodu robota, tj. $q_{počátek}$. K tomuto bodu jsou připojovány další uzly vznikající z náhodně vygenerovaného bodu $q_{náhodný}$. Nový bod vznikne tak, že se $q_{náhodný}$ spojí přímkou s $q_{nejbližší}$ a za splnění podmínek (dále popsány v podkapitole 5.4.3) je $q_{nový}$ připojen do stromu v definované vzdálenosti od $q_{nejbližší}$, jinak se volí nový $q_{náhodný}$.

Výhodou algoritmu je velmi rychlá konvergence k cílovému bodu i v poměrně složitém prostředí. Algoritmus se dá požít v prostoru libovolné dimenze, nejčastěji druhého a třetího řádu. Nevýhodou je poměrně kostrbatá trajektorie, která se dá upravit a vyhladit tak, aby byla reálně použitelná (viz kapitola 5.5). Další nevýhodou je ta, že v určitých prostředích algoritmus nemusí konvergovat, nebo jen velmi pomalu (viz 5.4.4)

5.4.3 Implementace RRT algoritmu pro autonomní průzkum

Vlastní implementace *RRT* algoritmu, jak znázorňuje algoritmus 2, je rozdělena do tří kroků. Algoritmus pro svou práci potřebuje znát mapu prostoru, kterou poskytne *SLAM* a taky svou polohu v mapě. V případě, že by neznal svou polohu v mapě, algoritmus by nemohl expandovat z počátečního bodu $q_{počátek}$ (obrázek 5.6) do koncového. Z toho vyplývá první krok algoritmu, tj. určit správně počáteční polohu a především určit cíl prozkoumávání (nastavit cílový bod). O vyhledávání cílových bodů pojednává podkapitola z 5.4.3 Vyhledávání cílových bodů.

Algoritmus 2: RRT Algoritmus

```
Data: Mapa M, Odometrie O
Result: Strom S, Trajektorie T
while true do
    if dataOdometrie and dataMapa then
        lokálníKoncovýBod ⟵ LokálníKoncovýBod();
        if not lokálníKoncovýBod then
            globalníKoncovýBod ⟵ GlobálníKoncovýBod();
        end if
        náhodný ⟵ M.NáhodnýBod();
        nejbližší ⟵T.NajdiNejbližšíUzel(náhodný);
        nový ⟵ PodmínkyProUzel(tolerance, okruh);
        if nový then
            S.ExpandujStrom(nejbližší, nový);
            if /nový - KoncovýBod/ < Konst then
                T ⟵ Trajektorie(nový);
            end if
        end if
    else
        čekání na data;
    end if
end while
```

Se znalostí mapy, počáteční a cílové pozice je možné začít prohledávat samotný prostor za účelem získání trajektorie do cílového bodu. K samotné expanzi stromové struktury (obrázek 5.6) je potřeba vytvořit náhodný bod[1], ke kterému se najde nejbližší bod stromové struktury. V tomto okamžiku je nutné ověřit zda nový uzel (z obrázku 5.6 $q_{nový}$) bude splňovat podmínky pro přidání do stromu. O podmínkách hovoří podkapitola z 5.4.3 Podmínky pro nový uzel. Nejjednodušší podmínkou je schopnost průjezdu robota z $q_{nejbližší}$ do $q_{nový}$, tj. zda úsečka spojující tyto body neprochází libovolnou překážkou. Vyhledávání nových uzlů se provádí tak dlouho, dokud se uzel $q_{nový}$ nepřiblíží koncovému bodu na vzdálenost menší než je mezi dvěma uzly ve stromu. Při splnění této podmínky přichází třetí část algoritmu.

Posledním krokem algoritmu je nalezení trajektorie ve stromu. Pokud se vyhledává trajektorie právě tehdy, kdy se přidá poslední uzel do dostatečné blízkosti ke koncovému bodu, potom existuje právě jedna cesta spojující $q_{počátek}$ a nově vzniklý uzel. Získání cesty je popsáno v kapitole 5.5. Existuje také vylepšený algoritmus *RRT**, který vyhledává nejefektivnější cestu i z ostatních blízkých uzlů cílovému

[1] Jedná se o takový bod, který náleží do určité oblasti kolem mapy a nemusí v ní nutně ležet

bodu. Tato vlastnost nebyla pro potřeby práce implementována.

Vyhledávání cílových bodů

Vyhledávání cílových bodů velice důležitou součástí algoritmu, která určuje chování robota a způsob prohledávání budovy. Libovolný bod je vybrán za cílový při splnění určitých podmínek.

- Musí ležet v dostatečné vzdálenosti od překážek
- Musí ležet ve známém prostředí
- Cílový bod musí ležet v blízkosti prostředí ve kterém robot ještě nebyl
- Nesmí se vybírat dva po sobě jdoucí cílové body, při kterých by musel robot projet celou známou budovu (neefektivní průzkum)

První podmínka je splnitelná jednoduše. V okolí prohledávaného bodu nesmí ležet žádná překážka. Velikost okolí je definovaná uživatelem a většinou se bude odvíjet od velikosti robota.

Druhá podmínka je přirozená a snadno ověřitelná. Robot by neměl a ani nemohl plánovat trajektorii tam, kde je pro něj neznámé prostředí.

Splnění třetí podmínky je o dost komplikovanější. V mapě je známé prostředí definováno[2] 0, neznámé -1 a známé[3] 100. Ke splnění třetí podmínky je potřeba aby v daném okolí prohledávaného bodu bylo určité procento neznámých (zajištění neznámého prostředí které může robot prozkoumat) a procento známého prostředí (zajištění bezpečnosti průjezdu). Předcházející tři podmínky splní v mapě množina bodů, která může být poměrně rozsáhlá, nebo nemusí obsahovat žádný koncový bod. V takovém případě se robot vrací zpět na startovací pozici, ve které může dostat další instrukce.

Čtvrtá podmínka vybírá z množiny koncových bodů ten, ke kterému bude plánována trajektorie. Filtrace bodů, které splňují předcházející podmínky je čitelná z algoritmu 2. Aby byl průzkum efektivní robot nejprve prohledává své lokální prostředí, tj. takové prostředí, kde má *Lidar* dosah. Pokud už není žádný neznámý bod v blízkosti robota, potom se za koncový bod zvolí nejvzdálenější koncový bod na mapě.

Výběr cílového bodu je značně nedokonalý a prvoplánový. Při výběru cílového bodu se neuplatňuje žádná inteligence, jedná se pouze o algoritmické řešení, které ovšem funguje, ale dalo by se využít jiných metod zefektivňující prohledávání budovy.

[2] Ve skutečnosti je to interval od <0,tolerance)

[3] Opět se jedná o interval <tolerance,100>

Podmínky pro nový uzel

V algoritmu 2 jsou ve funkci *PodmínkyProUzel* dva parametry, kde tolerance udává pravděpodobnost při níž je pixel mapy[4] považovaný za překážku. Druhý parametr je okruh udávající velikost poloviny strany čtverce sestrojeného kolem prohledávaného bodu.

Podmínky pro zařazení bodu $q_{nový}$ do stromu mohou být různé v závislosti na způsobu prozkoumávání budovy.

První přirozenou podmínkou je volné prostředí mezi uzly stromu. Splnění tohoto požadavku je jednoduché. Úsečka mezi $q_{nejbližší}$ a $q_{nový}$ nesmí procházet žádnou překážkou. Pro algoritmické řešení pracuje s parametrickou rovnicí přímky 5.14,

$$\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x_A & v_x \\ y_A & v_y \end{pmatrix} \begin{pmatrix} 1 \\ t \end{pmatrix} \tag{5.14}$$

kde vektor $\mathbf{v} = (v_x, v_y)$ je směrový vektor dán body A = $q_{nejbližší}$ a B = $q_{nový}$ a x_A, y_A jsou souřadnice bodu A. Parametr t $\in$ <0,1> a vzorkování se volí dostatečně malé vzhledem k rozměrům pixelů mapy. Pokud žádný bod úsečky neleží v obsazeném pixelu mapy je první podmínka prohlášena za splněnou.

Bod $q_{nový}$ samozřejmě musí ležet ve známém prostoru, jinak by se strom rozrůstal do nezmapovaných oblastí mapy. Dalším požadavkem na bod $q_{nový}$ je ten, aby ležel v dostatečné vzdálenosti od překážky a bylo tak možné bezpečně projet libovolnou trajektorii, která může ze stromu vzniknou. Dostatečný prostor od překážek také zabezpečuje případnou kolizi při vyhlazování trajektorie. O získání trajektorie a jejího vyhlazování pojednává kapitola 5.5 Trajektorie.

5.4.4 Výsledky RRT

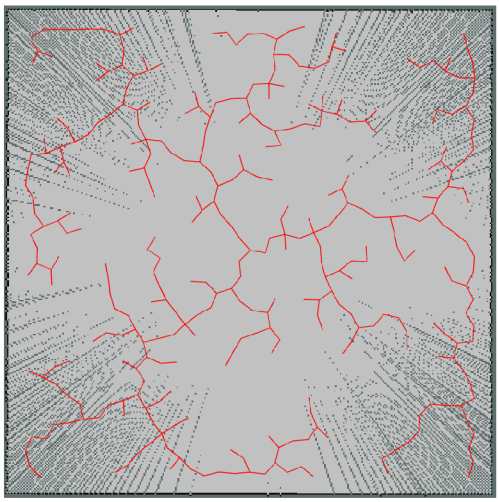
RRT algoritmus velice rychle konverguje k cílovému bodu ve velkých otevřených prostorech, jak je ukázáno na obrázku 5.7a. V takových případech strom rychle expanduje, jelikož mu nic nebrání v rozrůstání, tj. překážka nebo stěna. Stejně rychle algoritmus může konvergovat i v poměrně složitém prostředí (obrázek 5.7b). V takovém případě má strom mezi překážkami hodně volného prostoru, do kterého se může rozrůstat a konvergence k cílovému bodu je velice rychlá.

Slabinou *RRT* algoritmu jsou rychle se měnící prostředí a úzké skuliny.

