

Navigace mobilního robotu myAGV ve frameworku ROS

Bc. Filip Klapetek

Bakalářská práce
2025



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav automatizace a řídicí techniky

Akademický rok: 2024/2025

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	Bc. Filip Klapetek
Osobní číslo:	A22648
Studijní program:	B0714A150006 Aplikovaná informatika v průmyslové automatizaci
Specializace:	Inteligentní systémy s roboty
Forma studia:	Prezenční
Téma práce:	Navigace mobilního robotu myAGV ve frameworku ROS
Téma práce anglicky:	Navigation of the myAGV Mobile Robot within the ROS Framework

Zásady pro vypracování

1. Zpracujte rešerši mobilních robotů pro výuku
2. Seznamte se s metodami lokalizace a mapování v mobilní robotice
3. Seznamte se s prostředím ROS a navigačním balíčkem Nav
4. Navrhněte řešení pro navigaci mobilního robota v bludišti
5. Navržené řešení implementujte do mobilního robota MyAGV a otestujte autonomnost systému

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. SIEGWART, Roland; NOURBAKHSH, REZA, Illah a SCARAMUZZA, Davide. *Introduction to Autonomous Mobile Robots*. Second edition. MIT Press, 2011. ISBN 0262015358
2. CORKE, Peter. *Robotics, Vision and Control*. Second Edition. Berlin Heidelberg: Springer, 2017. ISBN 978-3-319-54412-0.
3. RICO, Francisco Martin. *A Concise Introduction to Robot Programming with ROS2*. CRC Press, 2023. ISBN 978-1-032-26720-3
4. KLANČAR, Gregor; ZDEŠAR, Andrej; BLAŽIČ, Sašo a ŠKRJANC, Igor. *Wheeled mobile robotics*. Elsevier, 2017. ISBN 978-0-12-804204-5.

Vedoucí bakalářské práce: **Ing. Jakub Novák, Ph.D.**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **13. prosince 2024**

Termín odevzdání bakalářské práce: **27. května 2025**

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan



prof. Ing. Vladimír Vašek, CSc. v.r.
ředitel ústavu

Ve Zlíně dne 13. prosince 2024

PROHLÁŠENÍ AUTORA BAKALÁŘSKÉ PRÁCE

Beru na vědomí, že

- odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb., v platném znění bez ohledu na výsledek obhajoby;
- bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému a bude dostupná k nahlédnutí;
- jedno vyhotovení bakalářské práce v listinné podobě bude ponecháno Univerzitě Tomáše Bati ve Zlíně k uložení;
- na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- podle § 60 odst. 1 autorského zákona má Univerzita Tomáše Bati ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- podle § 60 odst. 2 a 3 mohu užít své dílo – bakalářskou práci – nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tj. k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá; neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji, že

- jsem na bakalářské práci pracoval(a) samostatně a použitou literaturu jsem řádně citoval(a); v případě publikace výsledků budu uveden(a) jako spoluautor;
- odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou obsahově totožné.

Ve Zlíně, dne

.....
podpis autora

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a implementací autonomního navigačního systému mobilního robotu v rámci ROS. Účelem tohoto experimentu je podrobně popsat proces sestavení systému, který robotu umožňuje autonomně prozkoumat neznámé prostředí, vytvořit mapu, lokalizovat se a bezpečně se v něm pohybovat. V úvodní části této bakalářské práce je popsána oblast mobilní robotiky a její výhody jak v průmyslu, tak ve školství. Dále se teoretická část zaměřuje na základní principy lokalizace a mapování.

Praktická část práce se věnuje konkrétní implementaci systému na platformě myAGV, včetně tvorby síťové a softwarové architektury, konfigurace různých ROS balíčků a testování v reálném prostředí modelového bludiště. Výsledné řešení je doplněno rozšířením systému o funkcionalitu patrolování a návratu na výchozí pozici. Tato práce ukazuje, jak lze open source nástroje ROS použít k vytvoření funkčního a modulárního systému pro autonomního mobilního robotu.

Klíčová slova: mobilní robotika, ROS, myAGV, SLAM, AMCL, navigace, lokalizace, mapování

ABSTRACT

This bachelor thesis focuses on the design and implementation of an autonomous navigation system for a mobile robot using the ROS framework. The aim of this work is to provide a detailed description of the process of building a system that enables a robot to autonomously explore an unknown environment, create a map, localize itself, and navigate safely. The introductory part describes the field of mobile robotics and its benefits both in industry and education. The theoretical part focuses on the basic principles of localization and mapping.

The practical part then presents the specific implementation of the system on the myAGV platform, including the creation of network and software architecture, configuration of various ROS packages, and testing in a real-world maze environment. The final solution is extended with features such as patrolling and returning to a home position. This work demonstrates how open-source ROS tools can be used to create a functional and modular system for autonomous mobile robotics.

Keywords: mobile robotics, ROS, myAGV, SLAM, AMCL, navigation, localization, mapping

Rád bych poděkoval svému vedoucímu bakalářské práce Ing. Jakubovi Novákovi, Ph.D. za odborné vedení a vstřícný přístup při zpracování práce.

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	14
1 MOBILNÍ ROBOTIKA.....	15
1.1 AGV vs AMR	17
1.2 TECHNICKÉ VYBAVENÍ	18
1.3 APLIKACE.....	19
1.4 SOUČASNÉ TRENDY A VÝZVY	20
2 MOBILNÍ ROBOTIKA VE VÝUCE	22
2.1 PŘEHLED MOBILNÍCH ROBOTŮ PRO VÝUKU	23
2.1.1 MyAGV – Elephant Robotics	23
2.1.2 MyAGV 2023 – Elephant Robotics	24
2.1.3 TurtleBot 4	25
2.1.4 iRobot – Create 3	26
2.1.5 Husarion – ROSbot 3	27
2.1.6 AgileX – LIMO PRO	28
2.1.7 Hiwonder – JetAuto	29
2.2 SROVNÁNÍ MOBILNÍCH VÝUKOVÝCH ROBOTŮ	30
3 ROBOT OPERATING SYSTÉM.....	32
3.1 ARCHITEKTURA.....	33
3.1.1 Uzel (Node).....	34
3.1.2 Zprávy (Messages)	35
3.1.3 Témata (Topics)	36
3.1.4 Služby (Services)	37
3.1.5 Akce (Action).....	37
3.1.6 Master.....	38
3.1.7 Parameter Server	39
3.2 SOUBOROVÝ SYSTÉM (FILE SYSTEM)	40
3.2.1 Catkin	41
3.2.2 Pracovní prostředí (Workspace).....	42
3.3 KLÍČOVÉ NÁSTROJE A BALÍČKY V MOBILNÍ ROBOTICE.....	43
3.4 ROS 1 vs ROS 2	46
3.4.1 Využití ROS v praxi.....	47
4 LOKALIZACE A MAPOVÁNÍ V MOBILNÍ ROBOTICE.....	48
4.1 LOKALIZACE A JEJÍ METODY	48
4.1.1 Bayesův filtr	50
4.1.2 Markovova lokalizace	51
4.1.3 Rozšířený Kalmanův filtr (EKF).....	51
4.1.4 Monte Carlo Localization (MCL)	52

4.2	MAPOVÁNÍ	53
4.2.1	Metrické mapy.....	54
4.2.2	Topologické mapy	54
4.2.3	Hybridní mapy.....	55
4.3	SOUČASNÁ LOKALIZACE A MAPOVÁNÍ.....	56
II	PRAKTICKÁ ČÁST	58
5	NÁVRH ŘEŠENÍ.....	59
5.1	POŽADAVKY	59
5.2	PŘEDPŘIPRAVENÉ PROSTŘEDÍ NA ROBOTU	59
5.3	POUŽITÝ HW A SW	60
5.3.1	myAGV	60
5.3.2	Router	61
5.3.3	VirtualBox	62
5.3.4	TightVNC Viewer	63
5.4	ARCHITEKTURA SYSTÉMU	63
5.4.1	Síťová topologie	63
5.4.2	Softwarové rozvržení	65
6	VYTVOŘENÍ PROSTŘEDÍ PROJEKTU.....	67
6.1	NASTAVENÍ VIRTUÁLNÍHO STROJE A PROSTŘEDÍ ROS	67
6.1.1	Nastavení adres v LAN	69
6.1.2	Připojení pomocí SSH.....	69
6.1.3	Připojení pomocí VNC.....	70
6.1.4	Propojení ROS master/klient.....	71
6.2	OŽIVENÍ ROBOTU	72
6.3	SYNCHRONIZACE SYSTÉMOVÉHO ČASU	73
7	MAPOVÁNÍ NEZNÁMÉHO PROSTŘEDÍ.....	74
7.1	KONFIGURACE SLAM BALÍČKU	74
7.1.1	Telemetrie.....	75
7.2	NAVIGACE.....	76
7.2.1	Tvorba konfiguračních souborů	77
7.2.2	Nastavení vizualizace a testování.....	78
7.3	PRŮZKUM PROSTŘEDÍ	79
7.3.1	Balíček Explore_lite	79
7.3.2	Uložení výchozí polohy.....	80
7.3.3	Návrat na výchozí polohu a uložení mapy	81
7.3.4	Testování a vizualizace	83
8	NAVIGACE V BLUDIŠTI	84
8.1	LOKALIZACE ROBOTU	84
8.2	VYTVOŘENÍ TESTOVACÍHO BLUDIŠTĚ	85

8.2.1	Testování a vizualizace	87
8.3	PATROLOVÁNÍ.....	88
8.3.1	Nastavení bodů pro patrolování	88
8.3.2	Spuštění s možností návratu na výchozí polohu	90
8.3.3	Vyhýbání překážkám	90
9	ZHODNOCENÍ VÝSLEDKŮ	92
9.1	VYHODNOCENÍ VÝSLEDKŮ	92
9.2	MOŽNOSTI VYLEPŠENÍ.....	93
	ZÁVĚR	95
	SEZNAM POUŽITÉ LITERATURY.....	97
	SEZNAM OBRÁZKŮ	102
	SEZNAM TABULEK.....	104
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	105
	SEZNAM PŘÍLOH.....	106

ÚVOD

Robotika je jedním z nejrychleji se rozvíjejících oborů moderních technologií, jehož význam v posledních letech roste. Zatímco průmyslové roboty se uplatňují hlavně ve výrobních procesech, mobilní robotika nachází využití v různých oblastech, jako je logistika, zdravotnictví, autonomní průzkum neznámých prostředí nebo dohledové systémy. Díky své schopnosti pohybovat se v prostoru rozšiřují mobilní roboty možnosti automatizace mimo tradiční výrobní linky.

Významným milníkem pro rozvoj mobilní robotiky se stal nástup frameworku ROS (Robot Operating System), který standardizoval vývoj robotických aplikací. V minulosti si vývojáři museli jednotlivé komponenty (komunikaci se senzory, plánování trajektorie, řízení motorů atd.) řešit individuálně pro každou platformu, což výrazně zpomalovalo vývoj a omezovalo přenositelnost řešení. ROS umožňuje modulární přístup, opakované využití existujících balíčků a snadnou integraci různých zařízení a senzorů, čímž urychluje celkový vývoj, zvyšuje robustnost systému a jeho přenositelnost a otevírá nové cesty výzkumu.

Díky otevřenosti ROS a existenci množství volně dostupných nástrojů a robotických platform se mobilní robotika stala také důležitou součástí technického vzdělávání. Studenti se mohou zapojit do návrhu, programování a testování systémů, které odpovídají reálným průmyslovým standardům. Výuka tak není omezena pouze na teoretické znalosti, ale rozvíjí i praktické dovednosti v oblasti sensoriky, řízení, algoritmizace a mnoha dalších.

Tato práce se zaměřuje na vytvoření funkčního systému autonomní navigace mobilního robota myAGV ve frameworku ROS. V teoretické části je podrobně popsán samotný framework ROS, jeho architektura, principy komunikace mezi jednotlivými komponentami a nástroje, které usnadňují vývoj a ladění robotických aplikací. Dalším důležitým tématem, které se práce v teoretické části věnuje je lokalizace a mapování, které patří ke klíčovým schopnostem mobilních robotů. Popsány jsou zde základní principy filtrování a různé přístupy k mapování.

Na teoretickou část navazuje praktická realizace, která se snaží důkladně popsat celý proces vytváření systému schopného samostatně prozkoumat neznámé prostředí, vytvářet jeho mapu a poté se v ní lokalizovat a navigovat bezpečně dle potřeby. Tato část provádí čtenáře jednotlivými kroky od návrhu a konfigurace prostředí, přes implementaci klíčových komponent, až po jejich propojení a testování v reálných podmínkách a ukazuje pracovní proces při vytváření robotických aplikací ve frameworku ROS.

I TEORETICKÁ ČÁST

1 MOBILNÍ ROBOTIKA

Mobilní robot může být definován jako robot, který je schopen se sám přemístit v prostoru. Narozdíl od stacionárních robotů mají tedy velkou výhodu v podobě vysoké mobility. Obecně se dá z jakéhokoliv robotu udělat robot mobilní pomocí podvozku, pojezdů či jiného systému, který umožňuje pohyb robotu. [1]

„Mobilní systémy lze definovat jako systémy, které nejsou připojeny k síti.

a mohou se pohybovat v určitém prostoru“. [2]

Mobilní roboty lze rozdělit z několika hledisek, podle:

- Míry autonomnosti:
 - Teleoperované – plně řízené člověkem na dálku. Člověk ovládá veškeré výstupy robotu.
 - Semi-teleoperované – člověk poskytuje určité cíle a robot autonomně vykoná některé řídicí činnosti.
 - Autonomní – robotu je zadán určitý cíl a poté jej robot samostatně vykoná.
- Kinematiky:
 - Kolové, kráčející, létající, vodní (také podvodní), hybridní.
- Řídicí architektury (tj. jak robot na základě vstupů plánuje své akce):
 - Reaktivní architektura – přístup analýzy zdola nahoru. Robot má definované základní moduly, které představují nejnižší chování robotu. Z těchto modulů jsou sestavovány složitější vzorce chování. Jednotlivé moduly mají přímý přístup k datům senzorů. Samotné prostředí je jen zhruba namodelováno nebo vůbec.
 - Funkční dekompozice – přístup analýzy shora dolů. řízení je rozděleno na akce a události, které na sebe navazují. Prostor je navržen na různých úrovních abstrakce. Jednotlivé moduly tedy nepracují napřímo s daty ze senzoru.
 - Plánování na vyšší úrovni – založené na deklarativní reprezentaci znalostí o světě.
 - Neuronové sítě a genetické algoritmy.

- Hybridní systémy. [3][4]

Mobilní roboty lze považovat za autonomní, pokud jsou schopny samostatného pohybu v daném prostředí. Autonomie takového systému je však brána ze dvou pohledů:

- Z energetického hlediska – robot by měl mít na sobě připevněný zdroj energie.
- Z hlediska rozhodování – tedy samotný robot by měl být schopen vykonat určité rozhodnutí podle míry autonomie a na základě těchto rozhodnutí provést příslušné akce. [2]

Mobilní roboty přijímají určité příkazy od lidského operátora, které se mohou lišit podle míry autonomie. Robot reaguje na příkazy, zpracuje je a snaží se provést zadaný úkol a dílčí úkoly na nižších úrovních. V případě že se nevyskytne žádný nepředvídatelný problém je úloha dokončena za určitý časový interval. Příkazy, které robot přijímá mohou vypadat následovně:

- Požadovaná rychlost kol – s tím souvisí řídicí algoritmus daných senzorů (obvykle rotačních enkodérů) k zajištění správného otáčení kol robotu.
- Požadované podélné a úhlové rychlosti robotu – program si je vědom kinematického modelu robotu, aby mohl správně vypočítat dané rychlosti kol, tak, aby dosáhl požadované rychlosti robotu.
- Požadovaná trajektorie robotu – robot je schopen určit a také řídit svou polohu v prostředí. K lokalizaci robot využívá různé senzory, které jsou připevněny na něm samotném nebo se nacházejí v prostředí. Díky informacím ze senzorů je schopen určit svou přibližnou polohu a určit nejlepší trajektorii pohybu.
- Požadovaný chod uvnitř známého prostředí s potenciálními překážkami – robot má provést činnost ve známém prostředí, ve kterém se může vyskytovat statická nebo pohyblivá překážka. Robot je tedy schopen naplánování cesty a popřípadě upravit tuto cestu, pokud ho k tomu překážka donutí.
- Požadovaný chod uvnitř neznámého prostředí – robot nezná své prostředí a musí tedy provádět současně určování své polohy a sestavit mapu daného prostředí. Tento přístup je znám pod pojmem SLAM (simultaneous localization and mapping – současně probíhající lokalizování a mapování).

- Požadovaný úkol – Robot zahájí úkol, který má splnit. Pracuje v určitém prostředí, kde může spolupracovat s jinými roboty nebo lidmi. Robot musí mít představu o svých úkolech a také mezi nimi rozlišovat priority, a tedy být schopen přerušit určitý úkol a vykonat úkol s vyšší prioritou. Musí být obsaženo mnoho rozhodovacích procesů v samotném robotu.[2]

Robot jako autonomní není považován v prvních dvou bodech představených výše. [2]

1.1 AGV vs AMR

AGV (Automated Guided Vehicles) jsou autonomní vozidla, která se pohybují po předem definovaných trasách. Nejčastěji se využívají v průmyslové výrobě, ale také ve skladech a distribučních centrech pro opakované úkoly, jako je přeprava materiálu mezi výrobními linkami nebo přesun hotových produktů do skladů či expedičních zón. [5]

Tyto systémy jsou vybaveny senzory pro detekci okolí a překážek, avšak nedokážou autonomně měnit trasu a vyhýbat se překážkám. Jejich pohyb je striktně řízen a vyžaduje pevně stanovenou navigační infrastrukturu, v podobě drátů, magnetických pásek nebo přidavných senzorů. Tyto pevně dané trasy vyžadují rozsáhlou instalaci, která může být nákladná. AGV mají minimální palubní inteligenci a spoléhají se výhradně na definovanou cestu. Vozidla se mohou řídit jednoduchými programovacími pokyny, ale jen v omezené míře (např. vozidlo dokáže překážku na trase detekovat, ale ne objet, takže se vozidlo pouze zastaví) [5][6]

Ještě donedávna byla automaticky řízená vozidla jediným dostupným řešením pro automatizaci vnitropodnikové přepravy. AGV se tradičně uplatňovala ve velkých provozech s pevnou infrastrukturou, kde je klíčová pravidelná a opakovaná distribuce materiálu. Tento systém však vyžaduje vysoké počáteční investice a návratnost vložených prostředků je často dlouhodobá. [6]

Dnes však AGV čelí konkurenci autonomních mobilních robotů (AMR), které nabízejí vyšší flexibilitu, pokročilejší technologie a efektivnější provozní náklady. Přestože oba systémy slouží k přesunu materiálu, jejich princip fungování se výrazně liší. [6]

Autonomní mobilní roboty (AMR) jsou samostatně se pohybující roboti, kteří využívají pokročilé senzory a navigační technologie, aby se mohli volně pohybovat ve svém prostředí bez nutnosti pevných tras. [5]

Vozidla AMR jsou vybaveni obdobnými senzory jako AGV, jako je LiDAR a kamery, které jim umožňují rozpoznávat překážky, lidi a jiné objekty, a přizpůsobovat tak svou trasu v reálném čase nebo se případně přesměrovat. Avšak spolu se senzory je jejich nutnou součástí také sofistikovaný software, který slouží k sestavování map a k autonomní navigaci. Díky této flexibilitě jsou ideální pro dynamická prostředí (např. pro automatizovaný přesun zboží ve velkých skladech nebo řízení toku materiálu ve výrobních závodech), ale díky své míře autonomnosti je lze využívat i pro jiné účely, než pouze pro klasické úkony jako je přeprava materiálů a zboží ve skladech a výrobních halách. [5][6]

1.2 Technické vybavení

Mezi hlavní mechanické a elektronické součásti autonomního mobilního robotu patří:

- *„Mechanické části – pevné a pohyblivé (karoserie, kola, pásy, nohy atd.)*
- *Akční členy – elektrické motory (DC, krokové, servomotory atd.)*
- *Senzory – rotační enkodéry, snímače přiblížení a vzdálenosti, navigační jednotky, globální snímače navigačního systému atd.*
- *Výpočetní jednotky – mikrokontrolery, přenosné osobní počítače, vestavěné systémy atd.*
- *Napájecí jednotka – baterie, solární panely atd.*
- *Elektronika – řadiče akčních členů, měření senzorů, distribuce energie, telekomunikační elektronika“[2]*

Mimo zmíněné mechanické a elektronické součásti autonomního robotu, které tvoří hardware robotu je pro jeho správnou funkčnost klíčový software. Ten běží na výpočetní jednotce robotu a propojuje veškerý hardware a umožňuje jeho řízení, navigaci, komunikaci, autonomní rozhodování a interakci s okolím v reálném čase. [7]

Význam softwaru se ukazuje také ve správě robotických flotil (tedy skupin vzájemně pracujících robotů), která je stále více žádaná. V plně automatizovaných provozech často pracuje více robotů současně, a právě systémy pro řízení flotily umožňují jejich koordinaci, optimalizaci provozu a efektivní řízení celého ekosystému. [7]

Jeden z nejrozšířenějších frameworků pro vývoj softwaru v oblasti mobilní robotiky je ROS (Robot Operating System). ROS je standardem mezi open-source softwarovými platformami. Nabízí spoustu nástrojů pro zjednodušení vývoje složitých chování robotů.

Díky své modularitě a široké komunitní podpoře umožňuje rychlou integraci různých hardwarových komponent. [7]

ROS je taktéž využíván v rámci této práce, a je samostatně rozebrán v kapitole 3.

1.3 Aplikace

Mobilní roboty mají několik vlastností, které je činí velmi atraktivními. Mohou se dostat do nebezpečného prostředí bez přítomnosti člověka (např. minové pole, radioaktivní prostředí, hlubokomořský průmysl) nebo lze díky nim přistupovat do vzdálených a pro lidi nepřístupných prostředí (průzkum mimozemských planet, roboti v nano měřítku).

Robotické systémy mohou vykonávat fyzicky náročné úkoly místo lidí. Zavedení automatizace, robotiky a mobilních systémů může přispět ke zlepšení efektivity, produktivity, kvality produktů nebo služeb a snížit náklady na pracovní sílu.

V současnosti má mobilní robotika široké využití v různých oblastech. Díky rychlému rozvoji technologií se obor neustále rozšiřuje. Následuje seznam aplikací mobilních robotů:

- *„Zdravotnické služby, podpora operací (v situacích s rizikem infekce).*
- *Úklidové aplikace (vysávání, vytírání a zametání podlah).*
- *Aplikace v zemědělství – automatický sběr ovoce, sekání trávy, sazení semen.*
- *Údržba lesů a těžba dřeva.*
- *Obchody se spotřebním zbožím.*
- *Inspekce a dohled nad nebezpečnými oblastmi (detekce a deaktivace min v minových polích, revize jaderných reaktorů, čištění potrubí kanalizace).*
- *Vesmírné mise (družice, inspekce a servis družic, planetární mise průzkum planet).*
- *Průzkum mořské hlubiny (roboti pro kabeláž a kontrolu mořského dna).*
- *Roboty pro nakládání a vykládání zboží nebo materiálů z letadel, lodí, nebo nákladních vozidel.*
- *Vojenské roboty (průzkumné roboty, letadla a různé autonomní střely).*
- *Bezpečnostní roboti (strážní roboti pro kontrolu skladů, budov atd.)*

- *Pomoc starším lidem a zdravotně postiženým (autonomní invalidní vozíky, rehabilitační roboti).*
- *Spotřebitelské aplikace (robotičtí domácí mazlíčci, robotický fotbal atd.).*
- *Systémy ve výzkumných institucích zaměřené na učení a vývoj nových technologií a algoritmů.* “ [2]
- *Přeprava lidí formou autonomních dopravních prostředků.* [3]
- *Průmyslová automatizace.* [8]

Předvídat, jak bude budoucnost vypadat, bylo vždy těžké. Technologie a aplikace, které jsou dnes k dispozici si bylo obtížné představit i před deseti lety. Nicméně, v blízké budoucnosti, se očekává, že mobilní robotika, a především autonomní mobilní robotika se stane ještě více nedílnou součástí naší společnosti a každodenního života. Budou s námi pracovat v moderních továrnách budoucnosti, pomáhat s domácími pracemi, vozit po městech nebo např. zachraňovat životy při záchranných akcích. [2]

1.4 Současné trendy a výzvy

Mnoho trendů a výzev pro mobilní robotiku, lze převzít z výzev pro celou oblast robotiky. Avšak pro mobilní robotiku mají velký význam především umělá inteligence, mobilní manipulátory, humanoidní roboti a digitální dvojčata. [9]

Umělá inteligence a strojové učení již nejsou pouze teorie a dnes jsou součástí moderní robotiky. Tyto technologie umožňují efektivnější a přesnější řešení složitých úloh, čímž zvyšují přidanou hodnotu robotických systémů. Pomocí generativní AI lze roboty programovat prostřednictvím přirozeného jazyka, to výrazně usnadňuje jejich integraci. [10]

Hlavním účelem využití umělé inteligence v robotice je zlepšit zvládání proměnlivosti a neočekávaných situací v okolním prostředí, a to jak v reálném čase, tak při offline zpracování. Proto se umělá inteligence, včetně metod strojového učení, stále více využívají. Díky ní lze efektivně optimalizovat procesy, implementovat prediktivní údržbu nebo vyvíjet aplikace pro manipulaci s různorodými objekty za pomoci strojového vidění. [11][12]

Mobilní manipulátory, které spojuje pohyblivost autonomních robotů s precizností robotických ramen, zásadně transformují manipulaci s materiály a produkty ve výrobním prostředí. Tyto systémy jsou navrženy pro dynamické provozy, kde je nutná vysoká flexibilita a schopnost okamžité adaptace na změny. Díky pokročilým senzorům a

inteligentní navigaci se efektivně pohybují i ve složitých prostředích, což je klíčové například pro automobilový průmysl a logistiku. [10]

Spolupráce mezi lidmi a roboty zůstává jedním z hlavních trendů v robotice. Díky rychlému rozvoji senzorů, technologií počítačového vidění a inteligentních chapadel dokážou roboti v reálném čase reagovat na změny v prostředí, což jim umožňuje bezpečně spolupracovat s lidskými pracovníky. Kolaborativní roboti představují nový zdroj pro podporu zaměstnanců v oblastech s namáhavou činností, opakujícími se úkoly nebo pro práci v nebezpečných podmínkách. Spektrum kolaborativních aplikací se neustále rozšiřuje, neboť taktéž klesá počet kvalifikovaných zaměstnanců pro daný typ práce (např. svařování). [9]

Digitální dvojčata, jakožto digitální replika skutečného robotu umožňuje simulaci úkonů a jejich analýzu a optimalizaci v reálném čase bez potencionálního rizika. [12]

Humanoidní roboti zažívají velmi významný pokrok, již nyní zvládají širokou škálu úkolů v různých prostředích. Díky antropomorfní konstrukci se dvěma pažemi a nohama jsou schopni v určitých úkonech nahradit člověka nebo efektivně s lidmi spolupracovat. [9]

