České vysoké učení technické v Praze
Fakulta strojní

BAKALÁŘSKÁ PRÁCE

Malý mobilní robot pro mapování a lokalizaci

Praha 2020 **Martin Vejbora**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vejbora** Jméno: **Martin** Osobní číslo: **472973**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav mechaniky, biomechaniky a mechatroniky**

Studijní program: **Teoretický základ strojního inženýrství**

Studijní obor: **bez oboru**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Malý mobilní robot pro mapování a lokalizaci

Název bakalářské práce anglicky:

Small mobile robot for mapping and localization

Pokyny pro vypracování:

1) Seznamte s úlohou robotického mapování a lokalizace.
2) Seznamte se s prostředím ROS.
3) Vytvořte simulační model vybraného podvozku robota.
4) Ověřte možnosti použití kamery Intel Realsense d435i pro úlohu mapování a lokalizace.

Seznam doporučené literatury:

[1] Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part I, IEEE Robotics & Automation Magazine, vol. 13 (2), pp. 99–110, 2006.
[2] Realsense in ROS, [on-line] http://wiki.ros.org/RealSense, cit. 15-03-2020.
[3] Vrbský, L.: Experimentální model vozidla s asistentem pro couvání s přívěsem, bakalářská práce, FS ČVUT v Praze, 2016.
[4] Mikio, S.: Build an Autonomous Mobile Robot with the Intel RealSense Camera, ROS, and SAWR, [on-line] http://software.intel.com/en-us/articles/build-an-autonomous-mobile-robot-with- the-intel-realsense-ros-and-sawr,cit. 15-03-2020.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Petr Beneš, Ph.D., odbor mechaniky a mechatroniky FS

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **30.04.2020** Termín odevzdání bakalářské práce: **11.08.2020**

Platnost zadání bakalářské práce: _____

Ing. Petr Beneš, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Miroslav Španiel, CSc. podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

Anotační list

Jméno autora:	**Martin Vejbora**
Název bakalářské práce:	**Malý mobilní robot pro mapování a lokalizaci**
Anglický název:	**Small mobile robot for mapping and localization**
Akademický rok:	2019/2020
Studijní obor:	Teoretický základ strojního inženýrství
Ústav, odbor:	Ústav mechaniky, biomechaniky a mechatroniky Odbor mechaniky a mechatroniky
Vedoucí bakalářské práce:	Ing. Petr Beneš, Ph.D
Bibliografické údaje:	48 stran 29 obrázků 1 x CD příloha
Klíčová slova:	Současná lokalizace a mapování, SLAM, Intel RealSense D435i, Robot Operating System, ROS, Gazebo
Key words:	Simultaneous localization and mapping, SLAM, Intel RealSense D435i, Robot Operating System, ROS, Gazebo

Abstrakt: Tato práce se zabývá možností implementace mobilního robota pro úlohu současného mapování a lokalizace (SLAM). V úvodu práce je SLAM popsán a následně jsou nastíněny základní algoritmy pro jeho řešení. Práce také obsahuje stručné seznámení s prostředím ROSu (Robot Operating System). V další části je pomocí experimentu ověřena vhodnost kamery Intel RealSense D435i pro úlohu současného mapování a lokalizace. Na závěr je pro zvolený podvozek vytvořen model a provedena simulace SLAMu ve virtuálním prostředí Gazeba.

Abstract: The goal of this thesis is to design a mobile robot for simultaneous mapping and localization (SLAM). The introduction describes the SLAM problem and outlines basic algorithms for it. The thesis continues with a brief description of the ROS (Robot Operating System) environment. Then, the suitability of the Intel RealSense D435i camera for the task of simultaneous mapping and localization is verified with an experiment. In the end, a model of a small mobile robot is created and a simulation of the SLAM is performed with this robot in Gazebo.

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a použil pouze podklady uvedené v přiloženém seznamu literatury.

V Praze dne _____ _____

podpis

Poděkování

Rád bych poděkoval vedoucímu své bakalářské práce Ing. Petru Benešovi, Ph.D. za ochotu a čas, který mé práci věnoval, a především za to, že mi dodal optimismus ve chvílích, kdy jsem pochyboval, že se mi podaří práci úspěšně dokončit.

Dále bych rád poděkoval rodině za podporu při studiu a v neposlední řadě za pomoc s gramatickou stránkou práce.

Obsah

Seznam obrázků **8**

1 Úvod **9**
1.1 Mapování a lokalizace 9
1.2 Intel RealSense kamery 10
1.3 Model robota 10

2 Cíle práce **11**

3 SLAM **12**
3.1 Formální popis 12
3.1.1 Definice pravděpodobnostního SLAMu 14
3.2 Algoritmy 15
3.2.1 EKF SLAM 15
3.2.2 FastSLAM 17
3.2.3 Další algoritmy 22
3.2.4 Reprezentace mapy 23
3.3 Vhodné senzory 23

4 Robot Operating System **26**
4.1 Architektura 26
4.1.1 Topics 26
4.1.2 Services 27
4.1.3 Actions 28
4.2 Balíčky 28
4.3 Gazebo 29

5 Kamera Intel RealSense D435i **30**
5.1 Princip fungování 30
5.1.1 IMU 32
5.1.2 Vlastnosti 32
5.2 Spojení s ROSem 32
5.3 Ověření vhodnosti pro SLAM 32

6 Model robota **35**
6.1 Výběr podvozku . . . 35
6.1.1 Diferenciální podvozek . . . 35
6.1.2 Všesměrový podvozek . . . 36
6.1.3 Ackermannův podvozek . . . 37
6.1.4 Výběr podvozku . . . 37
6.2 Model podvozku pro Gazebo . . . 38
6.3 Simulace RealSense kamery v Gazebu . . . 39
6.4 Svět v Gazebu . . . 39
6.5 SLAM v simulovaném prostředí . . . 40
6.5.1 Výsledky simulace . . . 40
6.5.2 Nedostatky simulace . . . 41

7 Závěr **43**
7.1 Navazující práce . . . 43

Použité zdroje **45**

A Přílohy **48**

Seznam obrázků

3.1 Znázornění pohybu robota v neznámém prostředí 13
3.2 2D Gaussovo rozdělení 15
3.3 Odhad polohy robota a orientačních bodů podle Gaussova rozdělení 17
3.4 Lokalizace pomocí částicového filtru, čas t = 0 18
3.5 Lokalizace pomocí částicového filtru, čas t = 1 19
3.6 Lokalizace pomocí částicového filtru, čas t = 2 20
3.7 FastSLAM v rovině 22
3.8 Detail mřížky obsazenosti 23
3.9 Mřížka obsazenosti vytvořená mapováním Intel Research Labu . . 23

4.1 Diagram zachycující komunikaci uzlů pomocí topics 27
4.2 Diagram komunikace uzlů pomocí services 27
4.3 Diagram komunikace dvou uzlů pomocí actions 28
4.4 Screenshot Gazeba s robotem Husky 29

5.1 Kamera Intel RealSense D435i 30
5.2 Stereoskopické vidění 30
5.3 Scéna zachycená kamerou Intel RealSense D435i 31
5.4 Screenshot RtabMapViz. 34
5.5 Mřížka obsazenosti sestavená mapováním místnosti 34
5.6 Fotka mapovaného prostoru 34

6.1 Dvoukolový diferenciální podvozek 35
6.2 Husky robot se čtyřkolovým diferenciálním podvozkem 35
6.3 Příklad všesměrového podvozku 36
6.4 Diagram všesměrového podvozku 36
6.5 Porovnání Ackermannova a diferenciálního podvozku. 37
6.6 Základní model robota v Gazebu 38
6.7 Model robota s kamerou v Gazebu 39
6.8 Modelová místnost vytvořená pro simulaci v Gazebu 40
6.9 Průběh mapování simulovaného prostředí 41
6.10 Mřížka obsazenosti a 3D mapa vytvořená pomocí SLAMu v Gazebu 41

1. Úvod

Jednou ze současných výzev v mobilní robotice je autonomní navigace, která už v současné době nachází uplatnění v řadě aplikací. Příkladem mohou být autonomní automobily nebo roboti pro průzkum nebezpečných či zamořených prostor. Jedním z cílů této práce je seznámení se právě s problematikou autonomní navigace.

Aby se robot mohl samostatně navigovat, musí mít k dispozici alespoň následující informace:

- Mapu prostoru, kde se nachází, se všemi překážkami
- Svoji pozici na této mapě

S těmito informacemi pak může začít plánovat trasu. To znamená, jak se dostat ze současné pozice do určené cílové destinace tak, aby se při tom vyhnul všem překážkám.

1.1 Mapování a lokalizace

Mapu prostoru může robot získat dvěma způsoby. První variantou je, že mu **nahrajeme existující mapu** prostředí, kterou jsme předtím vytvořili na základě měření. Druhou variantou je, že necháme na robotovi, aby si **mapu sestavil sám**. Výhoda druhého přístupu je, že lze robota použít v jakémkoliv novém prostoru, bez nutnosti vytvářet a nahrávat nové mapy. Budeme se proto dále zabývat pouze tímto přístupem.

Lokalizaci na mapě můžeme také rozdělit do dvou hlavních kategorií. Pomocí **globální lokalizace** určujeme absolutní souřadnice pozice robota. Mezi zástupce patří například technologie GPS nebo určování polohy pomocí Wi-Fi hotspotů. Lokalizační zařízení na robotovi komunikuje s přístroji vhodně umístěnými v prostoru. Nevýhodou Wi-Fi hotspotů je, že vyžadují instalaci dalších zařízení ve zkoumaném prostoru. K nedostatkům GPS patří, že má špatný signál uvnitř budov, kde není přímá viditelnost mezi senzorem na robotovi a satelity. Oproti tomu **lokální metody** poskytují pouze lokální informaci, například vzdálenost od senzoru. Příkladem mohou být laserové skenery (lidary), radary nebo různé druhy kamer. Mezi lokální metody také patří enkodéry, pomocí kterých můžeme počítat otáčky kol, nebo jednotky IMU (Inertial Measurement Unit), které v sobě obsahují nejčastěji akcelerometr a gyroskop. Všechna uvedená zařízení mají společné,

že se jedná pouze o zařízení připevněná přímo na robotovi. Navíc jsou zpravidla přesnější než globální metody. V této práci se omezíme pouze na lokální metody [1].

V předchozích odstavcích jsme na základě rozboru výhod a nevýhod zvolili, že si náš robot bude vytvářet mapu prostředí sám a že nebudeme spoléhat na globální metody lokalizace. Robot si tedy bude vytvářet mapu a zároveň se v ní lokalizovat. Tento přístup se nazývá SLAM (Simultaneous Localization And Mapping). Do češtiny se označení překládá jako současná lokalizace a mapování. V této práci je jí věnována kapitola 3.

1.2 Intel RealSense kamery

Intel představil v nedávné době produktovou řadu kamer RealSense. Tyto kamery umožňují snímat prostorová data. My se zaměříme na otázku, zda jsou vhodné jako primární senzor pro SLAM. Výhodou těchto kamer oproti běžně používaným lidarům je, že jeden přístroj v sobě kombinuje jak snímání hloubkový dat, tak viditelného obrazu. To může být užitečné například pro spolupráci robot – člověk, kdy robot může zobrazovat 3D model prostředí zároveň s obrazem z kamery, což je pro člověka přirozenější zdroj informací.

Pro naší práci máme k dispozici kameru Intel RealSense D435i. V kapitole 5 se podíváme detailněji na specifikaci této kamery a pokusíme se ověřit její vhodnost pro simultánní lokalizaci a mapování.

Intel nabízí ke svým RealSense kamerám knihovnu s názvem Intel RealSense SDK 2.0. Jde o multiplatformní knihovnu napsanou v C/C++. K této knihovně jsou navíc k dispozici wrappery pro další jazyky a frameworky, mezi které patří C#, Python, MATLAB, Unity, Unreal Engine, ROS a další. Kompletní seznam a více informací lze nalézt na oficiálních stránkách Intelu [2] nebo na GitHub stránce knihovny [3], kde lze nalézt také wrappery navíc, které jsou zatím ještě ve vývoji. My jsme si pro naši práci zvolili ROS (Robot Operating System), a to hlavně kvůli jeho rozšířenosti v robotice. O ROSu více pojednává kapitola 4.

1.3 Model robota

V kapitole 6 vytvoříme model robota na vybraném podvozku. K tomuto modelu pak připojíme virtuální RealSense kameru a pomocí ROSu nasimulujeme úlohu současné lokalizaci a mapování.