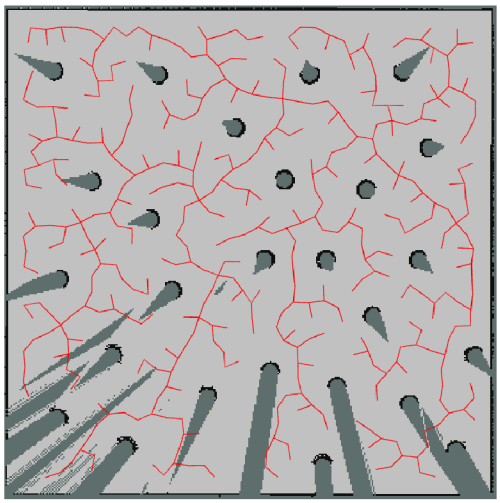
RRT algoritmus je závislý na dvou parametrech. Jedním z nich je vzdálenost uzlů a dalším je velikost okolí uvnitř nichž se nesmí nacházet žádná překážka[5]. A právě velikost okolí určuje, že se robot nebude pohybovat v okolí překážek a taky

[4] Jedná se o mapu typu mřížka obsazenosti

[5] Myšleno obsazený pixel mapy

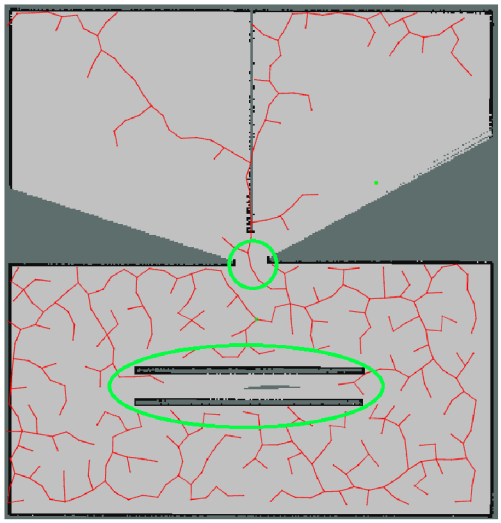


(a) RRT ve volném prostoru



(b) RRT v komplikovaném prostoru

Obr. 5.7: Algoritmus RRT v různých prostředích. Dobře konverguje v prázdných i relativně komplikovaných prostředích neobsahující úzké skuliny.



Obr. 5.8: RRT algoritmus v úzkém prostoru. Špatně prochází prostory ve kterých se nachází úzké skuliny.

to, že robot nebude projíždět úzkými průchody. Situace je znázorněna na obrázku 5.8 zeleně.

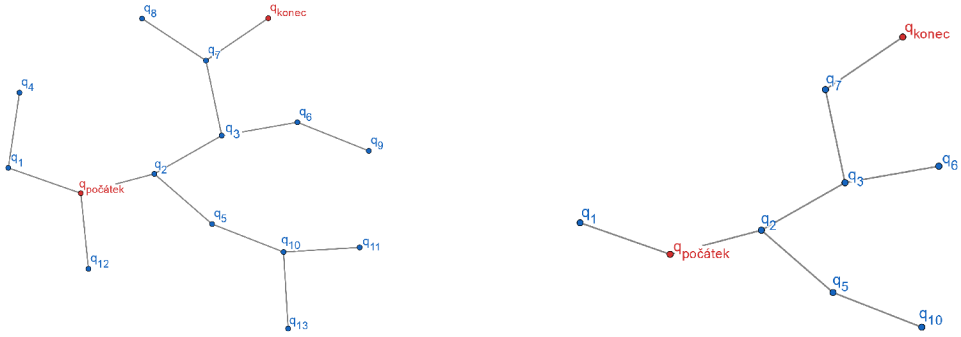
I když existuje průchod mezi úzkou překážkou (na obrázku 5.8 zelené kolečko) je pravděpodobnost průchodu úzkou skulinou značně menší než ve volném prostoru (spodní část obrázku 5.8).

V případě prohledávání prostoru, ve kterém existují úzké skuliny, jimiž má robot projet, nemusí algoritmus konvergovat, nebo konvergovat rychle popřípadě se zvyšuje

pravděpodobnost srážky robota s překážkou[6].

5.5 Trajektorie

Počáteční bod robota je automaticky brán za první uzel a k němu jsou přidávány další uzly. Po přidání nového uzlu $q_{nový}$ do okolí koncového bodu je dalším krokem extrahování trajektorie ze stromu, po které se robot bude pohybovat. Průběh algoritmu je zobrazený na obrázku 5.9. Při každém kroku se ze stromu odebere takový uzel, který má pouze jednoho souseda[7], kromě počátečního a koncového bodu. Pokračuje se tak dlouho, dokud neexistuje uzel s jedním sousedem.



(a) Původní strom

(b) První krok

Obr. 5.9: Ukázka získávání trajektorie ze stromu. Odebírají se ty uzly, které mají jen jednoho souseda s výjimkou počátečního a koncového bodu.

Vznikne posloupnost bodů, které jsou základem pro budoucí trajektorii. V dalších podkapitolách jsou vysvětleny možné interpretace těchto bodů.

Trajektorie se singulárními body

Při spojení po sobě jdoucích bodů přímkou (obrázek 5.9b po odstranění bodů q_1, q_5, q_{10} a q_6) vznikne trajektorie obsahující v uzlových bodech singulární body. Trajektorie není hladká, tj. v uzlových bodech není diferencovatelná.

Po trajektorii se robot ovšem může pohybovat, ale jeho pohyb není přirozený, jelikož by se v každém uzlovém bodě musel zastavit, otočit k dalšímu uzlovému bodu a znovu se rozjet. Nevýhody takového pohybu jsou patrné, obzvláště pokud by se ve stejném prostředí pohybovali lidé, kteří by nedokázali odhadnout pohyb robota, což by zvyšovalo pravděpodobnost kolize robota s člověkem.

[6]S parametrem velikost okolí rovnajícím se nule

[7]Soused je takový uzel, který je s spojený větví s druhým uzlem

I po nehladké trajektorii lze jet s robotem tak, aby jeho výsledná trajektorie byla rozumná (viz. 5.6.2). Jedná se o algoritmy, které v průběhu vykonávání pohybu upraví pohyb robota tak, aby zaobloval ostré hrany (algoritmus *Pure pursuit* [18] popsaný v kapitole 5.6 Řízení).

Dalším způsobem je vyhlazení trajektorie před započetím pohybu. O vyhlazování trajektorie pojednává následující podkapitola.

Metody vyhlazování trajektorie

Cílem vyhlazování trajektorie je z uzlových bodů vytvořit spojitou křivku, po níž se bude robot plynule pohybovat. Nejznámější metody na vyhlazování trajektorií jsou založeny na interpolaci. V podkapitole Trajektorie se singulárními body je používá lineární interpolace, tj. spojení po sobě následujících bodů přímkou, což je pro potřeby robotiky obvykle nežádoucí, jelikož nevzniká hladká trajektorie.

Interpolační techniky je možno rozdělit do dvou skupin. První skupinou jsou takové typy interpolace, které z uzlových bodů (nebo také uzlů interpolace) vytvoří trajektorii procházející všemi uzly interpolace. Mezi ně patří například *Lagrangeův interpolační polynom* nebo *Newtonův interpolační polynom*. Tyto dva typy interpolace jsou se zvyšujícím se počtem prvků v trajektorie poměrně náročné na implementaci. Další technikou spadající do první kategorie je *kubický spline*. *Kubický spline* spojuje vždy následující dva uzly interpolace kubickou křivkou tak, aby ve všech bodech, kde na sebe navazují dvě křivky, byly stejné derivace, což zajišťuje hladkost trajektorie. *Kubický spline* pro vyhlazování trajektorie vzniklé z *RRT* algoritmu není vhodný, jelikož uzly interpolace jsou vybírány náhodně[8] a mohla by vzniknout poměrně komplikovaná trajektorie.

Další skupinou jsou interpolační techniky nezaručující, že uzly interpolace prochází výsledná křivka. Trajektorie vzniklá z *RRT* algoritmu je často kostrbatá a je výhodné neprocházet všemi body trajektorie. Pro vyhlazování v této práci je použita *Bézierova křivka* [19]. V následující kapitola je detailně popsána.

Implementace Bézierovy křivky

Výhodou *Bézierovy křivky* [19] je její snadná implementace. Je definovaná obecně jako polynom n-tého stupně počítající s n+1 body, kde

$$K(t) = \sum_{i=0}^{N} B_{i,n}(t) P_i \qquad t \in < 0, 1 > \tag{5.15}$$

[8] Musí splnit podmínky pro přiřazení do stromu

$B_{i,n}(t)$ je Bernsteinův polynom.

$$B_{i,n}(t) = \binom{n}{i} t^i (1-t)^{n-i} \tag{5.16}$$

Při pohybu robota je jeho budoucí trajektorie neustále upravována na základě pozice robota a pozice cílových bodů, proto je *Bézierova křivka* implementována jako křivka třetího stupně, pro množinu bodů $\mathbf{P} = \{p_0, p_1, p_2, p_3\}$ takto

$$K(t) = \sum_{i=0}^{3} \binom{3}{i} t^i (1-t)^{3-i} P_i = (1-t)^3 p_0 + 3t(1-t)^2 p_1 + 3t^2(1-t)p_2 + t^3 p_3 \tag{5.17}$$

Trajektorie je množina bodů jdoucích za sebou. Pro polynom 5.17 se zvolí parametr t určující jednotlivé body křivky tak, aby se rychlost robota promítala do výsledné trajektorie. To znamená, že trajektorie je vzorkována body tak, aby robot za ekvidistantní časový okamžik urazil vzdálenost mezi těmito body.

Při vyhlazování trajektorie se může stát, že výsledná trajektorie nebude ležet v množině volných buněk $\boldsymbol{V}$. Tento problém se dá vyřešit různými způsoby. Prvním z nich je plánování trajektorie v dostatečné vzdálenosti od překážek 5.4.3. Nebo se může *Bézierova křivka* upravit tak, aby se zvýšila váha u zvolených bodů a výsledná trajektorie se více přiblížila uzlům interpolace a vyhnula se překážce.

5.6 Řízení

Řízení je jednou z nejzákladnějších částí softwaru robota. Po správně definovaných souřadnicových soustavách (5.1) ve kterých se robot může pohybovat je potřeba zajistit správný pohyb robota.

Pro vytvoření mapy prostředí, což je jedním z cílů práce, stačí teleoperovaný mobilní robot, kterého ovládá člověk na dálku. O řízení v manuálním režimu pojednává podkapitola 5.6.1.

V případě autonomního pohybu robota je vypočítaná trajektorie robota. Cílem řízení v autonomním módu (podkapitola 5.6.2) je kontrola správného pohybu robota po trajektorii.

5.6.1 Manuální mód

Jedná se o řízení bez zpětné vazby. V tomto režimu robot přijímá požadavky na pohyb, které uživatel posílá prostřednictvím klávesnice, nebo jiného ovladače. Ze znalostí rozměrů robota a jeho kinematiky (kapitola 4) vypočítává s frekvencí 50Hz,

tj. každých 20 ms, úhel natočení kol. V případě simulace robot posílá úhel natočení do simulátoru, který vykoná příslušný pohyb. V reálném případě se robotické platformě posílají rychlosti jednotlivých kol.