Velmi důležitou výzvou je udržitelnost a energetická efektivita robotů. Klade se důraz na použití ekologických materiálů a využívání energeticky úsporných systémů, díky čemuž se snižuje dopad na životní prostředí. Tato oblast je velmi důležitá, neboť robotika zažívá velký rozmach a využívání robotů se každým rokem zvyšuje. Proto je zásadní, aby globální trend o snižování uhlíkové stopy byl dodržován i v oblasti robotiky. [12]

2 MOBILNÍ ROBOTIKA VE VÝUCE

Výuka mobilní robotiky využívá robotiku k osvojení znalostí zábavnou a interaktivní cestou. Zároveň může usnadnit zvládnutí určitých předmětů prostřednictvím motivace k objevování a řešení problémů se kterými se mohou v rámci práce na mobilních robotech setkat. Mezi tyto předměty mohou patřit témata spojená se senzory, akčními členy, elektrotechnika, programování, algoritmizace. Získané teoretické znalosti jsou v rámci výuky mobilní robotiky aplikovány. [13]

Díky výuce mobilní robotiky se lze zábavnou formou naučit mnoho důležitých dovedností, mezi které patří:

Mechanika a elektrotechnika – Mechanika ztělesňuje vzájemné vztahy mezi různými kinematickými substrukturami a kinematikou celého robotu. Je důležité porozumět mechanickému součástem a jak spolu spolupracují jako celek. Mezi základní mechanismy patří servomotory, podvozek, odpružení, ložiska. V elektronické části jsou řídicí jednotky motorů, mikroprocesory, snímače vzdálenosti, senzory umožňující vidění jako LiDAR nebo kamery. [14]

Programování – Samotné programování robotu zahrnuje psaní příkazu a skriptů, díky kterým je robot ovládán v daném programovacím jazyce. Samotné dovednosti v oblasti programování přesahují rámec robotiky a zahrnují návrh, tvorbu, ladění a komentování kódu, a nakonec dokumentaci k programu. [14]

Týmová práce – Efektivní týmová práce je velmi důležitá v mnoha oblastech. Mezi klíčové dovednosti patří tvorba a ověřování nových nápadů, rozdělování rolí a odpovědností a společné vytváření znalostí pomocí pozorování, napodobování, brainstormingu. [14]

Řešení problémů – K řešení složitějších problémů jsou klíčové dovednosti trpělivost, vytrvalost a schopnost řešení problémů. V oblasti metodologie řešení problémů jsou klíčové vlastnosti tvorba účinných strategií pro řešení výzev, které se mohou během zpracování objevit. Stanovení vhodných dílčích cílů, využívání zpětné vazby robotu k identifikaci slabin současných postupů a schopnost včas rozeznat a upustit od neefektivních přístupů. [14]

Pohled ze strany robotu – To zahrnuje autonomii, integraci hardwaru a softwaru, řízení robotu programově, diagnostiku a další dovednosti. Tato oblast je důležitá pro pochopení samotných omezení robotů v rámci jejich možností v pohybu a omezeném rozsahu ve kterém operují. Do těchto omezení se řadí omezení ze strany senzorů, efektorů. Pohledem

robotu je myšleno to, jak se samotný robot „dívá“ a jak vnímá svět kolem sebe, tedy pochopit senzorická omezení a limity akčních členů, které definují jeho činnosti. Díky porozumění tohoto pohledu, může člověk, který s mobilním robotem pracuje může rozlišit různé typy chování, která mohou být reálně proveditelná a taková která jsou nerealizovatelná. [14]

Mobilní robotika má velký rozsah využití, konkrétně AMR (autonomní mobilní roboty) mohou být využity jak ve venkovních, tak vnitřních prostorech (viz kapitola 1.3 Aplikace). [15]

Různé aplikace mobilních robotů přináší odlišné navigační problémy. Pro autonomní mobilní roboty je klíčové vnímat a analyzovat pracovní prostředí, plánování tras, rozhodování a tvorba řídicích struktur, které jsou schopny adekvátně reagovat na získané informace. [16]

2.1 Přehled mobilních robotů pro výuku

Pro výuku mobilních robotů existuje celá řada modelů od různých výrobců. Liší se hlavně složitostí zpracování a jejich možnostmi, tedy jaké různé senzory obsahují. Zde budou porovnávány hlavně takové modely mobilních robotů, které se nejvíce přibližují k vlastnostem a cenové kategorii modelu myAGV od firmy Elephant Robotics, který je v rámci této práce dále využíván.

2.1.1 MyAGV – Elephant Robotics

Jedná se o mobilního robotu od firmy Elephant Robotics (viz Obrázek 1). Tento robot je využíván v rámci této práce. Mezi jeho základní vlastnosti patří:

- Velikost: 311x236x130 mm.
- Váha: 3,6 kg.
- Nosnost: 2 kg.
- 4 kola typu Mecanum.
- Motor: DC s hallovým enkodérem.
- Maximální otáčky motoru: 220 ot./min.
- Výdrž: 90 min (v navigačním režimu, jinak 120 min).
- Rychlost pohybu: 0,9 m/s.

- 360° LiDAR (na vzdálenost 0,12 – 8m), 5MP kamera.
- Rozsah detekce 8m.
- Raspberry Pi 4B.
- Konektory: 2x USB, 1x HDMI, 1x RJ45, I/O interface.
- Cena: Již se oficiálně neprodává. [17]



Obrázek 1 Mobilní robot myAGV [18]

2.1.2 MyAGV 2023 – Elephant Robotics

Jedná se o novější model mobilního robotu MyAGV. Změny se týkají hardwaru robotu, avšak jen minimálně, více změn lze zaznamenat v podpoře softwaru. Výrobce uvádí tyto vlastnosti:

- Velikost: 311,15x230x110 mm.
- Váha: 4,16 kg.
- Nosnost: až 5 kg.
- 4 kola typu Mecanum.
- Motor: BLDC s hallovým enkodérem.
- Maximální otáčky motoru: 220 ot./min.
- Výdrž: 181 min (v navigačním režimu, jinak 328 min).
- Rychlost pohybu: 0,9 m/s.

- 360° Lidar (na vzdálenost 0,12 – 8m), 5MP kamera.
- Rozsah detekce 8m.
- Raspberry Pi 4B nebo Jetson Nano.
- Konektory: 2x USB, 1x HDMI, 1x RJ45, 1x grove.
- Cena: 30 000 Kč. [19]

Tento model je vhodný pro výuku, neboť obsahuje systém blízký stolnímu pc. Systém myAGV má vestavěné prostředí a není potřeba dalších složitých konfigurací. K samotnému robotu lze taky připojit další periferie jako robotické rameno, displej, baterii nebo další senzory. Programovat lze pomocí blokového programování v myBlockly nebo pomocí pythonu přidáním potřebných knihoven od výrobce. [19]

2.1.3 TurtleBot 4

TurtleBot 4 (viz Obrázek 2) je nejnovější model řady TurtleBot. Model TurtleBot 4 se prodává ve dvou základních konfiguracích (Standard a Lite). Mezi hlavní rozdíly patří absence OLED displeje, desky k připojení dalších periferií a menší počet konektorů (u standard verze jsou navíc 4x USB, Raspberry Pi piny a ethernet konektor). Dále budou popsány vlastnosti TurtleBot4 Standard:

- Velikost: 341x339x351 mm.
- Váha: 3,9 kg.
- Nosnost: až 9 kg.
- Výdrž: 2,5 – 4 hodiny (závislé na zátěži).
- Rychlost pohybu: 0,31 m/s.
- 360° LiDAR, OAK-D-PRO kamera
- Další senzory: 2x přední nárazníkové zóny, 4x IR senzor pádu, 6x IR senzor překážek, 1x optický snímač ujeté vzdálenosti, 1x 3D gyroskop, 1x 3D akcelometr.
- Rozsah detekce 12 m.
- Raspberry Pi 4 B [20]
- Cena: 70 000 Kč (Lite model 42 000 Kč). [20]



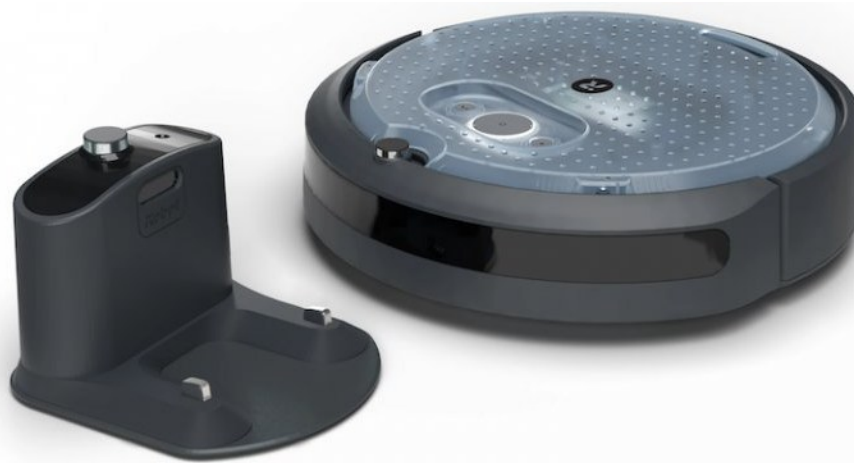
Obrázek 2 TurtleBot4 Standard (vlevo) a Lite (vpravo) [20]

TurtleBot 4 je open source platforma, lze tedy získat design a software zařízení. Je tedy vhodná volba pro výuku, neboť ji lze jednoduše vyzkoušet, rozšiřovat a měnit. K dispozici je taktéž dobře zpracovaná dokumentace od výrobce, která obsahuje detailní popis hardwaru, softwaru a obsahuje různé tutoriály. [9]

2.1.4 iRobot – Create 3

Nový model Create 3 (viz Obrázek 3) od společnosti iRobot nabízí mobilní robotickou platformu, která díky své modularitě umožňuje připojení dalšího hardwaru. Na přední straně lze připojit další skenery, kamerové moduly nebo jiné senzory. Mezi základní vlastnosti robotu patří:

- Velikost: 142,9 x 406,4 x 473,1 mm
- Váha: 4,68 kg.
- Nosnost: až 9 kg.
- Výdrž: do 2 hodin (závislé na zátěži).
- Rychlost pohybu: 0,31 m/s.
- Neobsahuje vestavěnou výpočetní jednotku a pro provoz ROS2 vyžaduje připojení externího počítače
- Senzory: 2x přední nárazníkové zóny, 4x IR senzor pádu, 7x IR senzor překážek, 1x optický snímač ujeté vzdálenosti, 1x 3D gyroskop, 1x 3D akcelometr. [21]
- Cena: 15 000Kč. [22]



Obrázek 3 iRobot Create 3 [21]

Software robotu pracuje na platformě ROS 2, která se stará o sběr a zpracování dat ze senzorů, ovládání akčních členů a zpracování požadavků od uživatele. Robot obsahuje již přednastavené autonomní funkce, jako automatické dokování, sledování stěn nebo reakce na překážky. Na webových stránkách výrobce lze najít manuál s popisem hardwaru, lekcemi nebo příklady softwaru. [21]

2.1.5 Husarion – ROSbot 3

ROSbot 3 (viz Obrázek 4) je mobilní robotická platforma pro výzkum a vzdělávání. Konstrukce robotu je velmi kvalitní a odolná vyrobená z 1,5 mm kovové slitiny. Ovladače systému ROS 2 jsou již předinstalované a nakonfigurované, a proto robot nabízí efektivní a snadnou platformu pro testování algoritmů, výuku robotiky nebo pro pokročilejší vědecké a průmyslové projekty. Samotné ovládání lze provádět přes webové rozhraní. Základní vlastnosti robotu jsou:

- Velikost: 200x233x197 mm.
- Váha: 2,84 kg.
- Nosnost: až 5 kg.
- Výdrž: 1,5 – 5 hodiny (závislé na zátěži).
- Rychlost pohybu: 1 m/s.
- 360° Lidar SLAMTEC C, 3D kamera Luxonis OAK-D Lite
- Další senzory: inteligentní senzor absolutní orientace (akcelometr a gyroskop), IR senzor vzdálenosti.

- Raspberry Pi 5 8 GB.
- Cena: 65 000 Kč. [23]



Obrázek 4 ROSbot 3 [23]

K modelu je na stránkách výrobce dokumentace společně se simulačním modelem v prostředí Gazebo, dále popis HW a SW částí modelu a další odkazy na různé manuály a návody. [23]

2.1.6 AgileX – LIMO PRO

Limo PRO od firmy AgileX Robotics usnadňuje jízdu na jakémkoli povrchu díky čtyřem nezávisle řízeným kolům. Zvládá jízdu na hladkém betonu nebo písčité cestě. Tato flexibilita je výhodná pro vývojáře zkoumající různé aplikace. Má vyspělou centrální výpočetní jednotku NVIDIA Orin Nano, díky ní je model schopen velmi zvládat náročné výpočty v rámci zpracování dat z více senzoru a SLAM mapování. Jeho hlavní vlastnosti jsou:

- Velikost: 322x220x251 mm.
- Váha: 4,8 kg.
- Nosnost: až 4,8 kg.
- Výdrž: 2,5 - 4 hodiny (závislé na zátěži).
- Rychlost pohybu: 1 m/s.
- EAI T-mini Pro LiDAR, kamera Orbbec Dabai
- Další senzory: inteligentní senzor absolutní orientace (akcelometr a gyroskop), IR senzor vzdálenosti.

- Procesor Intel NCU i7.
- Cena: 80 000 Kč.[24]



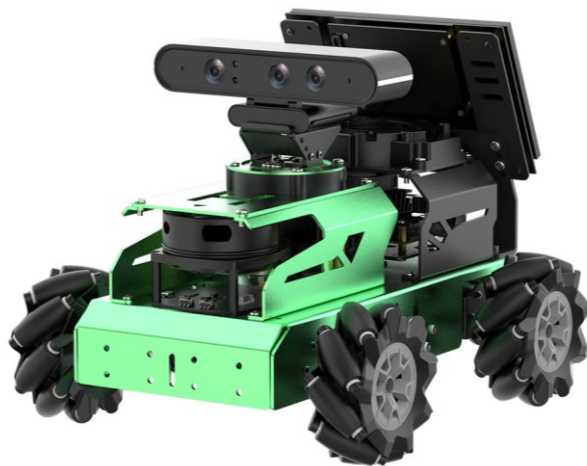
Obrázek 5 LIMO PRO [24]

Hlavní výhodou robotu je podpora ROS a GAZEBO platformy, taktéž kompatibilita s programovacími jazyky Python a C++. Platforma taktéž obsahuje demo ukázky modelů hlubokého učení, ROS mapování a navigace zajištění snadnějšího učení a implementace jednotlivých metod. [24]

2.1.7 Hiwonder – JetAuto

Jedná se o jednoduchou a levnou platformu, která má však vše potřebné pro výuku mobilních robotů. Veškeré hardware komponenty jsou kvalitní a na webových stránkách výrobce jsou taktéž rozsáhle tutoriály k robotu, které pokrývají základy ROS, metod mapování, strojového učení a další.

- Velikost: 302x260x256 mm.
- Váha: 3,5 kg.
- Výdrž: 60 minut (závislé na zátěži).
- 360° LiDAR, 3D kamera.
- Další senzory: inteligentní senzor absolutní orientace (akcelometr a gyroskop), IR senzor vzdálenosti.
- Raspberry Pi 5 8 GB nebo Jetson Nano nebo Jetson Orin Nano.
- Cena: 27 000 Kč. [25]



Obrázek 6 JetAuto [25]

Při objednání robotu si lze vybrat preferovaný typ kontroléru robotu. Výběrem kontroléru se taktéž změní jisté vlastnosti robotu, avšak změny jsou minimální. [25]

2.2 Srovnání mobilních výukových robotů

V Tabulce 1 lze vidět porovnání výše popisovaných mobilních robotů, které by byly vhodné k výuce mobilní robotiky. Jedná se o souhrn hlavních vlastností robotů, podle kterých lze vybrat vhodný robot podle potřeby.

Tabulka 1 – Srovnání hlavních vlastností vybraných mobilních robotů

Robot	Senzory	Počítač	Autonomie	Dokumentace	Podpora AI	Cena (Kč)
myAGV 2023	LiDAR, IMU, encodery, ToF	Raspberry Pi 4B	SLAM, navigace	Dostupná, ale omezená	Možná (doplňkové moduly)	30 000
TurtleBot 4	LiDAR, kamera, IMU	Raspberry Pi 4B / Jetson Nano	Autonomní navigace	Velmi dobrá (oficiální podpora ROS2)	Ano (Jetson Nano varianta)	40 000
iRobot Create 3	IR senzory, IMU, odometrie	Externí (nutné připojit)	Bez SLAM	Základní	Ne	15 000
Husarion ROSbot 3	LiDAR, IMU, encodery, ToF	Intel NUC / Jetson Xavier	Pokročilá navigace	Dostupná, ale méně rozsáhlá	Ano (Jetson Xavier)	65 000
AgileX LIMO PRO	LiDAR, kamera, IMU	Jetson Xavier NX	AI, SLAM	Pokročilá (průmyslová úroveň)	Ano (AI specializace)	80 000
Hiwonder JetAuto	Kamera, ToF, IMU	Jetson Nano/ Raspberry Pi	AI, CV	Základní (pro AI)	Ano (Jetson Nano varianta)	27 000

Jedno z hlavních kritérií je cena, od které se poté odvíjejí další vlastnosti robotů. Ve výběru jsou 3 cenové kategorie. Do 15 000 Kč se jedná o robot iRobot Create3. V cenovém rozmezí 30 000 – 40 000 jsou roboti TurtleBot 4, myAgv a Hiwonder JetAuto. V poslední nejdražší cenové kategorii jsou roboti od firmy Husarion a AgileX, kde jde vidět cenový rozdíl hlavně v kvalitnější a robustnější konstrukci robotu.

Důležitá vlastnost je vnitřní výpočetní jednotka robotů. Všichni roboti mají vestavěný počítač, buď různé verze raspberry Pi nebo Jetson od společnosti Nvidia, kromě

nejlevnějšího modelu iRobot Create 3, který k funkčnosti potřebuje externí počítač. Raspberry Pi je univerzální jednodeskový počítač pro běžné aplikace v rámci robotiky a zvládne jednoduché AI úlohy, zatímco jednodeskové počítače Jetson jsou specializované právě na AI metody (strojové učení, hluboké učení, počítačové vidění atd.).

Co se týče vybavenosti v rámci senzoričky jsou na tom roboti obdobně a základ mají všichni roboti podobný, avšak kvalita a počet senzorů koreluje s cenou robotu.

Starší verze myAGV (verze která je použita v rámci BP) oficiálně podporuje pouze ROS, novější verze má podporu ROS 2 stejně jako ostatní zmínění mobilní roboti.

Další důležitou vlastností při využití těchto robotů k výuce je jejich dokumentace poskytnutá výrobcem. Zde vyniká TurtleBot 4, který má také přímo oficiální podporu ROS2. Velmi pokročilou dokumentaci disponuje robot AgileX Limo Pro.

3 ROBOT OPERATING SYSTÉM

Robot Operating System (zkráceně ROS) je open-source, meta-operating systém, což znamená, že poskytuje služby, které by byly očekávány od operačního systému, včetně abstrakce hardwaru, nízko úrovněového řízení zařízení, implementace běžně používaných funkcí, předávání zpráv mezi procesy a správy balíčků. Poskytovány jsou také nástroje a knihovny pro získávání, sestavování, psaní a spouštění kódu na více počítačích. [26]

Hlavním cílem frameworku ROS je podpora opakovaného použití kódu v oblasti výzkumu a vývoje robotiky. Díky svým možnostem se rychle stal vývojovým standardem pro mnoho výzkumných institucí a firem zabývajících se robotikou. [27]

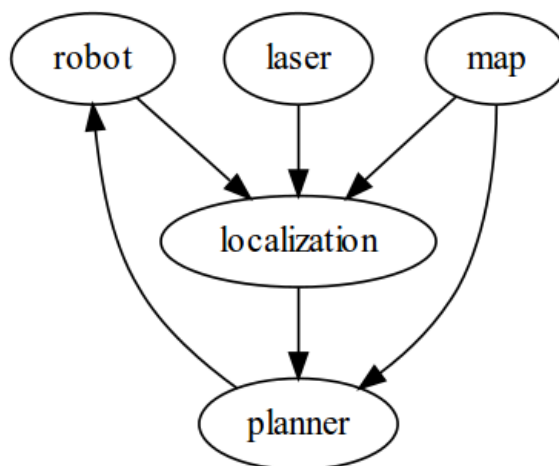
Jedním z původních motivačních scénářů, které významně ovlivnily návrh systému ROS, byl problém tzv. fetch an item (tedy přines předmět). Tento úkol představuje použití složitě robotu vybaveného více kamerami, laserovými skenery, manipulační paží a mobilní základnou s koly. Cílem robotu je pohybovat se v běžném prostředí domácnosti nebo kanceláře, vyhledat požadovaný objekt a doručit jej na určené místo. Analýza tohoto scénáře (stejně jako u mnoha jiných úloh v oblasti robotiky) vedle k důležitým poznatkům, které se později promítly do návrhových principů ROS:

- „Aplikační úlohu lze rozložit na mnoho nezávislých subsystémů, jako je navigace, počítačové vidění, uchopení atd.
- Tyto subsystémy mohou být použity pro jiné úlohy, například pro provádění bezpečnostních hlídek, úklid, doručování pošty atd.
- S vhodnými vrstvami abstrakce hardwaru a geometrie lze naprostou většinu aplikačního softwaru může běžet na jakémkoli robotu.“ [28]

Tyto primární cíle systému ROS, lze vyjádřit pomocí základní zobrazovací koncepce, grafu. ROS systém je tvořen různými programy (nodes, které běží současně a komunikují mezi sebou pomocí zpráv. Pro znázornění těchto vztahů je ideální využít právě zmíněný graf (tedy ROS graf) (viz Obrázek 7). Kde jednotlivé uzly grafů představují běžící programy (ROS nodes) a jednotlivé šipky znázorňují datový tok (tedy odesílání a příjem zpráv mezi uzly). [28]

Samotné detaily grafu nejsou pro pochopení podstaty důležité, obrázek slouží jen k ilustraci obecného principu systému ROS jako sítě spolupracujících uzlů. Toto grafové

zobrazení je natolik užitečné, že se v rámci ROS běžně využívá nejen pro vizualizaci systému, ale i při samotném vývoji aplikací. [28]



Obrázek 7 Příklad ROS grafu [29]

Výhodou ROS je podpora široké škály robotických platforem od pozemních a leteckých robotů až po humanoidní a podvodní systémy. Taktéž nabízí širokou podporu senzorů, které se běžně využívají v robotice (jako IMU – inerciální měřící jednotka, GPS přijímače, kamery a laserové senzory). Pro tyto senzory jsou k dispozici již hotové ovladače, což usnadňuje jejich integraci do robotických aplikací. [27]

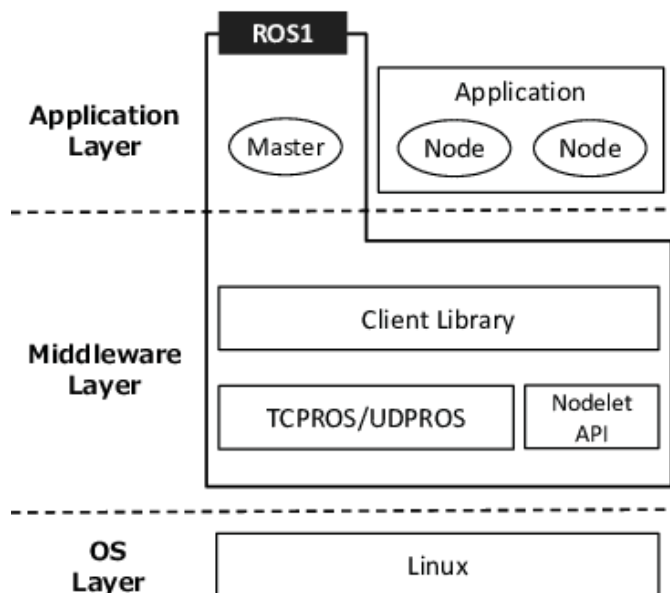
Jak již bylo zmíněno je ROS open-source software a je licencovaný pod BSD licenci. To znamená, že vývojáři mohou volně využívat existující kód pro různé robotické komponenty (např. pro plánování pohybu nebo systémy počítačového vidění). [27]

ROS je postaven na Ubuntu, a tudíž nenabízí hard real-time vlastnosti, ale podporuje soft real-time aplikace. Taktéž je kompatibilní s populárními open-source knihovnami, jako jsou OpenCV (pro počítačové vidění), Point Cloud Library (pro práci s 3D daty) nebo simulátor Gazebo. [27]

3.1 Architektura

ROS používá síť vzájemně propojených procesů, které spolu komunikují přímo mezi sebou (peer to peer). Tyto procesy mohou běžet i na různých počítačích a jsou volně propojeny pomocí komunikační infrastruktury ROS. Jednotlivé komponenty infrastruktury si vzájemně předávají data a zpracovávají je pomocí různých mechanismů. Mezi základní koncepty ROS implementace patří uzly (nodes), zprávy (messages), témata (topics) a služby (services). [27]

Na obrázku 8 lze vidět základní architekturu ROS systému, která je rozdělena do tří hlavních vrstev, vrstva operačního systému (OS layer), vrstva mezivrstvy (Middleware layer) a aplikační vrstva (Application layer). [30]



Obrázek 8 ROS architektura [30]

Operační systém tvoří základ, na kterém ROS běží. Middleware vrstva obsahuje komunikační protokoly (např. TCPROS a UDPROS) a klientské knihovny, které zajišťují přenos zpráv mezi jednotlivými komponentami systému. V této vrstvě se také nachází API pro práci s tzv. nodlety, které umožňují spouštění více uzlů ve stejném procesu. Aplikační vrstva reprezentuje konkrétní aplikace tvořené uzly, které mezi sebou komunikují prostřednictvím ROS Mastera. Ten slouží jako centrální koordinační bod, který zprostředkovává registraci a objevování jednotlivých uzlů. [30]

Takto koncipovaná struktura umožňuje flexibilní, modulární a distribuovaný návrh robotických systémů. Jednotlivé softwarové komponenty mohou běžet na různých zařízeních a komunikovat pomocí definovaných protokolů, což z ROS činí univerzální nástroj k vývoji robotických aplikací. [30]

3.1.1 Uzel (Node)

Uzlem (node) se v ROS rozumí samostatný softwarový proces, který vykonává specifické výpočetní úlohy. Každý uzel je tedy samostatná část programu (např. ovládání senzoru, část uživatelského rozhraní, algoritmus pro výpočet polohy robotu). ROS je navržen takovým

způsobem, aby podporoval jemně rozdělenou architekturu, což v praxi znamená, že se obvykle systém skládá z většího počtu menších uzlů. [27]

Použití uzlů přináší do systému několik výhod. Především posiluje modularitu a zvyšuje odolnost systému vůči chybám. Selhání jednoho uzlu neovlivní chod celého systému. Taktéž umožňuje skrýt detaily implementace jednotlivých částí, což přispívá k lepší přehlednosti a udržitelnosti systému. Každý uzel vystavuje pouze minimální rozhraní (API), přes které komunikuje s ostatními částmi systému. Pokud jednotlivé komponenty dodržují komunikační mechanismy ROS mohou poté být implementovány různými technologiemi. [27]