2. Cíle práce

V této kapitole uvádíme stručné shrnutí hlavních cílů práce, abychom se na ně mohli v dalších kapitolách snadno odkazovat.

Hlavním cílem práce je seznámit se s návrhem malých mobilních robotů pro autonomní navigaci a ověřit vhodnost kamery Intel RealSense D435i pro tuto aplikaci.

Práci můžeme rozdělit do následujících bodů:

1. Seznámení se s úlohou současného mapování a lokalizace.
2. Seznámení se s prostředím ROS.
3. Ověření možnosti použití kamery Intel RealSense D435i pro úlohu mapování a lokalizace.
4. Vytvoření simulačního modelu vybraného podvozku robota.

3. SLAM

V rámci požadavku 1 jsme si stanovili cíl seznámit se s úlohou současné lokalizace a mapování. A právě o tom pojednává tato kapitola.

Název současná lokalizace a mapování[1] vznikl překladem z anglického simultaneous localization and mapping, zkráceně SLAM. **Lokalizací** se rozumí určení aktuální pozice robota v prostoru, respektive určení souřadnic na mapě prostředí. Cílem **mapování** je pak nasnímat okolí robota a vytvořit model prostředí (mapu). Někdy se jako součást SLAMu bere i **plánování trasy** robota [4]. To je potřeba, pokud požadujeme, aby robot autonomně prozkoumal a sestavil mapu celého prostoru. V takovém případě si musí robot sám plánovat trasu k úsekům terénu, které ještě nejsou zmapované.

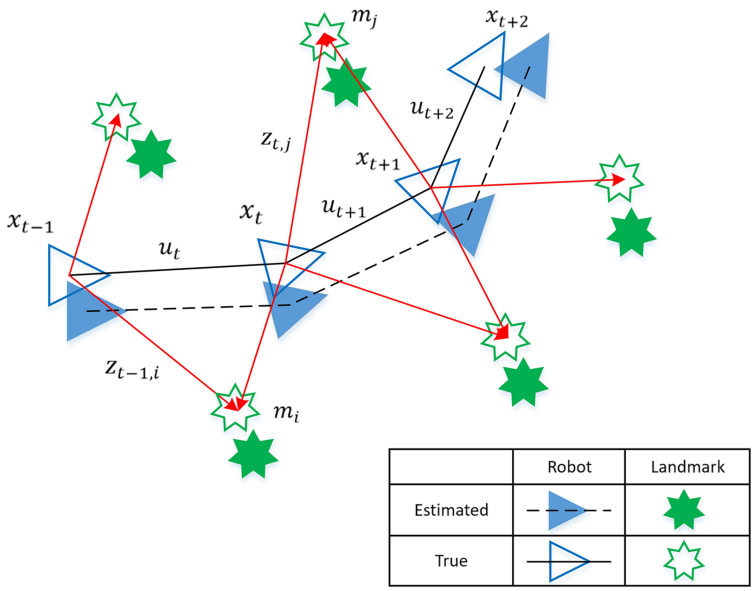
Obtížnost průběžné lokalizace a mapování spočívá v tom, že pro správnou lokalizaci robota je potřeba mít mapu prostředí. Zároveň je ale nezbytné znát přesnou polohu robota pro sestavení mapy. Mapu totiž sestavujeme na základě měření senzorů umístěných na robotovi. Často se to přirovnává k problému, zda byla dříve slepice nebo vejce [5].

3.1 Formální popis

Na obrázku 3.1 je znázorněna trajektorie pohybu robota neznámým prostředím v časech $\{t - 1, t, t + 1, t + 2\}$ (modré trojúhelníky). Zelené hvězdy reprezentují význačné orientační body v prostoru. Červené šipky pak představují pozorování těchto orientačních bodů pomocí senzorů umístěných na robotovi. Trojúhelníky a hvězdy bez výplně reprezentují skutečné pozice, kdežto vyplněné tvary představují domnělé pozice robota a bodů v prostoru, které robot odhadl při mapování prostoru.

Orientační body (v angličtině označované jako landmarks) jsou místa s charakteristickými rysy, která je robot schopen detekovat. Při snímání kamerou lze například v obraze hledat jednoduché geometrické útvary jako jsou čáry, kruhy, mnohoúhelníky apod. V reálném prostředí může jít například o hrany nábytku, rohy místností nebo okna. Při následujícím popisu SLAMu předpokládáme neměnnou polohu těchto bodů během mapování a lokalizace [7].

[1] Někdy se používá název průběžná lokalizace a mapování.



Obrázek 3.1: Znázornění pohybu robota v neznámém prostředí. Převzato [6].

Nejdříve zavedeme následující veličiny, které budeme používat při popisu.

- x_k: Vektor, jehož prvky jsou souřadnice polohy a natočení robota v prostoru.
- u_k: Vektor řídících instrukcí pro čas k. To jsou instrukce, kterými robot ovládá své pohony mezi časem $k-1$ a k, aby z pozice x_{k-1} přejel do pozice x_k. Místo řídících instrukcí lze v tomto vektoru použít hodnoty naměřené pomocí enkodérů v kolech. Tím eliminujeme část šumu vzniklou rozdílem mezi odeslanými instrukcemi a reálným pohybem kol.
- m_i: Vektor polohy a natočení i-tého orientačního bodu na mapě.
- z_{ik}: Vektor s naměřenou polohou i-tého orientačního bodu v čase k pomocí senzorů umístěných na robotovi.

Množinu všech pozic robota od času 0 do času k označíme $X_{0:k} = \{x_0, x_1, \ldots, x_k\} = \{X_{0:k-1}, x_k\}$. Obdobně označíme množinu všech ovládacích instrukcí $U_{0:k}$, polohu všech význačných bodů m a množinu pozorování všech význačných pozic od času 0 do času k $Z_{0:k}$.

Před vlastním popisem problému je třeba si uvědomit, že v reálném prostředí jsou měření senzorů zatížena šumem. Kromě toho je šumem zatížen i vlastní pohyb robota. Robot sice pošle přesné řídící signály, ale motory v podvozku nevykonají předepsaný pohyb s absolutní přesností. Další nejistoty pak můžou

být způsobeny například protáčením kol. Prokluz kol vzniká špatnou přilnavostí kol k povrchu nejen při přímé jízdě, ale především při zatáčení, a to zejména v případě diferenčního podvozku. Naše odhady polohy pak můžeme zpřesňovat díky tomu, že ze senzorů získáváme redundantní data (když změříme v různých časech stejný objekt). Z těchto důvodů nebudeme pracovat s přesnými polohami, ale s pravděpodobnostmi. Například $P(x_k)$ značí distribuční funkci reprezentující rozložení pravděpodobnosti polohy robota v našem prostředí v čase k.

3.1.1 Definice pravděpodobnostního SLAMu

Nyní již můžeme přistoupit k definování samotného problému. Při řešení SLAM nám jde o to, určit polohu a natočení robota a umístění orientačních bodů na mapě z naměřených dat. To můžeme definovat jako výpočet distribuční funkce pravděpodobnosti

$$P(x_k, m|Z_{0:k}, U_{0:k}, x_0) \tag{3.1}$$

Tato distribuční funkce popisuje rozložení pravděpodobnosti polohy a natočení robota a zároveň rozložení pravděpodobnosti polohy a natočení orientačních bodů. To představují členy x_k a m. Členy za svislou čárou znamenají, že tyto pravděpodobnosti určujeme v závislosti na historii všech měření $Z_{0:k}$, historii ovládacích instrukcí $U_{0:k}$ a startovní pozici robota x_0.

Rekurzivní popis

Pro výpočty je výhodnější řešit SLAM problém rekurzivně. K výpočtu distribuční funkce pravděpodobnosti v čase k 3.1 využijeme znalosti distribuční funkce v čase $k-1$

$$P(x_{k-1}, m|Z_{0:k-1}, U_{0:k-1}, x_0). \tag{3.2}$$

Pohybový a pozorovací model

Při výpočtu distribuční funkce pravděpodobnosti polohy se využívá znalosti snímačů, matematického modelu podvozku robota a jeho odezvy na řídící vstupy. Durrant-Whyte a Bailey odvodili v článku [8] rekurzivní výpočet pravděpodobnosti polohy v závislosti na modelu pozorování 3.3 a pohybovém modelu 3.4.

$$P(z_k|x_k, m) \tag{3.3}$$

$$P(x_k|x_{k-1}, u_k) \tag{3.4}$$

3.2 Algoritmy

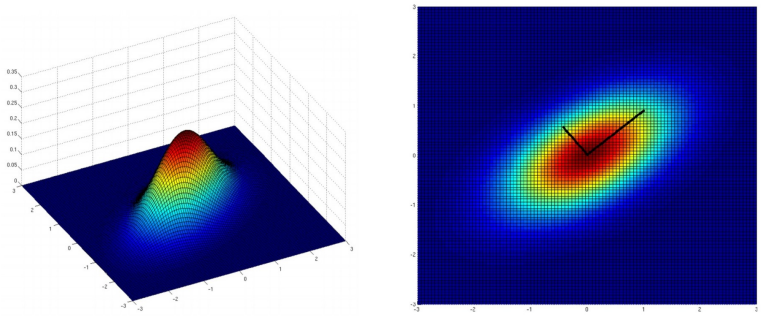
V teoretickém popisu problému jsme pozici robota a detekovaných bodů popisovali libovolným rozdělením pravděpodobnosti. Ovšem v reálné aplikaci nedokážeme přesně určovat potřebná libovolná pravděpodobnostní rozdělení. Pro praktické implementace tak vždy zavádíme různé aproximace. V následujícím textu si popíšeme popíšeme dva nejčastěji používané přístupy. První způsob aproximuje libovolná rozdělení pravděpodobnosti pomocí n-rozměrného Gaussova rozdělení. Tato aproximace vede k použití Kalmanova filtru, respektive rozšířeného Kalmanova filtru. Metoda řešení založená na použití rozšířeného filtru se pak nazývá EKF SLAM[2]. Druhý způsob je založen na diskrétním vzorkováním spojitého rozdělení pravděpodobnosti. Příkladem takového algoritmu je FastSLAM, využívající částicový filtr.

3.2.1 EKF SLAM

Jak už bylo zmíněno, EKF SLAM využívá Kalmanův filtr, který aproximuje rozdělení pravděpodobnosti polohy robota i orientačních bodů Gaussovým rozdělením[3]. Měření fyzikálních veličin má v praxi často právě tento typ rozdělení, proto je vhodné pro aproximaci. Hustota N-rozměrného Gaussova rozdělení má předpis

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \ \exp\left\{-\frac{1}{2}(x-\mu)^T \ \Sigma^{-1}(x-\mu)\right\} \tag{3.5}$$

a je plně charakterizováno kovarianční maticí Σ a vektorem středních hodnot μ. Kalmanův filtr hledá při aproximaci pravděpodobností právě hodnoty těchto parametrů. Příklad Gaussova rozdělení v rovině můžeme vidět na obrázku 3.2.



Obrázek 3.2: 2D Gaussovo rozdělení. Převzato [5].

[2]Název vznikl jako zkratka z anglického Extended Kalman Filter.
[3]Používá se také pojem normální rozdělení.

Rozšířený Kalmanův filtr

Kalmanův filtr můžeme použít pouze pro lineární systémy. Reálné funkce přechodu mezi stavy a závislosti měření ale mívají většinou nelineární charakteristiku. Pak je potřeba použít rozšířený Kalmanův filtr, který se liší od základního Kalmanova filtru tím, že tyto funkce linearizuje pomocí Taylorova rozvoje prvního řádu [7].

Vzájemná korelace orientačních bodů

Durrant-Whyte a Bailey [8] popisují, že při SLAMu vznikají největší chyby v konstruované mapě kvůli špatně určené pozici robota. Chyby způsobené nepřesností měření senzorů jsou pak řádově menší. To ovšem znamená, že chyby v odhadu pozic orientačních bodů jsou vysoce korelované. Jinými slovy vzájemná poloha dvou orientačních bodů $(m_i - m_j)$ je pravděpodobně určená s mnohem větší přesností než absolutní poloha orientačních bodů na mapě. Této charakteristiky můžeme využít pro zpřesňování mapy při opakovaném měření už jednou detekovaných orientačních bodů.