Algoritmus 3: Algoritmus Pure pursuit

```
Data: Trajektorie T, Odometrie O, Poloměr R
Result: Rychlost pravého kola P, rychlost levého kola L
while true do
    A' ⟵ NejbližšíBodTrajektorie();
    Průnik ⟵ PrůnikKružniceATrajektorie(A', R);
    rychlost ⟵ RychlostZTrajektorie();
    Fi_průnik ⟵ UhelRobotPrůnik();
    Fi_odometrie ⟵ O.ZískejOrientaci();
    Fi ⟵ modulo(Fi_průnik - Fi_odometrie + PI, 2*PI) - PI;
    if abs(Fi) < 0.5236 then
        LineárníRychlost ⟵ rychlost;
        ÚhlováRychlost ⟵ Fi/(R/speed);
        P, L ⟵ NastavRychlosti(LineárníRychlost, ÚhlováRychlost);
    else
        LineárníRychlost ⟵ 0;
        ÚhlováRychlost ⟵ Fi/(R/speed);
        if ÚhlováRychlost > rychlost then
            ÚhlováRychlost = rychlost;
        end if
        P, L ⟵ NastavRychlosti(LineárníRychlost, ÚhlováRychlost);
    end if
    NastavRychlostKol(P, L);
end while
```

5.6.2 Autonomní mód

V autonomním módu se robot musí rozhodovat a pohybovat sám. Rozhodovací proces je popsán v dřívějších kapitolách a výsledkem je trajektorie robota. Řídící program zpracovávající pohyb přijímá *topik* s trajektorií a odometrií. Jedná se tedy o zpětnovazební řízení.

Existují algoritmy, které se snaží řídit pohyb robota po trajektorii co nejpřesněji, tj. tak, aby se pozice robota v průběhu pohybu co nejméně lišila od naplánované trajektorie.

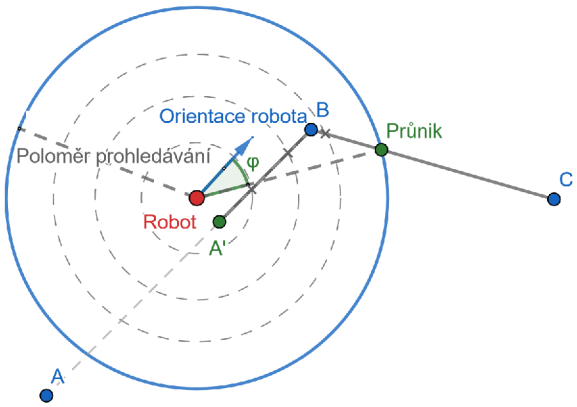
V této práci je implementovaný algoritmus, který přesně nesleduje naplánovanou trajektorii a snaží se jí co nejvíce linearizovat[9]. Řídicí algoritmus lze proto použít i

[9]Záleží na parametrech algoritmu, které se dají před vykonáním pohybu změnit.

na nehladké trajektorie. Algoritmus je detailně popsán v následující podkapitole.

Pure pursuit

Pure pursuit je algoritmus na sledování trajektorie [18]. Je založený na vyhledávání bodu na trajektorii v určité vzdálenosti od robotu, ke kterému robot směřuje svůj pohyb. Algoritmus je znázorněn na obrázku 5.10.



Obr. 5.10: Algoritmus Pure pursuit. Robot (symbolicky znázorněný světle modrou šipkou orientace robota) směřuje k bodu Průnik, který je dán kružnicí s poloměrem prohledávání a původní trajektorií. Robot přizpůsobuje svou rychlost lineární a úhlovou v závislosti na velikosti úhlu ϕ a poloměru prohledávání. Při pohybu tak dochází k vyhlazování pohybu.

Míra přilnavosti robota k trajektorii je dána poloměrem prohledávání. Jak je vidět z obrázku, čím větší poloměr prohledávání robot má, tím více se může robot od původní trajektorie odchýlit.

Algoritmus je založený na vyhledávání průniku kružnice s trajektorií. Obecně mohou existovat dva průniky, ale jen jeden z nich je považován za bod, ke kterému robot směřuje svůj pohyb. Algoritmus 3 nejprve vyhledává bod A' (jedná se o nejbližší bod trajektorie k robotovi), což hraje zásadní roli při vyhledávání průniku. V případě odříznutí trajektorie od začátku do bodu A', existuje už pouze jeden průnikový bod, viz 5.10.

Dalším krokem v algoritmu 3 je nalezení nejmenšího úhlu mezi orientací robota a a přímkou danou bodem Průnik a Robot. Z velikosti úhlu se rozhoduje zda se bude robot pohybovat po trajektorii, tj. má nenulovou lineární rychlost, nebo zda bude měnit pouze svou orientaci. Robot bude měnit pouze svou orientaci v několika

případech. Prvním z nich je případ, kdy se orientace robota liší od počáteční trajektorie o více než 0.526 radiánu, nebo při získání nové trajektorie, která plynule[10] nenavazuje na starou trajektorii.

5.7 Rviz

Rviz je vizualizační nástroj, který umožňuje zobrazovat vypočítaná nebo získaná data. Veškeré vizualizační nástroje v robotice jsou velice cenné, jelikož usnadňují lidem pochopit jak robot vnímá své okolí a proč se chová tak, jak se chová. V *Rvizu* jsou předpřipravené často používané *topiky*, jako například mapa, odometrie, model robota, trajektorie a podobně. V případě nutnosti vizualizace jiných dat, lze *Rviz* obohatit o vlastní pluginy, které popisují chování zobrazovaných dat. Obrázky 2.2, 4.2 a 5.7, kde strom (červeně) je vlastní implementace, jsou vzniklé vizualizací dat v *Rviz*u. Takto zobrazená data mají pro člověka vyšší vypovídající hodnotu než číselně zobrazená data.

[10] Úhel mezi orientací robota a novou trajektorií je menší než 0.5236 radiánu

6 Výsledky

Cílem práce je vytvoření mobilní robotické platformy a následné testování vybraných *SLAM* algoritmů v simulátoru a reálném světě. V předchozích dvou kapitolách byl popsán model robota používaný v simulaci, struktura reálného robota a jeho softwarové vybavení. Software robota umožňuje vytvářet mapy budov v manuálním a autonomním režimu. Proto jsou v této kapitole nejprve podrobně testovány vybrané *SLAM* algoritmy v simulaci, následně mapy vzniklé autonomně a poslední část se věnuje mapám z reálného prostředí.

Ke správnému otestování *SLAM* algoritmů je potřeba vymyslet exaktní metodu jednoznačně určující kvalitu mapy. Ačkoli vizuální hodnocení člověka je do značné míry vypovídající, protože dokáže odhalit významné nedostatky, nedokáže správně vyhodnotit detaily spojené s odhadem počáteční polohy robota a jeho správnou lokalizaci v prostředí.

Metoda pro hodnocení algoritmů se skládá z následujících čtyř bodů seřazených podle důležitosti.

- Nejdůležitějším faktorem, který rozhoduje o kvalitě vzniklé mapy je porovnání trajektorie, po které se robot reálně pohyboval a domnělé, vypočtené, trajektorie. Metoda určuje míru správné lokalizace robota v prostoru, která je nedílnou součástí správně vybudované mapy a umístění objektů v ní. K vyhodnocení se používá hodnota RMSE (root-mean-squre error) [20] , která udává jak se průměrně liší jednotlivé body trajektorie a je dána vztahem

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}[(x_{si} - x_i)^2 + (y_{si} - y_i)^2]}, \tag{6.1}$$

  kde x a y jsou souřadnice a index s udává pravou hodnotu souřadnice.
- Dalším kritériem pro hodnocení je přesnost v detekci překážek, která souvisí s rozostřením stěn. Čím více je překážka rozostřená, tím více je mapa nepřesná. [20]

  Rozostření stěn je skvěle viditelné v histogramu obsazenosti. Jedná se o takový histogram, který obsahuje pro všechny pravděpodobnosti[1] četnost výskytu buněk s touto pravděpodobností. Ideální mapa bude mít všechny obsazené buňky v rozmezí od 90% do 100%, kde 100% bude mít největší zastoupení a všechny volné buňky v rozmezí 0% až 10%.
- Dalším, celkově třetím, kritériem je vizuální hodnocení. Při vizuálním hodnocení se hodnotitel soustředí na výrazné chyba jako na obrázku 6.6. Obecně čím více rohů a hran mapa obsahuje, tím více je nepřesná [20].

[1] Od 1% do 99%. Interval je zvolen takto s předpokladem, že nejvíce buněk má pravděpodobnost obsazení právě 0% a 100% a více tak vyniknou detaily určující rozostření viz 6.1.1

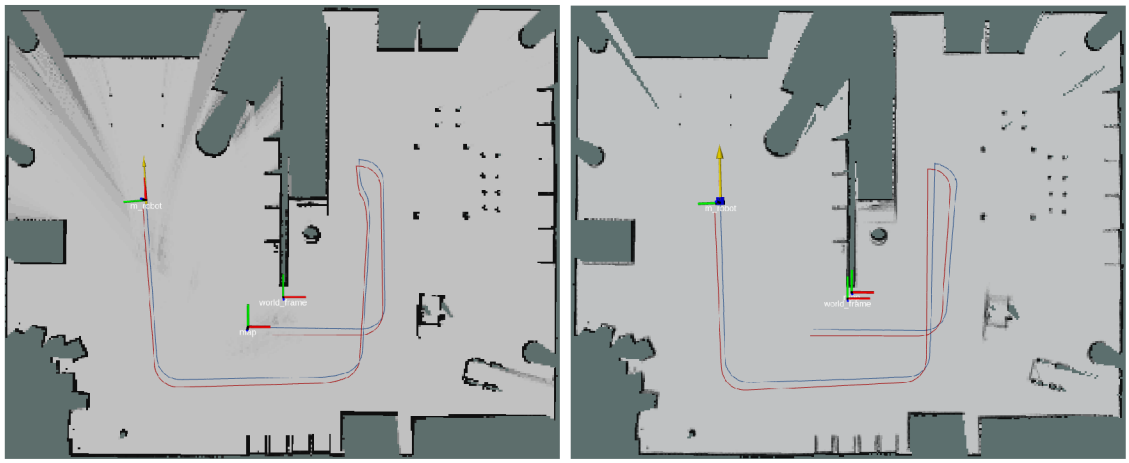
- Posledním a volitelným prvkem je výpočetní náročnost algoritmu.

6.1 Testování SLAM algoritmů v simulaci

Pro testování v simulaci byly zvoleny dva typy prostředí. Prvním z nich je komplexní interiér, ve kterém se vyskytují překážky různé velikosti a různých materiálů. Obecně je v tomto prostředí hodně volného prostoru a robot by neměl mít jakýkoli problém s vytvořením kvalitní mapy. Druhou budovou je extrémní případ, kdy je pro robota obtížné správně vyhodnotit data z *Lidaru* a lokalizovat se v mapě.

6.1.1 Komplexní interiér

Nejlepší výsledky všech algoritmů v tomto prostředí jsou ukázány na následujících obrázcích 6.1 a 6.2. Větší mapy jsou přiloženy do přílohy A. První z nich, tj. *Hector SLAM* a *Gmapping* využívají pro zobrazení mapy typu mřížka obsazenosti a lze získat až 101 stavů pravděpodobností obsazení. Pro tyto algoritmy je použitelný histogram obsazenosti, viz 6.1.1. Pro *Karto SLAM* implementovaný v *ROSu* nelze získat pravděpodobnost obsazení pro jednotlivé a buňky, a proto *Karto SLAM* využívá 3 stavovou mapu, tj. volná, obsazená a neznámá buňka, a nelze rozhodnout o rozostřenosti buněk.