Každý uzel je v systému jednoznačně identifikován svým jménem, které je zároveň klíčovým prvkem v komunikaci s nástroji v rámci frameworku ROS. Jednotlivé uzly mají kromě jména také svůj typ, který je obdobný třídě v objektově orientovaném programování. Typ uzlu se definuje názvem spustitelného souboru a názvem balíčku do kterého patří. Tato struktura vyžaduje určité konvence v návrhu balíčků a pojmenování jejich typů. [27]

Zadáním příkazu *rostop* v příkazovém řádku lze získat informace o uzlech běžících v daném systému a umožňuje zobrazit aktivní uzly nebo zkontrolovat jejich stav. [27]

3.1.2 Zprávy (Messages)

Jednotlivé uzly mohou v ROS systému spolu komunikovat formou přímé (peer to peer) výměny zpráv. Zpráva představuje definovanou datovou strukturu, která je určena k přenosu informací mezi uzly. ROS podporuje základní datové typy, jako jsou celá čísla, desetinná čísla, logické hodnoty a jejich pole. Zprávy mohou být tvořeny taky složenými typy (např. jinými zprávami nebo jejich poli – do libovolné hloubky vnoření). [27]

Každá zpráva je definovaná v tzv. message souboru, což je jednoduchý textový soubor s příponou *msg*, který specifikuje datová pole zprávy. Příklad zprávy reprezentující bod v prostoru z balíčku *geometry_msgs* může vypadat následovně (viz Obrázek 9). [27]

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Obrázek 9 Příklad zprávy [27]

Typy zpráv jsou pojmenovány podle konvence „název_balíčku/zpráva“ (např. *geometry_msgs/Point*). ROS na základě těchto definic generuje zdrojový kód pro práci se zprávami (jak v jazyce Python, tak v C++). [27]

K zobrazení informací o definovaných zprávách, lze využít příkaz *rosmg* v příkazovém řádku. [27]

3.1.3 Témata (Topics)

Výměna dat mezi uzly probíhá prostřednictvím komunikačního vzoru *Publisher/Subscriber* (vydavatel/odběratel). Uzel, který odesílá data, publikuje zprávy na tzv. téma (topic), což je pojmenovaný kanál, přes který proudí určitý typ zpráv. Publikované zprávy musí uzel nejprve na dané téma oznámit (advertise), to znamená, že zveřejní název tématu a typ zprávy, kterou bude posílat. Jiný uzel, který má o daná data zájem, se může na toto téma přihlásit (subscribe), tím že požádá ROS Mastera o připojení a jakmile je navázáno, tak všechny zprávy publikované na daném tématu jsou doručovány tomuto uzlu. [27][28]

Ke každému tématu může být zároveň přihlášeno více vydavatelů i více odběratelů. Zároveň jeden uzel může publikovat i odebírat zprávy z více témat najednou. Komunikační model je navržen tak, aby byl nezávislá a oddělený – jednotliví vydavatelé a odběratelé o sobě navzájem vůbec nemusí vědět, což výrazně zjednodušuje návrh distribuovaných systémů. [27]

Každé takové téma je úzce svázáno s určitým typem zprávy. Aby mohla komunikace probíhat, musí jak publikující, tak odebírající uzel používat přesně stejný typ zprávy (při navázání spojení dochází k automatické kontrole shody typů zprávy). [27]

Přenos zpráv mezi uzly může v systému ROS probíhat přes dva různé transportní protokoly: TCPROS (na bázi TCP/IP) a UDPROS (na bázi UDP). Ve výchozím nastavení se používá TCPROS, který garantuje spolehlivý přenos dat, narozdíl od UDPROS, který poskytuje nízkou latenci a nižší spolehlivost, protože nezaručuje doručení zprávy ani jejich pořadí doručení. Z těchto důvodů se UDPROS využívá pouze v teleoperační aplikaci. [27][31]

Pro sledování a analýzu témat slouží nástroj příkazové řádky „rostopic“, díky kterému lze zobrazit informace o jednotlivých tématech, včetně statistik výkonnosti jako např:

- „perioda přenosu zpráv (průměrná, maximální hodnota, směrodatná odchylka),
- stáří zpráv na základě časové známky v hlavičce (průměr, maximum, odchylka),

- *počet zahozených zpráv,*
- *přenesený objem dat (v bajtech).*“ [27]

3.1.4 Služby (Services)

Zatímco témata slouží v ROS pro jednosměrný a nepřetržitý přenos dat, tak služby slouží k zajištění obousměrné komunikace. Tedy odesílání požadavku a následné přijetí odpovědi. Tento mechanismus zvaný služby, odpovídá principu synchronního vzdáleného volání procedur (Remote Procedure Call – RPC), to umožňuje jednomu uzlu vyvolat funkci běžící v jiném uzlu. [27][28]

Služby umožňují interakci typu request/reply (požadavek/odpověď), uzly se tedy dělí na dvě role server a klient. Server je uzel, který službu poskytuje, definuje zpětné volání pro zpracování požadavku a službu zveřejní. Klient je uzel, který službu volá, přistupuje ke službě prostřednictvím lokálního rozhraní (proxy). [28]

Definice služby, tedy vstupy a výstupy této funkce se definují podobně jako v případě zpráv v tzv. service souboru, což je jednoduchý textový soubor s koncovkou srv, ze kterého ROS automaticky vygeneruje potřebné rozhraní (tzv. stub) pro využití služby jak na straně klienta, tak na straně serveru. [27][28]

Tento způsob komunikace se hodí zejména tehdy, když uzly potřebují jednorázově získat specifické informace, nebo když je potřeba provést konkrétní akci a ověřit její výsledek s předem odhadnutelnou dobou trvání (např. výpočty, které se mají delegovat na jiný počítač, nebo diskrétní akce robotu jako zapnutí senzoru nebo pořízení snímku ve vysokém rozlišení). [27][28]

Pomocí příkazů *rossrv* a *rosservice* lze zobrazit informace ohledně služeb. [27]

3.1.5 Akce (Action)

V předchozí části byly popsány služby, které se v systému používají k synchronní komunikaci typu požadavek/odpověď. Tento model je vhodný, když je potřeba rychle získat nebo nastavit hodnotu (např. dotaz stavu senzoru nebo změna konfigurace). Služby však nejsou ideální pro zpracování složitějších nebo časově náročnějších úloh. [28]

Nastane-li situace, kdy je úkolem robotu dojet na vzdálené místo, tak tento proces může trvat několik sekund, minut nebo i déle v závislosti na aktuální situaci prostředí (např. výskyt překážky apod.). Použití služby by v tomto případě vedlo k neefektivnímu chování. Volající

program odešle požadavek a poté je nucen čekat na odpověď bez jakékoli zpětné vazby, aniž by bylo možné sledovat průběh úlohy, upravit cíl nebo úlohu zrušit. [28]

Právě akce jsou určeny k asynchronnímu řízení chování robotu, které je časově nezávislé a orientované na dosažení cíle. Podobně jako služby využívají akce žádost (goal) a odpověď (answer), ale zároveň přidávají i průběžnou zpětnou vazbu (feedback). Navíc je také možno zrušení dané úlohy během jejího běhu (cancel). [28]

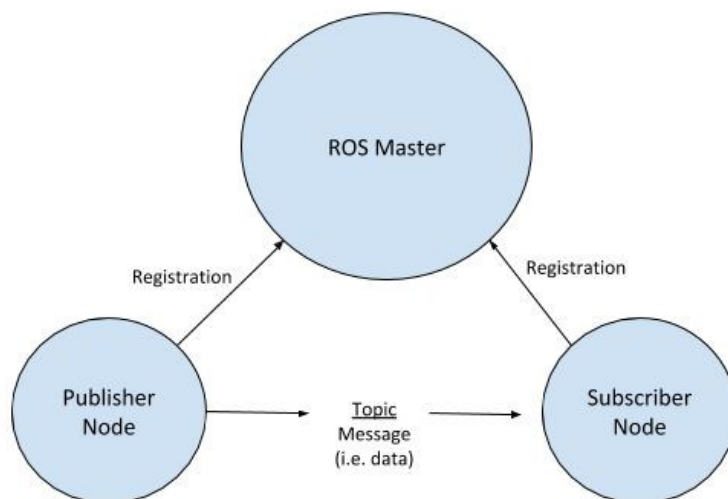
Z technického hlediska je akce implementována jako vyšší protokol nad sadou témat, typicky zahrnující:

- Goal – zadání cíle,
- Feedback – informace o průběhu úlohy,
- Result – finální výsledek,
- Cancel – zrušení cíle. [28]

Použití akce umožňuje programu odeslat robotu cíl, pokračovat v jiných činnostech a průběžně přijímat informace o tom, jak robot plní úkol. Jedním z nejčastějších praktických příkladů použití akcí v ROS je uzel *move_base*, který slouží k plánování a řízení pohybu mobilního robotu. Tento uzel přijímá cílové pozice pomocí akčního rozhraní, kde každé zadání cíle je reprezentováno jako akce. *move_base* průběžně informuje o průběhu plnění cíle (např. pozice robotu vzhledem k cílovému bodu), a pokud je potřeba, může volající uzel požadavek zrušit. Díky tomu je možné reagovat na dynamické změny v prostředí a zajišťovat bezpečný pohyb robotu. [28]

3.1.6 Master

ROS Master je klíčovou součástí systému, která zajišťuje registraci a vyhledávání uzlů. Bez něj by jednotlivé uzly neměly možnost navzájem se objevit a navázat mezi sebou komunikaci. Úlohou Masteru je tedy zprostředkovat inicializaci spojení mezi uzly. Jakmile se dva uzly prostřednictvím Masteru navzájem najdou, probíhá jejich další komunikace přímo (peer to peer), bez další účasti Masteru (viz Obrázek 10). [27][28]



Obrázek 10 Registrace jednotlivých uzlů pomocí ROS Master [32]

ROS Master poskytuje API založené na protokolu XML-RPC, které uzly využívají k registraci a dotazování na informace o ostatních uzlech. Tato rozhraní obsahuje funkce jako *registerService*, *registerSubscriber* nebo *registerPublisher*, které slouží pro registraci služeb, odběratelů a vydavatelů. [27]

Tyto nízkoúrovňové operace jsou zcela skryty za klientskými knihovnami (*roscpp* pro C++, *rospy* pro Python), takže běžný uživatel s nimi nepřichází do přímého kontaktu. Práce s Masterem je tedy z pohledu vývojáře plně automatizovaná a již nevyžaduje implementaci vlastního rozhraní. [27]

ROS Master je společně s Parameter Serverem vyvolán příkazem *roscore* v příkazové řádce. [28]

3.1.7 Parameter Server

Parameter Server je součástí systému ROS, která slouží jako sdílený klíč-hodnota uložisko, dostupné skrz síťové API. Jedná se o vícerozměrný slovník, který umožňuje jednotlivým uzlům ukládat a načítat parametry za běhu systému. [27]

Tento server je určen především pro práci s neměnnými nebo pomalu se měnícími konfiguračními daty, která nejsou binárního typu (typickými příklady jsou frekvence aktualizací, cesty k souborům atd.). Není optimalizován pro vysoký výkon, a proto není vhodný pro častě aktualizovaná data nebo velká datová pole. [27]

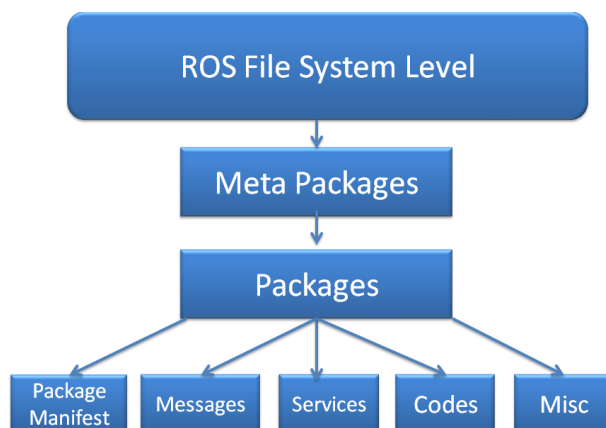
Výhodou je, že je globálně přístupný, tedy jeho obsah lze snadno prohlížet a upravovat pomocí nástrojů a skriptů. To usnadňuje kontrolu aktuálního konfiguračního stavu celého

systemu. Z technického hlediska je implementován pomocí XML-RPC (jako ROS Master) a běží jako součást Ros Masteru. [27]

3.2 Souborový systém (File System)

ROS není pouze vývojový framework, hovoří se o něm jako o metaoperačním systému, protože kromě knihoven a nástrojů poskytuje také funkcionality typické pro skutečný operační systém. Mezi tyto funkce patří například abstrakce hardwaru, správa balíčků nebo kompletní nástroj pro vývoj. Stejně jako běžný operační systém má i ROS specifickou strukturu souborového systému, která určuje, jak jsou jednotlivé části projektu uloženy na disku. Tato struktura usnadňuje organizaci kódu, sdílení balíčků a jejich opětovné použití.

Následující schéma znázorňuje, jak jsou soubory a adresáře v prostředí ROS uspořádány (viz Obrázek 11). [31]



Obrázek 11 ROS souborový systém [31]

Níže jsou vysvětleny jednotlivé bloky v souborové systému ROS projektů:

- **Balíčky (Packages):** Základní jednotkou v systému ROS je balíček. Každý balíček může obsahovat uzly (nodes), knihovny, konfigurační soubory, skripty, zprávy, služby a další soubory. Balíček je zároveň nejmenší jednotkou, která se v systému ROS kompiluje a distribuuje.
- **Manifest balíčku (package.xml):** Každý balíček obsahuje manifest, což je soubor package.xml, který definuje metadata o balíčku – jeho název, verzi, autora, licenci, závislosti na jiných balíčcích a další informace potřebné při sestavení a instalaci.
- **Metabalíčky (Metapackages):** Metabalíček představuje virtuální balíček, který sdružuje více souvisejících balíčků do jedné logické skupiny. Sám o sobě neobsahuje žádný zdrojový kód ani spustitelné soubory – jeho cílem je pouze organizace.

- Manifest metabalíčku: Funguje podobně jako manifest běžného balíčku, ale navíc definuje závislosti na skupině jiných balíčků (například pomocí tagu `<run_depend>`) a často obsahuje exportní direktivy.
- Zprávy (.msg): Vlastní typy zpráv, které se používají pro komunikaci mezi uzly, lze definovat ve složce msg uvnitř balíčku. Každý typ zprávy je uložen jako soubor s příponou .msg (např. *my_package/msg/MyMessageType.msg*). [31]
- Služby (.srv): Podobně jako zprávy lze ve složce srv definovat vlastní služby, které obsahují dvojici zpráv – požadavek a odpověď. Každý typ služby je uložen jako .srv soubor (např. *my_package/srv/MyServiceType.srv*).
- Repository (repozitáře): Většina ROS balíčků je spravována pomocí verzovacích systémů (VCS), jako jsou Git, Subversion (SVN) nebo Mercurial (hg). Repoziťář představuje sadu souborů verzovanou jako celek, a často obsahuje více balíčků.
- Misc (ostatní/různé): tato složka reprezentuje další podpůrné soubory jako jsou konfigurační soubory, dokumentace, skripty, testovací soubory, datové soubory, vizualizační konfigurace a další soubory potřebné pro běh nebo vývoj.

3.2.1 Catkin

Catkin je build systém využívaný v ROS. Jedná se o sadu nástrojů, která ROS využívá ke generování spustitelných programů, knihoven, skriptů a rozhraní, které mohou být dále využívány v jiných částech systému. [28]

Pokud je vývoj ROS systému prováděn v jazyce C++, je potřeba porozumět principům fungování catkinu podrobněji, protože hraje klíčovou roli při konfiguraci a kompilaci balíčků. Při vývoji v Pythonu je možné se mnoha detailům vyhnout, protože Python uzly obvykle nevyžadují překlad. Přesto je nutné do určité míry s catkin pracovat, zejména při nastavování struktury balíčků a závislostí. [28]

Catkin je postaven na systému CMake, což je běžně používaný open-source nástroj pro sestavení softwaru. Nad rámec základního CMake workflow přidává catkin vlastní CMake makra a skripty v jazyce Python, které poskytují rozšířenou funkcionalitu potřebnou pro vývoj ROS. [28]

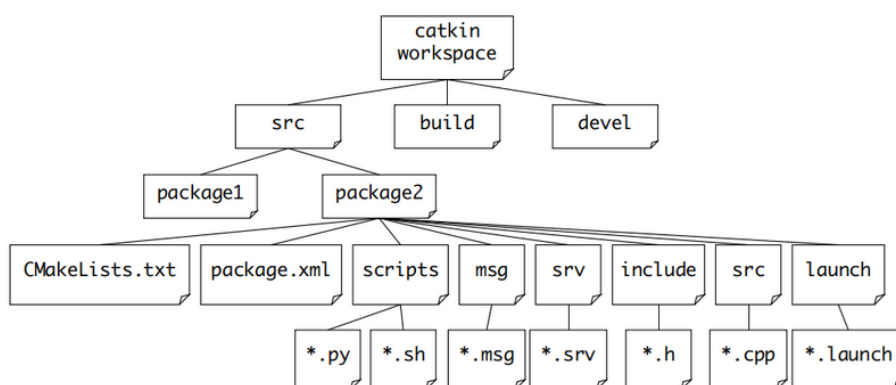
Pro hlubší pochopení je výhodou základní znalost CMake. Pro běžného uživatele ROS je zpravidla dostačující vědět, že každý balíček musí obsahovat dva důležité konfigurační

sobory *CMakeLists.txt* a *package.xml*, kde se definují závislosti, struktura a specifikace daného balíčku, aby vše správně fungovalo. [28]

3.2.2 Pracovní prostředí (Workspace)

V průběhu vývoje se používají různé nástroje balíku catkin, které automaticky generují potřebné složky a konfigurační soubory pro sestavení tzv. workspace (pracovní prostředí). Vytvoření workspace umožňuje organizaci projektu (viz Obrázek 12) a obvykle obsahuje složky:

- *src/*: obsahuje všechny ROS balíčky (stažené i vlastní) používané v daném workspace.
- *build/*: zde CMake ukládá meziprodukty kompilace (např. sestavovací programy, knihovny atd.). Tato složka je využívána hlavně při práci v jazyce C++.
- *devel/*: obsahuje množství souborů, přičemž nejdůležitějšími z nich jsou tzv. setup soubory (např. *setup.bash*, *setup.sh*, *setup.zsh*). Spuštěním těchto souborů se nakonfiguruje prostředí systému tak, aby pracovalo s tímto konkrétním workspace. Díky tomu lze používat kód a balíčky, které se ve workspace nacházejí. [28]



Obrázek 12 Příklad ROS workspace [33]

Workspace je tedy sada složek, do které se ukládají všechny související komponenty ROS projektu. Workspace musí být uživatelsky vytvořen a aktivně spravován. Uživatel může mít v systému více různých pracovních ROS workspace (např. pro různé projekty nebo účely).

V praxi je možné aktivně pracovat pouze v jednom workspace současně, protože pouze tento používaný workspace je „viditelný“. To znamená, že pouze tento workspace bude

používán při spouštění uzlů, načítání konfigurací nebo při ladění. Díky této modularitě je možné pohodlně spravovat i rozsáhlejší projekty nebo paralelně vyvíjet více částí systému bez jejich vzájemného ovlivnění. [28]

3.3 Klíčové nástroje a balíčky v mobilní robotice

Robot Operating System (ROS) není jen komunikační framework, ale také rozsáhlý ekosystém softwarových nástrojů a balíčků, který výrazně usnadňuje vývoj, testování a nasazení robotických aplikací. Tyto nástroje poskytují možnosti pro vizualizaci dat, simulaci chování robotu, správu souřadnicového systému nebo nahrávání a přehrávání meziuzlové komunikace. Kromě toho ROS obsahuje mnoho hotových softwarových balíčků, které implementují pokročilé algoritmy pro lokalizaci, mapování, plánování pohybu nebo řízení robotů. V následujícím přehledu jsou představeny nejpoužívanější nástroje používané k vývoji a testování mobilních robotických systémů v ROS:

- RVIZ (ROS Visualization) je univerzální 3D vizualizační nástroj používaný v systému ROS. Umožňuje zobrazovat různé datové toky ze senzorů a uzlů, jako jsou obrazy z kamer, mračna bodů, trajektorie nebo souřadnicové systémy. Díky tomu je velmi užitečný při ladění, teleoperaci a vývoji robotických aplikací. Vizualizace probíhá vždy vzhledem ke zvolenému referenčnímu rámci, přičemž konfigurace prostředí RVIZ může být uložena pro opakované použití. [28]
- GAZEBO je vykonný 3D simulační nástroj úzce integrovaný s prostředím ROS. Umožňuje realistickou simulaci robotických systémů včetně senzorů, fyziky a pohybu v prostoru. Na rozdíl od jednodušších 2D simulátorů (např. Stage) je Gazebo vhodný i pro složitější scénáře, jako je manipulace s objekty, interakce s okolím nebo provoz v nestrukturovaném prostředí. Využívá fyzikální enginy (např. ODE nebo Bullet) pro simulaci tuhých těles a umožňuje obousměrnou komunikaci s ROS pomocí balíčku *gazebo_ros*. Díky tomu je možné vývojový kód testovaný v simulaci použít i na reálném robotu bez nutnosti zásadních změn. [28]
- RQT (ROS QT) je modulární grafické rozhraní postavené na frameworku QT. Umožňuje implementaci různých grafických nástrojů ve formě pluginů. Umožňuje spouštět všechny existující GUI (grafické uživatelské rozhraní) nástroje jako okna v rámci jednoho integrovaného prostředí, což usnadňuje správu více oken na obrazovce současně. Mezi nejčastěji používané pluginy patří *rqt_graph* (vizualizace uzlů a témat), *rqt_plot* (grafy z ROS témat) a *rqt_reconfigure* (dynamické

nastavování parametrů). Framework je navržen jako snadno rozšiřitelný, takže vývojáři si mohou vytvářet vlastní pluginy podle konkrétních potřeb. [34]

- Rosbag je nástroj, který umožňuje zaznamenávat a následně přehrávat zprávy mezi uzly. Tento nástroj je velmi užitečný při ladění a vývoji algoritmů, protože umožňuje opakovaně testovat kód se stejnými vstupními daty bez nutnosti mít stále k dispozici reálného robotu. Pomocí rosbag lze zaznamenat senzorová data z reálného provozu a následně je využít k offline analýze nebo simulaci. To výrazně zjednodušuje vývoj a odhalování chyb v robotickém softwaru. [28]
- Roslaunch je nástrojem příkazového řádku, který umožňuje spouštět více uzlů najednou. Místo ručního spouštění každého uzlu pomocí *roslaunch* lze pomocí tzv. launch souborů (s příponou *.launch*) definovat celou sadu uzlů, jejich parametry další nastavení. *Roslaunch* následně tyto uzly spustí a monitoruje. Tento přístup je nezbytný při práci s komplexními systémy, kde je potřeba mít přehlednou a automatizovanou konfiguraci více komponent najednou. [28]

ROS poskytuje také oficiální funkční balíčky, které implementují klíčové algoritmy nezbytné pro provoz mobilních robotů. Tyto balíčky poskytují hotová řešení pro úlohy, jako je lokalizace, mapování, plánování pohybu, řízení atd. Následuje přehled nejdůležitějších ROS balíčků využívaných k vývoji mobilních robotických systémů v ROS:

- Navigation stack (zkráceně *nav stack*) je jeden z nejdůležitějších balíčků v systému ROS, který umožňuje mobilnímu robotu autonomní pohyb k cílové pozici bez kolizí. Tento systém kombinuje data z mapy, lokalizačního systému, senzorů a odometrie a na jejich základě plánuje optimální trasu a následně se jí snaží sledovat. Navigační cíl je zadán pomocí akce *MoveBaseGoal*, která definuje cílovou pozici a orientaci v mapě. Globální plánovač vytvoří cestu k cíli, zatímco lokální plánovač upravuje trajektorii v reálném čase na základě detekce překážek. Pokud dojde k nečekané překážce, systém je schopen trasu přeplánovat. [28]
- Slam_gmapping je ROS balíček z projektu gmapping, který umožňuje vytváření 2D map prostředí pomocí simultánní lokalizace a mapování (SLAM). Využívá algoritmus GMapping, založený na Rao-Blackwellized particle filteru, který odhaduje pravděpodobnou pozici robotu na základě senzorických dat a dosud vytvořené mapy. Mapy vzniklé tímto způsobem mohou obsahovat různé nepřesnosti způsobené chybami měření, stíny v datech nebo překážkami v prostředí, přesto však

zůstávají dostatečně použitelné pro navigaci. *slam_gmapping* je často používán společně s nástrojem *roscbag*, který umožňuje zaznamenat senzorová data a později je opakovaně přehrávat, což značně usnadňuje ladění parametrů mapování bez nutnosti neustálého pohybu robotu v reálném prostředí. [28]

- Tf (transform) je balíček v systému ROS, který slouží k řízení a správě souřadnicových systémů a jejich vzájemných transformací. Každý uzel může publikovat informace o transformaci mezi dvěma rámci (souřadnicových systémů), zatímco ostatní uzly tyto informace odebírají a získávají tak úplný obraz o aktuální konfiguraci robotu. Transformace se publikují pomocí zpráv typu *tfMessage* na téma */tf* a zahrnují názvy rámců, jejich vzájemnou polohu, orientaci a čas měření. Díky tomu lze správně pracovat se senzory, které poskytují data v různých časech a rámcích. Balíček tf poskytuje také knihovny pro práci s těmito daty – například *tfListener*, který uchovává historii transformací a umožňuje dotazování na vztahy mezi rámci v různých časech. Systém tf je velmi výkonný, ale může být i komplexní, a proto ROS obsahuje také nástroje pro jeho ladění a vizualizaci, jako je zobrazení transformací v RViz nebo výpisy na příkazové řádce. [28]
- Robot_state_publisher je balíček v ROS a slouží k publikování stavů robotu do systému transformací tf. Na základě aktuálních úhlů kloubů robotu a jeho kinematického modelu, definovaného v URDF (Unified Robot Description Format), tento balíček vypočítává a publikuje 3D pozice jednotlivých částí robotu. To umožňuje ostatním komponentám v systému ROS získávat aktuální informace o prostorovém uspořádání robotu, což je klíčové pro úlohy jako je plánování pohybu, vizualizace nebo simulace. I v mobilní robotice je důležitý pro správné zobrazování a výpočty v rámci systému tf, zejména pokud má robot pohyblivé senzory nebo je potřeba přesná vizualizace jeho struktury v nástrojích jako RViz. [35]
- Map_server je ROS balíček, který slouží ke čtení a publikování 2D map prostředí pro účely lokalizace a navigace. Načítá mapu ve formátu obrazového souboru (např. .pgm) spolu s konfiguračním souborem .yaml, který definuje měřítko a souřadnice. map_server následně publikuje mapu jako zprávu nav_msgs/OccupancyGrid na tématu /map, což umožňuje ostatním uzlům (např. amcl nebo move_base) využívat tuto mapu pro plánování pohybu nebo lokalizaci. Je to základní komponenta pro navigaci v předem známém prostředí. [36]

3.4 ROS 1 vs ROS 2

ROS 1 vytvořil Willow Garage roku 2007 a rychle si získal oblibu mezi vývojáři open-source robotiky po celém světě. S postupem času však vývojový tým identifikoval řadu klíčových nedostatků, jako je absence podpory pro reálný čas, nedostatečné zajištění bezpečnosti nebo složitá certifikace pro průmyslové nasazení. Implementace těchto funkcí by vyžadovala zásadní změny v architektuře ROS 1, které by narušily stabilitu celého systému. Proto byl vytvořen ROS 2 jako zcela nová generace systému, vyvinutý od začátku s cílem splnit i požadavky průmyslových aplikací, včetně podpory real-time systémů, bezpečnosti a distribuované architektury. [37]

Od vydání první LTS (dlouhodobá podpora) verze Foxy Fitzroy (2020) je ROS 2 aktivně vyvíjen s každoročními stabilními verzemi. Naproti tomu ROS 1 Noetic, vydaný rovněž v roce 2020, je poslední verzí systému ROS 1, s plánovaným ukončením podpory v roce 2025. Jeho hlavním cílem bylo zajistit přechod na Python 3 a poskytnout stabilní platformu pro projekty, které z různých důvodů nemohou okamžitě migrovat na ROS 2. [37]

ROS 2 přináší oproti svému předchůdci, ROS 1, několik klíčových vylepšení:

- Odstranění centrálního ROS Master: ROS 1 závisel na centrálním uzlu zvaném ROS Master pro řízení komunikace mezi uzly. V ROS 2 byl tento centrální bod odstraněn, což zvyšuje odolnost systému vůči selhání a eliminuje jediný bod selhání.
- Podpora reálného času (real-time) ROS 2 umožňuje psaní aplikací, které splňují požadavky na deterministické zpoždění. Díky použití DDS a vhodnému plánování vláken může být ROS 2 použit v soft real-time i hard real-time systémech.
- Multi-threaded executor (vícevláknové zpracování): Na rozdíl od ROS 1, kde každý uzel běží v jednom vlákne, ROS 2 umožňuje provádění callbacků paralelně, což zlepšuje výkon při práci se senzory, službami a plánováním současně.
- Podpora více platforem: Zatímco ROS 1 byl primárně určen pro Linux, ROS 2 je navržen tak, aby byl kompatibilní s různými operačními systémy, včetně Windows a macOS. Tato multiplatformní podpora usnadňuje integraci ROS 2 do různých prostředí a rozšiřuje jeho použitelnost.

