

Návrh a implementace autonomního vozidla využívajícího ROS

Martin Horák

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:	Martin Horák
Osobní číslo:	A22190
Studijní program:	B0714A150006 Aplikovaná informatika v průmyslové automatizaci
Specializace:	Inteligentní systémy s roboty
Forma studia:	Prezenční
Téma práce:	Návrh a implementace autonomního vozidla využívajícího ROS
Téma práce anglicky:	Design and Implementation of an Autonomous Vehicle Using ROS

Zásady pro vypracování

- Prostudujte základní principy autonomních vozidel a platformy ROS.
- Navrhněte a vytvořte mechanickou konstrukci vozidla s využitím technologie 3D tisku.
- Vyberte a integrujte senzory pro měření vzdálenosti a orientace.
- Implementujte navigační algoritmus zajišťující autonomní pohyb vozidla.
- Vytvořte řídicí systém vozidla v ROS a zajistěte jeho funkčnost.
- Otestujte vozidlo v reálných podmínkách a optimalizujte jeho chování.

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

Seznam doporučené literatury:

1. SIEGWART, Roland; NOURBAKHSH, Illah Reza a SCARAMUZZA, Davide. *Introduction to autonomous mobile robots*. 2nd ed. *Intelligent robotics and autonomous agents*. Cambridge, Mass.: MIT Press, c2011. ISBN 9780262295321. Dostupné také z: <https://proxy.k.utb.cz/login?url=http://ieeexplore.ieee.org/xpl/bkabstractplus.jsp?bkn=6267528>.
2. GOVERS, Francis X. *Artificial Intelligence for Robotics: Build Intelligent Robots That Perform Human Tasks Using AI Techniques*. Birmingham, UK: Packt Publishing, 2018. ISBN 9781788835701. Dostupné také z: <http://search.ebscohost.com/login.aspxdirect=true&scope=site&db=nlebk&AN=1881495&authtype=ip,shib&custid=s393675>
3. JOSEPH, Lentin a JOHNY, Aleena. *Robot operating system (ROS) for absolute beginners: robotics programming made easy*. Second edition. New York, NY: Apress, 2022. Dostupné z: <https://doi.org/9781484277508>.
4. NYCE, David S. *Position sensors*. Hoboken, New Jersey: John Wiley & Sons, [2016]. ISBN 9781523114832. Dostupné také z: <https://proxy.k.utb.cz/login?url=https://app.knovel.com/hotlink/toc/id:kpPS000032/position-sensors?kpromoter=marc>.
5. JOSEPH, Lentin. *Learning Robotics Using Python: Design, Simulate, Program, and Prototype an Autonomous Mobile Robot Using ROS, OpenCV, PCL, and Python, 2nd Edition*. 2nd ed. Birmingham: Packt Publishing, 2018. ISBN 9781788629973. Dostupné také z: <https://proxy.k.utb.cz/login?url=https://app.knovel.com/hotlink/toc/id:kpLRPDROS2/learning-robotics-using?kpromoter=marc>.
6. STAPLE, Danny. *Learn robotics programming: build and control AI-enabled autonomous robots using the Raspberry Pi and Python*. Second edition. Birmingham: Packt Publishing, 2021. ISBN 9781839217869. Dostupné také z: <https://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=2731479&authtype=ip,shib&custid=s3936755>.

Vedoucí bakalářské práce: **Ing. Jakub Husár, 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 zaměřuje na vývoj autonomního robotického vozidla, které využívá framework ROS (Robot Operating System). Robot je postaven na platformě Raspberry Pi a využívá 3D tištěné šasi s diferenciálním pohonem řízeným pomocí Arduino. Pro vnímání okolního prostředí je vybaven 2D LiDARem, který slouží k mapování prostoru a detekci překážek. Mapování prostředí a následná autonomní navigace byly realizovány pomocí nástrojů ROS. Robot je schopen plánovat trasu a bezpečně se pohybovat k zadanému cíli v prostředí obsahujícím jak statické, tak i dynamické překážky.