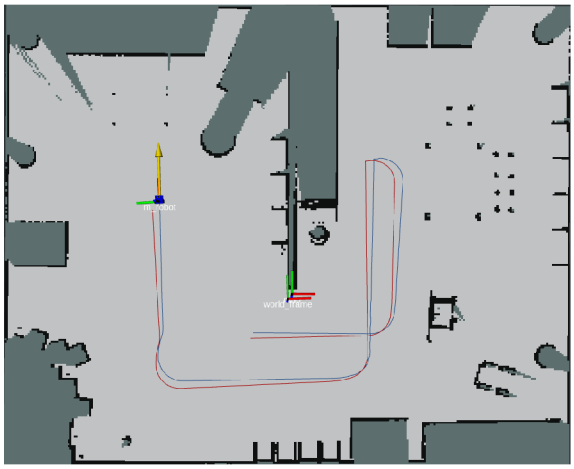


(a) Hector SLAM (b) Gmapping

Obr. 6.1: Výsledky testovaných SLAM algoritmů v komplexním prostředí se 102 stavovou mapou.

Srovnání trajektorií a RMSE

Každý algoritmus byl v jednom prostředí testován při dvou rychlostech. První průměrná rychlost robota byla 0.5 ms^{-1} a druhá 1 ms^{-1}. Vyšší rychlosti nebyly z důvodu omezené manévrovatelnosti v prostředí zahrnuty do testování. Průměrné odchylky RMSE odhadnuté trajektorie od skutečné trajektorie zobrazuje tabulka 6.1.



Obr. 6.2: Výsledky algoritmu KartoSLAM (3 stavová mapa) v simulaci.

Hodnota RMSE závisí na kvalitě odometrie a schopnosti *SLAM* algoritmu udržovat souřadnicovou soustavu mapy[2] a souřadnicovou soustavu robota (world_frame) ve stejné konfiguraci. Závisí také na délce a typu uražené trajektorie. Za předpokladu, že se robot pohybuje po dlouhých rovných úsecích, kdy přesnost odometrie klesá, bude výsledná hodnota RMSE větší. V tabulce 6.1 jsou kromě hodnot RMSE zahrnuty také absolutní hodnoty změn úhlů mezi počáteční souřadnicovou soustavou *world_frame* a *map*. Tato hodnota je zásadní při dlouhodobější lokalizaci a v ideálním případě je rovna nule, tj. počátek mapy a startovní pozice jsou stejně orientované. Testování probíhalo za stejných podmínek pro podobnou trajektorii.

Nejlepší shody trajektorií, a tedy i přesnosti lokalizace robota, dosahuje *Karto SLAM*, který dosahuje nejlepších výsledků i při vyšších rychlostech. Tato vlastnost je velice výhodná v menších prostředích, kdy je přesná lokalizace velice důležitá. Výsledné trajektorie všech tří algoritmů jsou k nalezení v příloze A.2.

Histogram obsazenosti

Důležitým faktorem ke kvalitní mapě je ostrá a přesná reprezentace překážek v ní. Míru rozostření překážek spolehlivě určuje histogram obsazenosti (obrázek 6.3). V histogramu je znázorněna procentuální četnost buněk se stejnou pravděpodobností

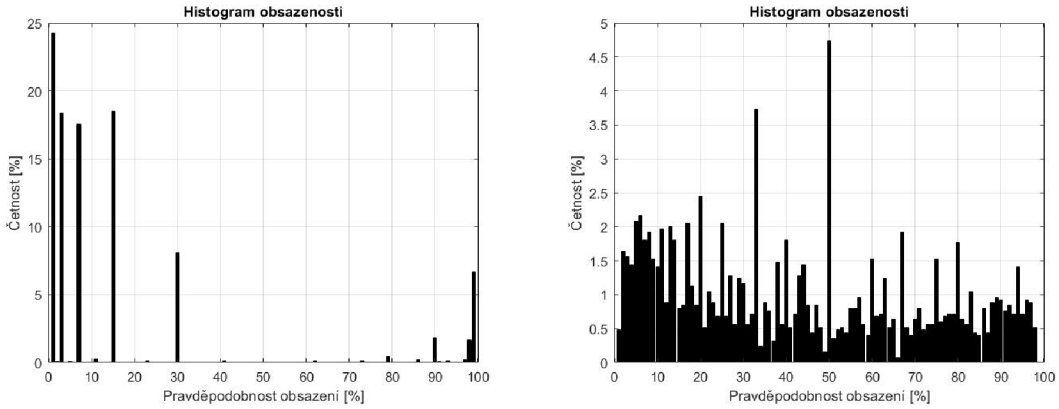
[2]Značenou na obrázcích 6.1 a 6.2 a v příloze A map

Tab. 6.1: RMSE srovnání trajektorií SLAM algoritmů

	RMSE [m]						
Průměrná rychlost	0.5 ms^{-1}			1 ms^{-1}			θ $[rad]$
Pokus	č.1	č.2	č.3	č.1	č.2	č.3	
`Hector SLAM`	0.2126	0.2167	**0.1816**	0.5593	0.3979	**0.3387**	0.0086
`Gmapping`	0.1852	0.1931	**0.1796**	**0.3749**	0.3881	0.4952	0.0192
`Karto SLAM`	**0.1638**	0.1793	0.1848	**0.2233**	0.3823	0.3100	0.0314

obsazení. Z histogramu jsou ořezány pravděpodobnosti 0% a 100%, jelikož mají největší procentuální zastoupení a průběhy zbylých pravděpodobností, určující právě ostrost mapy, by nebyly tak zřetelné.

Na následujícím obrázku jsou zobrazeny histogramy pro *Hector SLAM* a *Gmapping*. Pravděpodobnosti z *Karto SLAMu* nejsou v *ROSu* dostupné.



(a) Hector SLAM (b) Gmapping

Obr. 6.3: Histogramy obsazenosti. Pro každou pravděpodobnost obsazení buňky je určena četnost výskytu v mapě.

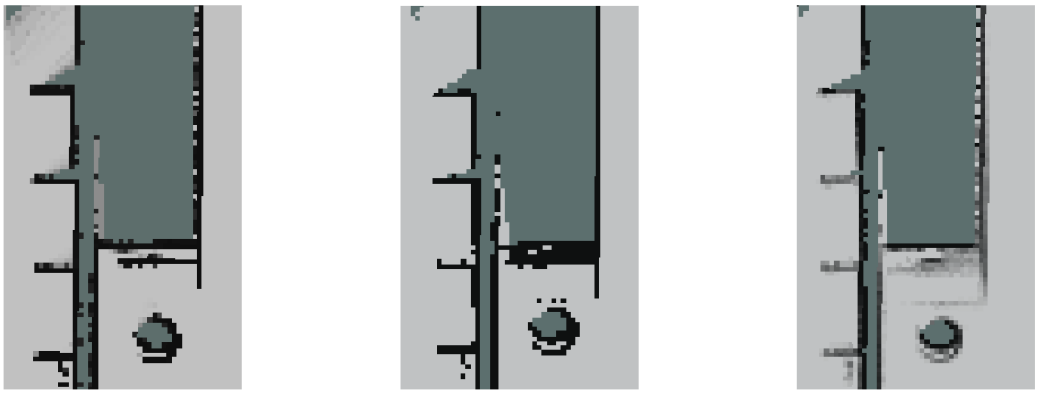
Z histogramů je patrné, že *Hector SLAM* má většinu hran ostrých, jelikož velké procentuální zastoupení buněk je s 98% a vyšší pravděpodobností obsazeno a četnost buněk s pravděpodobností obsazení mezi 30% až 90% je nízká.

U *Gmappingu* je situace jiná. Histogram ukazuje na rovnoměrné rozdělení četností pro jednotlivé pravděpodobnosti obsazení. Zvýšená četnost se projevuje jen v rozmezí od 0% až 30%, jelikož prázdných buněk je mnohonásobně více než obsazených. Takto rovnoměrně rozdělený histogram pravděpodobnosti se projevuje rozostřením některých překážek, viz 6.4c.

Hector SLAM oproti *Gmappingu* má výrazně ostřejší a přesnější detekci překážek. *Karto SLAM* nemá k dispozici data o pravděpodobnosti obsazení buněk a počítá jen s třemi stavy. Z obrázku 6.4 je vidět, že *Karto SLAM* má tendenci přidávat nebo ubírat překážky, které by byly běžně rozostřené jako v případě *Gmappingu*.

Viditelné chyby

Největšími problémy vzniklými na těchto mapách jsou viditelné na obrázku níže (6.4), kde došlo k protáhnutí hrany, což bylo pravděpodobně způsobeno malým množstvím význačných bodu při návratu robota od vrchní strany mapy ke spodní.



(a) Hector SLAM (b) Karto SLAM (c) Gmapping

Obr. 6.4: Chyby a rozostřenost SLAM algoritmů

Absence význačných bodů, tedy převážně hran, činí algoritmům největší problémy a pojednává o tom kapitola 6.1.2. Viditelné chyby vznikají především díky špatné lokalizaci.

CPU

Výpočetní náročnost může hrát velkou roli při výběru *SLAM* algoritmu. Ta závisí na velikosti a rozlišení mapy, což je vidět z grafu 6.5 u červené resp. modré křivky, reprezentující výpočetní náročnost *GMappingu* pracující s malou resp. velkou mapou. Experimenty byly prováděny se stejným rozlišením.

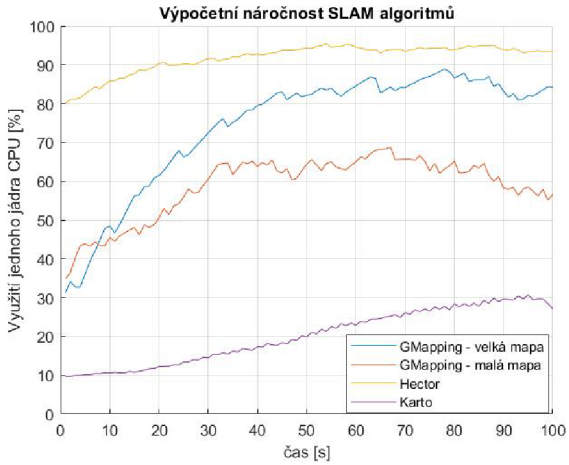
U *Karto SLAM* algoritmu nelze nastavit velikost mapy, s kterou se pracuje, jelikož se velikost mapy podle potřeby zvětšuje, což dokazuje rostoucí výpočetní náročnost algoritmu.

Hector SLAM je výpočetně nejnáročnější[3], ovšem jeho výpočetní náročnost není tolik ovlivněna velikostí mapy jako u *Karto SLAM* algoritmu, jehož výpočetní výkon

[3]Hodnota nad 100% znamená zapojení více jader procesoru

s rostoucí mapou roste rychleji než u zbylých dvou algoritmů. *Hector SLAM* algoritmus je navržený i pro pohyb s 6 stupni volnosti tzv. 6 DOF, což může negativně ovlivňovat výpočetní náročnost algoritmu.