- Zlepšená správa životního cyklu uzlů: ROS 2 umožňuje lepší kontrolu nad stavem uzlů díky stavovému automatu uzlů (např. stavy inactive, active, shutdown), který umožňuje lepší řízení nasazení a správy uzlů.
- Jednotné jádro pro klientské knihovny: v ROS 2 jsou všechny klientské knihovny postaveny nad společným jádrem (rcl), což zajišťuje jednotné chování napříč programovacími jazyky. Díky tomu je vývoj aplikací konzistentní bez ohledu na zvolený jazyk a je usnadněna i tvorba nových knihoven pro další jazyky. [38][39][40]

Tyto změny činí ROS 2 robustnějším a flexibilnějším rámcem pro vývoj robotických aplikací, přičemž řeší některé z hlavních omezení ROS 1 a přináší nové funkce a možnosti pro moderní robotické systémy.

3.4.1 Využití ROS v praxi

Robot Operating System (ROS) nachází široké uplatnění v různých oblastech, od akademického výzkumu až po průmyslové aplikace. Při rozhodování mezi použitím ROS 1 a ROS 2 je důležité zvážit specifické požadavky projektu a dostupnost podpory.

ROS 1 je dlouhodobě preferovanou volbou v akademických a výzkumných kruzích díky své rozsáhlé komunitě a množství dostupných balíčků. Jeho stabilita a široká dokumentace usnadňují rychlý vývoj prototypů a experimentálních aplikací. [37]

ROS 2 byl navržen již s ohledem na požadavky průmyslu, jako jsou real-time zpracování, bezpečnost a škálovatelnost. Díky podpoře pro více platform, včetně Windows a macOS, a použití DDS (Data Distribution Service) pro komunikaci, je ROS 2 vhodnější pro komerční nasazení v průmyslu a aplikace vyžadující vysokou spolehlivost. [39]

Pro organizace s rozsáhlým kódem v ROS 1 může být přechod na ROS 2 náročný. V takových případech je možné využít balíček *ros1_bridge*, který umožňuje komunikaci mezi ROS 1 a ROS 2. To umožňuje postupný přechod a testování nových funkcionalit v ROS 2, aniž by bylo nutné okamžitě opustit stávající infrastrukturu. [37]

Pro nové projekty je obecně doporučeno začít s ROS 2, zejména s ohledem na budoucí ukončení podpory ROS 1 a výhody, které ROS 2 přináší v oblasti výkonu a flexibility. Nicméně finální rozhodnutí, na jaké verzi systému ROS bude robotický systém postaven, by mělo vycházet z konkrétních požadavků na projekt, dostupnosti potřebných balíčků a plánovaného životního cyklu aplikace. [37]

4 LOKALIZACE A MAPOVÁNÍ V MOBILNÍ ROBOTICE

Autonomní mobilní roboty musí být schopny nejen vnímat své okolí, ale také určit svou polohu a vytvořit mapu prostředí, ve kterém se pohybují. Tyto dvě úlohy (lokalizace a mapování) tvoří základ každého navigačního systému a jsou klíčové pro plánování trajektorie i bezpečný pohyb v dynamickém prostředí [41]

Lokalizace znamená odhad aktuální pozice robotu vůči známé mapě, zatímco mapování se zabývá tvorbou této mapy na základě senzorických dat. Obě úlohy se často řeší současně v rámci tzv. SLAMu (Simultaneous Localization and Mapping) – tedy v situaci, kdy robot nezná ani svou polohu, ani prostředí a musí obojí odvodit za běhu. [41]

Vzhledem k jejich důležitosti byly vyvinuty různé teoretické přístupy i praktické implementace. Tato kapitola přehledně shrnuje základní principy lokalizace, mapování a SLAMu se zaměřením na mobilní robotiku a její realizaci v prostředí ROS. [42]

4.1 Lokalizace a její metody

Lokalizace je klíčová schopnost mobilního robotu, která mu umožňuje určit svou pozici a orientaci v prostředí. Typicky se jedná o určení souřadnic a natočení robotu vůči známému referenčnímu rámci, například mapě. Lokalizace vyjadřuje vztah mezi robotem a mapou – tedy mezi jeho lokálním a globálním souřadnicovým systémem (lokálním – je spjat s robotem a globálním – je spjat s prostředím). Tato informace je zásadní pro navigaci, vyhýbání se překážkám i plánování pohybu. [41]

Určit pozici a orientaci robotu lze pomocí odometrie, což je metoda, která využívá data ze senzorů pohybu (např. optické, magnetické enkodéry) nebo pomocí senzorů využívaných k inerciální navigaci (akcelometr, gyroskop), které měří zrychlení. Avšak tyto typy lokalizace jsou velmi nepřesné. Nejistota v měření, šum, skluz kol nebo neznámé vlastnosti prostředí vedou k tomu, že robot musí pracovat s pravděpodobnostním odhadem své polohy. Právě proto se v moderní robotice využívají pravděpodobnostní metody, které reprezentují víru (belief) o poloze jako rozdělení pravděpodobnosti napříč všemi možnými stavy. [43] [41]

Všechny lokalizační úlohy nejsou stejně náročné. Pro pochopení složitosti lokalizace je užitečné je rozdělit podle klíčových hledisek. Lokalizační problémy určené na základě počátečních znalostí o poloze, lze rozdělit do 3 kategorií:

- Sledování pozice: V tomto případě robot zná svou výchozí pozici a cílem je pouze průběžně sledovat změny jeho polohy, a to s ohledem na možné odchylky způsobené šumem v řízení. Protože odchylky bývají malé, často se používá přístup založený na unimodálním rozdělení pravděpodobnosti (např. Gaussovské). Tento typ lokalizace je označován jako lokální, protože chyba je malá a omezená na okolí skutečné pozice robotu.
- Globální lokalizace: Zde robot nezná svou počáteční polohu (byl například náhodně umístěn do neznámé části prostředí). Lokalizační algoritmus tedy nemůže předpokládat, že odchylka od skutečné polohy je malá. V tomto případě jsou unimodální rozdělení obvykle nevhodná, protože robot může být kdekoliv.
- Problém uneseného robotu: Robot během provozu „náhle zmizí“ z aktuálního místa a objeví se jinde (např. je ručně přenesen nebo ztratí orientaci). Klíčovým problémem zde je, že robot si myslí, že ví, kde je ale ve skutečnosti se nachází jinde. Zatímco v klasické globální lokalizaci robot ví, že neví, kde je, zde žije v omylu. Schopnost algoritmu se z takových situací zotavit je zásadní pro skutečně autonomní roboty. [41]

Další rozdělení můžeme udělat podle dynamiky prostředí:

- Statické prostředí je takové, kde se mění pouze poloha robotu a všechny ostatní objekty v rámci prostředí zůstávají na svém místě. Tento typ prostředí je velmi dobrý pro lokalizaci, protože lze použít efektivní pravděpodobnostní odhady s jednoduchým modelem prostředí.
- Dynamické prostředí obsahuje objekty, které mění svou polohu nebo stav v čase (např. lidé, nábytek, otevírající se dveře). V reálném světě je tento typ prostředí velmi častý. [41]

Třetím typem rozdělení je podle zásahu lokalizačního algoritmu do řízení robotu:

- Pasivní lokalizace znamená, že lokalizační modul pouze sleduje pohyb robotu, aniž by do něj sám zasahoval. Lokalizace se zde děje pouze na základě pozorování, bez přímého ovlivnění trajektorie.
- Robot je řízen aktivní lokalizací za účelem minimalizace nejistoty v odhadu své polohy, nebo vyhnutí se riziku, které by mohlo nastat, když by nepřesně lokalizovaný robot se mohl dostat do nebezpečné situace. [41]

Posledním typem lokalizačních úloh je podle počtu zapojených robotů v úloze lokalizace. Lokalizace samostatného robotu a lokalizace několik robotů. V druhém případě se jedná o velmi náročnou úlohu. [41]

4.1.1 Bayesův filtr

Základním a nejuniverzálnějším algoritmem pro výpočet pravděpodobnostního odhadu stavu je Bayesův filtr. Výsledkem každého kroku je tzv. „belief“ v daném čase t (ozn. $bel(x_t)$) – pravděpodobnostní rozdělení nad možnými stavy robotu v daném čase. Tento algoritmus umožňuje postupně, rekurzivně, aktualizovat odhad stavu robotu $bel(x_t)$ v čase t , který je vypočten z $bel(x_{t-1})$, společně s nejaktuálnějším vstupem řízení u_t a nejaktuálnějším měřením z_t ze senzorů (viz Obrázek 13). [41]

```

Algorithm Bayes_filter( $bel(x_{t-1}), u_t, z_t$ ):
  for all  $x_t$  do
     $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx$ 
     $bel(x_t) = \eta p(z_t | x_t) \overline{bel}(x_t)$ 
  endfor
  return  $bel(x_t)$ 

```

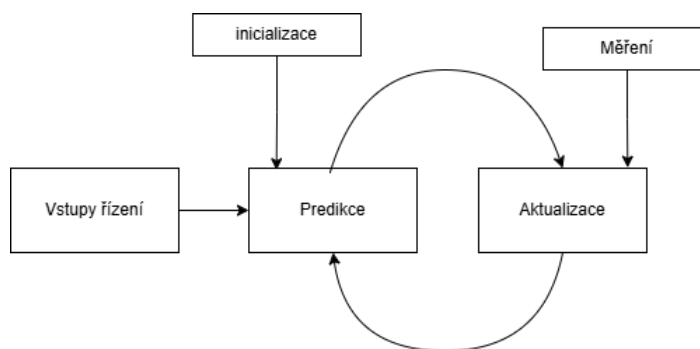
Obrázek 13 Obecný algoritmus pro Bayesův filtr [41]

Bayesův filtr pracuje ve dvou krocích:

- **Predikce** – V tomto kroku filtr pracuje s informací o řízení u_t (např. robot se posunul dopředu). Na základě předchozího odhadu stavu $bel(x_{t-1})$ a provedeného řízení u_t vypočítá nový odhad aktuálního stavu x_t . Tento výpočet se provádí integrováním (sečtením) všech možných předchozích stavů a zohlední se, s jakou pravděpodobností mohlo dané řízení způsobit přechod mezi nimi a novým stavem.
- **Aktualizace měřením** – Následně se odhad stavu x_t zpřesní na základě aktuálního měření senzorů z_t . Pro každý možný stav se vypočítá, jak pravděpodobné je, že by se v tomto stavu mohlo objevit právě naměřené z_t . tento výsledek se vynásobí predikovaným odhadem, a nakonec se celý výsledek normalizuje (aby se součet všech pravděpodobností rovnal 1). Tím vznikne odhad aktuálního stavu $bel(x_t)$. [41]

Bayesův filtr ve své obecné formě předpokládá výpočet dvou kroků – predikce a aktualizace měření – které vyžadují provedení integrace a násobení pravděpodobnostních rozdělení (viz Obrázek 14). Tyto výpočty jsou však v praxi proveditelné pouze v jednoduchých případech. Proto se pro reálné aplikace často volí omezení na konečný počet stavů (např. mřížka prostředí), nebo se používají aproximace jako Kalmanovy nebo

částicové filtry, které Bayesův filtr numericky přibližují a umožňují efektivní výpočet i v nelineárních nebo spojitých stavových prostorech. [41]



Obrázek 14 Obecné schéma Bayesova filtru [44]

4.1.2 Markovova lokalizace

Metody přímo vychází z obecného Bayesova filtru a jedná se o nejjednodušší aplikaci Bayesova filtru na problém lokalizace. Oproti základnímu algoritmu (viz Obrázek 13) má navíc vstupní parametr mapu prostředí (m). Tato mapa je klíčovou součástí sensor modelu, který popisuje pravděpodobnost, s jakou by robot v dané poloze a známém prostředí pozoroval určité měření. [41]

Tento algoritmus je natolik flexibilní, že dokáže řešit globální lokalizaci, sledování polohy i tzv. problém uneseného robotu. To vše za předpokladu, že prostředí je statické (nemění se během operace robotu).

4.1.3 Rozšířený Kalmanův filtr (EKF)

Rozšířený Kalmanův filtr je specifický případ Markovovy lokalizace. Na rozdíl od obecného Bayesovského přístupu, který pracuje s pravděpodobnostní distribucí, EKF reprezentuje víru $bel(x)$ o aktuálním stavu robotu pomocí dvou statistických momentů – střední hodnoty a kovarianční matice, která vyjadřuje nejistotu. [41]

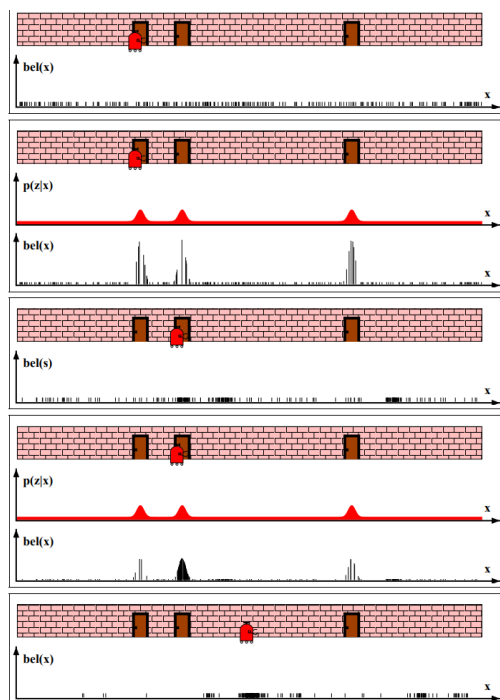
Tento algoritmus předpokládá, že prostředí je reprezentováno sadou výrazných a známých prvků (landmarků), vůči nimž robot provádí měření. V každém časovém kroku získává vektor pozorování (např. vzdálenost a úhel k jednotlivým orientačním bodům). V první variantě algoritmu se předpokládá, že všechny prvky jsou jednoznačně identifikovatelné. Později se přechází k obecnější variantě, kde identita landmarku není předem známa, robot místo toho pozoruje tzv. signaturu landmarku (např. barvu, tvar). [41]

4.1.4 Monte Carlo Localization (MCL)

MCL je jeden z nejpoužívanějších algoritmů pro lokalizaci. Je použitelný jak pro lokální, tak i globální lokalizační problémy. Tato metoda má pro svou jednoduchou implementaci vysokou spolehlivost. Základní princip MCL je postaven na částicovém filtru, přičemž každá částice reprezentuje možnou pozici robotu. Soubor částic tvoří přibližnou distribuci pravděpodobnosti pozice robotu v daném čase. Algoritmus iteruje ve třech hlavních krocích:

- **Predikce:** Na základě pohybového modelu se každá částice posune, aby odrazila nový odhad pozice po provedení řízení.
- **Aktualizace:** Pomocí senzoru se vyhodnotí, jak dobře daná částice odpovídá aktuálnímu měření – každé částici je přidělena váha podle míry shody.
- **Převzorkování:** Na základě těchto vah je vytvořen nový soubor částic – částice s vyšší váhou jsou vybírány častěji, zatímco méně pravděpodobné jsou vyřazeny. [41]

Příklad v jednodimenzionální chodbě (viz Obrázek 15) ukazuje, jak MCL postupně zužuje svou nejistotu – na začátku jsou částice rozmístěny náhodně po celé chodbě, ale s každým měřením a pohybem, se soubor částic koncentruje do místa, které nejlépe odpovídá pozorovaným údajům (např. detekce dveří). Váhy částic se aktualizují dle pravděpodobnosti měření a systém se postupně zaměřuje na nejpravděpodobnější oblast. [41]



Obrázek 15 Příklad lokalizace MCL [41]

4.2 Mapování

V předchozí části byly představeny metody pro určení polohy robotu v prostředí. Tyto principy však nejsou oddělené od úlohy mapování – naopak, schopnost vytvářet přesnou mapu prostředí často závisí na kvalitní lokalizaci a naopak. Mapování, podobně jako lokalizace, pracuje s nejistotou vyplývající ze senzorických měření a nepřesností pohybu robotu. [41]

Aby robot vytvořil mapu svého prostředí, musí být vybaven senzory, které dokážou vnímat jeho okolí – obvykle to jsou kamery, laserové skenery (2D a 3D), GPS nebo radar. Výsledky těchto měření jsou ale často zašuměné, uvnitř budov například nefunguje GPS. Navíc vzhledem k tomu, že se v procesu řízení neberou v úvahu vlivy prostředí, pohyb samotného robotu není přesný. [44]

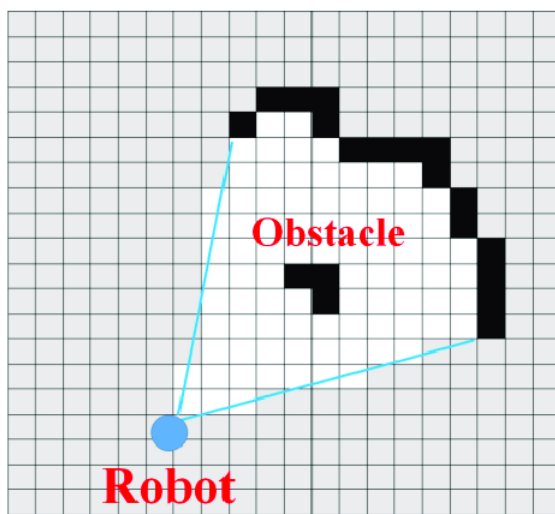
V ideálním případě lze přesnost mapy zlepšit kombinací mnoha měření a statistického zpracování. V praxi však nastávají problémy, protože chyby řízení a chyby měření jsou často vzájemně závislé. Proto se v průběhu času hromadí zkreslení odhadů polohy, což může negativně ovlivnit přesnost celkové mapy. Jednou z největších výzev při mapování je problém asociace dat, tj. identifikace, zda různá měření v různých časech zobrazují stejné místo. Pokud tento problém není správně vyřešen, budou chyby v mapě postupně přibývat, dokud se nezdaří určení polohy i mapování. Výpočetní složitost určování těchto shod omezuje použitelnost některých algoritmů SLAM na menší prostředí. Dalším komplikujícím faktorem je dynamická povaha prostředí. Mapovací algoritmy obvykle předpokládají, že prostředí je statické a dynamické změny považují za šum. Proto jsou obvykle spolehlivé pouze po omezenou dobu. [44]

Kromě přesnosti měření a stability prostředí ovlivňuje kvalitu mapování také strategie průzkumu. Robot by měl volit trajektorii nejen s ohledem na rychlost a spotřebu energie, ale také na to, jak dobře mu daná cesta umožní nasbírat data pro vytvoření mapy. V tomto kontextu rozlišujeme:

- Pasivní mapování, kde robot jen pozoruje prostředí a jeho trajektorie je dána jiným algoritmem (např. teleoperovaná jízda)
- Aktivní mapování, kdy robot aktivně upravuje svůj pohyb, aby zlepšil kvalitu lokalizace a zpřesnil mapu. Tento přístup je zvláště vhodný pro průzkum neznámých nebo symetrických prostor, kde je riziko chyb v odhadu vysoké. [44]

4.2.1 Metrické mapy

Metrické mapy slouží k přesné geometrické reprezentaci prostředí, ve kterém se mobilní robot pohybuje. Zaznamenávají polohu objektů a překážek v absolutních nebo relativních souřadnicích a umožňují tak robotu orientaci a plánování pohybu s vysokou přesností. Nejčastější formou metrické mapy je tzv. okupační mřížka (occupancy grid), která rozděluje prostor do buněk s přiřazenou pravděpodobností obsazení překážkou. Tato forma mapy je velmi využívána v systému ROS (viz Obrázek 16). [45][43]



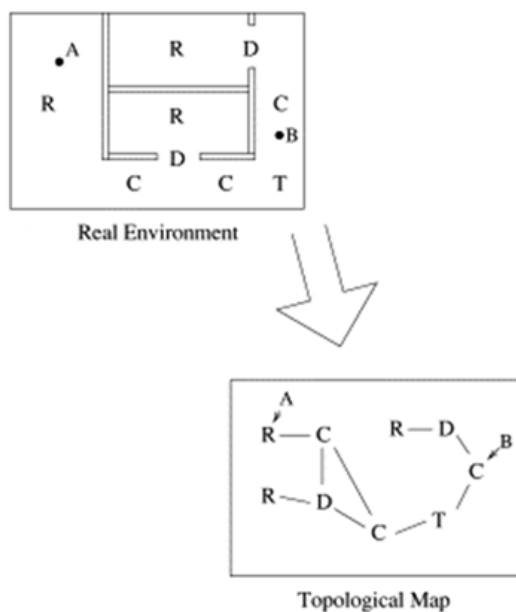
Obrázek 16 Příklad Metrické mapy formou okupační mřížky [46]

Výhodou metrických map je jejich detailnost a jednoznačnost, díky čemuž jsou snadno interpretovatelné i člověkem a vhodné pro algoritmy navigace. Na druhou stranu jsou paměťově náročné a jejich tvorba je citlivá na chyby v odhadu polohy robotu. Prostorová přesnost metrických map závisí na rozlišení mřížky – vyšší rozlišení poskytuje detailnější popis prostředí, ale vyžaduje více výpočetních prostředků. [43]

4.2.2 Topologické mapy

Topologická mapa reprezentuje prostředí jako graf, jehož uzly odpovídají jednotlivým rozlišitelným místům (místnostem, dveřím, chodbám) a hrany znázorňují možné cesty mezi nimi. Na rozdíl od metrických map neuchovávají přesné souřadnice objektů, ale pouze jejich vzájemné vazby.

Tento přístup nabízí abstraktní a méně detailní model, protože nevyžaduje přesné souřadnice (uchovává jen klíčová místa a jejich propojení). Model je výhodný zejména díky nízké výpočetní náročnosti, což z něj dělá vhodný nástroj pro plánování pohybu. Na obrázku 17 je příklad topologické části budovy. [45]



Obrázek 17 Příklad topologické mapy [45]

Na druhou stranu topologické mapy neposkytují přesnou informaci o geometrii prostředí a jejich úspěšnost závisí na schopnosti správně rozpoznat jednotlivá místa. Jejich využití je proto omezené v případech, kde je potřeba detailní model prostředí nebo kde dochází k častému výskytu podobných struktur, což může způsobit chyby v rozpoznávání. [43]

4.2.3 Hybridní mapy

Hybridní mapy kombinují výhody metrické a topologické reprezentace, a proto jsou v mobilní robotice velmi populární. Jejich cílem je propojit přesnost metrických map s efektivitou a abstrakcí map topologických v rámci celkového plánování. Tím vzniká flexibilní a škálovatelný model vhodný pro složitější prostředí. [43]

Běžným postupem je například vytvoření metrické mřížky obsazenosti, která se následně rozdělí na oblasti – uzly topologické mapy. Spojnice mezi těmito uzly pak odpovídají přístupovým cestám (např. dveřím, chodbám). Jiným přístupem je struktura, kde lokální metrické mapy popisují jednotlivé části prostředí (např. místnosti) a globální topologie propojuje tyto části pomocí bran nebo přechodových bodů. [43]

Výhodou hybridních map je možnost přesné navigace na lokální úrovni i efektivního plánování ve větších prostředích. Umožňují také lepší interpretaci informací a přirozenější integraci s lidskou představou prostoru. Na druhou stranu je jejich implementace složitější, protože vyžaduje sladění dvou různých způsobů reprezentace. [43]

4.3 Současná lokalizace a mapování

Problém SLAM nastává v situaci, kdy robot nemá k dispozici mapu prostředí, ani nezná svou vlastní polohu. Místo toho má přístup pouze k senzorickým měřením a příkazům řízení. Název „simultánní lokalizace a mapování“ přesně vystihuje podstatu problému – robot musí současně vytvářet mapu prostředí a zároveň se v ní lokalizovat. Tento problém má charakter bludného kruhu – k tomu, abychom se mohli lokalizovat, potřebujeme mapu, ale k tomu, abychom mohli vytvořit mapu, potřebujeme znát svoji polohu. Přestože se to zdá jako zásadní překážka, moderní přístupy využívající pravděpodobnostní metody umožňují SLAM efektivně řešit. [41][47]

Stav systému v úloze SLAM zahrnuje jak konfiguraci robotu, tak souřadnice orientačních bodů (landmarků), které byly dosud pozorovány. SLAM tedy umožňuje postupně rozšiřovat mapu a současně upřesňovat vlastní lokalizaci, a to i v prostředích bez absolutních polohových údajů (např. bez GPS). [41]

Zásadním problémem je správné zacházení s nejistotou určení pozice robotu a mapy. Ta vzniká v důsledku nepřesností senzorických dat. Filtrační metody založené na pravděpodobnostních metodách, které se k tomu účelu využívají již byly popsány v podkapitole 4.1. Tyto pravděpodobnostní metody se taktéž využívají v problematice SLAM. Mezi hlavní přístupy patří SLAM-EKF (rozšířený Kalmanův filtr), částicové metody (ozn. FASTSLAM) a tzv. grafové SLAMy:

- EKF-SLAM (Extended Kalman Filter SLAM): je klasický přístup, který se hodí pro situace, kde je v prostředí jen omezený počet orientačních bodů (tzv. landmarků). Tento algoritmus si v každém okamžiku udržuje nejlepší odhad nejen pozice robotu, ale i poloh všech známých orientačních bodů. Vše se přitom ukládá pomocí středních hodnot a míry nejistoty (kovariančních matic). Problém nastává, když landmarků začne být hodně – složitost výpočtů totiž rychle roste a stává se pro běžný hardware příliš náročná.
- FastSLAM (částicový SLAM): odděluje odhad polohy robotu od mapování okolí. K odhadu pozice používá tzv. částice (množinu možných hypotéz, kde by robot mohl být) a každá z nich si drží vlastní mapu orientačních bodů. Díky tomuto rozdělení je algoritmus vhodný i pro složitější a rozsáhlejší prostředí. Ale i tento přístup má svou slabinu a to, že výpočetní náročnost roste

kvadraticky, takže při opravdu velkém množství landmarků může být problém udržet výkon.