Klíčová slova: autonomní, ROS, robot, senzory, vozidlo, navigace

ABSTRACT

This bachelor thesis focuses on the development of an autonomous robotic vehicle using the ROS (Robot Operating System) framework. The robot is built on the Raspberry Pi platform and uses a 3D printed chassis with differential drive controlled by Arduino. It is equipped with a 2D LiDAR for detection of the surrounding environment, which is used for space mapping and obstacle detection. Mapping and subsequent autonomous navigation in the pre-mapped space were implemented using ROS tools. The robot is able to plan a path and safely navigate to a specified destination in an environment containing static and dynamic obstacles.

Keywords: autonomous, ROS, robot, sensors, vehicle, navigation

Rád bych na tomto místě poděkoval vedoucímu práce Ing. Jakubu Husárovi, Ph.D., za odborné vedení, cenné rady a čas, který mi věnoval při zpracování této práce. Mé poděkování patří také rodičům za jejich trvalou podporu během studia.

„Little drops of water make the mighty ocean.“

Julia Carney

OBSAH

ÚVOD.....	13
I TEORETICKÁ ČÁST	14
1 AUTONOMNÍ VOZIDLO	15
2 MOBILNÍ ROBOTI.....	16
2.1 PODVOZKY MOBILNÍCH ROBOTŮ	17
2.2 KOLOVÉ PODVOZKY	18
2.2.1 Diferenciální podvozek	18
2.2.2 Všesměrový podvozek	19
2.3 KRÁČEJÍCÍ PODVOZKY	19
2.4 TRAJEKTORIE POHYBU	20
2.4.1 AGV	21
2.4.2 AMR.....	21
3 MIKROPOČÍTAČE	23
3.1 FUNGOVÁNÍ MIKROPOČÍTAČE.....	23
3.1.1 Sběrnice.....	23
3.1.2 Paměť programu.....	24
3.1.3 Paměť dat	24
3.1.4 Periferní obvody	24
3.1.5 Adresový prostor	25
3.2 PLATFORMA RASPBERRY PI	25
3.2.1 Typy desek Raspberry Pi	26
3.3 PLATFORMA ARDUINO	26
3.3.1 Využití.....	27
3.3.2 Typy desek Arduino	28
3.3.3 Vývojové prostředí Arduino IDE.....	29
4 ROBOT OPERATING SYSTEM.....	30
4.1 ARCHITEKTURA ROS	31
4.1.1 Uzly a parametry	31
4.1.2 Témata.....	33
4.1.3 Služby a akce.....	34
4.1.4 Workspace a balíčky ROS	36
4.2 PROSTOROVÉ TRANSFORMACE A VIZUALIZACE V ROS	37
4.2.1 Transformace.....	37
4.2.2 URDF	38
4.2.3 RVIZ	39
4.2.4 Gazebo.....	40
4.3 ALTERNATIVY A SOUVISEJÍCÍ PROJEKTY	42
4.3.1 Robotické platformy založené na ROS	42

4.3.2	Frameworky podobné ROS	42
II	PRAKTICKÁ ČÁST	43
5	KONSTRUKCE AUTONOMNÍHO VOZIDLA	44
5.1	MODEL VOZIDLA	45
5.1.1	Návrh modelu	45
5.1.2	3D tisk	46
5.2	HARDWAROVÉ VYBAVENÍ ROBOTA	47
5.2.1	Raspberry Pi	48
5.2.2	Baterie a napájení	49
5.2.3	Arduino UNO a pohon kol	50
5.2.4	Lidar	51
6	PROGRAMOVÉ VYBAVENÍ ROBOTA.....	52
6.1	ZAŘÍZENÍ A JEJICH UMÍSTĚNÍ V SÍTI	53
6.1.1	Instalace Ubuntu a ROS	53
6.1.2	Umístění v lokální síti	54
6.1.3	Tvorba workspace a balíčku.....	56
6.2	TVORBA SOFTWAREOVÉHO MODELU	57
6.2.1	Robot State Publisher	57
6.2.2	Struktura URDF modelu	58
6.3	SOFTWARE SIMULOVANÉHO MODELU	60
6.3.1	Teleoperování robotického vozidla	60
6.3.2	Instalace aplikace Gazebo	61
6.3.3	Řídící systém pro simulované prostředí	64
6.3.4	Spuštění simulace robota v prostředí Gazebo Ignition.....	66
6.4	SOFTWARE REÁLNÉHO ROBOTA	67
6.4.1	Lidar	67
6.4.2	Kamera	68
6.4.3	Arduino.....	69
6.4.4	Řídící systém reálného robotického vozidla	70
6.4.5	Spuštění reálného robota	72
6.5	MAPOVÁNÍ / LOKALIZACE / NAVIGACE	73
6.5.1	SLAM.....	74
6.5.2	Lokalizace	75
6.5.3	Navigace.....	77
7	TESTOVÁNÍ AUTONOMNÍHO VOZIDLA.....	80
7.1	TESTOVÁNÍ NA STATICKÝCH PŘEKÁŽKÁCH.....	81
7.1.1	Mapování.....	81
7.1.2	Nastavení navigace.....	82
7.1.3	Statické překážky – navigace	83
7.2	TESTOVÁNÍ NA DYNAMICKÝCH PŘEKÁŽKÁCH	85
7.2.1	Nastavení navigace.....	85