Zajímavý průběh je vidět u *GMappingu*, kde křivka zpočátku roste, i přes to, že algoritmus pracuje se stejně velkou mapou. Růst je způsobený objevováním většího množství neznámého prostředí a mohou tak vznikat lokální maxima[4].



Obr. 6.5: Výpočetní náročnost SLAM algoritmů

6.1.2 Dlouhé chodby

Největším problémem *SLAM* algoritmů jsou prostředí podobající se uzavřené smyčce tzv. Loop-closure. Algoritmus má problém navázat nově vznikající chodbu, podle dat ze senzorů, na již vzniklou, a proto mohou vznikat dvojité chodby, které jsou navzájem posunuty a nevzniká tak uzavřené prostředí, ve kterém se robot skutečně pohybuje.

Hector SLAM algoritmus může ke své lokalizaci používat pouze data ze senzorů. Výsledná mapa je ukázána na obrázku 6.6a. Robot se pohyboval v uzavřené smyčce[5]. Z výsledku jsou patrné dvě chyby. První z nich je nepřesné navázání nově vzniklé chodby na původní (nová chodba je ohraničena červenými daty ze senzoru) a druhou chybou je vznik falešné, horní a neuzavřené chodby, která vznikla na začátku pohybu. Na obrázku jsou zaznačeny dvě trajektorie. Modrá značí domnělý pohyb robota v mapě a červená skutečný pohyb a tvar chodeb.

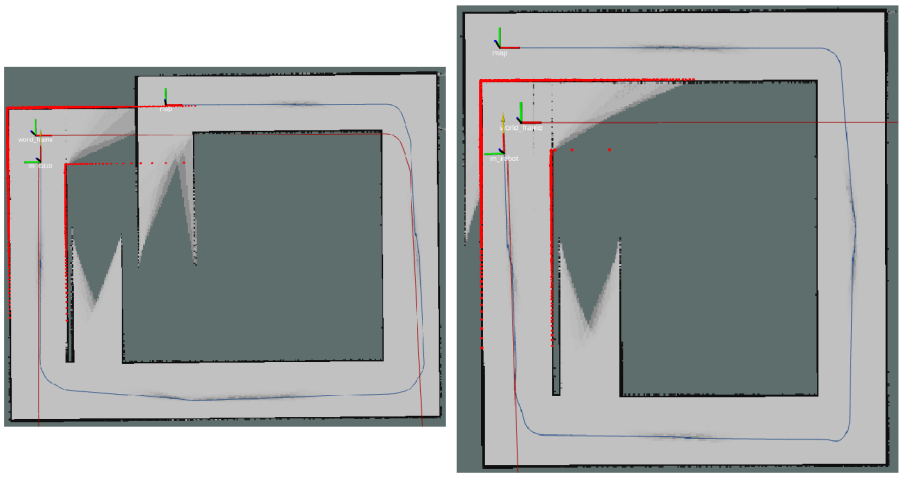
Skutečná trajektorie zaznačena na 6.6a není v mapě zachycena celá. Tento jev je způsobený zkracováním chodeb vlivem špatné asociace dat z *Lidaru*. V případě,

[4]Definice myšleného lokálního maxima je brána v delším časovém intervalu a nekopíruje matematickou definici

[5]Spodní neuzavřená chodba je součástí prostředí a nevznikla vlivem chyby

kdy se robot pohybuje rovnoběžně se stěnami, která nemá žádný význačný bod a dosah *Lidaru* je menší než délka chodby, dochází ke špatné asociaci dat a domnělému zastavení robota, neboť jsou data z *Lidaru* po celou dobu pohybu stejná. Robot se ve skutečnosti pořád pohybuje a v okamžiku, kdy senzor získá nová, odlišná data, robot se podle *SLAM* algoritmu začne znovu pohybovat. Chodba je proto zkrácena o délku danou časem, kdy robot dostává téměř totožná data a rychlostí robota.

Řešením je přidat informace o pohybu a poloze robota. Výsledek mapy vytvořené *Hector SLAM* algoritmem je na obrázku 6.6b.



(a) Hector SLAM bez odometrie (b) Hector SLAM s odometrií

Obr. 6.6: Hector SLAM v dlouhých chodbách

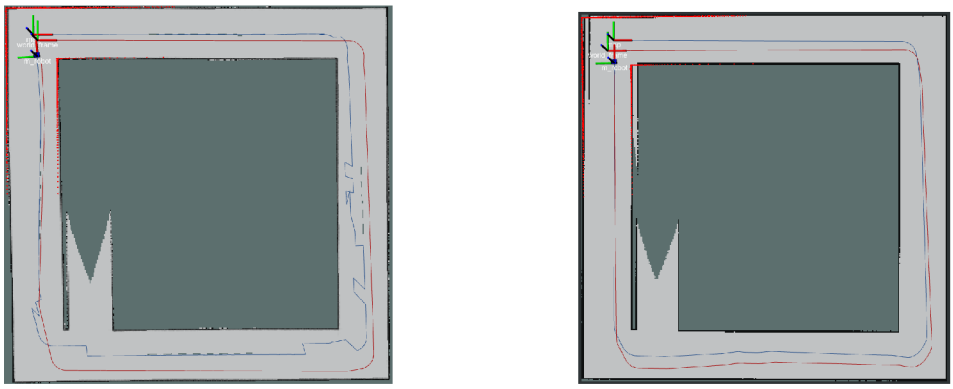
Situace se výrazně zlepšila, obzvláště v odhadu počáteční pozice, ale k efektu zkracování chodeb dochází stále. Je tedy patrné, že *Hector SLAM* algoritmus klade větší prioritu na data ze senzorů.

Zbylé dva algoritmy musí ke své správné funkci použít data z odometrie a výsledné mapy jsou na obrázcích 6.7.

Výsledný odhad počáteční pozice a lepší návaznost chodeb v uzavřených smyčkách je nejlepší u algoritmu *GMapping*. Nevznikají žádné falešné chodby ani jejich zdvojení. Největším problémem *Gmappingu* v tomto prostředí je skoková změna odhadu počáteční pozice mapy plynoucí z nespojité domnělé trajektorie robota (modrá).

Dobrou lokalizaci opět ukázal *Karto SLAM*, u kterého sice vznikly drobné nepřesnosti v navazování chodeb, ale ty nejsou pro správnou funkci algoritmu podstatné.

Vysoká a nepředpokládaná hodnota RMSE (tabulka 6.2) u *Hector SLAMu* s odometrií je dána lepším odhadem pozice oproti *Hector SLAMu* bez odometrie a současně se vyskytujícím efektem zastavení robota.



(a) Gmapping (b) Karto SLAM

Obr. 6.7: SLAM algoritmy v dlouhých chodbách s odometrií

Tab. 6.2: RMSE srovnání trajektorií SLAM algoritmů v uzavřené smyčce

	RMSE [m]
Hector SLAM bez odometrie	2.5928
Hector SLAM s odometrií	3.6517
Gmapping	0.6927
Karto SLAM	0.4232

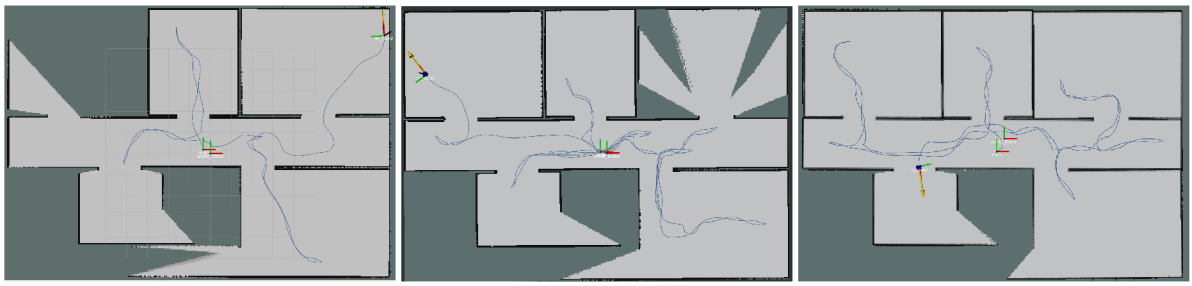
6.2 Autonomní vytváření mapy prostředí

Autonomní průzkum je založený na vyhledávání cílových bodů, ke kterým je směřována trajektorie robota. Vyhledávání těchto bodů závisí na kvalitě mapy. V případě nekvalitní nebo pomalu se tvořící mapy může docházet k repetitivnímu prozkoumávání známých míst.

Autonomní průzkum a tvorba mapy probíhala v budově obsahující pouze stěny a prostředí je tady chudé na význačné body. Každý experiment byl ukončen při možné srážce robota se stěnou a znehodnocení výsledků. Výsledné mapy jsou zobrazeny níže 6.8.

Kvalita a ostrost zobrazení překážek odpovídá výsledkům v kapitole 6.1.1. Největší změnou, po technické stránce mapy, je v hodnotách *RMSE*, které shrnuje tabulka (6.3).

Jak bylo naznačeno v kapitole 6.1.1, důležitým parametrem při lokalizaci v mapě je správný odhad úhlu natočení robota. *Hector SLAM* nejpřesněji zpracovává data z *Lidaru*, a proto nejlépe odhaduje natočení robota v prostoru. Navíc lze k němu připojit i data z *IMU* (Inertial measurement unit), tedy informace o zrychlení v



(a) Hector SLAM (b) Karto SLAM (c) Gmapping

Obr. 6.8: Mapa vzniklá z autonomního průzkumu

Tab. 6.3: RMSE srovnání trajektorií SLAM algoritmů - autonomní průzkum

	RMSE [m]
Hector SLAM	0.3925
Gmapping	0.7275
Karto SLAM	0.3088

jednotlivých osách, což může vylepšit výsledný odhad. Proto při delší a složitější trajektorii nedochází k takovému zhoršení vzhledem k testování v kapitole 6.1.1 oproti ostatním algoritmům.

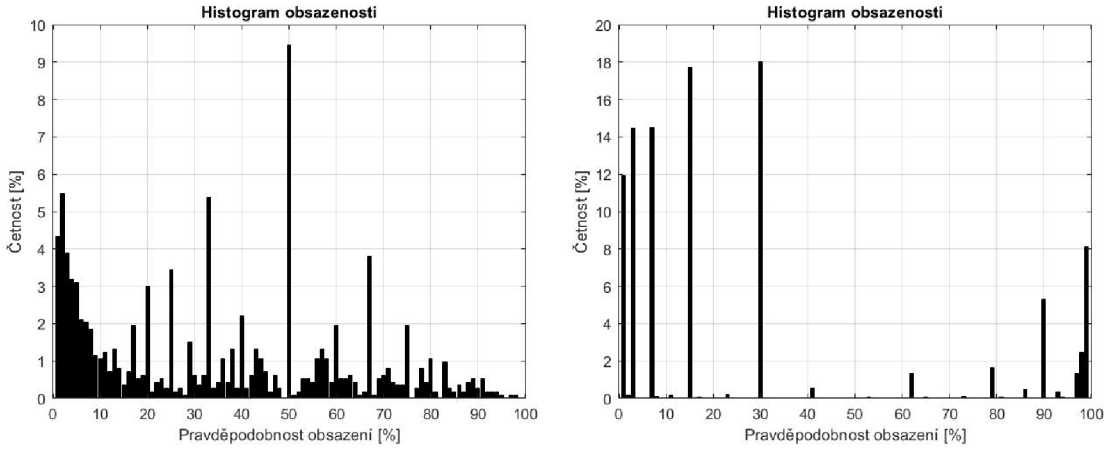
U *Karto SLAM* algoritmu dochází k častějšímu přejíždění ve známých prostorách, jelikož nejsou nová data z *Lidaru*, tedy nové neznáme prostory, dostatečně rychle zmapovány s dostatečnou přesností. Tato vlastnost pravděpodobně vychází z nižší spotřeby výpočetního výkonu.