- Grafový SLAM: představuje celý problém jako graf, kde uzly odpovídají pozicím robotu nebo landmarkům a hrany představují měření ze senzorů, která mezi nimi vytvářejí spojení. Cílem je pak optimalizovat celý graf tak, aby všechny pozice co nejlépe odpovídaly realitě. Velkou výhodou tohoto přístupu je možnost zpětně opravit chyby (např. když se robot vrátí na známé místo a zjistí, že jeho odhad polohy byl nepřesný). [41][43][47]

SLAM představuje zásadní přístup k navigaci mobilních robotů v prostředí, které není předem zmapované. Umožňuje robotu současně odhadovat svou vlastní polohu i vytvářet mapu okolí pouze na základě senzorických dat a řídicích vstupů. V této kapitole byly popsány tři hlavní přístupy k řešení problému SLAM, a to EKF-SLAM, FastSLAM a grafový SLAM. Včetně jejich výhod, nevýhod.

V praxi je SLAM využíván v celé řadě robotických systémů a jeho implementace jsou dostupné i v populárních robotických frameworkách, jako je například ROS. V rámci ROS existuje více balíčků implementujících SLAM algoritmy – mezi nejpoužívanější patří například gmapping, cartographer nebo slam_toolbox, které umožňují jak 2D, tak i 3D mapování. Zároveň poskytují nástroje pro vizualizaci, ladění parametrů a analýzu výsledků lokalizace a mapování.

V následující praktické části této práce bude představeno nasazení SLAM algoritmů v systému ROS na mobilním robotu myAGV.

II PRAKTICKÁ ČÁST

5 NÁVRH ŘEŠENÍ

Tato kapitola popisuje celkový návrh řešení autonomní navigace mobilního robotu myAGV ve frameworku ROS, které bylo realizováno v rámci praktické části práce. Návrh řešení obsahuje nejprve vypsání jednotlivých požadavků na tuto práci. Obsahuje popis jednotlivých komponent použitých při realizaci, včetně zvoleného hardwaru a softwarových nástrojů. Dále popisuje architekturu systému, způsob propojení jednotlivých částí a role jednotlivých ROS balíčků.

5.1 Požadavky

Na začátku práce byly stanoveny následující požadavky na praktickou část práce, podle kterých byl zhotoven tento návrh řešení:

- Možnost vzdáleného ovládání a vizualizace v RVIZ.
- Schopnost robotu vytvářet mapu neznámého prostředí pomocí SLAM.
- Schopnost robotu autonomně nasnímat neznámé prostředí a poté se vrátit na domovskou pozici.
- Schopnost robotu lokalizovat se v dříve uložené mapě.
- Schopnost robotu autonomně se pohybovat v mapě a reagovat na překážky.

5.2 Předpřipravené prostředí na robotu

Na robotu myAGV byl již při jeho převzetí předinstalovaný operační systém Ubuntu 18.04 spolu s distribucí ROS Melodic, která je s tímto systémem kompatibilní. Součástí prostředí byl také připravený ROS pracovní prostor s názvem *myagv_ros*, který obsahuje základní implementaci pro ovládání robotu a sběr senzorických dat a ukázky SLAM nebo navigace. Jedná se o image systému, který je přímo dodáván výrobcem Elephant Robotics k robotu myAGV. Z tohoto prostředí bylo v rámci práce využito tyto jednotlivé komponenty:

- *Myagv_odometry_node* z balíčku *myagv_odometry*, který zpracovává data z enkodérů a publikuje zprávy typu *nav_msgs/Odometry* a odpovídající transformace TF
- *Joint_state_publisher* a *robot_state_publisher*, které slouží k publikaci stavů kloubů a popisu robotu na základě načteného URDF modelu *myAGV.urdf*

- *Static_transform_publisher*, který zajišťuje statickou transformaci mezi rámci *base_footprint* a *base_link*.
- Spuštění LIDAR senzoru, které je realizováno pomocí *X2.launch* z balíčku *ydlidar_ros_driver*, který publikuje zprávy typu *sensor_msgs/LaserScan*.
- Balíčky *myagv_teleop* a *myagv_ps2* pro ovládání robotu pomocí klávesnice nebo ovladače.
- URDF (Unified Robot Description Format) model robotu myAGV, je to formát založený na XML, který slouží k popisu fyzické struktury robotu.

Toto prostředí představovalo výchozí bod pro další návrh praktické části práce.

5.3 Použitý HW a SW

Z hlediska hardwaru se jedná především o mobilní robotu myAGV, který slouží jako autonomní platforma vybavená potřebnými senzory a výpočetní jednotkou. Kromě samotného robotu bylo důležité také vývojové prostředí v podobě osobního počítače s virtuálním strojem, na kterém je patřičná verze operačního systému Linux. K robotu je taktéž možnost se připojit vzdáleně k ploše systému, který běží na robotu pomocí aplikace TightVNC Viewer a nakonec router, který zajišťuje komunikaci mezi těmito prvky.

5.3.1 myAGV

Hlavním zařízením této práce je mobilní robot myAGV, jedná se taktéž o první mobilní platformu společnosti Elephant Robotics, navrženou pro výzkum, výuku a vývoj v oblasti mobilní robotiky. Jeho konstrukce a softwarová podpora umožňují implementaci navigačních aplikací, od základních úloh až po komplexní scénáře zahrnující manipulaci s objekty. [17]

Robot je vybaven všesměrovými koly typu Mecanum, které umožňují omnidirekcionální pohyb, tedy pohybu vpřed, vzad, do stran a rotaci kolem vlastní osy. Tato schopnost je dosažena pomocí nezávislého řízení otáček jednotlivých kol, což zajišťuje větší schopnost manévrovatelnosti v omezených prostorech. [48]

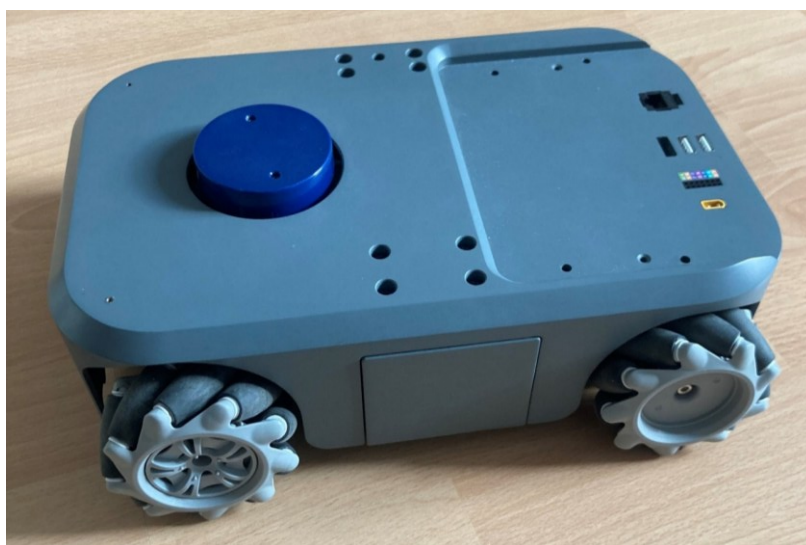
Konstrukce robotu je tvořena plně zakrytovaným kovovým rámem, který poskytuje zvýšenou odolnost a ochranu vnitřních komponent. Díky modulárnímu designu a rozšiřujícím rozhraním (jako jsou GPIO, USB, HDMI, Ethernet) je možné snadno připojit

další zařízení, jako jsou robotická ramena myCobot, mechArm, čímž vznikne tzv. kompozitní robot, schopný pohybu a manipulace s objekty. [48]

Robot je vybaven 2D LiDAR senzorem s 360° pokrytím a dosahem až 8 metrů, který se využívá k mapování a navigaci. Dále obsahuje 5MPx kameru, která umožňuje vizuální detekci a rozpoznání objektů. Výpočetní jednotkou je Raspberry Pi 4B, které poskytuje výkon pro provoz základních operací přímo na robotu. Pro náročnější programy je nutno propojit robot s výkonnější pracovní stanicí prostřednictvím Wi-Fi sítě. [17]

Robot běží na ubuntu verzi 18.04 s ROS1 verzi Melodic. Umožňuje tedy využití široké škály existujících balíčků a nástrojů pro tuto verzi ROS1.

Pro účely této práce byla z robotu myAGV odstraněna vrchní konstrukční část, která fyzicky omezovala výhled 2D LiDARu a tím bránila plnohodnotnému 360° skenování prostoru. Úprava výrazně usnadnila práci na robotu a následné testování. Výsledný stav robotu po úpravě je zobrazen na Obrázku 18.



Obrázek 18 Robot myAGV po úpravě

(zdroj: vlastní)

5.3.2 Router

Pro zajištění stabilního a rychlého spojení mezi robotem myAGV a vývojovým prostředím byl použit router TP-Link Archer AX17 (viz Obrázek 19), který podporuje moderní standard Wi-Fi 6 (802.11ax). Ten nabízí vyšší přenosové rychlosti, nižší latenci a vyšší efektivitu při komunikaci s více zařízeními současně, což je pro ROS systém klíčové. [49]



Obrázek 19 Použitý router [49]

(zdroj: vlastní)

Router poskytuje dvoupásmový provoz (2,4 Ghz a 5 Ghz) s kombinovanou rychlostí až 1500 Mb/s, což zajišťuje dostatečnou šířku pásma pro simultánní přenos dat jako jsou zprávy z LiDARu, vizualizace v RVIZ a dalších ROS témat. [49]

Použití tohoto konkrétního typu routeru nemá žádný velký význam. Tato implementace by mohla být provedena s jakýmkoliv routerem, který umožňuje LAN i Wi-Fi připojení, podporuje nastavení statických IP adres a má dostatečný výkon a pokrytí pro danou aplikaci.

5.3.3 VirtualBox

Pro účely vývoje a testování softwaru byl využit virtualizační nástroj Oracle VirtualBox, jedná se o multiplatformní řešení pro spouštění operačních systémů v podobě virtuálních strojů. Umožňuje provoz více nezávislých operačních systémů současně na jednom fyzickém zařízení, což je užitečné především při vývoji a testování softwaru bez nutnosti úpravy hlavního systému počítače. [50]

VirtualBox podporuje širokou škálu operačních systémů včetně různých distribucí Linuxu, které jsou často využívány v oblasti robotiky. Výhodou softwaru je jednoduchá instalace, snadné vytváření snapshot (zálohy virtuálního stroje). Lze tedy simulovat nebo testovat aplikace izolovaně, aniž by jakkoli ohrozil stabilitu a bezpečnost hlavního systému. [50]

V rámci této práce byla pomocí VirtualBoxu vytvořen Virtuální stroj s Ubuntu 18.04, na kterém byl následně nainstalován framework ROS Melodic. Virtuální stroj fungoval jako

hlavní výpočetní jednotka celého systému. Byly na něm spuštěny výpočetně náročnější uzly ROS (jako SLAM, AMCL, plánování trasy a vizualizace).

VirtualBox byl zvolen pro možnost rychlého a izolovaného nastavení vývojového prostředí bez nutnosti zásahu do hlavního operačního systému počítače. Oproti pouhému připojení k robotu přes vzdálenou plochu poskytuje VM výrazně vyšší výpočetní výkon a stabilnější běh náročných ROS uzlů.

5.3.4 TightVNC Viewer

Pro vzdálený přístup k systému robotu myAGV byl využit nástroj TightVNC Viewer, který umožňuje vzdálenou správu grafického rozhraní přes síť. TightVNC Viewer je open-source nástroj založený na protokolu VNC (Virtual Network Computing) a je určen pro vzdálené ovládání desktopového prostředí jiného zařízení na stejné LAN. [51]

Pomocí toho nástroje bylo možné se připojit z vývojového počítače přímo na operační systém běžící na robotu a provádět různé operace bez potřeby přímého připojení pomocí HDMI kabelu. To umožňuje spouštění a monitorování ROS uzlů přímo na robotu, práci s terminály a se správci souborů a ladění konfigurací bez potřeby fyzického připojení monitoru, klávesnice a myši.

Tento konkrétní nástroj byl použit díky své rychlé možnosti připojení ke grafickému prostředí a možnost vzdálené interakce i při omezeném přístupu k samotnému zařízení. Výhodou tohoto nástroje je možnost nastavení komprese přenosu podle potřeby rychlosti nebo kvality.

5.4 Architektura systému

Architektura robotického systému se skládá ze síťového rozvržení jednotlivých komponent a rozvržení softwarových částí v rámci systému.

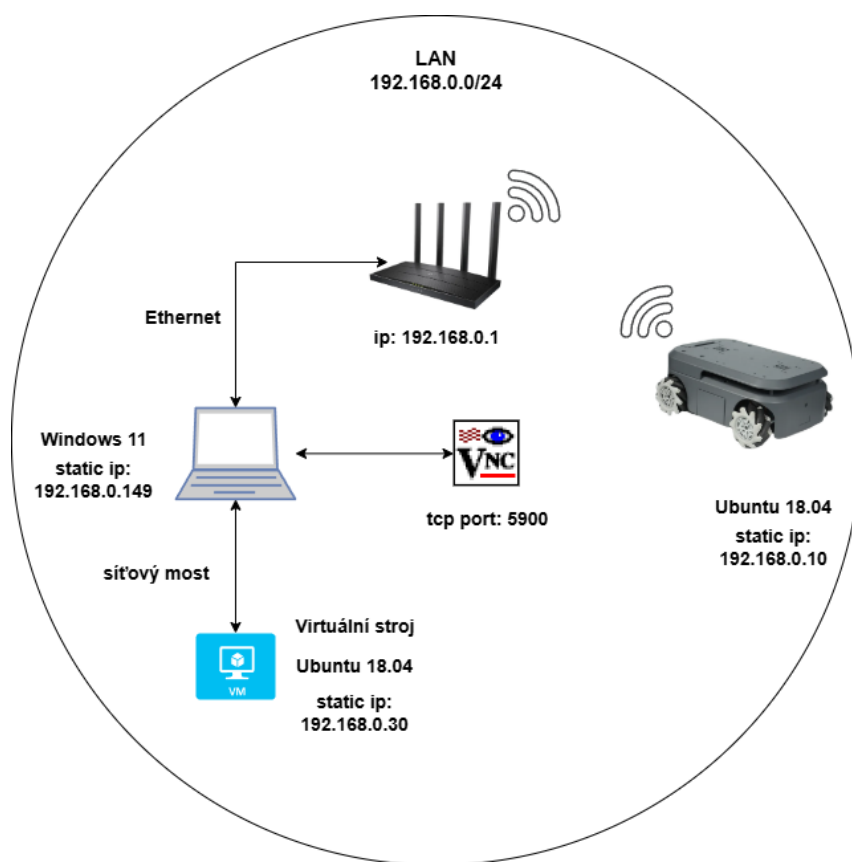
5.4.1 Síťová topologie

Síťová architektura navrženého systému je klíčovým prvkem k zajištění správné komunikace mezi jednotlivými komponentami v navrženém systému. Na obrázku 21 je schematicky znázorněno konkrétní rozložení sítě LAN a propojení všech použitých prvků.

Základem celé infrastruktury je router TP-Link Archer AX17, který tvoří jádro lokální sítě s adresací v rozsahu 192.168.0.0/24. Tento router zajišťuje jak připojení k internetu, tak

propojení robotu s vývojovým prostředím na osobním počítači. Všechna zařízení mají nastaveny statické IP adresy pro zajištění spolehlivé komunikace mezi nimi.

Mobilní robot myAGV je připojen k routeru prostřednictvím Wi-Fi a má přiřazenou statickou IP adresu 192.168.0.10. Na robotu běží operační systém Ubuntu 18.04 s nainstalovaným ROS Melodic. Zároveň zde běží *roscore* jako systemd služba, což zajišťuje, že robot funguje jako ROS master hned po jeho spuštění. Kromě toho jsou na robotu spuštěny senzory a odometrický uzel, který poskytuje data dalším komponentům v síti. Pro případy, kdy je nutné přistupovat ke grafickému rozhraní robotu, byl použit nástroj TighVNC Viewer, který zpřístupňuje vzdálené připojení k ploše robotu přes TCP port 5900.



Obrázek 20 Síťová architektura

(zdroj: vlastní)

Vývojové prostředí tvoří notebook s operačním systémem Windows 11 (se statickou IP adresou 192.168.0.149), na kterém běží virtualizační nástroj Oracle VirtualBox. V něm je spuštěn virtuální stroj s Ubuntu 18.04 (který má statickou IP adresu 192.168.0.30), kde jsou provozovány všechny výpočetně náročné uzly ROS systému. Notebook je připojen do LAN pomocí ethernetu a připojení virtuálního stroje do této LAN sítě je provedena prostřednictvím síťového mostu (bridge).

Díky tomuto rozdělení jsou výpočetně náročné procesy přeneseny na výkonnější zařízení, zatímco na robotu běží pouze základní služby. Komunikace mezi zařízeními je zajištěna pomocí ROS proměnných *ROS_MASTER_URI* a *ROS_IP*, které jsou nastavovány jak na stroji, tak na virtuálním stroji, automaticky pomocí skriptů při startu systému. Toto rozdělení umožňuje snadné ladění, testování a správu systému a současně přispívá k vyšší stabilitě a přehlednosti celé architektury systému.

5.4.2 Softwarové rozvržení

Systém je založen na frameworku ROS, který využívá komunikaci mezi jednotlivými uzly pomocí témat a zpráv. ROS umožňuje modulární rozdělení úloh mezi jednotlivé uzly a komponenty. Cílem návrhu softwarové architektury je jasné přiřazení úkolů systému.

Níže jsou uvedeny hlavní logické bloky systému s jejich stručným popisem funkce:

- Senzorická vrstva: uzly zpracovávající data z 2D Lidaru a enkodérů. Tato data slouží jako vstup pro odometrii, SLAM a lokalizaci
- Odometrie: uzly zajišťující převod dat z UART rozhraní do standardizované ROS zprávy (např. *nav_msgs/Odometry*).
- TF: balíček *tf* zajišťuje správu souřadnicových transformací mezi jednotlivými rámci jako je *map*, *odom*, *lidar_link*, *base_footprint* a další. Vytváří také potřebné transformace v rámci tzv. *tf* stromu.
- SLAM: pomocí uzlu *slam_gmapping* zajišťuje vytváření mapy a aktualizuje pozici robotu v mapě pomocí dat z enkodérů a LiDARu.
- Lokalizace: o lokalizaci se stará balíček *nav* a konkrétně *amcl*, který využívá částicový filtr pro učení se aktuální polohy robotu vzhledem k již existující mapě.
- Navigace: hlavní uzel *move_base* zajišťuje globální a lokální plánování trasy a vytváří tzv. *costmapy*. Spolupracuje s mapou, lokalizací a odhadem pozice robotu.
- Průzkum prostředí: k autonomnímu průzkumu neznámé lokace je použit balíček *explore_lite*, který automaticky generuje cílové body pro pokrytí celého neznámého prostoru a tvorbu výsledné mapy.

- Uložení mapy: po dokončení mapování je možné pomocí nástroje *map_server* uložit výslednou mapu ve formátu .pgm a .yaml pro další možnosti použití.
- Vizualizace: nástroj RVIZ slouží k vizualizaci všech komponent systému – od sensorických dat přes trajektorie, zobrazení mapy až po plánované cílové body.

Důležitou součástí architektury navrženého systému je rozdělení ROS balíčků mezi jednotlivá zařízení s ohledem na výpočetní nároky jednotlivých úloh. Robot myAGV funguje jako centrální bod komunikace mezi uzly. Na robotu běží *roscore*, senzory a základní uzly jako odometrie. Veškeré výpočetně náročné komponenty, jako SLAM, lokalizace, navigace nebo vizualizace běží na virtuálním stroji na počítači, který k ROS masteru přistupuje jako klient. Toto rozdělení umožňuje efektivní využití výpočetního výkonu, lepší kontrolu nad laděním systému, stabilnější provoz autonomního robotu.

6 VYTVOŘENÍ PROSTŘEDÍ PROJEKTU

Tato kapitola popisuje implementaci systému navrženého v předchozí části. Zaměřuje se na postupné nasazení jednotlivých komponent v prostředí ROS, vytvoření pracovní sítě, nastavení prostředí ROS, konfigurací balíčků, vytváření vlastních uzlů a skriptů. Kapitola byla psána tak, aby bylo možné jednotlivé kroky snadno zopakovat.

6.1 Nastavení virtuálního stroje a prostředí ROS

Pro vývoj, testování a správu robotu byl využit virtuální stroj (VM, virtual machine) s operačním systémem Ubuntu 18.04. Do něj poté byla nainstalována ROS distribuce Melodic. [52]

Nejdříve nezbytné stáhnout instalační obraz (ISO) systému Ubuntu. Vzhledem k použití ROS Melodic byla zvolena verze Ubuntu 18.04 LTS (Bionic). ISO obraz byl připojen při vytváření nové VM ve VirtualBoxu.

V programu Oracle VirtualBox byl vytvořen nový VM, kde bylo potřeba vložit stažený ISO obraz, po vložení ISO obrazu VirtualBox automaticky rozpozná typ a verzi operačního systému.

VM byl hardwarově nakonfigurován následovně:

- Operační paměť 16 GB.
- Počet CPU: 12.
- Hard Disk: 40 GB.

Po dokončení a vytvoření VM bylo třeba v nastavení VM nastavit tyto parametry:

- V kartě Display
 - Video paměť: 128 MB.
 - Povolit 3D akceleraci
- V kartě Síť
 - Povolit síťovou kartu a připojit ji k „Síťový most“.
 - Výběr typu síťové karty, podle typu připojení PC k internetu (ethernet nebo WiFi).
 - Vybrat „Kabel připojen“.

Připojením síťové karty k síťovému mostu bylo zajištěno přímé připojení rozhraní virtuálního stroje s fyzickou sítí, jako by byl samostatný fyzický počítač. VM dostala vlastní IP adresu ze stejné podsítě jako ostatní zařízení. To je důležité k následné komunikaci s robotem myAGV. [53]

Po nastavení všeho potřebného a zapnutí VM bylo potřeba projít základní instalaci operačního systému, během které bylo nastavené heslo „admin“. Poté se přešlo k instalaci ROS Melodic.

Nejprve bylo zapotřebí přidat do systému nový zdroj softwarových balíčků (tzv. repozitář), odkud bude možné nainstalovat ROS Melodic. Konkrétně byl vytvořen nový soubor s adresou ROS repozitáře, který systém používá při instalaci a aktualizaci balíčku přes apt. [54]

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu bionic main"
> /etc/apt/sources.list.d/ros-latest.list'
```

Pro plnou instalaci ROS Melodic byl použit následující příkaz.

```
sudo apt install ros-melodic-desktop-full
```

Po dokončení instalace ROS bylo potřeba zajistit, aby se při každém otevření terminálu automaticky aktivovalo prostředí ROS Melodic pomocí inicializačního skriptu *setup.bash*.

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
```

Tento příkaz přidá do konfiguračního souboru *~/.bashrc* příkaz pro načtení skriptu *setup.bash*, který nastaví proměnné prostředí potřebné pro běh ROS. Tento skript nastavuje potřebné systémové proměnné (např. cesty k balíčkům, knihovnám a nástrojům ROS), aby bylo možné spouštět příkazy jako *roscore*, *roslaunch* nebo *roslaunch* bez další potřebné konfigurace. Dalším příkazem se načtou změny provedené v tomto souboru do aktuálního terminálu, takže není nutné zavírat a znovu otevírat terminál.

```
source ~/.bashrc
```

Pro další nastavování a používání prostředí ROS je taktéž vhodné nainstalovat nástroj *roscdep*, který slouží ke správě a instalaci systémových závislostí ROS balíčků. [54]

```
sudo apt install python-roscdep -y
```

A následuje inicializace *roscdep*, která se spouští pouze jednou.

```
sudo roscdep init
```

Po úspěšném instalování ROS, lze správnost celé instalace ověřit pomocí spuštění příkazu *roscore* v terminálu. Poté byl vytvořen ROS workspace, nejdříve byla vytvořena adresářová struktura.

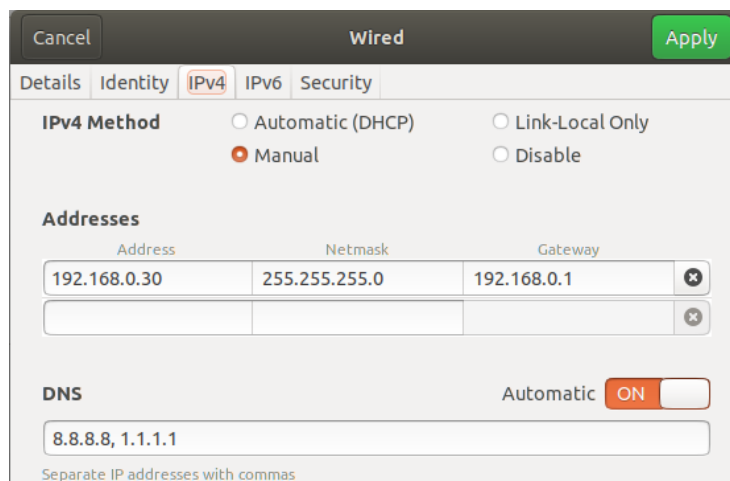
```
mkdir -p ~/ros/src  
cd ~/ros
```

V adresáři `ros` byly příkazem `catkin_make` zkompileovány přítomné balíčky v pracovním prostoru. Po jeho spuštění byly zkompileovány všechny balíčky ve složce `src/`, čímž vznikly dvě samostatné složky (*build/* s dočasnými soubory a *devel/* s výsledky překladu). Po tomto kroku byl workspace připraven k práci.