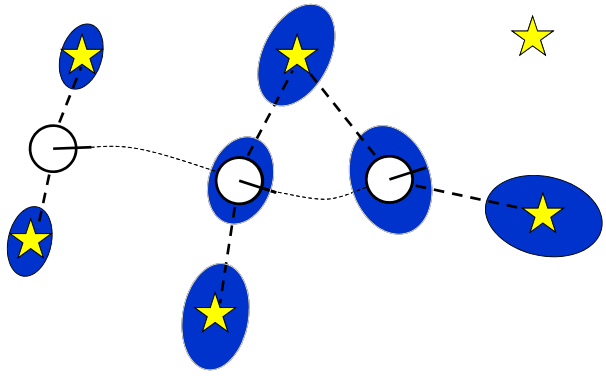
Pro popis konkrétního příkladu se vrátíme k obrázku 3.1. Robot na pozici x_k změří polohu orientačních bodů m_i a m_j. V následujícím kroku x_{k+1} znovu detekuje orientační bod m_j. Toto nové měření umožní robotovi zpřesnit odhad svojí polohy i polohy změřeného orientačního bodu. Díky tomu, že známe relativně přesně vzájemnou polohu bodů m_i a m_j, můžeme navíc propagovat zpřesnění polohy m_j a upřesnit i polohu bodu m_i, přestože ho v aktuálním kroku robot nevidí. Pokud máme už více detekovaných orientačních bodů, jejichž poloha je korelovaná s m_i nebo m_j, můžeme pokračovat a rekurzivně zpřesnit také polohu těchto bodů.

Korelace odhadu vzájemné polohy orientačních bodů s výhodou využívá EKF SLAM. I matice používané při výpočtu jsou konstruované s ohledem na tuto znalost. Prvky hlavní kovarianční matice mají význam neurčitosti polohy robota a orientačních bodů vůči mapě. Kromě toho matice obsahuje i prvky reprezentující kovarianci (lineární závislost) mezi vzájemnou polohou robota a jednotlivými orientačními body, stejně jako mezi orientačními body navzájem. Rozměry této matice jsou pak $(3+2N) \times (3+2N)$, kde N je počet nalezených orientačních bodů [9].

Tím, že při výpočtu EKF SLAM využíváme znalosti korelace vzájemné polohy orientačních bodů, zpřesňujeme s každým měřením polohu všech orientačních bodů na mapě. To je zjevnou výhodou algoritmu. Nevýhodou naopak je, že kvůli

tomu musíme přepočítat celou kovarianční matici po každém pozorování byť i jediného orientačního bodu. Z toho vyplývá vysoká časová složitost algoritmu $\mathcal{O}(N^2)$, kde N je počet orientačních bodů.

Použití EKF SLAM pro odhad polohy robota a orientačních bodů podle Gaussova rozdělení můžeme vidět na obrázku 3.3.



Obrázek 3.3: Odhad polohy robota a orientačních bodů podle Gaussova rozdělení. Převzato [10].

3.2.2 FastSLAM

FastSLAM aproximuje spojité rozložení pravděpodobnosti pomocí konečného počtu částic. Každá částice pak vlastně představuje hypotézu, že na jejím místě se nachází robot. Algoritmus FastSLAM po každém měření senzorů částice ohodnotí podle toho, jak dobře odpovídají naměřeným hodnotám. Částice, které vyjdou jako velmi nepravděpodobné, jsou nahrazeny novými. K tomu se využívá částicový filtr.

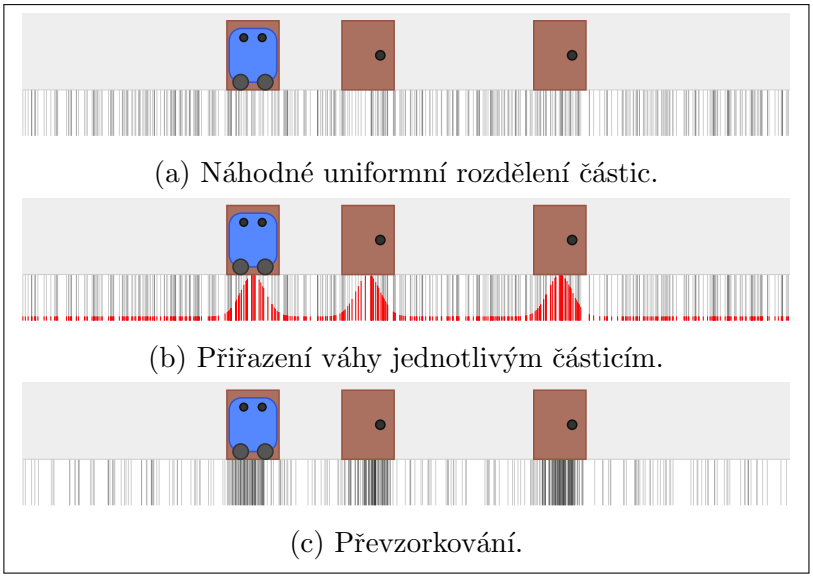
Příklad lokalizace

Princip fungování částicového filtrování si ukážeme na příkladu lokalizace robota v jednoduchém jednodimenzionálním prostředí. Představme si, že máme robota, jehož senzor dokáže detekovat pouze to, zda robot stojí přede dveřmi nebo ne. Takového robota umístíme do známé kruhové chodby s několika dveřmi a budeme sledovat proces jeho lokalizace pomocí částicového filtru. (Protože máme k dispozici mapu chodby, jde v tomto příkladu pouze o lokalizaci, nikoliv o plný SLAM.)

t = 0

Na začátku inicializujeme naši jednodimenzionální mapu částicemi podle rovnoměrného rozdělení pravděpodobnosti, což odpovídá tomu, že zatím nevíme nic o poloze našeho robota a všechny pozice na mapě jsou tak stejně pravděpodobné. Mohli bychom inicializovat částice pravidelně se stejnými rozestupy, ale v praxi se častěji pozice částic generují pomocí generátorů pseudonáhodných čísel. Tento typ algoritmů se nazývá Monte Carlo metody. Inicializační krok s použitím náhodných pozic můžeme vidět na obrázku 3.4a.

V dalším kroku již využijeme měření senzorů. Z obrázků 3.4 vidíme, že v čase $t = 0$ senzor robota s největší pravděpodobností detekoval dveře. Na základě porovnání naměřené hodnoty a skutečné mapy přiřadíme částicím váhy. Pro každou částici vlastně určujeme, jak moc je pravděpodobné, že by robot naměřil hodnoty, které naměřil, pokud by stál na její pozici. Jinými slovy váha je pravděpodobnost, s jakou se robot nachází na pozici dané konkrétní částicí. Na obrázku 3.4b vidíme červeně vyznačené váhy pro částice z předchozího kroku. Čím delší je červená část čáry, tím pravděpodobnější je, že se robot nachází na příslušné pozici. Ze stejného obrázku vidíme, že některá místa jsou už velmi nepravděpodobná a je tedy zbytečné, abychom pro ně používali tolik částic. Výhodnější by bylo zvýšit přesnost lokalizace použitím více částic pro místa s větší pravděpodobností.



(a) Náhodné uniformní rozdělení částic.

(b) Přiřazení váhy jednotlivým částicím.

(c) Převzorkování.

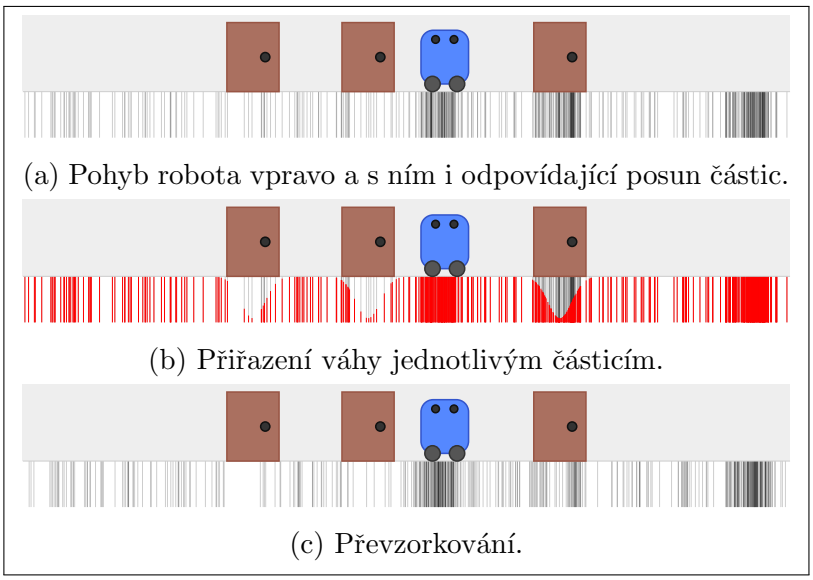
Obrázek 3.4: Kroky v čase t = 0. Převzato [11].

K tomu slouží další krok algoritmu, který se nazývá převzorkování. V něm vybíráme nové částice z původních částic podle váhy. Částice můžeme vybírat

opakovaně. Při správném nastavení algoritmu vybereme částice s velkou vahou opakovaně a naopak některé částice s malou vahou nevybereme vůbec. Stav výpočtu po tomto kroku je zobrazen na obrázku 3.4c.

t = 1

Mezi časem $t = 0$ a časem $t = 1$ přejel robot v našem příkladu doprava. Tato vzdálenost, navíc zatížená šumem, odpovídá ovládací instrukci u_1. Posuneme tedy částice na mapě stejným směrem o dráhu danou instrukcí u_1 a pro každou částici navíc přidáme nějaký šum. Každou částici posuneme tedy trochu jinak. Tento šum se často modeluje Gaussovým rozdělením. Situaci po posunutí doprava můžeme vidět na obrázku 3.5a. Tentokrát robot s největší pravděpodobností změřil, že stojí před zdí. Přiřadíme odpovídající váhy částicím (obrázek 3.5b) a vytvoříme novou generaci částic převzorkováním (obrázek 3.5c).



(a) Pohyb robota vpravo a s ním i odpovídající posun částic.

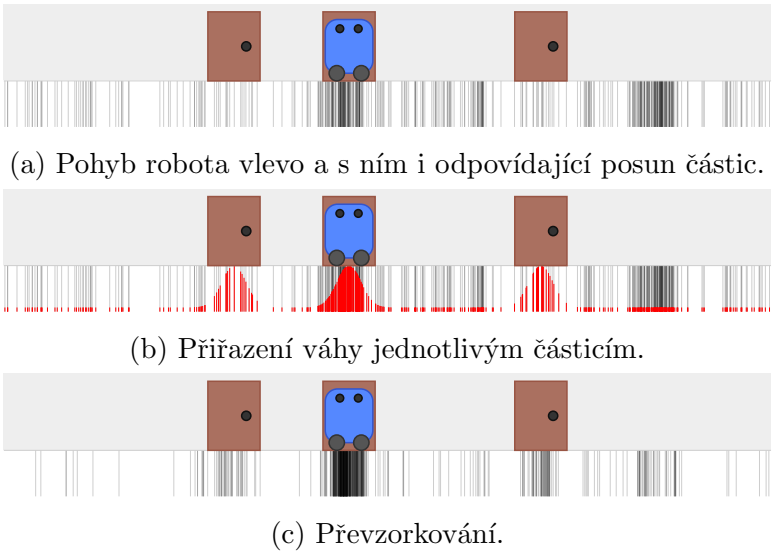
(b) Přiřazení váhy jednotlivým částicím.

(c) Převzorkování.

Obrázek 3.5: Kroky v čase t = 1. Převzato [11].

t = 2

Mezi časy $t = 1$ a $t = 2$ se robot přesunul doleva. Přesun částic, přiřazení váhy a převzorkování pro čas $t = 2$ můžeme vidět na obrázku 3.6. Na posledním obrázku 3.6c je vidět, že algoritmus už začíná konvergovat ke správné pozici a určovat, že robot stojí před druhými dveřmi [11].



(a) Pohyb robota vlevo a s ním i odpovídající posun částic.

(b) Přiřazení váhy jednotlivým částicím.

(c) Převzorkování.

Obrázek 3.6: Kroky v čase t = 2. Převzato [11].

Shrnutí částicového filtru

Kroky částicového filtrování, které jsme pozorovali v předchozím příkladu můžeme shrnout následovně. Před začátkem lokalizace je potřeba inicializovat částice (nejčastěji podle uniformního rozložení pravděpodobnosti). Po každém naměření nových hodnot pomocí senzorů se pak provádějí tyto výpočty [12]:

1. Posunutí částic podle pohybového modelu robota. (Pokud se robot před aktuálním měřením pohyboval.)
2. Přiřazení váhy částicím podle naměřených hodnot ze senzorů.
3. Převzorkování částic podle počtu a váhy částic z aktuální generace.