Vyšší hodnota *RMSE* u *Gmapping SLAM* algoritmu je způsobená nejdelší uraženou trajektorií, jelikož u autonomního prozkoumávání dlouho dobu nedocházelo k možnosti kolize a po zastavení robota už nedošlo k takovému opravení vzájemné polohy souřadnicových soustav *map* a *world_frame* jako u zbylých dvou algoritmů. I přes to byla hodnota v průběhu mapování vyšší než u *Hector* a *Karto SLAM* algoritmů.

6.3 Testování SLAM algoritmů v reálném prostředí

Nejdůležitějším aspektem pro zhodnocení kvality *SLAM* algoritmů podle kritérií popsaných v kapitole 6 je hodnota RMSE odchylek domnělé a pravé trajektorie. V simulaci jde reálná trajektorie jednoduše získat, ale v reálném testování jde pravá trajektorie získat obtížně a nepřesně, a proto by výsledky nebyly vypovídající. Nejvíce

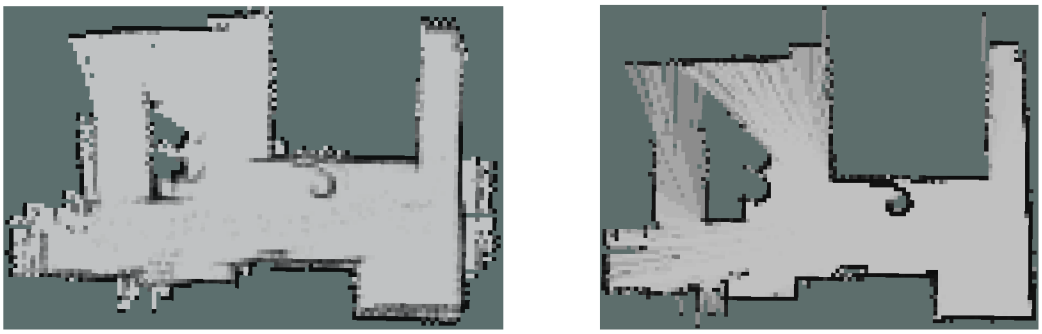
vypovídajícím kritériem k ohodnocení algoritmů v reálném prostředí je histogram obsazenosti.



(a) Gmapping (b) Hector SLAM

Obr. 6.9: Histogramy SLAM algoritmů v reálném prostředí

Výsledky získané z reálného prostředí korelují s výsledky ze simulace. V reálném prostředí se zvětšují rozdíly mezi přednostmi a slabinami jednotlivých algoritmů. Překážky vytvořené *Hector SLAM* algoritmem jsou převážně ostré a volný prostor je rovněž správně detekován. U *Gmappingu* je vidět značné rozostření překážek a taky nejistota při stanovení volného prostoru. Výsledky z histogramů potvrzují i výsledné mapy zobrazené na obrázcích níže (6.10).



(a) Gmapping (b) Hector SLAM

Obr. 6.10: SLAM algoritmy v reálném prostředí

Další kritérium použité k ohodnocení kvality algoritmů je vizuální zhodnocení. U *Gmappingu* jsou na první pohled viditelné chyby značící významně horší kvalitu vytvořené mapy oproti *Hector SLAM* algoritmu.

Závěr

V průběhu práce bylo dosaženo několika důležitých milníků vedoucích ke splnění cíle práce, a to vytvoření mobilní robotické platformy se schopností rozběhnout již existující 2D SLAM algoritmy implementované v ROSu (Robot Operating System) a otestovat je v simulátoru reálné fyziky a v reálném prostředí.

Jedním z těchto milníků bylo vytvoření modelu robota v simulátoru Gazebo spolupracujícího s frameworkem ROS. Současně s modelem robota byly nastaveny stejné fyzikální vlastnosti senzoru Lidar podle snímače RPLIDAR A1, který byl použit při reálném testování.

Po úspěšné simulaci modelu robota s příslušnými senzory v simulátoru bylo nezbytné vytvořit základní software. Na základě kinematických vlastností diferenciálně řízeného dvoukolového robota vznikl řídící *node* zajišťující pohyb robota pomocí ovládání, které dostane od uživatele prostřednictvím klávesnice.

S tímto základním softwarem už mělo smysl implementovat jeden z vybraných SLAM algoritmů - Hector SLAM. Ten je založený pouze na datech z Lidaru nevyžadující externí lokalizaci. Další vybrané algoritmy, tj. Gmapping a Karto SLAM, používají k jejich správné funkci data o poloze robota. V práci proto byla implementována základní odometrie.

Dalším významným milníkem byla implementace autonomního průzkumu budov s účelem zmapovat všechny části budovy. Autonomní průzkum je založený na pravděpodobnostním plánování trajektorie využívající Rapidly-exploring Random Tree neboli RRT. Tento algoritmus rychle konverguje k cílovému bodu i v relativně komplikovaném prostředí. Největší slabinou algoritmu RRT, a tedy i autonomního průzkumu, je nalezení trajektorie v úzkých uličkách. Po nalezení trajektorie k cílovému bodu určující budoucí pohyb robota, který je vyhledáván na základě dosavadně získané mapy a směřující k neprozkoumaným místům, bylo nutné do řídícího *nodu* přidat algoritmus ke sledování trajektorie robotem. Trajektorie nalezená pomocí algoritmu RRT obsahuje singulární body, ve kterých není trajektorie diferencovatelná, a proto by se v každém singulárním bodě musel robot zastavit, změnit svojí orientaci a znovu se rozjet k dalšímu singulárnímu bodu a pohyb robota by tak nebyl plynulý. Tento problém byl vyřešen spojitou aproximací trajektorie Bézierovou křivkou nebo řídícím algoritmem Pure Pursuit.

K vyhodnocení kvality SLAM algoritmů pomocí vzniklých map neexistuje jednotné pravidlo, a proto bylo v práci stanoveno kritérium kvality podle něhož se určuje přesnost a kvalita mapy. Kritéria jsou určena následujícími body

- Srovnání pravé a domnělé trajektorie robota v mapě
- Histogram obsazenosti určující rozostření překážek
- Vizuální hledání jednoznačných chyb

- Výpočetní náročnost algoritmu

Algoritmy byly testovány v simulaci v několika případech. První z nich je běžná budova obsahující hodně význačných a rozličných bodů a druhým případem je extrémní prostředí mající za cíl znemožnit vytvoření mapy.

V první budově jsou výsledné mapy všech SLAM algoritmů velice podobné. Podle prvního kritéria, tj. nejmenší rozdíl pravé a domnělé trajektorie robota, byly rozdíly všech algoritmů vzhledem k délce uražené trajektorie minimální, ale nejlepších výsledků dosahoval Karto SLAM, který má navíc pro malé budovy nejnižší výpočetní náročnost. Vzhledem ke kvalitě vytvořené mapy, podle druhého kritéria, měl nejlepší a nejstabilnější výsledky Hector SLAM. Jeho výhodou je možnost zlepšení výsledků pomocí IMU a použití pro roboty s až šesti stupni volnosti.

V druhém extrémním případě se robot pohyboval v dlouhých chodbách chudých na význačné body. Problém nastává v případě, kdy je délka chodeb delší, než je dosah Lidaru a kdy se robot pohybuje rovnoběžně se stěnami. V takovém případě robot dostává stejná data a je nemožné správně vytvořit mapu, jelikož si robot myslí, že se nepohybuje a dochází ke zkracování chodeb. Řešením je přidání lokalizační techniky nezávislé na datech ze senzoru, v tomto případě odomtrie.

Výsledky map v reálném světě korelují s výsledky v simulaci, jen se prohlubuje rozdíl mezi přednostmi a slabinami jednotlivých algoritmů.

Literatura

[1] QUIGLEY, Morgan, Brian GERKEY a William SMART. Programming Robots with ROS: A PRACTICAL INTRODUCTION TO THE ROBOT OPERATING SYSTEM [online]. Sebastopol, California: O'Reilly Media, 2015 [cit. 2019-11-04]. ISBN 978-1-449-32389-9.

[2] ROS Wiki. ROS.org [online]. 2008 [cit. 2019-11-10]. Dostupné z: http://wiki.ros.org/

[3] DUDEK, Gregory a Michael JENKIN. Computational Principles of Mobile Robotics. 2nd Edition. New York: Cambridge University Press, 2010. ISBN 978-0-521-69212-0.

[4] CHHOTRAY, Animesh, Manas K. PRADHAN, Krishna K. PANDEY a Dayal R. PARHI. Kinematic Analysis of a Two-Wheeled Self-Balancing Mobile Robot. Proceedings of the International Conference on Signal, Networks, Computing, and Systems. New Delhi: Springer India, 2016, 2016-10-15, , 87-93. Lecture Notes in Electrical Engineering. DOI: 10.1007/978-81-322-3589-7_9. ISBN 978-81-322-3587-3. Dostupné z: http://link.springer.com/10.1007/978-81-322-3589-7_9

[5] Gazebo Tutorials: Get Started. Http://gazebosim.org/: Robot simulation made easy. [online]. Open Source Robotics Foundation, 2014 [cit. 2019-11-21]. Dostupné z: http://gazebosim.org/tutorials?cat=get_started

[6] RIISGAARD, Sren a BLAS, Morten Rufus. SLAM for Dummies [online]. [cit. 2020-04-24]. Dostupné z: https://dspace.mit.edu/bitstream/handle/1721.1/36832/16-412JSpring2004/NR/rdonlyres/Aeronautics-and-Astronautics/16-412JSpring2004/A3C5517F-C092-4554-AA43-232DC74609B3/0/1Aslam_blas_report.pdf

[7] MONTEMERLO, Michael a Sebastian THRUN. FastSLAM: A Scalable Method for the Simultaneous Localization and Mapping Problem in Robotics [online]. Berlín: Springer-Verlag Berlin Heidelberg, 2007 [cit. 2020-04-24]. ISBN 978-3-540-46399-3.