6.1.1 Nastavení adres v LAN

Pro vzdálené připojení k robotu bylo na něm zapotřebí nastavit statickou IP adresu, aby se poté bylo možné k němu automaticky připojit. Proto bylo nejdříve potřeba se k němu připojit pomocí HDMI a nastavit na něm statickou IP adresu v konfiguračním souboru `etc/netplan/50-cloud-init.yaml`. V tomto souboru bylo nastaveno připojení k WiFi připojení přes rozhraní `wlan0`, vypnutí přiřazení adresy od DHCP serveru, nastavení statické IP adresy, brány, DNS serveru a připojení k jednotlivým WiFi. Kvůli záloze zde byly nastaveny dva přístupové body. LAN síť s SSID „Moje_2.4GHz“ a záložní „MyAGV_hotspot“.

Nastavení IP adresy na VM probíhalo pomocí GUI. Konkrétně v nastavení sítě v síťového rozhraní ethernet. V nastavení ethernet v záložce IPv4, byla nastavena IP adresa, maska, brána a DNS server (viz Obrázek 21).



Obrázek 21 Nastavení statické IP adresy VM

(zdroj: vlastní)

6.1.2 Připojení pomocí SSH

Pro připojení k robotu z VM byla použita služba SSH (secure shell), je to označení síťového protokolu, který slouží k bezpečnému vzdálenému přihlášení do jiného zařízení přes síť. Robot může být ovládán z terminálu na VM. [55]

Pro snadné připojení k robotu pomocí VM byla použita zkratka „alias“, která umožňuje definovat jednoduchý název pro delší příkaz.

```
alias robot="ssh ubuntu@192.168.0.10"
```

Tento alias byl vložen do `~/.bashrc`, takže bude aktivní při každém zapnutí terminálu. Poté pro vzdálené připojení k robotu z VM stačí napsat v terminálu příkaz `robot` a po vyžádání heslo „Elephant“.

6.1.3 Připojení pomocí VNC

Pro připojení k vzdálené ploše pomocí VNC bylo zapotřebí umožnit připojení i bez fyzického propojení s monitorem pomocí HDMI kabelu. K tomu bylo zapotřebí na robotu nainstalovat

```
sudo apt install x11vnc xserver-xorg-video-dummy
```

- `x11vnc` – umožní vzdálený přístup ke grafickému výstupu. [56]
- `xserver-xorg-video-dummy` – poskytuje virtuální obrazovku v případě, že není připojen fyzický monitor. [57]

Dále bylo nutné vytvořit konfigurační soubor pro použití ovladače dummy, který simuluje výstupní grafické zařízení. Tento ovladač umožňuje spuštění X serveru i v případě, že není připojen žádný fyzický monitor. Konfigurační soubor byl umístěn do adresáře `/etc/X11/xorg.conf.d/` pod názvem `99-dummy.conf`. [56][57]

X server představuje základní vrstvu grafického systému v prostředí Linux a zajišťuje zobrazování grafického rozhraní a zpracování vstupů z periferních zařízení (např. myš a klávesnice). Ve zmíněném konfiguračním souboru jsou definovány parametry virtuálního monitoru, které odpovídají běžnému fyzickému displeji. Mezi tyto parametry patří horizontální a vertikální obnovovací frekvence, barevná hloubka a rozlišení. [58]

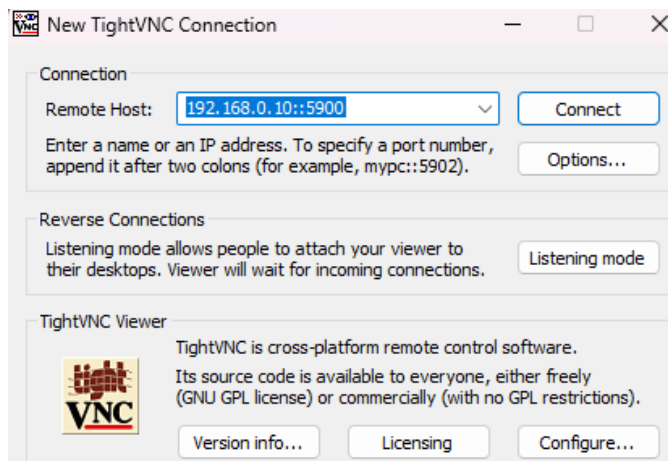
Tato konfigurace zajistí, že se při startu systému inicializuje virtuální obrazovka s rozlišením 1280×720 pixelů. To umožní běh grafického prostředí i bez fyzického HDMI výstupu, což je zásadní pro funkčnost VNC.

Aby se VNC server `x11vnc` spustil automaticky při startu systému, byla vytvořena systemd jednotka `/etc/systemd/system/x11vnc.service`. Tato jednotka zajišťuje připojení k běžícímu grafickému prostředí (na displeji :0) bez požadavku na zadání hesla a s podporou více současných připojení. Pokud dojde k chybě nebo ukončení procesu, služba se automaticky restartuje.

Systemd jednotka byla povolena a spuštěna pomocí následujících příkazů.

```
sudo systemctl daemon-reexec
sudo systemctl enable x11vnc.service
sudo systemctl start x11vnc.service
```

Následně bylo možné z počítače pomocí programu TightVNC přistoupit ke vzdálené ploše robotu zadáním jeho IP adresy a defaultního portu 5900 (viz Obrázek 22).



Obrázek 22 Připojení k myAGV pomocí VNC

(zdroj: vlastní)

6.1.4 Propojení ROS master/klient

Pro správnou komunikaci mezi ROS masterem a ROS klientem byla potřeba správně nastavit ROS proměnné, jak na VM, tak na robotu. ROS proměnné jsou:

- ROS_MASTER_URI – je adresa, kde běží *roscore*.
- ROS_HOSTNAME – IP adresa zařízení, které spouští uzly. [59]

ROS proměnné byly nastaveny v souboru `~/.bashrc`. Jedná se o spouštěcí skript, který se automaticky spustí při otevření nového terminálu. Nejprve byly nastaveny proměnné na VM.

```
export ROS_MASTER_URI=http://192.168.0.10:11311
export ROS_HOSTNAME=192.168.0.30
```

Port 11311 je defaultně rezervovaný pro běh ROS Masteru.

```
export ROS_MASTER_URI=http://192.168.0.10:11311
export ROS_HOSTNAME=192.168.0.10
```

V takovém nastavení ROS proměnných nastává problém, protože se nastavují v souboru `~/.bashrc`, takže se provedou pouze po otevření terminálu. To na VM nevadí,

protože při práci se vždy terminál spouštěl, ale na robotu to nebylo vždy nutné. Proto byla vytvořena vlastní systemd jednotka, která nastavuje proměnné prostředí a spouští *roscore* při startu systému. Tento správce služeb je zodpovědný za spouštění a správu služeb po startu systému i během jeho běhu. Proto jeho použití je ideální pro tento případ, kdy bylo zapotřebí nastavit ROS proměnné hned po startu systému robotu. Nejprve byla vytvořena služba *roscore.service*.

```
sudo nano /etc/systemd/system/roscore.service
```

Tato služba se spustí po inicializaci sítě a běží pod uživatelem ubuntu. Před spuštěním čeká 3 sekundy, aby se stabilizovalo síťové připojení, a následně načte prostředí ROS. Pomocí ROS proměnných definuje, že *roscore* poběží na dané IP adrese a bude dostupný ostatním zařízením v síti. Pokud by služba spadla, systemd ji automaticky znovu spustí.

Správné propojení bylo ověřeno příkazem *rostopic list* ve VM, po zapnutí robotu se správně zobrazoval téma */rosout* a */rosout_agg*.

6.2 Oživení robotu

K oživení robotu byl vytvořen launch soubor *myagv_active.launch*, který slouží k inicializaci základních komponent robotu myAGV v prostředí ROS. Spuštění launch souboru se provede následujícím příkazem.

```
roslaunch myagv_ros myagv_active.launch
```

Tento launch soubor nejprve spouští uzel *myagv_odometry_node* z balíčku *myagv_odometry*, který zpracovává údaje z enkodérů a publikuje odometrii a transformační rámce. Následně je načten URDF model robotu ze souboru *myAGV.urdf*, který je uložen v balíčku *myagv_urdf*. Tento popis robotu využívají uzly *joint_state_publisher* a *robot_state_publisher* k publikaci aktuálních stavů kloubů a k výpočtu transformačního stromu TF, který definuje prostorové vztahy mezi jednotlivými částmi robotu. Dále je pomocí *static_transform_publisher* publikována statická transformace mezi rámci *base_footprint* a *base_link*, které slouží jako základní referenční body robotu pro lokalizaci a navigaci. Taktéž je spuštěn kontrolér laserového skeneru pomocí souboru *X2.launch* z balíčku *ydliider_ros_driver*, který inicializuje senzor a publikuje data typu *sensor_msgs/LaserScan*.

Tento oživovací launch soubor zajišťuje kompletní spuštění odometrie, popisu robotu a získání dat z LiDAR senzoru a jeho nutné ho spustit pomocí VNC nebo SSH připojení pro

následnou práci s robotem. Po jeho spuštění je prostředí připraveno pro vyšší úlohy jako je mapování, lokalizace a navigace, jejich vývoj probíhá kompletně ve VM.

6.3 Synchronizace systémového času

Důležitou součástí nastavení prostředí byla synchronizace systémového času mezi robotem a VM. ROS velmi spoléhá na časová razítka u všech zpráv a pokud není čas mezi zařízeními synchronizovaný, může docházet k chybnému zobrazení dat ve vizualizaci RVIZ, nesprávnému sloučení senzorických dat a následnému problému s lokalizací nebo navigací.

K synchronizaci byla využita služba chrony. Nejprve bylo zapotřebí nainstalovat chrony na VM i na robotu. [60]

```
sudo apt install chrony
```

Na VM se v */etc/chrony/chrony.conf* nastaví VM jako časový server. [60]

```
allow 192.168.0.0/24
```

Na robotu se v */etc/chrony/chrony.conf* nastaví robot jako klient, který se připojí k časovému serveru. [60]

```
server 192.168.0.30 iburst
```

7 MAPOVÁNÍ NEZNÁMÉHO PROSTŘEDÍ

Tato část popisuje praktický postup vytvoření mapy prostředí pomocí mobilního robota v systému ROS. Postup byl realizován ve VM a byl rozdělen do několika fází. Nejprve byl do pracovního prostředí stažen a nakonfigurován SLAM balíček. Následně byla ověřena správná funkce SLAM pomocí manuálního ovládání robota, které sloužilo k procházení prostoru a sběru dat pro vytvoření mapy. Po úspěšném ověření byl systém rozšířen o plánovací komponentu *move_base*, která umožňuje ovládat robot pomocí zadávání cílů ve vizualizaci RVIZ.

Pro automatický průzkum prostředí byl použit balíček *explore_lite*, který umožňuje robotu samostatně prozkoumat dosud neznámé oblasti. Po dokončení průzkumu byla uložena aktuální mapa prostředí. Při začátku průzkumu byla uložena výchozí pozice robota, která sloužila jako referenční bod pro návrat po ukončení průzkumu neznámého prostředí. Celý proces je vizualizován pomocí nástroje RVIZ, kde bylo připraveno vlastní rozložení pro přehledné sledování získaných dat ze senzorů, aktuální trajektorie a generování mapy prostředí.

7.1 Konfigurace SLAM balíčku

Pro účely mapování byl využit balíček *slam_gmapping*, který poskytuje implementaci algoritmu SLAM na základě Rao-Blackwellizovaného částicového filtru (RBPF), který odpovídá přístupu FastSLAM, který je zmíněn v kapitole 4.3. Algoritmus využívá data získaná z LiDARu a odometrie k odhadu pozice robota a tvorbě mapy. [61]

Základní princip spočívá v tom, že každá částice v rámci filtru reprezentuje hypotézu o trajektorii robota. Pro každou z těchto trajektorií je následně vypočítaná odpovídající mapa prostředí. Oproti klasickému SLAMu, který musí odhadovat celou mapu a trajektorii v jednom společném prostoru, RBPF odděluje tyto úlohy a tím výrazně snižuje výpočetní náročnost. [61]

Prvním krokem byla instalace balíčku *slam_gmapping* do pracovního prostředí projektu a kompilace. [62]

```
cd ~/ros/src
git clone https://github.com/ros-perception/slam_gmapping.git
cd ~/ros
catkin_make
```

Pro snadné spouštění byl vytvořen vlastní balíček s názvem *my_slam*, který obsahuje uživatelsky vytvořené launch soubory, konfigurační soubory, skripty, rviz konfigurace a vytvořené mapy.

```
cd ~/ros/src
catkin_create_pkg my_slam rospy std_msgs sensor_msgs nav_msgs tf
slam_gmapping
cd ~/ros/src/my_slam
mkdir -p launch config maps rviz scripts
cd ~/ros
catkin_make
```

Pro otestování SLAM algoritmu byl v balíčku *my_slam* vytvořen vlastní launch soubor s názvem *slam.launch*. Tento soubor spouští SLAM algoritmus pomocí balíčku *slam_gmapping* a zároveň inicializuje vizualizaci RVIZ. Dále definuje základní strukturu uzlů a jejich parametrů, které jsou nezbytné pro úspěšnou lokalizaci a mapování prostředí.

První spuštěný uzel pochází z balíčku *slam_gmapping* a provádí samotný SLAM algoritmus. V rámci něj jsou definovány následující klíčové části:

- Rámce (*base_frame* a *odom_frame*) určují názvy transformační rámců robotu, které se používají k převodu dat mezi senzory a mapou.
- Parametry mapy – ovlivňující frekvenci aktualizace, počet částic v algoritmu, rozlišení výsledné mapy a velikost mapy
- Parametry laseru – specifikují nastavení pro skenování okolí, které má vliv na kvalitu detekce překážek a následné zpracování měření.
- Další parametry pro jemnější ladění, které definují chování algoritmu při pohybu robotu (např. po jaké vzdálenosti nebo otočení se provede aktualizace).

[63]

Po spuštění uzlu *slam_gmapping* je následně spuštěn i nástroj RVIZ s přednastavenou konfigurací v *rviz/slam_config.rviz*, která zajišťuje přehledné zobrazení dat ze SLAMu.

7.1.1 Telemetrie

Důležité je ověřit funkčnost SLAM v reálném provozu. Pro sběr dat potřebných k vytvoření mapy bylo nejprve potřeba umožnit robotu manuální pohyb v prostředí. K tomuto účelu byl využit bezdrátový ovladač typu PS2, který umožňuje uživateli intuitivně ovládat pohyb robotu ve všech směrech.

Aby ROS mohl přijímat vstupy z ovladače, je nutné mít v systému nainstalovaný balíček *joystick*. Tento balíček obsahuje systémové nástroje pro práci s klasickými USB nebo bezdrátovými herními ovladači. Zajišťuje, že zařízení je správně rozpoznáno a může být použito jako vstup pro ROS uzly, které pracují s joystickem. [64]

```
sudo apt-get install joystick
```

Pro propojení ovladače PS2 a robotu byl následně použit připravený launch soubor od výrobce, který se nachází v pracovním prostředí *myagv_ros* na robotu. [64]

```
roslaunch myagv_ps2 myagv_ps2.launch
```

7.2 Navigace

Po úspěšném otestování funkčnosti SLAM algoritmu bylo dalším krokem rozšíření systému o schopnost navigace robotu v zároveň vytvářející se mapě. Cílem bylo vytvořit takové prostředí, ve kterém bude možné:

- Generovat mapu.
- Robot lze navigovat zadáváním cíle „2D Nav Goal“ v RVIZ.

Pro zajištění navigace v mapě generované pomocí SLAM bylo zapotřebí nainstalovat kompletní navigační balíček *navigation*, který obsahuje tyto komponenty:

- *Move_base* – hlavní uzel pro plánování a řízení robotu.
 - *Amcl* – algoritmus pro lokalizaci robotu v známé mapě (místo toho je použit SLAM, který mapu generuje v reálném čase).
 - *Map_server* – poskytuje uloženou mapu ve formátu okupační mřížky.
 - *Costmap_2d* – moduly pro práci s překážkami a plánování tras.
 - Další pomocné komponenty (např. recovery chování, planovače trajektorie).
- [65]

Instalace proběhla do pracovního prostředí *ros*. Balíček *navigation* nainstaluje všechny závislosti potřebné pro plnou funkcionalitu navigačního systému v ROS.

```
sudo apt-get install ros-melodic-navigation
```

Pro správné a pohodlné spuštění navigačního systému byly vytvořeny dva launch soubory v balíčku *my_slam*:

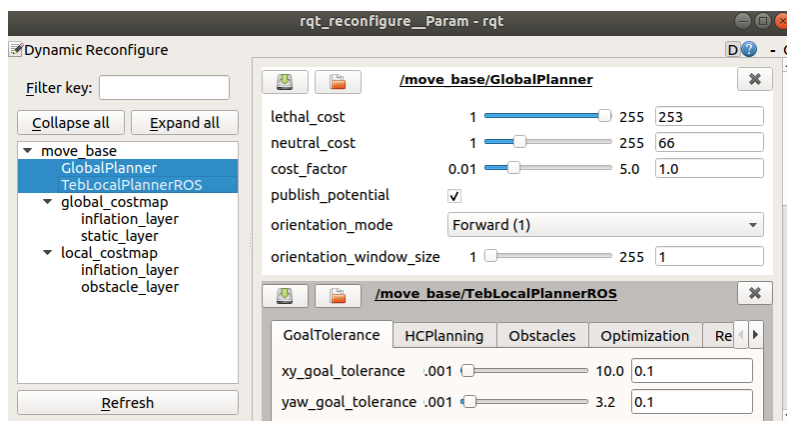
- *Slam_navigation.launch* - který vychází ze *slam.launch*, ale spouští také *move_base.launch* a spouští RVIZ konfiguraci *slam_navigation.rviz*.
- *Move_base.launch* – slouží ke spuštění hlavního navigačního uzlu *move_base*, který integruje plánovací a řídicí komponenty pro autonomní pohyb robotu v prostředí. Nastavuje použití plánovače GlobalPlanner a TebLocalPlanner v rámci uzlu *move_base* a načítá veškeré YAML soubory z adresáře *config/*.

7.2.1 Tvorba konfiguračních souborů

Konfigurační soubory ve složce *config/* byly vytvořeny s využitím nástroje *rqt_reconfigure*, který se spouští následujícím příkazem.

```
roslaunch rqt_reconfigure rqt_reconfigure
```

Tento nástroj umožňuje během běhu systému interaktivně upravovat parametry plánovačů a costmap a okamžitě sledovat jejich vliv na chování trajektorie a vykreslené costmapy (viz Obrázek 23).



Obrázek 23 Příklad konfigurace parametrů pomocí *rqt_reconfigure*

(zdroj: vlastní)

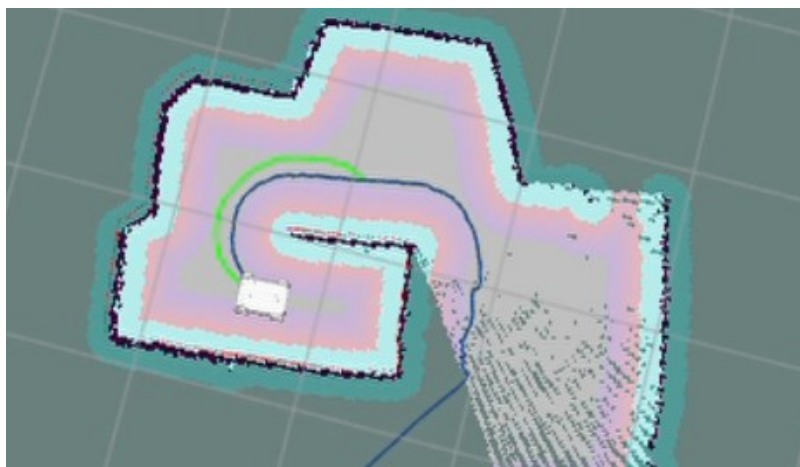
Mezi klíčové konfigurační soubory patří:

- *Local_costmap_params.yaml* - definuje parametry lokální costmapy, která slouží pro krátkodobé plánování trasy v bezprostředním okolí robotu. Costmapa je dynamická, aktualizuje se podle dat ze senzorů a slouží k vyhnutí se překážkám. Zahrnuje např. šířku a výšku oblasti, frekvenci aktualizace, rozlišení, tvar robotu, vzdálenost rozšiřující překážky a použité pluginy pro aktualizace překážky atd. [66]

- *Global_costmap_params.yaml* - popisuje obvykle statickou mapu, ale v této implementaci je dynamická, protože mapa se generuje za běhu programu. Slouží k následnému plánování celé trasy od aktuální pozice ke zvolenému cíli. Nastavuje referenční rámce pro navigaci, zvětšení překážek kvůli bezpečnosti rozlišení mapy atd. [66]
- *Global_planner_params.yaml* – obsahuje nastavení globálního plánovače trasy, který využívá globální costmapu a hledá optimální trasu mezi počáteční a cílovou pozicí. Nastavuje se zde např. algoritmus plánování, způsob práce s překážkami a frekvence přepočtu trasy. [66]
- *Teb_local_planner_params.yaml* – slouží ke konfiguraci lokálního plánovače TEB (time elastic band), který v reálném čase upravuje trajektorii podle dynamiky robotu a aktuální situace v okolí. Nastavují se zde maximální rychlosti, vzdálenosti od překážek, chování při zatáčení nebo vyhýbání a další parametry ovlivňující přesnost a bezpečnost pohybu. [67]

7.2.2 Nastavení vizualizace a testování

Pro otestování spolupráce navigace se SLAM byl vytvořen konfigurační soubor v složce *rviz/* s názvem *slam_navigation.rviz* pro vizualizaci, který se spouští launch souborem *slam_navigation.launch*. V RVIZ lze vidět pozici robotu, nasnímanou mapu, globální trajektorii (modrá čára), lokální trajektorii (zelená čára) a globální mapu, která je znázorněna jako costmapa a má barvu modrou až růžovou podle vzdálenosti od překážky (viz Obrázek 24). Ve vizualizaci lze pomocí příkazu „2D Nav Goal“ zadat cíl pro pohyb robotu.



Obrázek 24 Vizualizace SLAM s navigací

(zdroj: vlastní)

Po úspěšné exploraci prostoru lze manuálně uložit mapu pomocí balíčku *map_server* do určité uživatelské složky. Map_server uloží mapu do dvou souborů (černobílý obrázek mapy a konfigurační soubor). [68]

```
roslun map_server map_saver -f ~/ros/src/my_slam/maps/mapa
```

7.3 Průzkum prostředí

Tato část se věnuje autonomnímu průzkumu neznámého prostředí robotem bez nutnosti uživatelského zásahu. Cílem bylo rozšířit předchozí řešení o schopnost robotu samostatně detekovat nezmapované oblasti a postupně je procházet za účelem vytvoření úplné mapy.

Klíčovým nástrojem k tomu byl balíček *explore_lite*, který umožňuje průzkum hranic známého a neznámého prostoru (tzv. frontier-based exploration). Pro úplnou autonomnost řešení byly napsány skripty pro uložení počáteční polohy a pro návrat na tuto polohu při dokončení průzkumu a na závěr se uloží prozkoumaná mapa. [69]

7.3.1 Balíček Explore_lite

Tento balíček implementuje jednoduchou metodu průzkumu neznámého prostoru. Tento přístup umožňuje robotu automaticky vyhledávat hranice mezi známým a neznámým prostorem (tzv. frontieri) a postupně do těchto oblastí plánovat trasy pomocí *move_base*. Cílem je pokrýt co nejvíce prostoru a vytvořit úplnou mapu bez nutnosti ručního zásahu. [69]

Balíček *explore_lite* je navržen jako nadstavba nad *move_base* a nevyžaduje žádnou speciální konfiguraci. Využívá existující mapu (generovanou SLAM) a plánovač *move_base* pro generování cílů, ke kterým robot postupně jede. Po dosažení cíle je automaticky vybrán další frontier dokud nejsou žádné další neprozkoumané oblasti. [69]

Pro použití balíčku *explore_lite* ho bylo nutné nejdříve nainstalovat.

```
sudo apt-get install ros-melodic-explore-lite
```

Takto stažený balíček obsahuje i připravený launch soubor *explore.launch* k použití. V launch souboru byly modifikovány tyto parametry:

- Frekvence plánovače – pro frekvenci plánování nových cílů.
- Nastavení důležitosti, které ovlivňují výběr frontier cíle. Byl potlačen vliv směrové orientace a cíl se vybíral pouze podle vzdálenosti a velikosti oblasti.
- Byla nastavena minimální velikost průzkumné oblasti na 0,5m. [69]

V uživatelském balíčku *my_slam* byl vytvořen launch soubor *slam_explore.launch*, který spouští SLAM, navigaci, exploraci, vizualizaci a potřebné skripty.

7.3.2 Uložení výchozí polohy

Pro uložení výchozí polohy byl vytvořen python skript v balíčku *my_slam* ve složce *scripts/* s názvem *save_home.py*, který je spuštěn v launch souboru *slam_explore.launch*. Skript uloží výchozí pozici robotu v souřadnicovém systému mapy. Po spuštění skriptu dojde nejprve k inicializaci ROS uzly a vytvoření posluchače transformačního systému tf, který umožňuje získat polohu a orientaci robotu vůči mapě. Skript následně čeká na dostupnost transformace mezi rámci *map* a *base_footprint* a poté ji pomocí funkce `lookupTransform()` načte (viz Obrázek 25).

```
try:
    listener.waitForTransform("map", "base_footprint", rospy.Time(0), rospy.Duration(5.0))
    (trans, rot) = listener.lookupTransform("map", "base_footprint", rospy.Time(0))
except (tf.LookupException, tf.ConnectivityException, tf.ExtrapolationException):
    rospy.logerr("TF Transform error!")
    return
```

Obrázek 25 Načtení aktuálních souřadnic

(zdroj: vlastní)

Výsledná poloha a orientace robotu jsou zapsány do struktury *pose*, která je následně uložena do YAML souboru s názvem *home_pose.yaml* ve složce *config/* (viz Obrázek 26).

```
pose = {
    'position': {
        'x': trans[0],
        'y': trans[1],
        'z': trans[2],
    },
    'orientation': {
        'x': rot[0],
        'y': rot[1],
        'z': rot[2],
        'w': rot[3],
    }
}
with open('/home/ros/ros/src/my_slam/config/home_pose.yaml', 'w') as file:
    yaml.dump(pose, file)
```

Obrázek 26 Struktura *pose* a její uložení do .yaml souboru

(zdroj: vlastní)

Výsledný soubor *home_pose.yaml* lze vidět na Obrázku 27.

```

orientation:
  w: 0.9993408949682665
  x: 0.0
  y: 0.0
  z: 0.03630117965058666
position:
  x: -0.06090215309634983
  y: -0.008865960057165001
  z: 0.0

```

Obrázek 27 Uložená výchozí poloha a orientace robotu

(zdroj: vlastní)

7.3.3 Návrat na výchozí polohu a uložení mapy

Pro návrat robotu na výchozí polohu je v launch souboru *slam_explore.launch* taktéž spuštěn skript *explore_monitor.py*, který monitoruje, zda jsou k dispozici aktivní frontieri. Skript obsahuje inicializaci ROS uzlu a nastavení parametrů, jako minimální časová prodleva pro návrat a tolerance vzdálenosti. Skript odebírá data ze dvou hlavních témat */explore/frontiers* pro získání aktuálních cílů průzkumu a */odom* pro získání aktuální polohy robotu.