Popis FastSLAMu

V předchozí části jsme viděli použití částicového filtru pro lokalizaci v příkladu, kde jsme měli k dispozici mapu. Nyní se podíváme, jak by se tento filtr dal využít pro řešení celého SLAM problému. První přímočarou možností by bylo modelovat částicemi pozice robota a zároveň pozice orientačních bodů. Pro výpočty by ovšem bylo potřeba mít exponenciální počet částic vzhledem k počtu orientačních bodů, což není výpočetně zvládnutelné.

Místo toho lze využít předpokladu, že tvorba mapy je závislá hlavně na poloze robota. Pokud bychom znali přesnou trajektorii robota, umíme mapu

sestavit relativně snadno. Můžeme to vyjádřit pomocí distribučních funkcí pravděpodobnosti tak, že základní model rozložení pravděpodobnosti polohy robota a orientačních bodů 3.1 rozdělíme na dvě výpočtově nezávislé části podle pravidla $P(x_1, x_2) = P(x_2|x_1)P(x_1)$ do následujícího vztahu [8]:

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0)P(m|X_{0:k}, Z_{0:k}) \tag{3.6}$$

První člen složené pravděpodobnosti představuje odhad pozice robota. Druhý člen určuje rozložení pravděpodobnosti polohy orientačních bodů za předpokladu znalosti trajektorie robota ($X_{0:k}$). Druhý člen můžeme ještě faktorizovat[4] přes jednotlivé orientační body

$$P(X_{0:k}, m|Z_{0:k}, U_{0:k}, x_0) = P(X_{0:k}|Z_{0:k}, U_{0:k}, x_0) \prod_{i=1}^{N} P(m_i|X_{0:k}, Z_{0:k}). \tag{3.7}$$

Pozici robota z rovnice 3.7 budeme odhadovat pomocí částicového filtru. Pro každou částici pak budeme konstruovat vlastní mapu. Jedna částice vlastně odpovídá hypotéze, kudy přesně se robot pohyboval. Díky tomu, že částic máme relativně hodně, je velká pravděpodobnost, že některé částice budou dostatečně blízko skutečné pozice robota. Při vytváření mapy můžeme tedy předpokládat znalost přesné polohy robota. Vzájemné polohy orientačních bodů proto nejsou korelované, jako tomu bylo u EKF SLAMu. Zde budeme naopak považovat polohu orientačních bodů za navzájem nezávislou.

Váhy budeme částicím přiřazovat na základě toho, jak jejich mapa a jejich aktuální poloha na mapě odpovídá naměřeným hodnotám. Na základě vah poté provedeme převzorkování, při němž vybíráme částice s velkou vahou s větší pravděpodobností než částice s malou vahou. Takto vytvořené nové částice přebírají od původní částice i její mapu. Stává se tedy, že získáme více stejných částic se stejnou mapou. Jejich výpočet se začíná lišit až v dalším kroku, kdy posouváme částice podle pohybu robota. Každé částici totiž přiřadíme trochu jinou odchylku od teoretického pohybu robota.

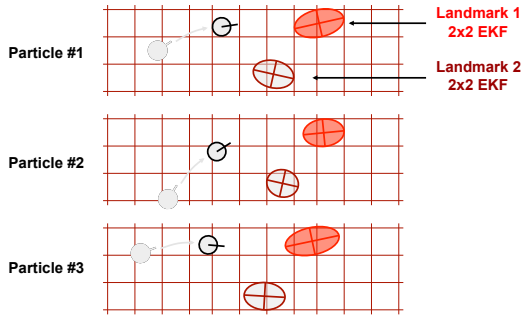
Ve FastSLAMu se pro vytváření mapy používá také rozšířený Kalmanův filtr. Avšak pro každý orientační bod se počítá vlastní nezávislý odhad (nepočítá se kovariance mezi jednotlivými orientačními body jako EKF SLAM). Pro každý orientační bod se tedy používá vlastní rozšířený Kalmanův filtr dimenze 2. Pro K částic a N orientačních bodů je potřeba $K \cdot N$ Kalmanových filtrů. To evokuje, že časová složitost FastSLAMu by měla být $\mathcal{O}(KN)$. S použitím stromové datové

[4]Tato faktorizace se nazývá Rao-Blackwellizace.

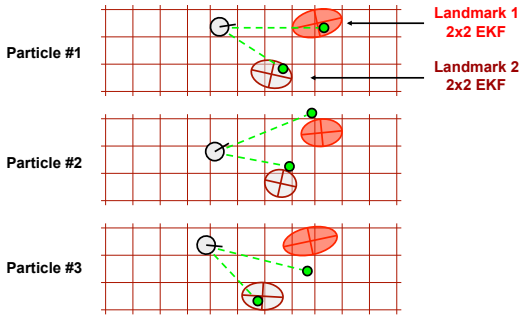
struktury pro uchovávání odhadů poloh orientačních bodů lze však naprogramovat FastSLAM dokonce se složitostí $\mathcal{O}(K \log N)$ [13].

Příklad FastSLAM

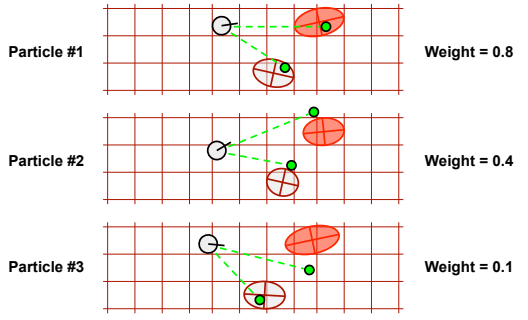
Příklad algoritmu FastSLAM v rovině můžeme vidět na obrázcích 3.7, kde jsou zvlášť zobrazeny mapy pro tři různé částice. Na prvním obrázku 3.7a je zachycen pohyb robota ze tří různých poloh, reprezentovaných různými částicemi. Na dalším obrázku 3.7b je do mapy promítnutá poloha orientačních bodů, kterou robot změřil. Na třetím obrázku 3.7c jsou přiřazeny váhy jednotlivým částicím podle toho, jak moc odpovídá jejich pozice naměřeným hodnotám. Na posledním obrázku 3.7d jsou upravené 2x2 rozšířené Kalmanovy filtry odhadující polohu orientačních bodů.



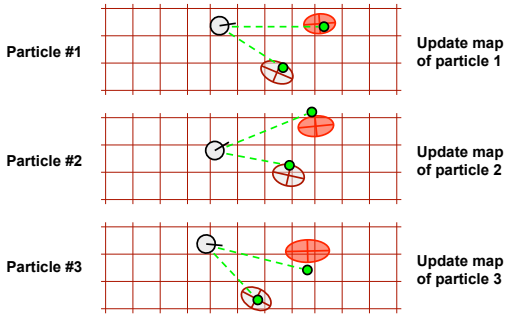
(a) Pohyb robota.



(b) Promítnutí naměřených dat.



(c) Přiřazení váhy částicím.



(d) Úprava EKF na mapách.

Obrázek 3.7: FastSLAM v rovině. Převzato [14].

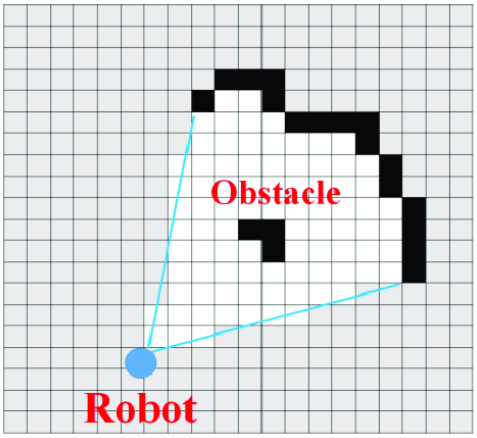
3.2.3 Další algoritmy

V předchozí části jsme popsali dva z nejpoužívanějších algoritmů pro řešení současné lokalizace a mapování. Zmiňme ještě novější a taky často používaný

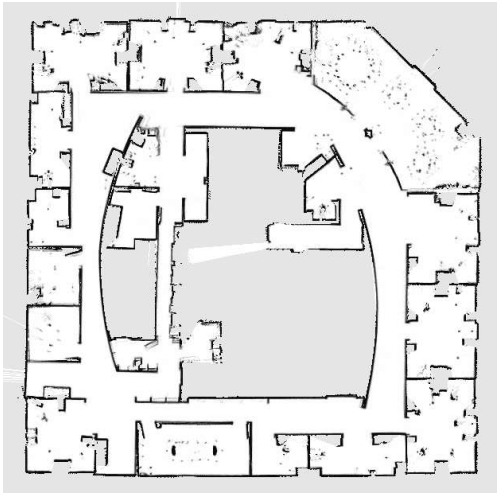
GraphSLAM, který využívá k reprezentaci problému graf [15].

3.2.4 Reprezentace mapy

V předchozí části jsme se omezili pouze na mapování a lokalizaci pomocí orientačních bodů. Jinou alternativou, kterou lze použít s některými algoritmy (například FastSLAM), je využívat všech detekovatelných překážek v prostoru a budovat takzvanou mřížku obsazenosti (occupancy grid). V tomto případě rozdělíme mapu mřížkou na množství bodů a ke každému bodu si uložíme, zda jsme v něm detekovali překážku nebo ne, jako na obrázku 3.8. Případně si do každého políčka mřížky můžeme ukládat pravděpodobnosti toho, zda je na daném místě překážka. Příklad reálně sestavené mapy můžeme vidět na obrázku 3.9.



Obrázek 3.8: Detail mřížky obsazenosti. Převzato [16].



Obrázek 3.9: Mřížka obsazenosti vytvořená mapováním Intel Research Labu. Převzato [17].

Mřížka obsazenosti je vhodná v prostředí s větší hustotou překážek nebo pokud jsou orientační body podobné a hrozila by jejich záměna. Nevýhodou jsou naopak vyšší paměťové i výpočetní nároky [18].

3.3 Vhodné senzory

Aby byl robot schopný současné lokalizace a mapování, je potřeba ho vybavit vhodnými senzory. V této sekci projdeme nejčastěji používané senzory a zmíníme některé jejich výhody a nevýhody. Je třeba poznamenat, že volba vhodného senzoru závisí hlavně na prostředí, ve kterém se bude robot pohybovat. V dnešní

době jsou vyvíjeni roboti s funkcí SLAMu pro vnitřní a venkovní pozemní aplikace, stejně jako pro podvodní a letecké využití. Další proměnnou, kterou je třeba vzít v úvahu při výběru senzoru, jsou světelné podmínky, které jsou na škále od intenzivního slunečního světla až po úplnou tmu například v podzemí [19].

Akustické senzory

Akustické senzory (sonary) jsou jedny z nejlevnějších senzorů, fungují na většině povrchů i materiálů. Nevýhodou je ovšem nízká úroveň detailů a krátký dosah. Své uplatnění nacházejí většinou pro podvodní aplikace.

Laserové senzory

Laserové skenery (lidary) se těší velké popularitě v mobilní robotice. Hlavním důvodem je, že mají velmi dobrou přesnost. Vhodné jsou hlavně pro pozemní aplikace, uvnitř budov i ve venkovních prostorech. Jejich nevýhodou je, že nefungují spolehlivě, pokud mají překážky lesklý, odrazivý povrch nebo pokud jsou z transparentního materiálu.

Kamery

Pro současnou lokalizaci a mapování lze využít také běžné kamery. Princip fungování je takový, že na základě měření z různých pozic získáváme hloubkovou informaci čistě pomocí výpočtů v softwaru. Pro SLAM s použitím kamery jako hlavního senzoru se používá označení visual SLAM a jedná se o oblast s velmi aktivním výzkumem. Při řešení visual SLAM se využívají postupy z počítačového vidění jako je detekce prvků a hran, rozpoznávání obrazu a další. Tato metoda zatím neposkytuje tak přesné hloubkové informace jako při použití jiných zde popsaných senzorů. Ovšem zřejmou výhodou je dostupnost senzorů (kamer), které jsou už navíc často součástí dronů nebo robotů ovládaných člověkem [4].