[8] List of moments of inertia. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-04-14]. Dostupné z: https://en.wikipedia.org/wiki/List_of_moments_of_inertia

[9] ROBOPEAK TEAM a SHANGHAI SLAMTEC. RPLIDAR A1: Low Cost 360 Degree Laser Range Scanner Introduction and Datasheet. Rev.1.0. Shanghai Slamtec, 2016. Dostupné z: <https://www.robotshop.com/media/files/pdf/rplidar-a1m8-360-degree-laser-scanner-development-kit-datasheet-1.pdf>

[10] BEN-ARI, Moti. A Tutorial on Euler Angles and Quaternions [online]. Version 2.0.1. California, USA: Department of Science Teaching Weizmann Institute of Science, 2014 [cit. 2020-04-15]. Dostupné z: https://www.weizmann.ac.il/sci-tea/benari/sites/sci-tea.benari/files/uploads/softwareAndLearningMaterials/quaternion-tutorial-2-0-1.pdf

[11] SKALKA, Marek. Srovnání lokalizačních technik [online]. Praha, 2011 [cit. 2020-04-16]. Dostupné z: http://marek.sk.sweb.cz/lokalizace/index.html. Diplomová práce. Univerzita Karlova v Praze Matematicko-fyzikální fakulta. Vedoucí práce David Obdržálek.

[12] WINKLER, Zbyněk. Odometrie. Robotika.cz [online]. Robotika.cz, 2005 [cit. 2020-04-16]. Dostupné z: https://robotika.cz/guide/odometry/cs

[13] SIEGWART, Roland a Illah NOURBAKHSH. Introduction to Autonomous Mobile Robots [online]. Cambridge, Massachusetts: The MIT Press, 2004 [cit. 2020-04-18]. ISBN 9780262195027.

[14] Prohledávání do hloubky. Algoritmy.net [online]. [cit. 2020-04-17]. Dostupné z: https://www.algoritmy.net/article/1378/Prohledavani-do-hloubky

[15] Prohledávání do šířky. Algoritmy.net [online]. [cit. 2020-04-17]. Dostupné z: https://www.algoritmy.net/article/1399/Prohledavani-do-sirky

[16] Dijkstrův algoritmus. Algoritmy.net [online]. [cit. 2020-04-17]. Dostupné z: https://www.algoritmy.net/article/5108/Dijkstruv-algoritmus

[17] Bézier curve. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2020-04-17]. Dostupné z: https://en.wikipedia.org/wiki/B%C3%A9zier_curve

[18] Implementation of the Pure Pursuit Path Tracking Algorithm. Pennsylvania: The Robotics Institute Camegie Mellon University Pittsburgh, 1992. [cit. 2020-04-17]. Dostupné z: https://www.ri.cmu.edu/pub_files/pub3/coulter_r_craig_1992_1/coulter_r_craig_1992_1.pdf

[19] LAVALLE, Steven M. Rapidly-Exploring Random Trees: A New Tool for Path Planning [online]. Ames, IA USA: Department of Computer Science Iowa State University [cit. 2020-04-17]. Dostupné z: http://msl.cs.illinois.edu/~lavalle/papers/Lav98c.pdf

[20] ANTON, Filatov, Filatov ARTYOM, Krinkin KIRILL, Chen BAIAN a Molodan DIANA. 2D SLAM Quality Evaluation Methods [online]. St. Petersburg, Russia: Saint-Petersburg Electrotechnical University, 2017 [cit. 2020-04-21]. Dostupné z: https://www.researchgate.net/publication/319002279_2D_SLAM_Quality_Evaluation_Methods

[21] NAMOSHE, Molaletsa, Oduetse MATSEBE a Nkgatho TLALE. Feature extraction: techniques for landmark based navigation system. Sensor Fusion and its Applications [online]. Croatia: Sciyo, 2010, s. 347-373 [cit. 2020-04-26]. ISBN 978-953-307-101-5. Dostupné z: https://pdfs.semanticscholar.org/549a/8b14bcbf8b59e655480de2caa353da3ecaca.pdf

[22] An Iterative Closest Points Algorithm for Registration of 3D Laser Scanner Point Clouds with Geometric Features. Sensors [online]. 2017, , 3-7 [cit. 2020-05-23]. DOI: 10.3390/s17081862. Dostupné z: https://www.researchgate.net/publication/319072034_An_Iterative_Closest_Points_Algorithm_for_Registration_of_3D_Laser_Scanner_Point_Clouds_with_Geometric_Features

[23] CALONDER, Michael. EKF SLAM vs. FastSLAM - A Comparison [online]. Lausanne: Swiss Federal Institute of Technology, Computer Vision Lab [cit. 2020-05-23]. Dostupné z: https://infoscience.epfl.ch/record/146805/files/ekf_fastslam_comp.pdf

[24] BECKER, Alex. KalmanFilter.net [online]. 2018 [cit. 2020-05-24]. Dostupné z: https://www.kalmanfilter.net

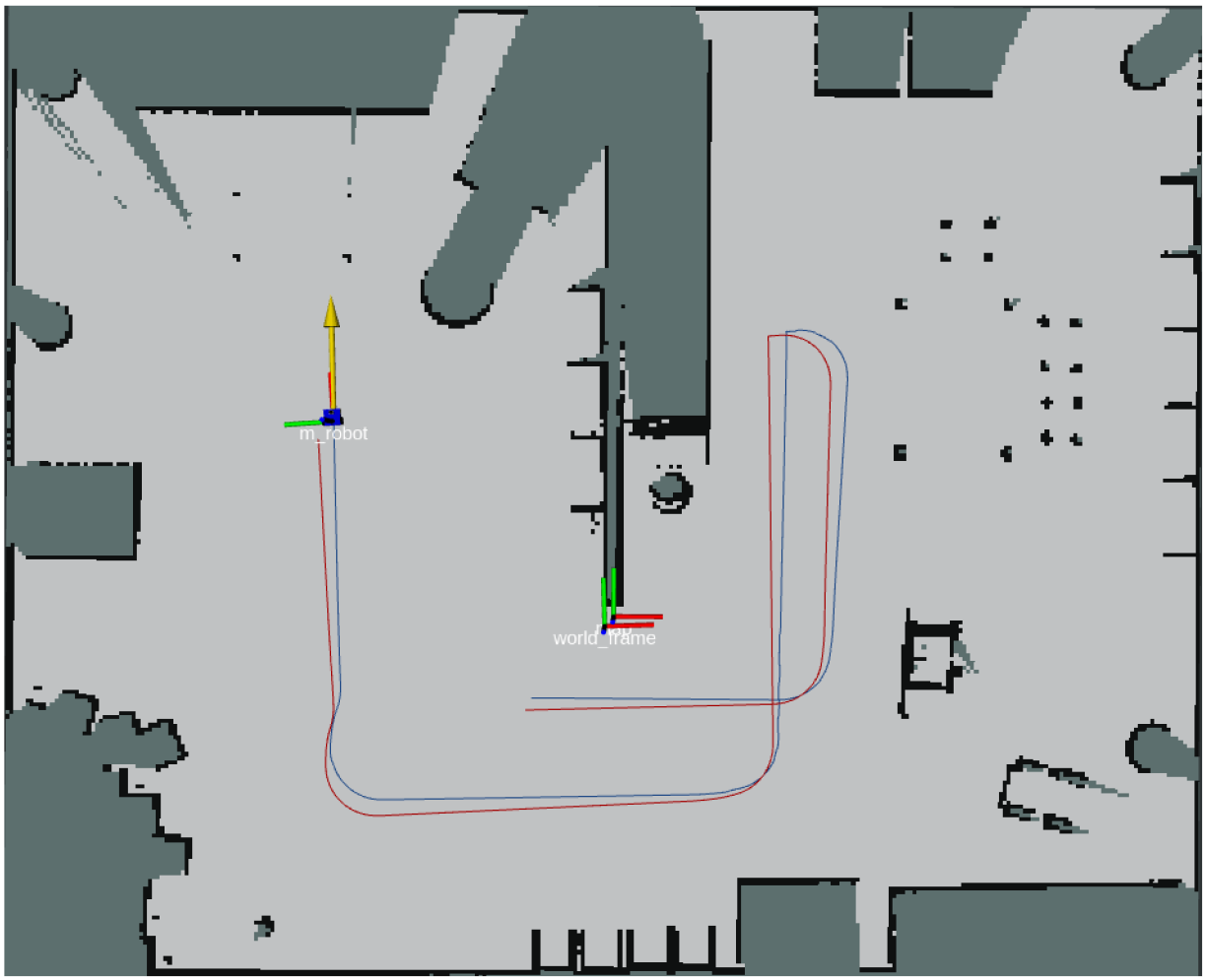
[25] SU, Huaicheng. FastSLAM An Improved Particle FilteringAlgorithm for Simultaneous Localization andMapping: A Survey [online]. University of Windsor: School of Computer Science [cit. 2020-05-24]. Dostupné z: https://pdfs.semanticscholar.org/8e89/01197eeee430531c7661f236f06b5cd97bb1.pdf

Seznam symbolů, veličin a zkratek

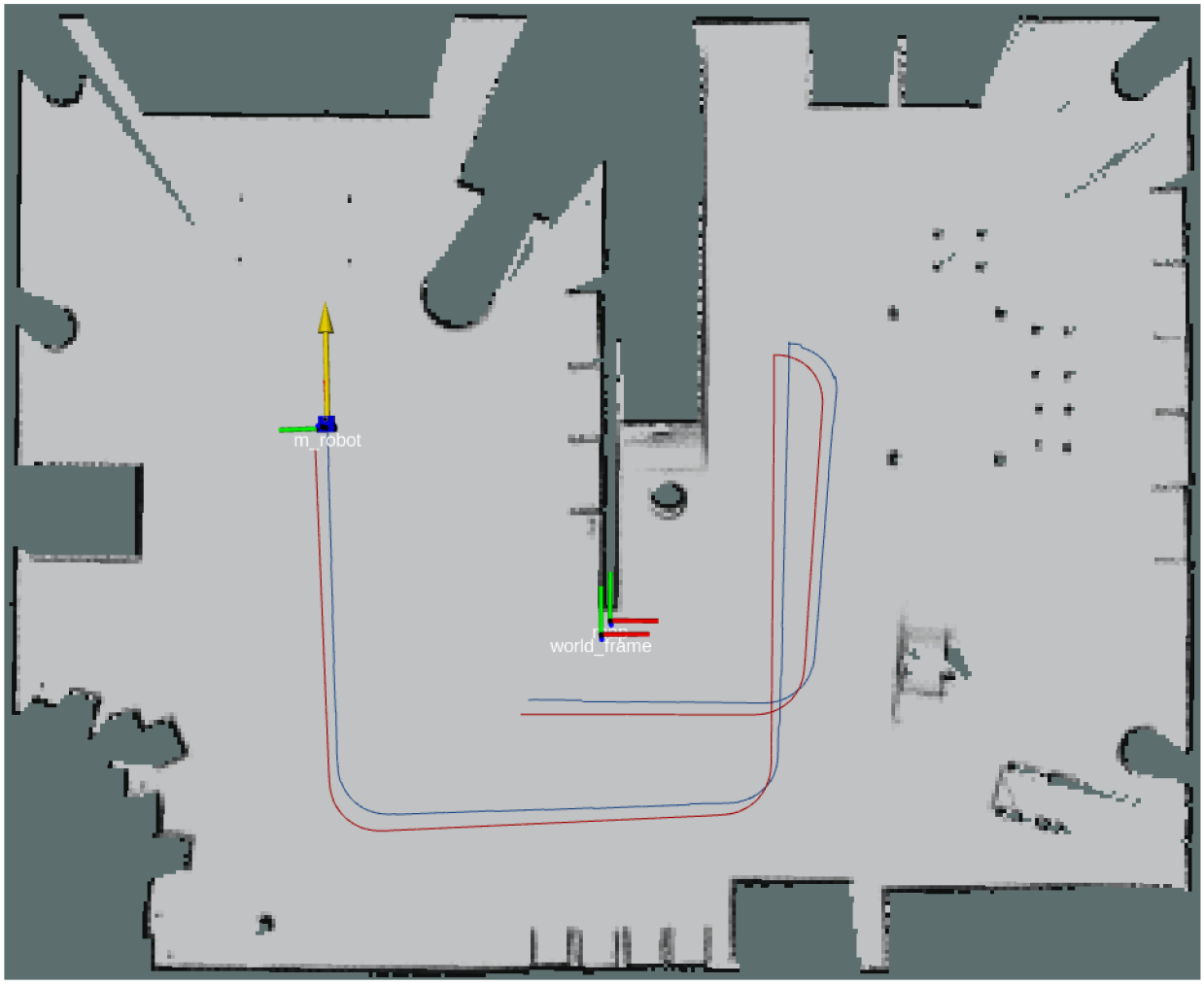
SLAM	Simultaneous localization and mapping
ROS	Robot Operating System
v_r	Obvodový rychlost pravého kola
v_l	Obvodový rychlost levého kola
D	Rozteč mezi koly
R	Poloměr kružnice pohybu robota
v	Rychlost robota
ω	Úhlová rychlost robota
p	Pozice robota v prostoru
x	X-ová souřadnice pozice robota v prostoru
y	Y-ová souřadnice pozice robota v prostoru
θ	Natočení robotu
URDF	Unified robot description format
XACRO	XML Makro
LIDAR	Light Detection And Ranging
R	Matice rotace
$\boldsymbol{O}$	Množina všech obsazených mřížek v mřížce obsazenosti
$\boldsymbol{V}$	Množina všech průchodných mřížek v mřížce obsazenosti
$\boldsymbol{N}$	Množina všech neznámých mřížek v mřížce obsazenosti
RMSE	Root-mean-square error
IMU	Inertial measurement unit