Klíčovou částí skriptu je funkce *monitor_loop()*, která zajišťuje neustálé sledování stavu průzkumu a rozhodování o jeho ukončení. V této smyčce se ve dvou různých situacích vyhodnocuje, zda má být zahájen návrat robotu na výchozí pozici a uložena mapa prostředí. První podmínka sleduje, zda uplynula delší doba než 10 sekund bez přijetí jakékoliv zprávy z tématu */explore/frontiers*, který informuje o detekovaných cílech průzkumu. Jedná se o bezpečnostní mechanismus, který může indikovat chybu v systému a pokud se robot nenachází v blízkosti své uložené výchozí polohy, tak je spuštěn skript *go_home.py* a následně *map_server* pro uložení mapy do předem určeného adresáře (viz Obrázek 28)

```

def monitor_loop(self):
    while not rospy.is_shutdown():
        now = rospy.Time.now()
        elapsed = now - self.last_change_time

        # Watchdog: žádná zpráva z frontier topicu delší dobu než 10s
        if now - self.last_frontier_msg_time > self.frontier_timeout:
            rospy.logwarn("[Watchdog] No frontier messages for %.1f sec.",
                           self.frontier_timeout.to_sec())
            if not self.is_at_home():
                rospy.loginfo("running go_home.py and saving map")
                subprocess.Popen(["roslaunch", "my_slam", "go_home.py"])
                rospy.sleep(2.0)
                subprocess.Popen(["roslaunch", "map_server", "map_saver",
                                   "-f", rospkg.RosPack().get_path('my_slam') + "/maps/mapa"])
            else:
                rospy.loginfo("Already at home")
        break

```

Obrázek 28 Funkce *monitor_loop()* – ztráta komunikace

(zdroj: vlastní)

Další podmínka vyhodnocuje návrat robotu na výchozí pozici a uložení mapy prostředí podle toho, zda se za posledních 5 sekund nezměnil počet detekovaných frontierů, tedy robot již nenašel další nové oblasti k průzkumu. Tento scénář reprezentuje běžné dokončení průzkumu. Pokud robot není v cílové pozici, opět se spustí návratový skript *go_home.py* a uložení mapy pomocí *map_serveru*. V opačném případě, pokud se robot nachází ve výchozí poloze, se proces pouze ukončí (viz Obrázek 29).

```
# Standardní ukončení podle prázdných frontierů
if self.no_frontier_start is not None and (now - self.no_frontier_start > self.wait_duration):
    rospy.logwarn("No change in frontiers for %.1f sec",
                  self.wait_duration.to_sec())
    if not self.is_at_home():
        rospy.loginfo("Calling go_home.py and saving map")
        subprocess.Popen(["rosrun", "my_slam", "go_home.py"])
        rospy.sleep(2.0)
        subprocess.Popen(["rosrun", "map_server", "map_saver",
                           "-f", rospkg.RosPack().get_path('my_slam') + "/maps/mapa"])
    else:
        rospy.loginfo("Already at home")
    break
```

Obrázek 29 Funkce *monitor_loop()* – vyčerpání frontierů

(zdroj: vlastní)

Skript *go_home.py* slouží k automatickému odeslání cílové pozice robotu na základě pozice uložené v souboru *home_pose.yaml*. Skript vytvoří Publisher na téma */move_base_simple/goal*, který využívá navigační systém *move_base* k přijetí cílových souřadnic ve formátu *PoseStamped*. Následně se načte obsah YAML souboru obsahujícího uloženou pozici a orientaci robotu v mapě. Tato data jsou vložena do zprávy *PoseStamped*, která zahrnuje jak souřadnice, tak i směr natočení (viz Obrázek 30).

```
def send_home_goal():
    rospy.init_node('go_home_node')
    pub = rospy.Publisher('/move_base_simple/goal', PoseStamped, queue_size=1)
    rospy.sleep(1.0)

    with open('/home/ros/ros/src/my_slam/config/home_pose.yaml', 'r') as file:
        pose_data = yaml.safe_load(file)

    goal = PoseStamped()
    goal.header.frame_id = "map"
    goal.header.stamp = rospy.Time.now()
    goal.pose.position.x = pose_data['position']['x']
    goal.pose.position.y = pose_data['position']['y']
    goal.pose.position.z = pose_data['position']['z']
    goal.pose.orientation.x = pose_data['orientation']['x']
    goal.pose.orientation.y = pose_data['orientation']['y']
    goal.pose.orientation.z = pose_data['orientation']['z']
    goal.pose.orientation.w = pose_data['orientation']['w']

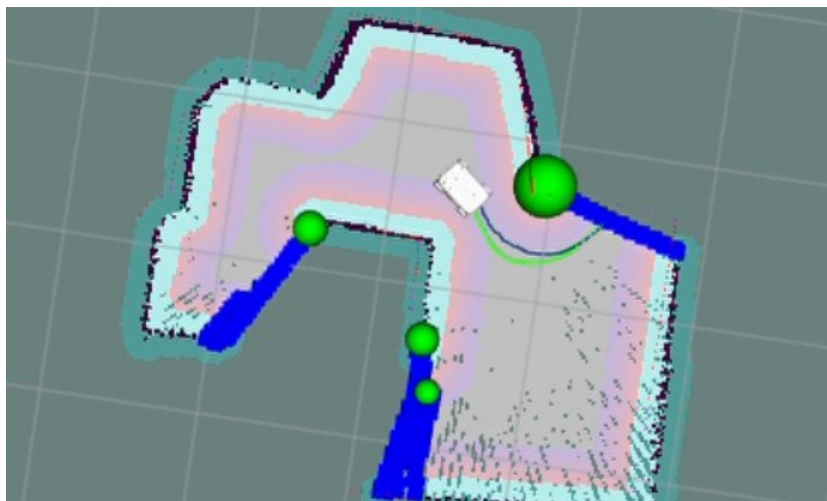
    pub.publish(goal)
    rospy.loginfo("Home goal sent!")
```

Obrázek 30 Funkce pro návrat na výchozí polohu

(zdroj: vlastní)

7.3.4 Testování a vizualizace

Pro testování autonomního průzkumu v reálném prostředí byla použita stejná RVIZ konfigurace jako v případě pro SLAM navigaci, akorát se zde ještě navíc zobrazují frontie (tmavě modrý úsek označuje hranici mezi známou a neznámou částí mapy a zelený kruh reprezentuje cílové body) (viz Obrázek 31).



Obrázek 31 Vizualizace průzkumu mapy

(zdroj: vlastní)

8 NAVIGACE V BLUDIŠTI

Tato část úzce navazuje na předchozí část, kdy je již vytvořená mapa prostředí, konkrétně v modelu bludiště. Cílem bylo umožnit robotu přesně určit svou pozici v prostředí a následně se pohybovat v mapě po zadání cíle v RVIZ nebo mezi zadanými body v RVIZ (patrolování).

Systém je rozdělen do několika částí. Nejprve byla načtena dříve uložená mapa prostředí, následně byl robot lokalizován pomocí algoritmu AMCL, který je součástí navigačního balíčku. AMCL algoritmus umožňuje odhadnou aktuální pozici na základě porovnání senzorických dat s mapou. Poté byla použita a modifikována plánovací komponenta *move_base* pro tuto konkrétní aplikaci.

Součástí řešení bylo i vytvoření vlastního skriptu pro možnost patrolování robotu po předem definovaných bodech. Tento skript taktéž obsahuje možnost návratu do výchozí polohy. Celý proces byl vizualizován pomocí nástroje RVIZ.

8.1 Lokalizace robotu

Pro zajištění schopnosti robotu pohybovat se ve známé mapě bylo nezbytné zajistit přesnou lokalizaci jeho aktuální pozice v prostředí. K tomu byl využit algoritmus AMCL (Adaptive Monte Carlo Localization), který je součástí standardního balíčku *navigation*. Tento algoritmus využívá částicový filtr pro odhad pozice robotu na základě porovnání senzorických dat z LiDARu s informacemi uloženými ve statické mapě. [70]

AMCL vychází z metody Monte Carlo Localization (MCL), která je detailně popsána v kapitole 4.1.4 Monte Carlo Localization. Zatímco klasický MCL pracuje s pevně daným počtem částic, AMCL jejich počet adaptivně mění v závislosti na míře nejistoty odhadu polohy robotu. Tím dochází k výrazné optimalizaci výpočetní náročnosti při zachování dostatečné přesnosti. [41]

V rámci této části byl vytvořen nový balíček s názvem *my_navigation_new*, který slouží jako pracovní prostor pro konfiguraci navigace v bludišti. Součástí balíčku je hlavní launch soubor s názvem *bringup.launch*, který spouští důležité komponenty jako *map_server*, který načte uloženou mapu a publikuje ji, balíček *amcl*, který se stará o lokalizaci robotu a jeho konfigurační soubor *amcl_params.yaml*. Dále ještě spouští *move_base*, která je obdobná jako v předchozí kapitole, kromě nastavení *static_map* na

hodnotu True, kvůli tomu, že mapa je již načtena. V poslední řadě spouští nakonfigurovaný RVIZ pro vizualizaci této implementace.

Parametry pro AMCL byly definovány v konfiguračním souboru `config/amcl_params.yaml`, který se nachází v uživatelském balíčku `my_navigation_new`. Mezi hlavní nastavené parametry patří:

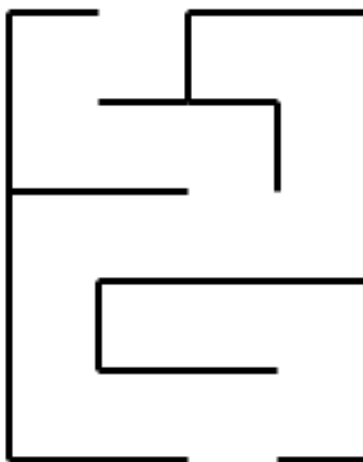
- Nastavení rozsahu počtu částic.
- Určení adaptivní rychlosti počtu částic.
- Nastavení typu modelu na holonomní platformu.
- Nastavení šumu z odometrie.
- Nastavení vyhodnocování měření z LiDARu. [70]

Parametry byly laděny pomocí nástroje *rqt_reconfigure* tak, aby byla lokalizace stabilní a robot neztrácel svou pozici ani při jízdě úzkými pasážemi bludiště.

8.2 Vytvoření testovacího bludiště

Pro sestavení testovacího prostředí ve formě bludiště byly využity kartonové panely o rozměrech přibližně 56x22 cm a 35x22 cm. Jednotlivé části byly spojeny 3D vytištěnými spojkami, které umožnily snadnou konstrukci bludiště bez nutnosti lepení.

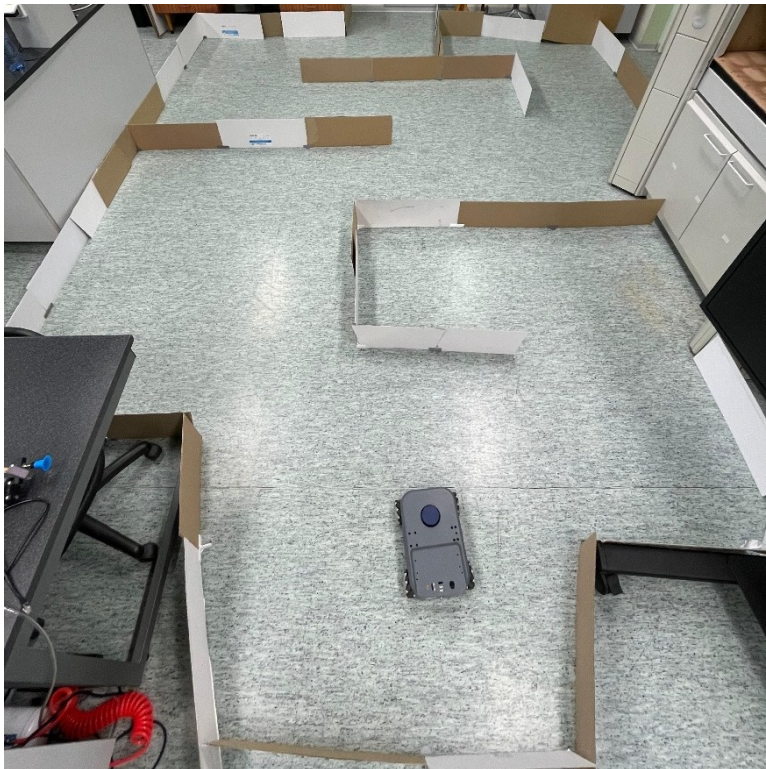
S ohledem na dostupné množství materiálu, rozměrů testovací plochy a nutnosti ponechání určitého prostoru pro prohyb robotu, byl navržen model bludiště, který obsahuje základní prvky jako jsou zatáčky, slepé uličky i průchozí koridory (viz Obrázek 32).



Obrázek 32 Návrh bludiště

(zdroj: vlastní)

Na obrázku 33 je zachycen reálný vzhled sestaveného bludiště po jeho postavení. Prostředí bylo pečlivě ohraničeno, aby nedocházelo k možným nepřesnostem během snímání prostoru,

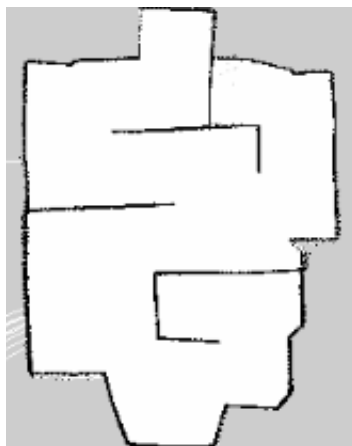


Obrázek 33 Sestavené bludiště

(zdroj: vlastní)

Po sestavení bludiště byl v tomto prostředí realizován proces autonomního mapování, jak bylo popsáno v kapitole 7.3 Průzkum prostředí. Robot během průzkumu samostatně procházel jednotlivé části bludiště, detekoval neprozkoumané oblasti a generoval mapu. Po dokončení průzkumu byla výsledná mapa automaticky uložena pomocí *map_serveru*.

Výslednou mapu, která slouží jako podklad pro lokalizaci, navigaci a patrolování v této kapitole, lze vidět na Obrázku 34. Mapa zachycuje půdorys bludiště, rozlišené průchody a představuje klíčový prvek pro implementaci lokalizace.



Obrázek 34 Uložená mapa

(zdroj: vlastní)

8.2.1 Testování a vizualizace

Po nakonfigurování lokalizačního systému pomocí algoritmu AMCL následovalo ověření jeho funkčnosti v reálném prostředí. Testování probíhalo v bludišti (viz Obrázek 33), jehož struktura odpovídala dříve uložené mapě prostředí získané pomocí SLAM a průzkumu (viz Obrázek 34). Cílem bylo ověřit přesnost lokalizace robotu při pohybu v členitém prostředí a správnou reakci systému na změny polohy.

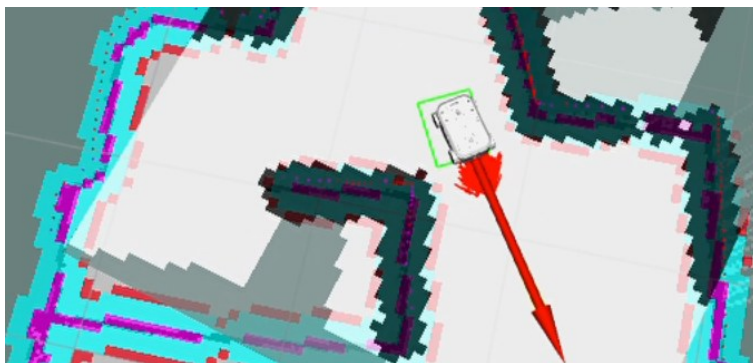
Po spuštění lokalizačního systému pomocí launch souboru *bringup.launch* v balíčku *my_navigation_new* byl taktéž spuštěn nakonfigurovaný RVIZ, který poskytuje přehled o stavu lokalizace. Před začátkem je důležité určit přibližnou startovací polohu robotu pomocí tlačítka „2D pose estimate“. Na obrázku 35 lze vidět stav lokalizačního systému AMCL bezprostředně po zadání odhadu polohy. Tato fáze představuje okamžik, kdy byla ručně určená přibližná poloha robotu a algoritmus AMCL zahájil odhad skutečné pozice. Každá šipka představuje jednu částici algoritmu, a tedy jednu hypotézu o poloze robotu.



Obrázek 35 Počáteční poloha částí při AMCL

(zdroj: vlastní)

Po zadání cíle v RVIZ se červené šipky, které reprezentovaly rozprostřený rozptyl částic s různými hypotézami o poloze robotu se nyní sloučily do jednoho místa. Tato změna vizualizace indikuje, že algoritmus dosáhl vysoké míry jistoty ohledně aktuální pozice robotu v mapě, teda že většina částic se shodla na jednom konkrétním odhadu pozice a orientace (viz Obrázek 36).



Obrázek 36 AMCL po zlokalizování pozice

(zdroj: vlastní)

8.3 Patrolování

Další funkcionalita systému spočívá v tom, že robot je schopen se autonomně pohybovat mezi předem definovanými body v mapě. Toto využití simuluje tzv. patrolování robotu v určitém objektu. Na této aplikaci lze taky názorně předvést reakci na objevení překážky v trajektorii robotu.

Patrolovací systém využívá stávající nastavení lokalizace ve známém prostředí a poté pomocí na navigace a skriptu, který sbírá jednotlivé cílové body zadané uživatelem a taktéž detekuje pokyn pro návrat na výchozí polohu robotu.

Pro přehlednost byl vytvořen balíček *my_patrolling*, který obsahuje obvyklou adresářovou strukturu jako v předchozích případech. Je zde launch soubor *patrolling.launch*, který na začátku spustí skript *save_home.py*, který načte výchozí polohu (viz kapitola 7.3.2 Skript pro uložení výchozí polohy).

8.3.1 Nastavení bodů pro patrolování

Dále skript *patrol_control.py*, představuje hlavní řídicí prvek pro zajištění patrolování robotu mezi předem definovanými body v prostředí. Je navržen tak, aby umožnil nastavení počáteční lokace robotu, jednoduché zadání cílů uživatelem přes RVIZ tlačítkem „Publish Point“, jejich uložení do souboru a následné autonomní projíždění těchto bodů.

Nejprve je uživatel v terminálu vyzván pro zadání počáteční polohy, kterou pomocí zadání písmena „e“ potvrdí (viz Obrázek 37) a poté se uloží novou aktualizovanou výchozí poloha robotu.

```
[INFO] [1746365592.758624]: Zadej v RViz pomocí '2D Pose Estimate' počáteční pozici a stiskni 'e' pro potvrzení uložení.
```

Obrázek 37 Vyzvání k zadání počáteční polohy

(zdroj: vlastní)

Poté je uživatel v terminálu vyzván pro zadání bodů libovolného počtu bodů v RVIZ, které mu po kliknutí zobrazí v terminálu (viz Obrázek 38) a jsou uloženy v *config/waypoints.yaml* spustí patrolovací režim, který postupně projíždí jednotlivé body.

```
[INFO] [1746365626.140567]: Zadej body v RViz pomocí 'Publish Point'. Stiskni 'e' pro spuštění patrolu. 'h' → návrat domů.  
[INFO] [1746365630.452566]: Ukládám bod 1: x=1.22, y=1.20  
[INFO] [1746365633.212401]: Ukládám bod 2: x=2.21, y=-0.42  
[INFO] [1746365636.322221]: Ukládám bod 3: x=3.01, y=1.37
```

Obrázek 38 Vyzvání k zadání bodů pro patrolování a následně jejich výpis

(zdroj: vlastní)

Robot poté projíždí jednotlivými body pomocí funkce *patrol_loop()*, která postupně posílá jednotlivé cíle klientovi *move_base*. Robot po dosažení cíle čeká 1 sekundu a poté pokračuje k dalšímu bodu (viz Obrázek 39).

```
def patrol_loop():  
    global external_goal_sent  
  
    client = actionlib.SimpleActionClient('move_base', MoveBaseAction)  
    client.wait_for_server()  
  
    rospy.loginfo("Spouštím patrol mezi {} body.".format(len(waypoints)))  
    while not rospy.is_shutdown() and not external_goal_sent:  
        goal = MoveBaseGoal()  
        goal.target_pose.header.frame_id = "map"  
        goal.target_pose.header.stamp = rospy.Time.now()  
        goal.target_pose.pose.position.x = wp['x']  
        goal.target_pose.pose.position.y = wp['y']  
        goal.target_pose.pose.orientation.w = 1.0  
        rospy.loginfo("Naviguji na bod: {}".format(wp))  
        client.send_goal(goal)  
        client.wait_for_result()  
        rospy.sleep(1.0)
```

Obrázek 39 Smyčka pro sledování zadaných bodů

(zdroj: vlastní)

8.3.2 Spuštění s možností návratu na výchozí polohu

Pro funkci libovolné možnosti návratu domů, lze tento proces zapnout pomocí skriptu *launch_with_go_home.py* pomocí následujícího příkazu

```
roslun my_patrolling launch_with_go_home.py
```

Tento skript je spuštěn od začátku a umožňuje návrat robotu do výchozí polohy během běhu aplikace pomocí stisku tlačítka „h“, které spustí skript *go_home.py* (viz kapitola 7.3.3 Skripty pro návrat na výchozí polohu a uložení mapy) (viz Obrázek 40).

```
while True:
    if select.select([sys.stdin], [], [], 0)[0]:
        c = sys.stdin.read(1)
        if c == 'h':
            print("\nStisknuto 'h' → posílám robota domů")
            subprocess.call(['roslun', 'my_patrolling', 'go_home.py'])
```

Obrázek 40 Kontrola vstupu pro návrat na výchozí polohu

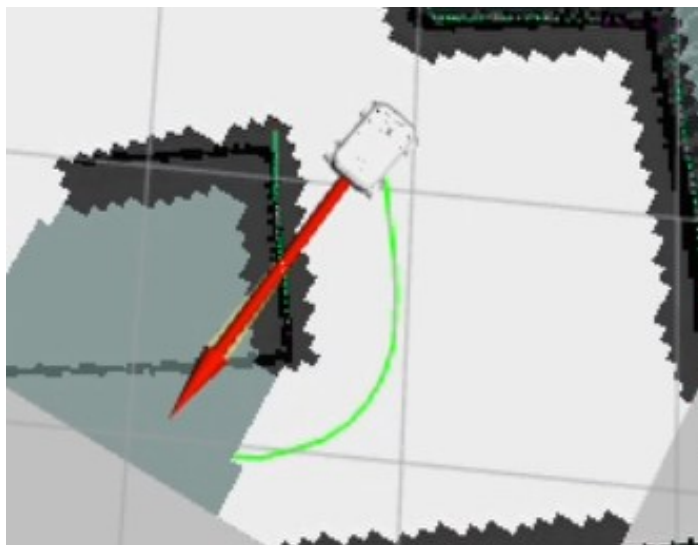
(zdroj: vlastní)

Toto opatření umožňuje větší autonomnost systému pro testování, kdy se robot po zadání příkazu vrátí zpět na výchozí polohu a taky jako bezpečnostní prvek v případě zaseknutí robotu či jiné technické závadě.

8.3.3 Vyhýbání překážkám

Implementovaný systém patrolování umožňuje názorně demonstrovat schopnost robotu autonomně se vyhýbat překážkám během pohybu mezi cíli. Díky využití dynamické lokální costmapy a plánovače *teb_local_planner* (viz kapitola 7.2.1 Tvorba konfiguračních souborů) dokáže robot reagovat v reálném čase na změny ve svém okolí a upravit trajektorii tak, aby se vyhnul kolizím, a přitom zachoval směr k zadanému cíli.

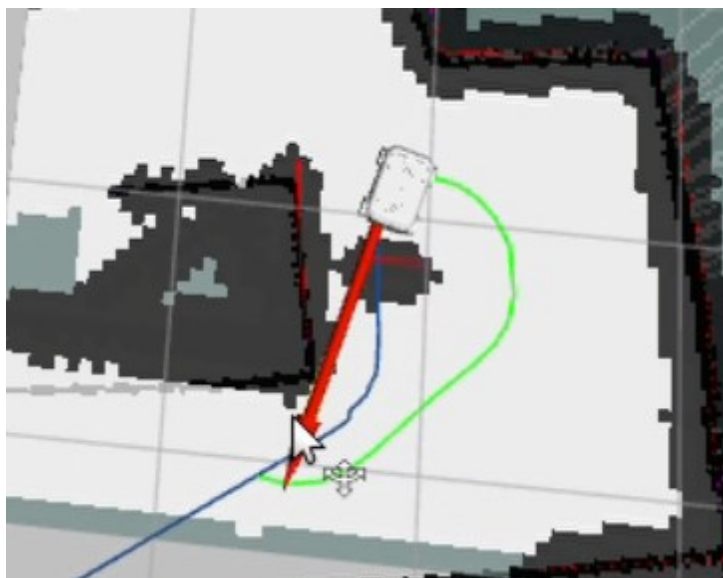
Testování schopnosti vyhnutí se překážky bylo provedeno při spuštění patrollingu mezi 2 body v bludišti. První průjezd byl proveden bez jakýchkoliv překážek, přičemž robot zvolil nejkratší trasu mezi zadanými body (viz Obrázek 41).



Obrázek 41 Průjezd bez překážky

(zdroj: vlastní)

V následujícím průjezdu byla do dráhy robotu umístěna papírová krabice, která představovala novou překážku. Díky aktivní lokální costmapě byla tato překážka detekovaná senzorem a zahrnuta do plánování. Výsledek toho je, že lokální plánovač nenásledoval globální plánovač, ale místo toho změnil trajektorii a zvolil alternativní cestu kolem překážky (viz Obrázek 42).



Obrázek 42 Vyhnutí se překážce

(zdroj: vlastní)

9 ZHODNOCENÍ VÝSLEDKŮ

Byl realizován systém autonomní navigace robotu myAGV ve frameworku ROS. Výsledné řešení splnilo stanovené požadavky, které byly stanoveny při návrhu systému. Systém byl implementován modulárně s důrazem na oddělení výpočetně náročných úloh do vývojového prostředí běžícího na VM. Robot byl využit především pro sběr dat ze senzorů a jako ROS master. Systém byl testován v reálném modelu bludiště. Během práce byly vytvořeny balíčky pro SLAM, navigaci a patrolování, přičemž každý z nich byl doplněn vlastními launch soubory, konfiguračními soubory, skripty a vizualizací.

Vývoj probíhal postupně od ověření systému až po lokalizaci a navigaci v mapě. Nakonec byl vytvořen systém patrolování mezi body s možností návratu na výchozí uloženou pozici.

9.1 Vyhodnocení výsledků

V rámci práce bylo úspěšně implementováno několik praktických aplikací, kterou mohou simulovat reálné využití mobilního robotu ve skutečných situacích:

- Autonomní průzkum prostředí s návratem na výchozí polohu – využití v záchranných akcích v neznámých prostorech, inspekce v pro člověka nebezpečných zónách.
- Navigace ve známém prostředí a reagování na vzniklé překážky – využití v průmyslové automatizaci nebo ve skladech.
- Patrolování ve známém prostředí – využití jako bezpečnostní dohled v místech s omezeným přístupem, robot pravidelně objíždí a kontroluje kritická místa v elektrárnách či datových centrech.