RGB-D

Poslední skupinou, kterou zde zmíníme, jsou RGB-D senzory. Ty umožňují získávat jak obraz z kamery, tak hloubková data. Mezi takové senzory patří například Microsoft Kinect nebo modely z produktové řady Intel RealSense. Mezi přednosti RGB-D senzorů patří především jejich cenová dostupnost. Více se těmto senzorům věnuje kapitola 5.

Pomocné senzory

Pro zlepšení odhadu polohy robota lze použít v kombinaci s výše popsanými senzory navíc ještě enkodéry v motorech pohonu nebo inerciální měřicí jednotku (IMU)[5]. Při měření polohy pouze pomocí IMU jednotky se i během relativně krátkého času akumulují chyby a vzrůstá nepřesnost v odhadu polohy (drift). Stejně tak se časem akumuluje i chyba dat získaných procesem odometrie z enkodérů.

Naopak výhodou IMU a enkodérů je nenáročnost výpočtů a rychlejší měření hodnot. Lze tak například využít kombinaci lidaru a IMU jednotky, kdy mezi dvěma měřeními lidarem můžeme několikrát upravit polohu pomocí dat z IMU.

[5]Inerciální měřicí jednotka v sobě obsahuje gyroskopy, akcelerometry a někdy i magnetometry. IMU měří hlavně úhlovou rychlost a lineární zrychlení. Abychom však z těchto údajů určili polohu a natočení, musíme naměřená data jednou, respektive dvakrát, integrovat, čímž se zvětší i velmi malá počáteční chyba měření.

4. Robot Operating System

Dalším z cílů naší práce je seznámit se s prostředím ROS (Robot Operating System). Přestože by se to podle názvu mohlo zdát, ROS není operační systém. Jedná se o soubor nástrojů a knihoven používaných při vývoji robotů.

ROS vznikl v roce 2007 na Standfordské univerzitě. Od jeho počátků se kolem něj vytvořila rozsáhlá komunita lidí z akademické sféry, ale i z průmyslu.

Mezi výhody ROSu patří, že je open-source a tudíž je k dostání zdarma. ROS původně oficiálně podporoval pouze linuxovou distribuci Ubuntu. V současné době je ovšem také aktivně vyvíjen ROS 2, který přináší mimo jiné i podporu pro Windows 10 a MacOS X [20]. Avšak ROS 2 není plně kompatibilní s originálním ROSem, a tak nemůže automaticky používat balíčky původního ROSu. Větší podporu tak má zatím stále ROS 1.

4.1 Architektura

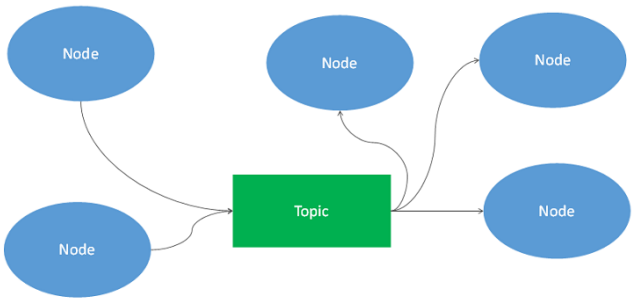
Robot Operating System je distribuovaný systém, který se snaží o to, aby byl co možná nejmodulárnější. Základním stavebním prvkem je uzel (*node*). Uzel si můžeme představit jako proces, který provádí například výpočet nějakého algoritmu nebo ovládá motor. Uzly je možné programovat v různých programovacích jazycích. Nejčastěji používané a zároveň nejlépe podporované jsou C++ a Python.

4.1.1 Topics

Uzly mezi sebou mohou sdílet informace posíláním zpráv. Tyto zprávy se posílají prostřednictvím *topics*, což jsou vlastně pojmenované komunikační kanály. Každý *topic* má přiřazenu jednu datovou strukturu, pomocí které se přes něj komunikuje. Takovou strukturou zde může být například text, číslo nebo i složitější datová struktura definovaná uživatelem. Každý *topic* je pak jednoznačně identifikovatelný svým jménem a typem dat, které posílá.

Uzly se k *topics* registrují jako odesílatelé (*publisher*) a příjemci (*subscriber*). *Publisher nodes* jsou často spojeny se senzory, jako jsou kamery nebo enkodéry. *Listeners* pak tato data nějak dále zpracovávají. K jednomu tématu se může registrovat libovolný počet odesílajících i přijímajících uzlů. *Topic* se dvěma *publisher* a třemi *listener* uzly můžeme vidět na obrázku 4.1.

Tato architektura, kdy o sobě uzly nic nevědí, jenom spolu komunikují přes *to-*



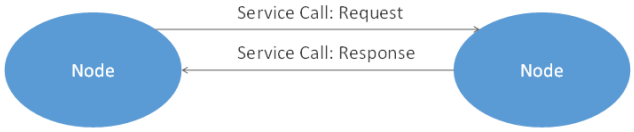
Obrázek 4.1: Diagram zachycující komunikaci uzlů pomocí topics. Převzato [21].

pics, vyniká svojí modularitou. Pokud máme například uzel, který přijímá zprávy z lidaru a implementuje EKF SLAM, můžeme ho nahradit uzlem, který přijímá stejný typ dat a implementuje FAST SLAM, aniž bychom museli měnit cokoliv dalšího v našem projektu.

To, že zprávu odeslanou přes *topic* obdrží správní *listeners*, zajišťuje hlavní uzel (*master*). Ten musí být přítomen v každé běžící instanci ROSu.

4.1.2 Services

Topics se hodí pro většinu komunikací mezi uzly. Avšak někdy je lepší mít možnost řídit, kdy se jaká data posílají. Proto byl vytvořen druhý typ komunikace, takzvané *services*. V tomto případě jsou data odesílána jen tehdy, když si je jiný uzel vyžádá. Uzel, který nabízí tento typ komunikace, se nazývá *service server*, jeho protějšek pak označujeme jako *service client*. Komunikace probíhá tak, že klient pošle serveru žádost, načež od něj dostane odpovědí požadovaná data. Znázorněno je to na obrázku 4.2.



Obrázek 4.2: Diagram komunikace uzlů pomocí services. Převzato [21].

Jednoduchým příkladem použití *services* může být uzel, který převádí imperiální jednotky na metrické. Je dobré poznamenat, že v případě *services* se datový typ zpráv, které posílá klient, může lišit od typu zpráv, které posílá server.

Nevýhodou *services* je, že jsou vykonávány synchronně a jsou blokující. Běh programu se zdánlivě zastaví, dokud nepřijde odpověď od serveru. Proto nemohou být v *service* serverech vykonávány žádné složité a dlouhotrvající výpočty. To řeší následující způsob propojení uzlů.

4.1.3 Actions

Actions fungují podobně jako *services*, uzly jsou také rozděleny na klienta a server. Rozdíl spočívá v tom, že *actions* mohou být vykonávány asynchronně. Při komunikaci pomocí *actions* se používají tři typy zpráv – *goal*, *feedback* a *result*.

Tento typ komunikace by mohl být využit například pro ovládání robota ve skladu. Řídící uzel pošle robotovi *goal* zprávu, ve které specifikuje místo, kam má robot dojet. Robot si vypočítá dráhu a začne se pohybovat směrem k zadanému cíli. Průběžně posílá *feedback* zprávy, obsahující zbývající vzdálenost k cíli. Když robot dojede na místo určení, pošle *result* zprávu o úspěšném dokončení požadavku. Tento typ komunikace je znázorněn na obrázku 4.3.

Obrázek 4.3: Diagram komunikace dvou uzlů pomocí actions. Převzato [21].

Více informací o ROS *nodes* a komunikaci mezi nimi lze najít v online kurzu Hello (Real) World with ROS z Technické univerzity Delft [1].

4.2 Balíčky

Software používaný v ROSu je organizovaný do balíčků. Balíček může obsahovat uzly, knihovny, nástroje, atd. Balíček je vlastně malý nezávislý program. Využívání systému balíčků přináší tu výhodu, že software, který se zabalí do balíčku, je lehce přenositelný do jiného projektu. Díky velké komunitě existuje přes 3000 volně dostupných balíčků.

Jedním z balíčků je Rviz (ROS Visualization), nástroj užitečný pro zobrazování 2D i 3D dat.

4.3 Gazebo

Kromě řízení reálných robotů lze v ROSu tyto roboty také simulovat. Nejčastěji se k tomu využívá program Gazebo. Ten je vyvíjený stejnou společností jako ROS, a tak je jejich propojení na vyšší úrovni než propojení s jinými simulačními programy.

Gazebo umožňuje vytvořit venkovní nebo vnitřní prostředí, které pak simuluje pomocí vestavěného fyzikální enginu. Kromě toho umožňuje simulovat také data ze senzorů a to včetně generování šumu.

Pro vytváření modelů robotů do Gazeba se používá jazyk URDF (Unified Robot Description Format), který je postavený na XML. Roboti se v URDF definují pomocí prvků *links* a *joints*. *Links* reprezentují jednotlivé díly robotů, jako například rám, kolo nebo segment robotického ramene. *Links* můžou mít podobu základních tvarů jako je kvádr, válec nebo koule. Druhou možností je naimportovat přesné 3D modely vytvořené v CAD programu.

Links prvky se pak spojují dohromady pomocí *joints*, které umožňují definovat typ vazby. Na výběr je mezi pevnou a několika druhy pohyblivých vazeb. Příkladem pevné může být spojení dvou sešroubovaných částí šasi robota, příkladem pohyblivé pak rotační vazba mezi rámem a kolem. Ukázka simulace robota v Gazebu je na obrázku 4.4.

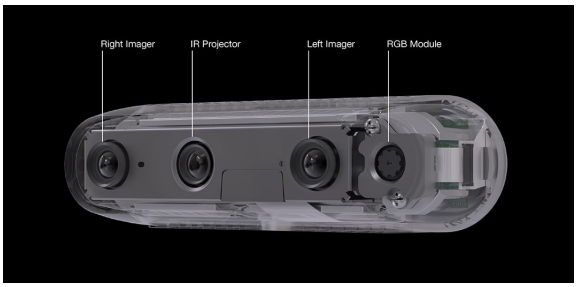
Obrázek 4.4: Screenshot Gazeba s robotem Husky. Převzato [22].

5. Kamera Intel RealSense D435i

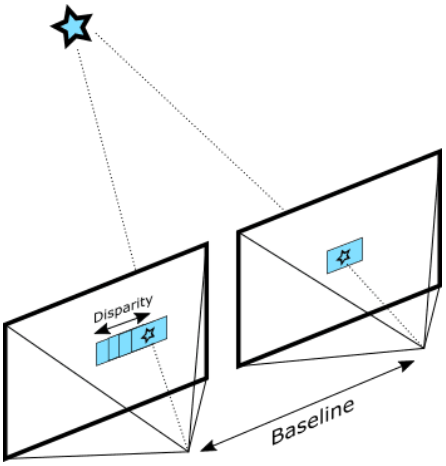
Jak už bylo zmíněno v úvodu, v naší práci používáme kameru Intel RealSense D435i. Tato kapitola popisuje princip fungování kamery, její propojení s ROSem a posouzení vhodnosti pro SLAM.

5.1 Princip fungování

Intel RealSense D435i v sobě obsahuje dva stejné snímače, které dokáží pořizovat fotky ve vlnových délkách na pomezí mezi viditelným a infračerveným spektrem [23]. Jak můžeme vidět na obrázku 5.1, objektivy těchto snímačů jsou umístěny na opačných stranách přední části kamery. Díky tomu snímá každá kamera trochu jiný obraz, což ilustruje nákres 5.2. Objekt na obrázku z levé kamery je posunutý lehce vpravo oproti stejnému objektu na obrázku z pravé kamery. Snímané objekty, které jsou blíž ke kameře jsou vzájemně posunuty více než objekty, které jsou dál od kamery. Díky tomu, že vzdálenost obou objektivů je známá, tak lze z vzájemného posunutí na snímcích dopočítat, jak daleko je konkrétní objekt. Na stejném principu funguje i lidský zrak, kde nám oči umístěné vedle sebe umožňují vnímat hloubku prostoru. Příklad snímku s vypočítanými hloubkovými daty je na obrázku 5.3c.



Obrázek 5.1: Kamera Intel RealSense D435i. Převzato [2].



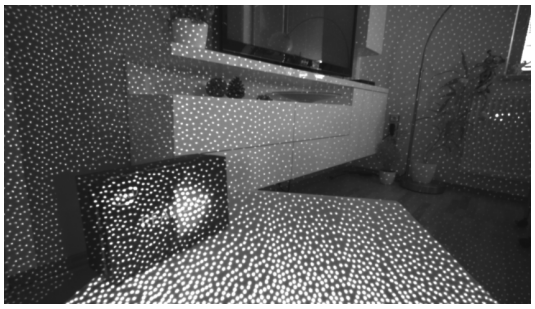
Obrázek 5.2: Stereoskopické vidění. Převzato [2].