A Výsledné mapy získané ze simulace

A.1 Výsledná mapa



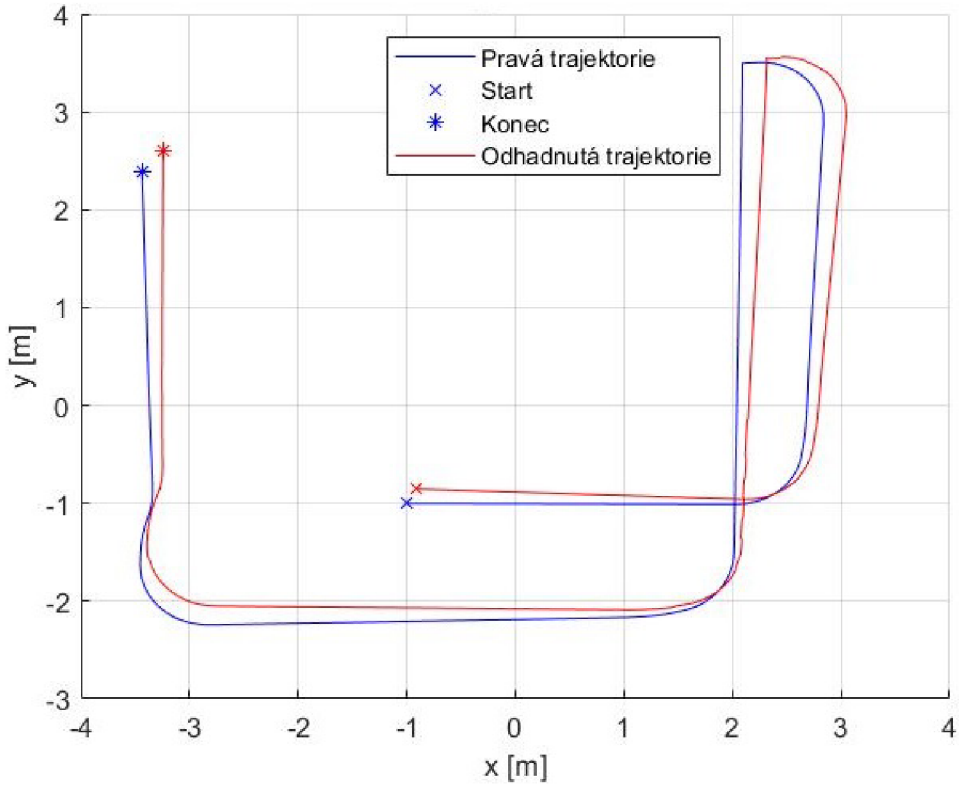
Obr. A.1: Mapa vzniklá pomocí Karto SLAM algoritmu



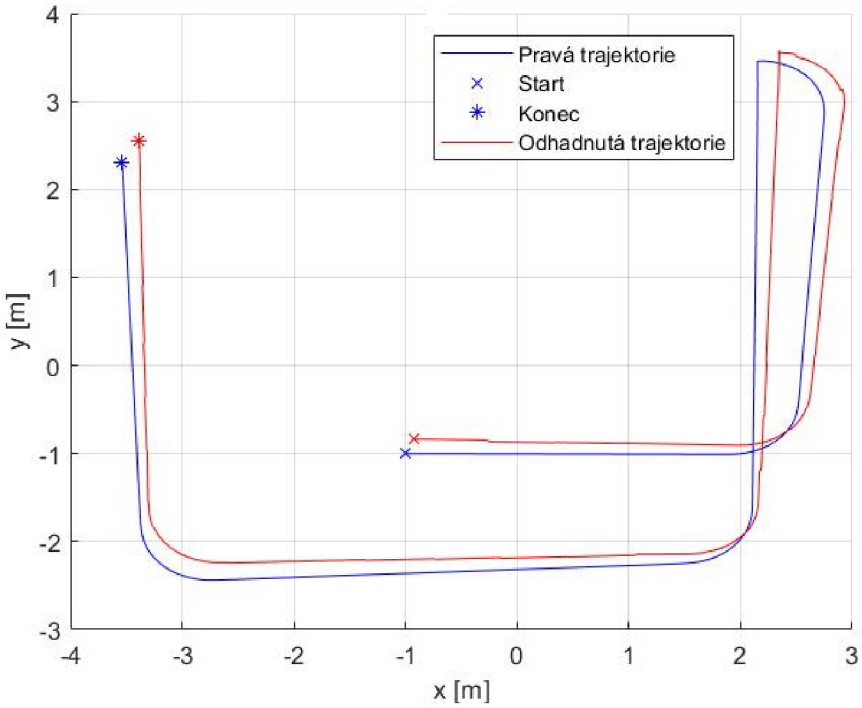
Obr. A.2: Mapa vzniklá pomocí Gmapping algoritmu

Obr. A.3: Mapa vzniklá pomocí Hector SLAM algoritmu

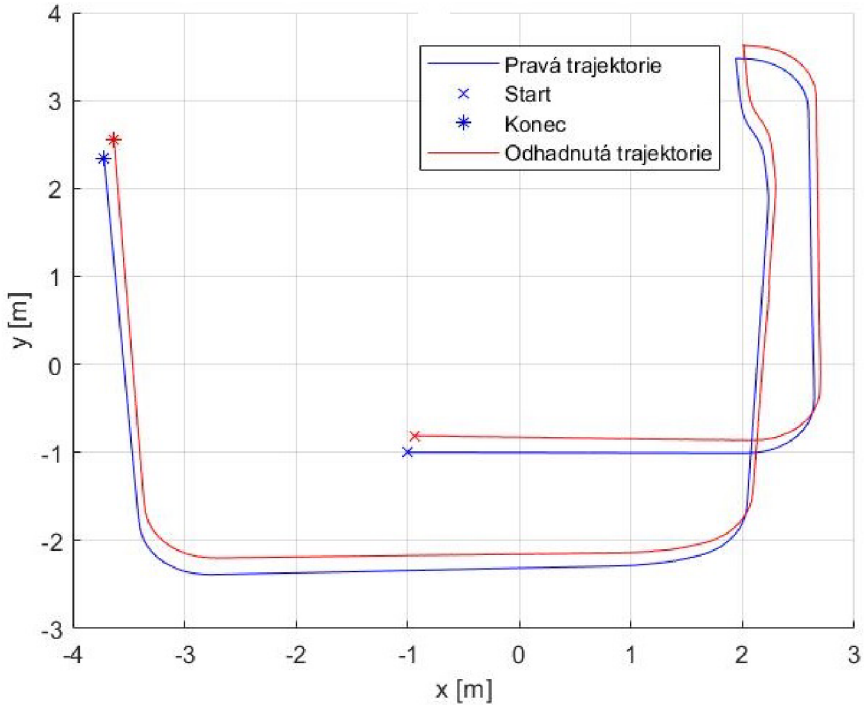
A.2 Trajektorie



Obr. A.4: Karto SLAM trajektorie



Obr. A.5: Gmapping trajektorie



Obr. A.6: Hector SLAM trajektorie

B Výsledné mapy získané ze reálného světa

Obr. B.1: Mapa vzniklá pomocí Hector SLAM algoritmu v reálném světě

Obr. B.2: Mapa vzniklá pomocí Gmapping algoritmu v reálném světě

C Adresářová struktura zdrojových souborů

```
core - obsahuje všechen software potřebný pro reálného robota
├── m_exploration - složka obsahující zdrojové kódy pro autonomní průzkum
│   (struktura všech složek je stejná)
│   ├── include - obsahuje hlavičkové soubory
│   ├── launch - obsahuje Launchfily
│   ├── src - obsahuje zdrojové cpp soubory
│   ├── CMakeLists.txt
│   └── package.xml
├── keyboard_driver
│   └── Zdrojové kódy pro manuální ovládání
├── m_controller
│   └── Zdrojové kódy řízení robota
├── m_odometry
│   └── Zdrojové kódy zajišťující odometrii
├── m_robot
│   └── Zdrojové kódy pro implementaci na reálného robota
├── slam_quality
│   └── Zdrojové kódy zjišťující kvalitu map
├── hector_slam
│   └── Externí zdrojové kódy pro Hector SLAM
├── slam_gmapping
│   └── Externí zdrojové kódy pro Gmapping SLAM
├── slam_karto
│   └── Externí zdrojové kódy pro Karto SLAM
└── rplidar_ros
    └── Externí zdrojové kódy ovládající RPLIDAR
simulation - obsahuje všechny soubory potřebné k simulace robotu
├── gazebo_loc
│   └── Zdrojové kódy poskytující absolutní lokalizaci v simulátoru
└── simulation
    ├── launch
    ├── models
    ├── rviz
    └── worlds
```