Na základě praktického testování bylo zároveň identifikováno několik slabých míst současné implementace. Systém funguje dobře v ideálních podmínkách uzavřeného prostředí s dobře snímanými hranicemi a se stabilním připojením k síti, avšak v některých případech se projeví určité nedostatky:

- Výpočetní zátěž systému – v náročnějších scénářích docházelo k občasnému mírnému zpoždění přenosu nebo zpracování dat, což může vést k neoptimálním rozhodnutím, proto je pohyb robotu nastaven na rychlost, při které tyto nedostatky mají jen minimální vliv.

- Omezení balíčku *explore_lite* – přestože je tento balíček vhodný pro účely průzkumu, jeho chování může být v reálných aplikacích někdy problematické. Během testování se několikrát stalo, že došlo k jeho „zaseknutí“, kdy přestal generovat nové cíle, ale zároveň nedošlo ke správnému ukončení procesu pohybu, což způsobilo, že robot pokračoval v nechtěném pohybu.
- Jako nedostatek a určitou nevýhodu této implementace lze považovat závislost na LAN síti, jelikož výpočetně náročné procesy nejsou prováděny přímo na robotovi, ale na vzdáleném počítači. Pro správnou funkčnost systému je proto nutné stabilní síťové spojení, což může omezit pohyb robotu na určitou vzdálenost.

Naopak implementování balíčků AMCL a SLAM prokázalo velmi spolehlivou funkčnost při testování. SLAM vytvářel přesnou mapu prostředí v reálném čase a AMCL umožnil stabilní lokalizaci robotu i při mírné chybě odhadu počáteční polohy. Tyto základní komponenty systému jsou robustní a tvoří pevný základ pro další možná vylepšení.

Videa z jednotlivých aplikací jsou součástí příloh této práce v adresáři *videa*, kde je možné sledovat jejich reálný průběh bludištěm společně s vizualizací v RVIZ. V příloze se také nachází pracovní prostředí *ros*, které bylo vyexportováno z VM a adresářová struktura tohoto pracovního prostředí (viz Příloha I).

9.2 Možnosti vylepšení

Během implementace a testování se ukázalo několik oblastí, kde by bylo možné na tuto práci navázat a vylepšit:

- Integrace inerciální měřicí jednotky (IMU) – zahrnutí IMU by výrazně pomohlo vylepšit data získaná z odometrie, které by podpořilo lokalizaci, která se v této implementaci hodně spoléhá na data získaná z LiDARu. Vzhledem k tomu, že myAGV platforma používá kola typu Mecanum, která umožňují všestranný pohyb, sloužilo by IMU ke kompenzaci setrvačných jevů, především při rychlých změnách směru nebo na kluzkém povrchu.
- Přechod na ROS2 – Novější verze frameworku ROS přináší vyšší výkon s podporou real-time komunikace a nejnovější podporu balíčků, které jsou aktivně vyvíjeny. V dlouhodobém horizontu by migrace systému na ROS 2

zvýšila stabilitu, další možnosti rozšiřitelnosti systému a zvýšila celkovou relevanci projektu.

- Využití kamery a strojového vidění – využití kamery v rámci autonomních aplikací (jako detekce QR kódu nebo detekce nepovolaných osob).
- Zvýšení autonomie systému – Přidáním mechanismů pro rozpoznání chybových stavů. Využití tzv. recovery behaviors (chování robotu při zotavení) v rámci *move_base*, které slouží ke zotavení robotu např. při srážce s překážkou.

ZÁVĚR

Tato bakalářská práce se zaměřila na návrh a realizaci systému autonomní navigace mobilního robotu pomocí frameworku ROS. Téma mobilní robotiky patří mezi klíčové oblasti moderní automatizace, s rostoucím významem nejen v průmyslových aplikacích, ale také ve veřejném prostoru, službách, zdravotnictví či vzdělávání. Práce ukazuje, že s využitím dostupných open-source nástrojů a cenově dostupných robotických platforem je možné sestavit funkční a modulární systém, který dokáže samostatně mapovat neznámé prostředí, lokalizovat se v něm a autonomně se pohybovat.

V teoretické části bylo nejprve řešeno téma mobilní robotiky, kde byl popsán rozdíl mezi AGV a AMR, byly zde představeny hlavní aplikace mobilní robotiky a jaké jsou její aktuální trendy a výzvy. Práce se taktéž zabývala využitím mobilní robotiky ve výuce, kde hrají platformy mobilních robotů významnou roli. Mobilní robotika hraje ve vědecké a akademické sféře významnou roli ve spojení teoretických znalostí získaných během studia s praktickou realizací, taktéž pomáhá rozšířit robotiku mezi širší veřejnost. Byly představeny a popsány platformy mobilních robotů, které jsou a mohou být využívány právě v rámci výuky jako vzdělávací platforma.

Dále byly popsány základní pojmy a principy z oblasti ROS, lokalizace a mapování, včetně porovnání rozdílů mezi ROS 1 a ROS 2. Součástí bylo i zdůvodnění proč je ROS výhodnou volbou při vývoji robotických systémů, a to nejen díky své modularitě, rozsáhlé komunitě a dostupnosti nástrojů, ale také díky jeho vhodnosti pro výuku a experimenty. Teoretický základ poskytl pevný rámec pro praktickou realizaci.

Praktická část práce byla zaměřena na vytvoření kompletního systému autonomní navigace, který může sloužit jako názorný příklad práce s frameworkem ROS. Byly zde řešeny všechny důležité aspekty vývoje od návrhu síťové architektury, nastavení vývojového prostředí ve virtualizovaném systému, propojení s robotem myAGV, až po vytvoření vlastních ROS balíčků a jejich konfigurace. Součástí implementace byly fáze mapování neznámého prostředí pomocí SLAM, lokalizace v dříve nasnímané mapě prostřednictvím AMCL, navigace pomocí komponenty move_base, a dále funkcionality jako patrolování mezi body nebo návrat na výchozí pozici pomocí python skriptů. Celý systém byl testován v reálných podmínkách modelového bludiště, kde se využívala vizualizace v nástroji RViz pro ověření správné funkce všech komponent. Výsledkem je funkční a modulární systém, na který lze snadno navázat.

Testování systému v reálném prostředí prokázalo spolehlivou funkčnost základních komponent, zejména SLAM a AMCL, které poskytovaly stabilní a přesná data o poloze robotu. Zároveň ale testování odhalilo i určitá omezení, jako např. občasné zpoždění způsobená výpočetní náročností, nebo nestabilní chování balíčku `explore_lite`, který může za určitých podmínek způsobit nechtěné pohyby robotu. Tato slabá místa byla podrobně analyzována a v práci jsou uvedeny návrhy na možná zlepšení, jako je integrace IMU pro lepší odhad pohybu robotu, využití tzv. `recovery behaviors` v rámci `move_base`, přechod na ROS 2 pro zvýšení stability a výkonu nebo rozšíření systému o strojové vidění. V neposlední řadě se jako praktické omezení ukázalo samotné rozdělení výpočetních částí systému a tedy jeho závislost na síťovém připojení k LAN síti, se kterou je potřeba počítat pro skutečné aplikace s tímto implementovaným systémem.

Závěrem lze říct, že všechny stanovené cíle práce, jak ze samotného zadání práce, tak i z vlastní iniciativy, byly úspěšně splněny. Přínos této práce spočívá především v detailním popisu celého procesu návrhu a implementace systému pro autonomní navigaci mobilního robotu v prostředí ROS (od teoretických základů až po reálné testování). Práce zároveň poukazuje na praktické výzvy, se kterými se lze při vývoji podobného systému setkat, identifikuje jejich příčiny a nabízí konkrétní návrhy možného řešení. Výsledný systém tak může sloužit jako základ pro další rozvoj, ale i jako výukový materiál, který studentům přiblíží reálné aspekty práce s mobilní robotikou a frameworkem ROS.

SEZNAM POUŽITÉ LITERATURY

- [1] RUMÍŠEK, Pavel. *AUTOMATIZACE (roboty a manipulátory)*. Online. VUT v Brně, 2003. Dostupné z: https://ust.fme.vutbr.cz/tvareni/img/opory/emm_mechanizace_a_automatizace_roboty_rumisek.pdf. [cit. 2025-01-08].
- [2] KLANČAR, Gregor; ZDEŠAR, Andrej; BLAŽIČ, Sašo a ŠKRJANC, Igor. *Wheeled mobile robotics*. Elsevier, 2017. ISBN 978-0-12-804204-5.
- [3] ŠTĚPÁN, Petr. *Mobilní robotika, Úvod do mobilní robotiky*. Online. ČVUT v Praze, 2010. Dostupné také z: <https://cw.fel.cvut.cz/old/media/mkr/lessons/lesson1.pdf>.
- [4] KOŠNAR, Karel. Mobilní robotika. Online. In: . ČVUT v Praze. Dostupné z: <http://old.roznovskastredni.cz/dwnl/pel2007/06/Kosnar.pdf>. [cit. 2025-01-08].
- [5] *Introduction to AMRs and AGVs*. Online. Vecna Robotics. Dostupné z: <https://www.vecnarobotics.com/amr-vs-agv/>. [cit. 2025-03-02].
- [6] *AGV vs. AMR - jaký je mezi nimi rozdíl?* Online. MiR. Dostupné z: <https://mobile-industrial-robots.com/cs/blog/agv-vs-amr>. [cit. 2025-03-02].
- [7] *Mobile robotics software: Advancements and applications*. Online. Robotnik. 2024. Dostupné z: <https://robotnik.eu/mobile-robotics-software-advancements-and-applications/>. [cit. 2025-03-04].
- [8] *Robotika a automatizace s MiR: Budoucnost průmyslu*. Online. MiR. Dostupné z: <https://mobile-industrial-robots.com/cs/blog/robotika-a-automatizace>. [cit. 2025-03-03].
- [9] *Top 5 Robot Trends 2024*. Online. International Federation of Robotics. 2024. Dostupné z: <https://ifr.org/ifr-press-releases/news/top-5-robot-trends-2024>. [cit. 2025-03-03].
- [10] *ROBOTY 2025: Nejnovější trendy v robotice a příležitosti pro český průmysl*. Online. Svět průmyslu. 2025. Dostupné z: <https://svetprumyslu.cz/roboty-2025-nejnovejsi-trendy-v-robotice-a-prilezitosti-pro-cesky-prumysl/>. [cit. 2025-03-03].
- [11] *Pět současných trendů v robotice*. Online. AUTOMA. 2023. Dostupné z: https://www.automa.cz/cz/casopis-clanky/pet-soucasnych-trendu-v-robotice-2023_02_0_14333/. [cit. 2025-03-03].
- [12] *Robotic trends in 2025: innovations transforming industries*. Online. Robotnik. 2025. Dostupné z: <https://robotnik.eu/robotic-trends-in-2025-innovations-transforming-industries/>. [cit. 2025-03-03].
- [13] CRNOKIĆ, Boris; GRUBIŠIĆ, Miroslav a VOLARIĆ, Tomislav. DIFFERENT APPLICATIONS OF MOBILE ROBOTS IN EDUCATION. Online. In: . International Journal on Integrating Technology in Education, 2017. Dostupné také z: https://www.researchgate.net/publication/320285579_Different_Applications_of_Mobile_Robots_in_Education.
- [14] NOURBAKHSH, Illah R.; CROWLEY, Kevin; WILKINSON, Katie a HAMNER, Emily. The educational impact of the Robotic Autonomy mobile robotics course. Online. 2003. Dostupné z: https://www.ri.cmu.edu/pub_files/pub4/nourbakhsh_illah_2003_1/nourbakhsh_illah_2003_1.pdf. [cit. 2025-01-08].
- [15] *Introduction to Mobile Robotics Lecture 1*. Online. Rose-hulman. Dostupné z: <https://www.rose->

- hulman.edu/~berry123/Courses/ECE425/Spring07_files/Lecture1%20-%20Introduction.pdf. [cit. 2025-01-08].
- [16] CONG, Zhaofeng. Exploring mobile robotics: historical applications and future. Online. Dostupné také z: https://www.researchgate.net/publication/372207639_Exploring_mobile_robotics_historical_applications_and_future.
- [17] *Elephant Robot Product Brochure*. Online. Elephant Robotics. 2022. Dostupné z: <https://download-elephantrobotics.oss-cn-shenzhen.aliyuncs.com/software/Handbook/myAGV%20Product%20Brochure220826.pdf>. [cit. 2025-02-16].
- [18] *Elephant Robotics*. Online. 2023. Dostupné také z: <https://www.elephantrobotics.com/>.
- [19] *MyAGV JN 2023*. Online. Elephant Robotics. Dostupné z: <https://www.elephantrobotics.com/en/myagv-2023-jn-specifications-en/>. [cit. 2025-02-16].
- [20] *Turtle Bot*. Online. Dostupné také z: <https://www.turtlebot.com/>.
- [21] *IRobot Education*. Online. Dostupné také z: <https://edu.irobot.com/what-we-offer/create3>.
- [22] *IROBOT CREATE 3 EDU ROBOT*. Online. Projekt MEDIA. Dostupné také z: https://www.projektmedia.cz/irobot-create-3-edu-robot/?gad_source=1&gclid=Cj0KCQiAr7C6BhDRARIsAOUKifhBlqnURtCSGaSCyLjUTyWo6FAeNHT4dEG3Nvp08oqLJuVniys_fKQaAr6nEALw_wcB.
- [23] *ROSbot 3*. Online. Husarion. Dostupné také z: <https://store.husarion.com/products/rosbot-3?variant=54336172360029>.
- [24] *LIMO PRO*. Online. AgileX Robotics. Dostupné z: <https://global.agilex.ai/products/limo-pro>. [cit. 2025-02-16].
- [25] *Hiwonder JetAuto*. Online. HiWonder. Dostupné z: https://www.hiwonder.com/products/jetauto?variant=39962923860055&srsId=AmBOoovsPlx0-tw8bOniSRws_5gS7HZobTRS07qjY0quBJ7QJKaD91-. [cit. 2025-02-16].
- [26] *ROS/Introduction*. Online. ROS.org. 2018. Dostupné z: <https://wiki.ros.org/ROS/Introduction>. [cit. 2025-03-18].
- [27] SERRANO, Daniel. Introduction to ROS. Online. Dostupné z: <https://www.sto.nato.int/publications/STO%20Educational%20Notes/STO-EN-SCI-271/EN-SCI-271-06.pdf>. [cit. 2025-03-18].
- [28] QUIGLEY, Morgan; GERKEY, Brian a SMART, William D. *Programming Robots with ROS*. Online. O reilly. [cit. 2025-03-18].
- [29] *Getting Started with Robot Operating System (ROS)*. Online. ERC Handbook. Dostupné z: <https://erc-bpgc.github.io/handbook/automation/ROS/ros/>. [cit. 2025-03-30].
- [30] *ROS Architecture*. Online. Linkedin. 2023. Dostupné z: <https://www.linkedin.com/pulse/ros-architecture-muhamad-elalawy-osncf/>. [cit. 2025-03-29].
- [31] LENTIN, Joseph a CACACE, Jonathan. *Masterin ROS for Robotics Programming*. Online. Třetí vydání. Packt, 2021. ISBN 978-1-80107-102-4. [cit. 2025-03-30].

- [32] *How to Create a Publisher Node in ROS Noetic*. Online. Automatic Addison. 2020. Dostupné z: <https://automaticaddison.com/how-to-create-a-publisher-node-in-ros-noetic/>. [cit. 2025-03-30].
- [33] *Part 2: 7 Simple Steps to Create and Build Your First ROS Package*. Online. Medium. 2021. Dostupné z: <https://medium.com/swlh/7-simple-steps-to-create-and-build-our-first-ros-package-7e3080d36faa>. [cit. 2025-03-31].
- [34] *Rqt*. Online. ROS.org. 2016. Dostupné z: <https://wiki.ros.org/rqt>. [cit. 2025-04-02].
- [35] *Robot state publisher*. Online. ROS.org. 2020. Dostupné z: http://wiki.ros.org/robot_state_publisher. [cit. 2025-04-02].
- [36] *Map server*. Online. ROS.org. 2020. Dostupné z: http://wiki.ros.org/map_server. [cit. 2025-04-02].
- [37] *ROS1 vs ROS2, Practical Overview For ROS Developers*. Online. The Robotics Back-End. Dostupné z: <https://roboticsbackend.com/ros1-vs-ros2-practical-overview/>. [cit. 2025-04-03].
- [38] *ROS 1 vs ROS 2 What are the Biggest Differences?* Online. MODEL-PRIME. 2023. Dostupné z: <https://www.model-prime.com/blog/ros-1-vs-ros-2-what-are-the-biggest-differences>. [cit. 2025-04-03].
- [39] *ROS2 vs. ROS1— key differences and which one is better?* Online. Medium. 2022. Dostupné z: <https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a>. [cit. 2025-04-03].
- [40] RICO, Francisco Martín. *Concise Introduction to Robot Programming with ROS2*. Online. CRC Press, 2023. ISBN 978-1-032-26720-3. [cit. 2025-05-20].
- [41] THRUN, Sebastian; BURGARD, Wolfram a FOX, Dieter. *Probabilistic robotics*. Cambridge, Massachusetts: MIT Press, [2006]. ISBN 978-0-262-20162-9.
- [42] CORKE, Peter I. *Robotics, vision and control: fundamental algorithms in Python*. 3rd edition. Springer tracts in advanced robotics. Cham, Switzerland: Springer, [2023]. ISBN 978-3-031-06468-5.
- [43] *Údržba map pro dlouhodobou navigaci mobilních robotů*. Online, Bakalářská práce. Praha: ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE, 2018. Dostupné z: <https://dspace.cvut.cz/bitstream/handle/10467/76139/F3-BP-2018-Halodova-Lucie-Udrzba%20map%20pro%20dlouhodobou%20navigaci%20mobilnich%20robotu.pdf>. [cit. 2025-04-07].
- [44] *PRŮBĚŽNÁ LOKALIZACE A MAPOVÁNÍ POMOCÍ MOBILNÍHO ROBOTU*. Online, DOKTORSKÁ PRÁCE. Brno: VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ, 2008. Dostupné z: <https://dspace.vut.cz/items/4b648610-1cf8-412c-933c-7fa481b9a9f8>. [cit. 2025-04-07].
- [45] FILLIAT, David a MEYER, Jean-Arcady. Map-based navigation in mobile robots:: I. A review of localization strategies. Online. *Cognitive Systems Research*. Dostupné z: <https://www.sciencedirect.com/science/article/pii/S1389041703000081#FIG3>. [cit. 2025-04-07].
- [46] *Occupancy grid map*. Online. ResearchGate. Dostupné z: https://www.researchgate.net/figure/Occupancy-grid-map_fig1_321326800. [cit. 2025-04-07].
- [47] SIEGWART, Roland; NOURBAKHS, REZA a ILLAH A SCARAMUZZA, Davide. *Introduction to Autonomous Mobile Robots*. Online. Second edition. MIT Press, 2011. ISBN 0262015358. [cit. 2025-05-20].

- [48] *Elephant Robotics myAGV Autonomní robot*. Online. RPishop.cz. Dostupné z: <https://rpishop.cz/myagv/5128-elephant-robotics-myagv-autonomni-robot.html#tab-description>. [cit. 2025-05-07].
- [49] *TP-Link Archer AX17 AX1500 Wi-Fi 6*. Online. Alza. Dostupné z: <https://www.alza.cz/tp-link-archer-ax17-ax1500-wi-fi-6-d12555413.htm>. [cit. 2025-05-07].
- [50] *Oracle VirtualBox*. Online. ORACLE. Dostupné z: <https://www.oracle.com/cz/virtualization/virtualbox/>. [cit. 2025-05-07].
- [51] *What is TightVNC?* Online. TightVNC Software. Dostupné z: <https://www.tightvnc.com/>. [cit. 2025-05-07].
- [52] *ROS Melodic Morenia*. Online. Ros.org. 2018. Dostupné z: <http://wiki.ros.org/melodic>. [cit. 2025-05-17].
- [53] *Chapter 6. Virtual Networking*. Online. Virtualbox. Dostupné z: <https://www.virtualbox.org/manual/ch06.html>. [cit. 2025-05-17].
- [54] *Ubuntu install of ROS Melodic*. Online. ROS.org. 2020. Dostupné z: <http://wiki.ros.org/melodic/Installation/Ubuntu>. [cit. 2025-05-17].
- [55] *What is SSH (Secure Shell) and How Does It Work?* Online. TechTarget. Dostupné z: <https://www.techtarget.com/searchsecurity/definition/Secure-Shell>. [cit. 2025-05-17].
- [56] *X11vnc*. Online. Wiki.archlinux. Dostupné z: <https://wiki.archlinux.org/title/X11vnc>. [cit. 2025-05-17].
- [57] *How to run X server using xserver-xorg-video-dummy driver on Ubuntu*. Online. TechOverflow. Dostupné z: <https://techoverflow.net/2019/02/23/how-to-run-x-server-using-xserver-xorg-video-dummy-driver-on-ubuntu/>. [cit. 2025-05-17].
- [58] *What is the X server?* Online. AskUbuntu. Dostupné z: <https://askubuntu.com/questions/7881/what-is-the-x-server>. [cit. 2025-05-17].
- [59] *ROSEnvironmentVariables*. Online. ROS.org. 2019. Dostupné z: <http://wiki.ros.org/ROS/EnvironmentVariables>. [cit. 2025-05-17].
- [60] *How to serve the Network Time Protocol with Chrony*. Online. Ubuntu Server. Dostupné z: <https://documentation.ubuntu.com/server/how-to/networking/serve-ntp-with-chrony/index.html>. [cit. 2025-05-17].
- [61] GRISSETTI, Giorgio; STACHNISS, Cyrill a BURGARD, Wolfram. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. Online. 2007, s. 12. Dostupné z: <http://www2.informatik.uni-freiburg.de/~stachnis/pdf/grisetti07tro.pdf>. [cit. 2025-05-18].
- [62] *Slam gmapping*. Online. GitHub. 2025. Dostupné z: https://github.com/ros-perception/slam_gmapping. [cit. 2025-05-18].
- [63] *Gmapping*. Online. ROS.org. 2019. Dostupné z: <http://wiki.ros.org/gmapping>. [cit. 2025-05-18].
- [64] *MyAGV-Handle control*. Online. Elephant Robotics. Dostupné z: <https://docs.elephantrobotics.com/docs/gitbook-en/13-AdvancedKit/13.2MobileCompoundRobot/13.2.5ps2control.html>. [cit. 2025-05-18].
- [65] *Navigation*. Online. ROS.org. 2020. Dostupné z: <http://wiki.ros.org/navigation>. [cit. 2025-05-18].
- [66] ZHENG, Kaiyu. ROS Navigation Tuning Guide. Online. In: . 2016. Dostupné z: <https://kaiyuzheng.me/documents/navguide.pdf>. [cit. 2025-05-18].

- [67] *Teb_local_planner*. Online. ROS.org. 2020. Dostupné z: http://wiki.ros.org/teb_local_planner. [cit. 2025-05-18].
- [68] *Map_server*. Online. ROS.org. 2020. Dostupné z: http://wiki.ros.org/map_server. [cit. 2025-05-18].
- [69] *Explore_lite*. Online. ROS.org. 2017. Dostupné z: http://wiki.ros.org/explore_lite. [cit. 2025-05-18].
- [70] *Amcl*. Online. ROS.org. 2020. Dostupné z: <http://wiki.ros.org/amcl>. [cit. 2025-05-18].

SEZNAM OBRÁZKŮ

Obrázek 1 Mobilní robot myAGV [18]	24
Obrázek 2 TurtleBot4 Standard (vlevo) a Lite (vpravo) [20]	26
Obrázek 3 iRobot Create 3 [21]	27
Obrázek 4 ROSbot 3 [23].....	28
Obrázek 5 LIMO PRO [24]	29
Obrázek 6 JetAuto [25]	30
Obrázek 7 Příklad ROS grafu [29].....	33
Obrázek 8 ROS architektura [30].....	34
Obrázek 9 Příklad zprávy [27]	35
Obrázek 10 Registrace jednotlivých uzlů pomocí ROS Master [32].....	39
Obrázek 11 ROS souborový systém [31].....	40
Obrázek 12 Příklad ROS workspace [33]	42
Obrázek 13 Obecný algoritmus pro Bayesův filtr [41]	50
Obrázek 14 Obecné schéma Bayesova filtru [44].....	51
Obrázek 15 Příklad lokalizace MCL [41]	52
Obrázek 16 Příklad Metrické mapy formou okupační mřížky [46].....	54
Obrázek 17 Příklad topologické mapy [45]	55
Obrázek 18 Robot myAGV po úpravě.....	61
Obrázek 19 Použitý router [49].....	62
Obrázek 20 Síťová architektura	64
Obrázek 21 Nastavení statické IP adresy VM	69
Obrázek 22 Připojení k myAGV pomocí VNC	71
Obrázek 23 Příklad konfigurace parametrů pomocí <code>rqt_reconfigure</code>	77
Obrázek 24 Vizualizace SLAM s navigací	78
Obrázek 25 Načtení aktuálních souřadnic	80
Obrázek 26 Struktura <i>pose</i> a její uložení do <code>.yaml</code> souboru	80
Obrázek 27 Uložená výchozí poloha a orientace robotu	81
Obrázek 28 Funkce <code>monitor_loop()</code> – ztráta komunikace	81
Obrázek 29 Funkce <code>monitor_loop()</code> – vyčerpání frontierů	82
Obrázek 30 Funkce pro návrat na výchozí polohu	82
Obrázek 31 Vizualizace průzkumu mapy	83
Obrázek 32 Návrh bludiště	85
Obrázek 33 Sestavené bludiště	86
Obrázek 34 Uložená mapa	87

Obrázek 35 Počáteční poloha části při AMCL	87
Obrázek 36 AMCL po zlokalizování pozice	88
Obrázek 37 Vyzvání k zadání počáteční polohy	89
Obrázek 38 Vyzvání k zadání bodů pro patrolování a následně jejich výpis	89
Obrázek 39 Smyčka pro sledování zadaných bodů	89
Obrázek 40 Kontrola vstupu pro návrat na výchozí polohu	90
Obrázek 41 Průjezd bez překážky	91
Obrázek 42 Vyhnutí se překážce	91

SEZNAM TABULEK

Tabulka 1 – Srovnání hlavních vlastností vybraných mobilních robotů	30
---	----

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

SLAM	Simultaneous Localization And Mapping
AGV	Automated Guided Vehicles
AMR	Autonomous Mobile Robot
LiDAR	Light Detection and Ranging
ROS	Robot Operating System
AI	Artificial Intelligence
DOF	Degrees Of Freedom
IMU	Inertial Measurement Unit
API	Application Programming Interface
RVIZ	Ros Visualization
DDS	Data Distribution Service
EKF	Extended Kalman filter
MCL	Monte Carlo Localization
LAN	Local Area Network
VNC	Virtual Network Computing
IP	Internet Protocol
VM	Virtual Machine
GUI	Graphical User Interface
DHCP	Dynamic Host Configuration Protocol
SSH	Secure Shell
AMCL	Adaptive Monte Carlo Localization

SEZNAM PŘÍLOH

Příloha P I: Adresářová sktruktura pracovního prostoru ros na VM

PŘÍLOHA P I: ADRESÁŘOVÁ STRUKTURA PRACOVNÍHO PROSTORU ROS NA VM