K tomu aby kamera rozpoznala, kde jsou na levém a pravém obraze stejné objekty, hledá význačné body, které se shodují v obou snímcích. Takovými body

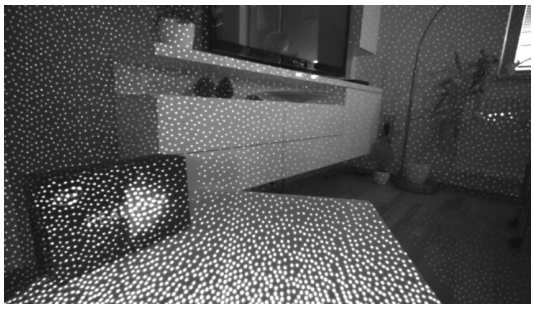
můžou být například výrazné přechody mezi tmavými a světlými pixely nebo základní geometrické tvary a přímky.

Popsaný princip rozpoznávání odpovídajících míst na snímcích funguje nejlépe, pokud mají snímané objekty výraznou texturu. Pokud je naopak součástí scény předmět s jednolitým povrchem, například jednobarevná zeď, tak by kamera tímto způsobem dokázala vypočítat pouze vzdálenost k jeho hranám. Vzdálenost doprostřed plochy by už určit nedokázala.[1]

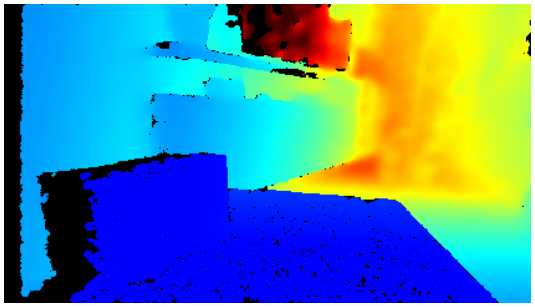
Aby nebyla kamera odkázaná pouze na textury snímaných předmětů, tak ještě promítá do prostoru před sebe určitý vzor. Konkrétní vzor z teček, který promítá RealSense D435i, můžeme vidět na snímcích z infračervených kamer 5.3a a 5.3b. Vestavěný projektor používá světlo s vlnovou délkou kolem 850nm, které je pro lidské oko neviditelné, protože se nachází na pomezí mezi viditelným a infračerveným světlem [2]. Další výhodou popsaného promítání je, že částečně osvětlí scénu, čímž pomůže kameře při určování vzdálenosti za zhoršených světelných podmínek.



(a) Snímek z levé infračervené kamery



(b) Snímek z pravé infračervené kamery



(c) Hloubkový snímek vypočítaný z infračervených snímků

(d) Snímek scény ve viditelném spektru zachycený RGB kamerou

Obrázek 5.3: Scéna zachycená kamerou Intel RealSense D435i

[1]Vzdálenost k bodům uvnitř nasnímané plochy nelze aproximovat ze vzdáleností okrajů. Obecně nemusí být nasnímaná plocha rovná jako v případě zdi, snímaný objekt může být například koule nebo sloup.

Kromě výše popsaných senzorů a projektoru obsahuje naše kamera ještě jednu klasickou RGB kameru pro snímání viditelného spektra, snímek z ní je na obrázku 5.3d. Obrazová data z této kamery mohou být namapovaná na získané hloubkové pixely. Tím se vytvoří struktura, která se nazývá colored point cloud, kde má každý pixel nejen souřadnice v prostoru, ale i barvu.

5.1.1 IMU

Součástí kamery RealSense D435i je kromě výše zmíněných kamer a projektoru ještě IMU jednotka. Ta pomocí akcelerometrů měří zrychlení a pomocí gyroskopů úhlovou rychlost celého zařízení.

5.1.2 Vlastnosti

Podle výrobce dokáže kamera, kterou máme k dispozici, měřit vzdálenost od 10 cm do 10 m. Ovšem je třeba počítat s tím, že pro předměty vzdálenější než 4 metry začíná znatelně narůstat chyba měření [2].

Výhodou, kterou naší kameře přináší popsaný princip fungování, je, že kamera funguje jak za dobrého světla tak i za mírně zhoršených světelných podmínek. Naopak nevýhodou je, že špatně detekuje průhledné a odrazivé materiály. Tento jev můžeme vidět v horní části obrázků 5.3, kde vypnutá televize odráží okolní světlo, a proto je její vzdálenost určena špatně.

5.2 Spojení s ROSem

Intel poskytuje balíček s názvem *realsense2_camera*, který umožňuje používání RealSense kamer v prostředí ROSu. Před instalací samotného balíčku je nejdříve potřeba na počítač s ROSem nainstalovat knihovnu *Intel RealSense SDK 2.0*. Dalším krokem pak je stáhnout kód balíčku *realsense2_camera* z GitHubu [24] a zkompilovat ho. Podrobný návod celé instalace lze nalézt na zmíněné GitHub stránce balíčku.

5.3 Ověření vhodnosti pro SLAM

Vhodnost kamery Intel RealSense D435i pro SLAM vyzkoušíme experimentem, při kterém budeme mapovat reálnou místnost. Výstupem tohoto mapování

by měla být mřížka obsazenosti, kterou bychom mohli dále použít pro navigaci robota. K implementaci zkušebního SLAMu použijeme následující balíčky:

- *realsense2_camera* – výše popsaný balíček s podporou RealSense kamer pro ROS. Uzel z tohoto balíčku čte data z připojené kamery a publikuje je pro ostatní uzly ve formě *topics*.

- *rtabmap_ros* – implementace SLAMu, která využívá reprezentaci problému grafem. Tento balíček se stará o lokalizaci kamery a budování mapy. Pokaždé když dostane *rtabmap* uzel *topic* s novými obrazovými a hloubkovými daty z kamery, tak je porovná se stávající mapou. Pokud dokáže určit, na které místo ve 3D mapě nová data patří, připojí je a upraví podle nich odhad pozice kamery.

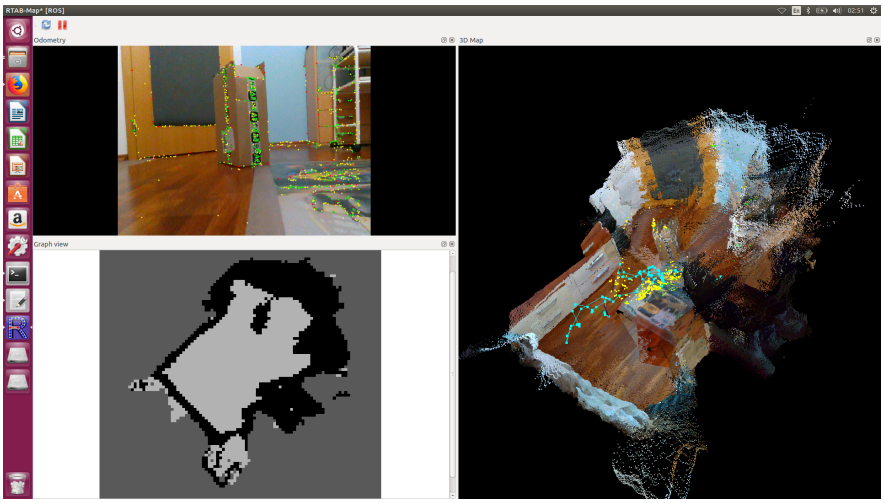
  Rtabmap_ros umožňuje také převádět 3D mapu na 2D mřížku obsazenosti. V této mřížce světlé pixely představují vodorovné plochy, po kterých se může robot pohybovat. Zatímco černé pixely reprezentují místa vybíhající nahoru, která jsou pro robota překážkou.

- *imu_filter_madgwick* – výpočet natočení přístroje z hodnot úhlové rychlosti a lineárního zrychlení, které měří IMU jednotka.

- *robot_localization* – polohová data získaná několika způsoby porovnává a spojuje pro získání přesnější polohy. Z tohoto balíčku využijeme uzel *ukf_localization_node*, kterému budeme posílat odhad polohy a natočení ze SLAM algoritmu a odhad polohy z IMU dat upravených *imu_filter_node* uzlem. Kombinací obou odhadů pak *ukf_localization_node* zpřesní odhad polohy a natočení kamery.

S těmito balíčky pak můžeme spustit ještě *rviz* nebo *rtabmapviz*. Oba fungují obdobně a oba slouží k vizualizaci naměřených dat a vytvářené mapy. My jsme se rozhodli využít *rtabmapviz*, protože běžel při experimentu plynuleji než *rviz*. Výše popsaný postup, který používáme k současné lokalizaci a mapování, je převzat od vývojářů Intelu [24].

Vlastní experiment pak probíhá tak, že s kamerou ručně otáčíme a posouváme, čímž simulujeme pohyb robota. Díky tomu se posouvá plocha, kterou kamera snímá při mapování. Na obrázku 5.4 můžeme vidět screenshot programu *rtabmapviz* z průběhu mapování. V levém horním okně je aktuální obraz z kamery s vyznačenými orientačními body. V levém spodním okně je pak již téměř hotová

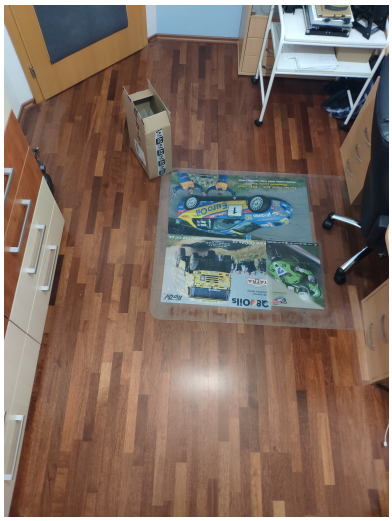


Obrázek 5.4: Screenshot RtabMapViz.

mřížka obsazenosti a v pravém okně je 3D mapa místnosti s vyznačenou drahou, kudy se kamera při mapování pohybovala.

Výslednou mřížku obsazenosti s vyznačenou aktuální polohou kamery můžeme vidět na obrázku 5.5. Volný prostor je zobrazen světlejší barvou zatímco překážky jsou černou. Skutečná mapovaná místnost je zachycena na vedlejším obrázku 5.6. V místnosti jsou po obvodu překážky, které vystupují různě daleko do volného prostoru. Uprostřed je pak kartonová krabice simulující překážku v prostoru. Z porovnání obou obrázků vidíme, že mapování proběhlo úspěšně. Volný prostor i překážky jsou zaznamenány správně a s poměrně velkou přesností.

Obrázek 5.5: Mřížka obsazenosti sestavená mapováním místnosti



Obrázek 5.6: Fotka mapovaného prostoru

6. Model robota

V této kapitole se věnujeme simulaci malého mobilního robota pro mapování a lokalizaci. Nejdříve popíšeme několik typů podvozku, z nichž vybereme jeden vhodný. Pro něj pak vytvoříme model v Gazebu, ke kterému připojíme virtuální RealSense kameru. Na závěr pak simulací ověříme, zda je vybraná kombinace podvozku a kamery vhodná pro naši úlohu.

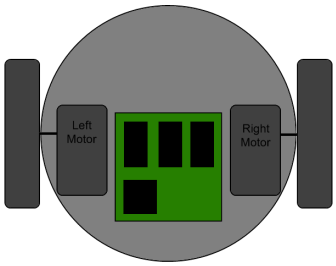
6.1 Výběr podvozku

V mobilní robotice se používá množství typově rozdílných podvozků. My projdeme tři základní, ze kterých vybereme ten nejvhodnější pro naši aplikaci.

6.1.1 Diferenciální podvozek

Prvním typem podvozku je diferenciální. Roboti s tímto typem podvozku se dokáží otáčet i na místě. To je dáno způsobem zatáčení, kdy se kola na jedné straně otáčí opačným směrem než kola na straně druhé.

Nejčastěji je diferenciální podvozek konstruován jako čtyřkolový nebo dvoukolový. Schéma diferenciální podvozku se dvěma koly je na obrázku 6.1. Takové podvozky mívají často ještě třetí pomocné všesměrové kolo, které není nijak poháněno. Nevýhodou dvoukolových podvozků je, že nejsou vhodné do většího terénu. Naopak podvozky se čtyřmi koly jsou použitelné i v terénu, příklad takového robota je na obrázku 6.2.



Obrázek 6.1: Dvoukolový diferenciální podvozek. Převzato [25].

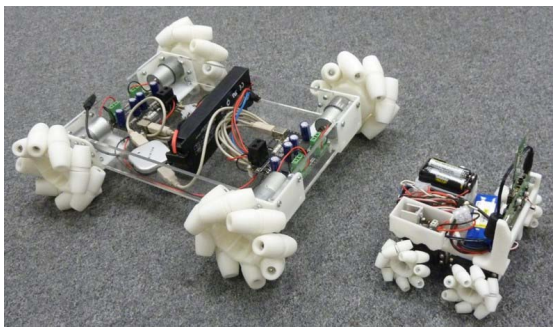
Obrázek 6.2: Husky robot se čtyřkolovým diferenciálním podvozkem. Převzato [26].

Diferenciální podvozky obecně se vyznačují jednoduchou konstrukcí a snadným řízením. Naopak mezi nevýhody může být bráno, že potřebují zvláštní motor pro každé kolo. Respektive u čtyřkolové verze lze použít dva motory, každý pro pohon kol na jedné straně podvozku.

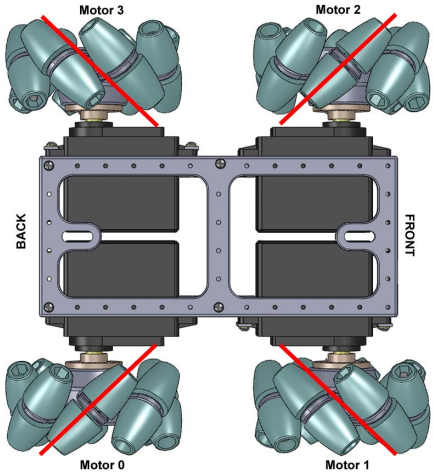
6.1.2 Všesměrový podvozek

Dalším používaným typem pohonu jsou všesměrové podvozky. Příklad robota s tímto podvozkem vidíme na obrázku 6.3.

Kromě toho, že se všesměrové podvozky dokáží otáčet na místě, tak navíc dokáží jet libovolným směrem, například přímo do boku. To je zajištěno použitím speciálních kol. Jedním takovým druhem jsou kola s válečky po obvodu, které jsou natočeny o 45 stupňů. Tato kola jsou pak namontována navzájem zrcadlově, jak můžeme vidět na obrázku 6.4. Pokud se zadní kola v této konfiguraci začnou otáčet opačným směrem než přední, dochází k protáčení válečků a výsledkem je pohyb robota do strany.



Obrázek 6.3: Příklad všesměrového podvozku. Převzato [27].

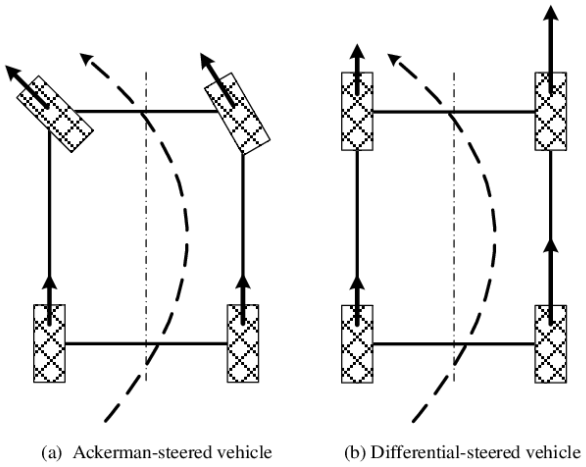


Obrázek 6.4: Diagram všesměrového podvozku. Převzato [27].

Výhoda všesměrových podvozků spočívá v tom, že se dokáží velmi efektivně pohybovat i v omezeném prostoru. Naopak nevýhodou tohoto podvozku je složitější konstrukce kol, nevhodnost do terénu a potřeba zvláštního motoru pro pohon každého kola.

6.1.3 Ackermannův podvozek

Poslední námi probíraný typ je podvozek automobilového typu neboli Akcermannův podvozek. Tento typ má na rozdíl od předchozích podvozků na přední nápravě natáčecí kola, pomocí kterých zatáčí. U Ackermannova podvozku se všechna kola točí vždy stejným směrem a jakékoliv protáčení nebo smýkání je na rozdíl od předchozích typů nežádoucí. Porovnání principu zatáčení Ackermannova a diferenciálního podvozku je na obrázku 6.5.



Obrázek 6.5: Porovnání Ackermannova a diferenciálního podvozku. Převzato [28].

Jednou z výhod Ackermannova podvozku pro roboty je, že na řízení stačí jen dva motory – jeden na zatáčení kol a druhý na pohon. Hlavní nevýhodou pak je, že se roboti s tímto podvozkem nedokáží otočit na místě, a tak potřebují více prostoru pro manévrování.

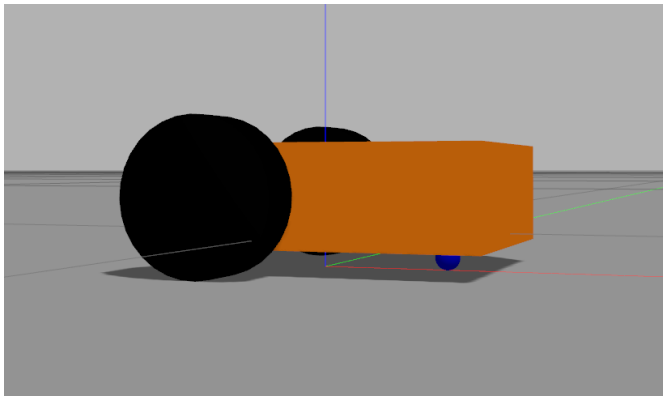
6.1.4 Výběr podvozku

Z výše představených podvozků zvolíme pro naší simulaci diferenciální typ, konkrétně jeho dvoukolovou verzi. Hlavním důvodem je, že se jedné o podvozek s nejjednodušší konstrukcí a ovládáním. Navíc tento typ poskytuje možnost otočení na místě, v budoucnu by tedy pro něj bylo jednodušší vytvořit plánovač trasy.

6.2 Model podvozku pro Gazebo

Jak už bylo zmíněno dříve, roboti pro Gazebo se vytvářejí pomocí jazyka URDF. V tomto jazyce se definují součásti robota (*links*), které se spojují dohromady pomocí *joints*. Každé součásti můžeme navíc přiřadit různé vlastnosti, z nichž základní jsou barva, hmotnost, momenty setrvačnosti vůči základním osám a drsnost povrchu. Takto definované fyzikální vlastnosti poté ovlivňují chování robota při pohybu a interakci s okolím.

Náš robot se skládá ze čtyř základních částí. Nejdříve vytvoříme kvádr reprezentující tělo robota. K němu připojíme rotačními vazbami dva válce představující poháněná kola. Nepoháněné všesměrové kolo simulujeme kuličkou, které nastavíme nulové tření, a opět ji připojíme k tělu robota pomocí rotační vazby. Vytvořeného robota můžeme vidět na obrázku 6.6.



Obrázek 6.6: Základní model robota v Gazebu

Abychom mohli našeho robota ovládat, je třeba k němu připojit *differential_drive_controller* plugin, jemuž přiřadíme odkazy na kola a tělo robota. Na základě ovládacích vstupů pak tento plugin definuje úhlové rychlosti jednotlivých kol. Pro získání ovládacích vstupů využijeme balíček *teleop_twist_keyboard*. Ten nám v součinnosti s *differential_drive_controllerem* umožní ovládat robota pomocí klávesnice.

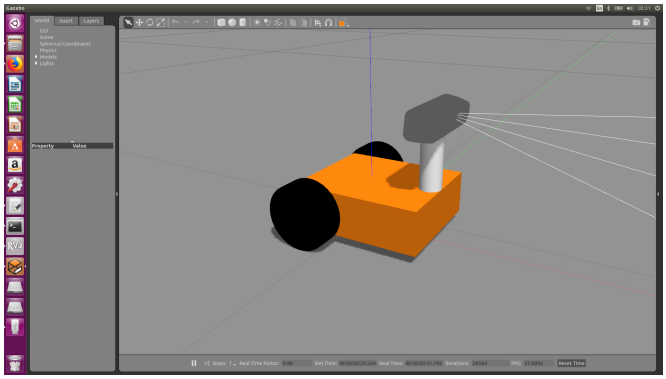
Popsaný robot byl vytvořen na základě e-knihy Autonomous Navigation of ROS Robot od Kiran Pally [29].

6.3 Simulace RealSense kamery v Gazebu

Předtím než na našeho robota přidáme kameru, tak ještě vytvoříme držák. Ten zajistí, že kamera bude ve vyvýšené pozici a bude tak mít lepší rozhled. Model držáku vytvoříme jako *link* ve tvaru válce, který připojíme k tělu robota pevnou vazbou.

Pro simulaci kamery použijeme balíček *realsense_gazebo_plugin* [30]. Ten implementuje pouze kameru Intel RealSense D435 bez IMU jednotky. To ovšem pro naše použití nevadí. Pokud zvládne simulovaný robot SLAM v této konfiguraci, tak přidáním IMU by se situace buď zlepšila, anebo bychom data z ní nepoužívali.

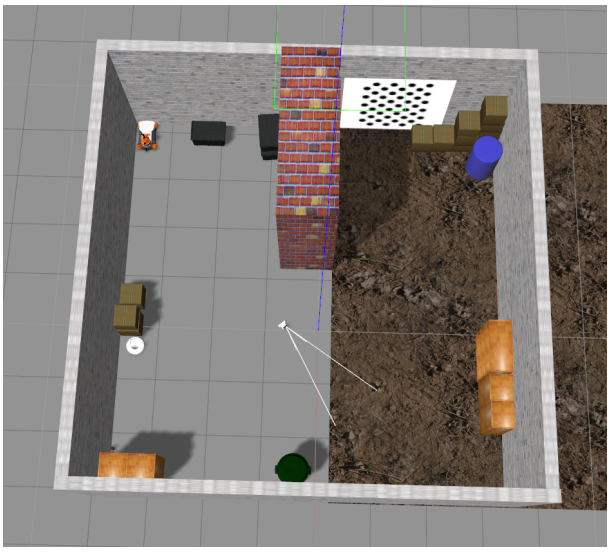
Vlastní model kamery připojíme pomocí další pevné vazby na vršek připraveného držáku. Model robota včetně kamery je na obrázku 6.7.



Obrázek 6.7: Model robota s kamerou v Gazebu

6.4 Svět v Gazebu

Gazebo umožňuje vytvářet nová prostředí, do kterých si můžeme umisťovat vlastní 3D modely. My pro naší simulaci vytvoříme čtvercovou místnost s jednou zdí vystupující do poloviny volného prostoru. Tato dělící zeď zajistí, že robot bude muset v rámci mapování projet dvě oddělené části. Při vytváření modelové místnosti je důležité, aby obsahovala dostatek orientačních bodů podle, kterých se může provádět SLAM. Proto jsme přidali několik objektů podél zdí. Naše modelová místnost je na obrázku 6.8.



Obrázek 6.8: Modelová místnost vytvořená pro simulaci v Gazebu

6.5 SLAM v simulovaném prostředí

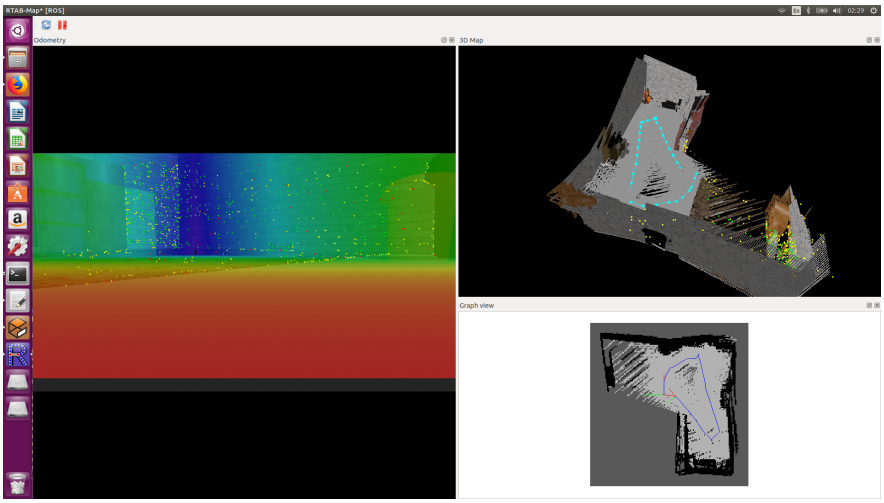
Nyní nám zbývá pouze vytvořit spouštěcí skript pro naši simulaci. Ve skriptu nejdříve definujeme spuštění Gazebo balíčku s vytvořenou místností a s naším robotem. Dále definujeme spuštění balíčku pro mapování a lokalizaci. Pro tento účel použijeme balíček *rtabmap_ros*, který jsme již vyzkoušeli v kapitole 5 při mapování skutečné místnosti. Tímto balíčkem je náš spouštěcí skript kompletní.

V rámci této kapitoly jsme tedy vytvořili model robota, kterého můžeme ovládat vstupy z klávesnice. K němu jsme připojili virtuální kameru, jenž umožňuje v kombinaci s balíčkem *rtabmap_ros* vytvářet mapu vymodelovaného světa. Tím máme vše připravené a můžeme přistoupit k samotnému experimentu.

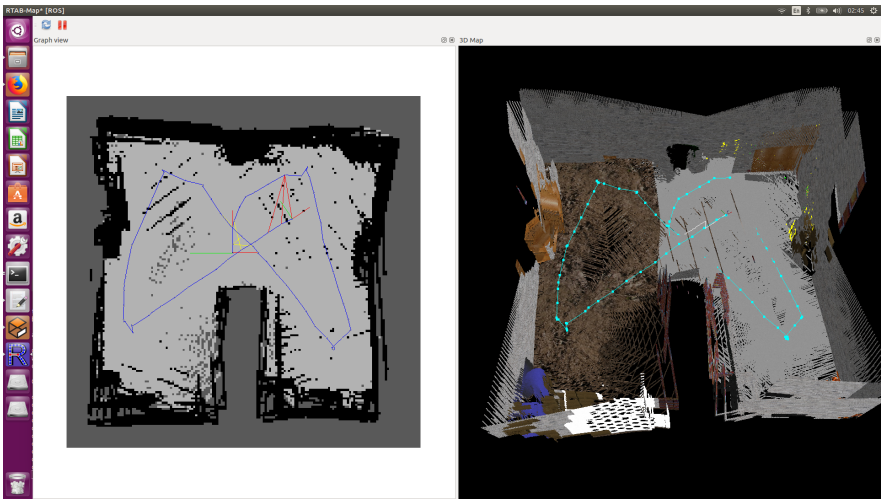
6.5.1 Výsledky simulace

Na obrázku 6.9 je vidět průběh mapování a lokalizace v simulovaném světě. V levé části je aktuální záběr kamery s vyznačenými orientačními body. Na pravé straně je pak rozpracovaná 3D a 2D mapa, obě s vyznačenou trajektorií robota.

V experimentu se nám podařilo uspokojivě namapovat celý prostor virtuální místnosti. Výsledná 3D mapa a 2D mřížka obsazenosti je na obrázku 6.10. Pokud tuto mapu a mřížku porovnáme s místností z obrázku 6.8, vidíme, že obě namapované struktury odpovídají naší modelové místnosti. Celkově tedy experiment můžeme považovat za úspěšný.



Obrázek 6.9: Průběh mapování simulovaného prostředí



Obrázek 6.10: Mřížka obsazenosti a 3D mapa vytvořená pomocí SLAMu v Gazebu

6.5.2 Nedostatky simulace

V průběhu mapování se opakovaně stalo, že SLAM algoritmus určil dvě odlišné části pokoje jako stejné, a proto vytvořil špatnou mapu. Pokud byl robot u zdi a v zorném poli kamery nebyl zároveň se zdí žádný výrazný předmět, který by odlišil toto místo, tak se stávalo, že algoritmus chybně namapoval tuto a protější zeď přes sebe. Chybovost se snížila přidáním dalších předmětů do modelové místnosti.

To vede k domněnce, že chyby jsou způsobeny tím, že v našem modelu pokoje je relativně malý počet předmětů a všechny zdi mají stejnou texturu. Zeď na jedné straně místnosti tedy vypadá naprosto stejně jako na ostatních třech stranách.

V reálném prostředí by tyto chyby hrozit neměly, jelikož žádné dvě reálné zdi nevypadají úplně stejně. Navíc je obvykle v reálném prostředí více předmětů než v naší simulaci, podle těchto předmětů pak SLAM algoritmus může jednotlivá místa odlišovat.

7. Závěr

Na závěr práce projdeme jednotlivé cíle vytyčené v kapitole 2 a pokusíme se zhodnotit jejich splnění. Po vyhodnocení se pak ještě zaměříme na nedostatky práce a navrhneme možné směry navazující práce.

1. V rámci kapitoly 3 jsme se seznámili s úlohou současného mapování a lokalizace. Tato kapitola poskytuje popis problému a také stručný úvod do algoritmů používaných při řešení SLAMu.

2. Kapitola 4 obsahuje popis prostředí ROSu včetně představení balíčku Gazebo, který se používá v ostatních částech práce.

3. V kapitole 5 jsme provedli experimentální mapování a lokalizaci v reálném prostředí. Tento experiment ověřil, že kameru Intel RealSense D435i lze využít pro SLAM.

4. Na základě poznatků z kapitoly 5 jsme v kapitole 6 pokračovali vytvořením modelu robota a provedením simulace současné lokalizace a mapování v prostředí Gazeba. Tento experiment napovídá, že kameru by mělo jít použít pro SLAM v mobilní robotice. Balíček pro ROS vytvořený v rámci kapitoly 6 je součástí přílohy A.

7.1 Navazující práce

Situace okolo Covidu-19 způsobila, že po většinu psaní této práce byly uzavřeny školy. Nepřistoupilo se tedy k sestavení skutečného robota. Další práce by tak měla směřovat tímto směrem. Většina kódu, který je použit pro mapování v simulačním prostředí, by se dala využít i pro reálného robota. Pro řízení skutečného robota by pak bylo potřeba implementovat ještě ovládání motorů pohonu.

Před vlastní stavbou robota by bylo možné vytvořit CAD modely součástí skutečného robota a těmi nahradit zástupné geometrické objekty v modelu pro Gazebo. Tímto krokem bychom mohli před samotnou stavbou ještě ověřit celkový návrh robota.

Další možností pro budoucí rozšíření práce je vytvoření plánovače trasy, který by automaticky navigoval robota k neprozkoumaným částem mapy. Tím bychom

z robota udělali plně autonomní stroj, který by dokázal sám zmapovat neznámý prostor.

Použité zdroje

[1] Delft University of Technology. Hello (Real) World with ROS – Robot Operating System. `https://ocw.tudelft.nl/courses/hello-real-world-ros-robot-operating-system`, 2019. [cit. 2020-5-16].

[2] Intel. Intel RealSense Technology. `https://www.intelrealsense.com/`, 2020. [cit. 2020-5-23].

[3] Intel developers. Wrappers for Intel RealSense Technology. `https://github.com/IntelRealSense/librealsense/tree/master/wrappers`, 2020. [cit. 2020-5-24].

[4] Jorge Fuentes-Pacheco, Jose Ascencio, and J. Rendon-Mancha. Visual simultaneous localization and mapping: A survey. *Artificial Intelligence Review*, 43, 11 2015.

[5] Miroslav Kulich. Materiály k předmětu mobile and collective robotics. `https://cw.fel.cvut.cz/old/_media/courses/b3m33mkr/slam.pdf`, 2016. [cit. 2020-5-24].

[6] Xu Lei, Bin Feng, Guiping Wang, Weiyu Liu, and Yalin Yang. A Novel FastSLAM Framework Based on 2D Lidar for Autonomous Mobile Robot. *Electronics*, 9:695, 2020.

[7] Tomáš Neužil. *Průběžná lokalizace a mapování pomocí mobilního robotu*. Disertační práce, Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky., 2008. [cit. 2020-5-14].

[8] H. Durrant-Whyte and T. Bailey. Simultaneous localization and mapping: part I. *IEEE Robotics & Automation Magazine*, 13(2):99–110, 2006.

[9] Petr Štrunc. Simultánní lokalizace a mapování v budově. `http://hdl.handle.net/11025/27253`, 2017. [cit. 2020-6-16].

[10] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Přednášky ke knize Probabilistic Robotics. `http://www.probabilistic-robotics.org/`, 2006. [cit. 2020-6-12].

[11] Wikipedia contributors. Monte carlo localization — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Monte_Carlo_localization&oldid=904812688, 2019. [cit. 2020-6-15].

[12] Andreas Lindholm. Particle filter explained without equations.

[13] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. 11 2002.

[14] Cyrill Stachniss. Materiály k předmětu Robot Mapping. http://ais.informatik.uni-freiburg.de/teaching/ws12/mapping/, 2019. [cit. 2020-6-25].

[15] Sebastian Thrun and Michael Montemerlo. The graph slam algorithm with applications to large-scale mapping of urban structures. *I. J. Robotic Res.*, 25:403–429, 05 2006.

[16] Tae Nam, Jae Shim, and Young Cho. A 2.5d map-based mobile robot localization via cooperation of aerial and ground robots. *Sensors*, 17:2730, 11 2017.

[17] Giorgio Grisetti, Gian Diego Tipaldi, Cyrill Stachniss, Wolfram Burgard, and Daniele Nardi. Speeding-up rao-blackwellized slam. pages 442–447, 01 2006.

[18] Kai Wurm, Cyrill Stachniss, and Giorgio Grisetti. Bridging the gap between feature- and grid-based SLAM. *Robotics and Autonomous Systems*, 58:140–148, 02 2010.

[19] T.J. Chong, X.J. Tang, C.H. Leng, Mohan Yogeswaran, O.E. Ng, and CHONG Yu Zheng. Sensor Technologies and Simultaneous Localization and Mapping (SLAM). *Procedia Computer Science*, 76:174–179, 12 2015.

[20] Dirk Thomas. Changes between ROS 1 and ROS 2. http://design.ros2.org/articles/changes.html, 2020. [cit. 2020-7-30].

[21] Mikio Sakemoto. Build an Autonomous Mobile Robot with the Intel RealSense Camera, ROS, and SAWR. https://software.intel.com/content/www/us/en/develop/articles/build-an-autonomous-mobile-robot-with-the-intel-realsense-camera-ros-and-saw.html, 2017. [cit. 2020-8-1].

[22] Goncalo Cabrita, Raj Madhavan, and Lino Marques. A framework for remote field robotics competitions. *Proceedings - 2015 IEEE International Conference on Autonomous Robot Systems and Competitions, ICARSC 2015*, pages 192–197, 05 2015.

[23] OmniVision. OV9282 Image Sensor. `https://www.ovt.com/sensors/OV9282`, 2020. [cit. 2020-8-5].

[24] Intel developers. ROS Wrapper for Intel RealSense Devices. `https://github.com/IntelRealSense/realsense-ros`, 2020. [cit. 2020-8-6].

[25] Cognitoware. Bayes Filters for a Differential Drive Robot. `http://cognitoware.com/tutorials/DifferentialDrive.htm`, 2009. [cit. 2020-8-3].

[26] ClearPath Robotics. Husky Unmanned Ground Vehicle. `https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`, 2020. [cit. 2020-8-3].

[27] Olaf Diegel. OddBot omni-directional mecanum wheeled robot. `https://www.oddguitars.com/oddbot.html`, 2011. [cit. 2020-8-4].

[28] Xiaodong wu, Min Xu, and Lei Wang. Differential speed steering control for four-wheel independent driving electric vehicle. pages 1–6, 05 2013.

[29] Kiran Palla. Autonomous Navigation of ROS Robot. `https://kiranpalla.com/autonomous-navigation-ros-differential-drive-robot-simulation/`, 2020. [cit. 2020-8-7].

[30] PAL Robotics and Salah Missri. Intel RealSense Gazebo ROS plugin. `https://github.com/pal-robotics/realsense_gazebo_plugin`, 2020. [cit. 2020-8-7].

A. Přílohy

CD přiložené k práci obsahuje následující přílohy:

- `/robot` – Balíček vytvořený v rámci této práce. Obsahuje model robota, model místnosti sestavený v prostředí Gazeba a *launch files*, které slouží pro spuštění simulace SLAMu.
- `/README.md` – soubor popisující instalaci a spuštění balíčku v ROSu.
- `/thesis.pdf` – elektronická verze této práce.