7.2.2	Dynamické překážky – navigace.....	86
ZÁVĚR		88
SEZNAM POUŽITÉ LITERATURY		89
SEZNAM OBRÁZKŮ		92
SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK		94
SEZNAM PŘÍLOH.....		95

ÚVOD

Autonomní vozidla a roboti představují v dnešní době jednu z nejvýznamnějších technologických oblastí současnosti, která nabývá potenciálu zásadně ovlivňovat způsob, jakým lidé na téhle planetě cestují, pracují a obecně žijí. Vývoj takovýchto technologií směřuje k vytváření systémů schopných samostatného rozhodování a interakce s okolím, snažících se co nejblíže napodobení lidského chování, ne-li přímo jeho podobě.

Nežli si však začneme představovat hrůzné scénáře jako ze sci-fi filmu, ve kterých získá nadvládu nad lidstvem armáda autonomních robotů, zastavme se u toho, co nám tyto technologie v dnešní době přináší a nadále i přinášet budou u většiny odvětví, od zdravotnictví až po zemědělství. Například v logistice a průmyslu přináší mobilní roboti transformaci skladů a distribučních center v tom, že automatizují rutinní úkoly jako je třídění a přeprava. V dopravě pak autonomní automobily slibují zefektivnění dopravy a snížení počtu nehod na silnicích. Tento trend naznačuje, že se podobné přístupy stávají nepostradatelnou součástí dnešní společnosti přinášející efektivitu, bezpečnost, kvalitu ale i spoustu komplikací.

Autonomní technologie tudíž doprovází spousta technologických i společenských výzev. Z technického hlediska jde zejména o spolehlivost, schopnost zpracovávat velké objemy dat a adaptaci na nepředvídatelné situace, které mohou nastat zejména ve chvílích, kdy autonomní systém volí mezi různými možnými následky. Z hlediska společenských aspektů jde o otázky právní odpovědnosti při selhání nebo dopady na zaměstnanost v odvětvích, ve kterých budou místa nahrazeny automatizací. Ať už se tedy vývoj těchto technologií bude odehrávat kamkoli, je nutné myslet na společenskou stránku věci, abychom předešli rizikům spojenými s pokrokem.

Tato práce si klade za cíl prozkoumat principy fungování autonomních vozidel, jejich způsobu řízení a v neposlední řadě snahu sestavit a naprogramovat mobilní vozidlo, které bude schopno pohybu a orientace v uzavřeném prostředí s možností lokalizování a navigování v namapovaném prostředí. Významem práce je pochopení a aplikování funkčních konceptů vedoucích k hlubšímu porozumění problematice a rozvoji dovedností v oboru.

I TEORETICKÁ ČÁST

1 AUTONOMNÍ VOZIDLO

Pojem „autonomní vozidlo“ je v dnešní době nejčastěji spojován s rychlým vývojem a narůstající oblibou samostatně pohybujících se automobilů, schopných částečného nebo plnohodnotného řízení bez nutnosti jakékoli asistence člověka. Ve skutečnosti však tento termín zahrnuje širší spektrum do určité míry samostatně fungujících zařízení, pohybujících se podle předem stanovených nebo vypočítaných trajektorií.

Ať už si tedy představíme auto bez řidiče, které se bude v nedaleké budoucnosti samo pohybovat po silnicích, vozík samostatně rozvázející balíky po logistickém skladu přepravní společnosti bez nutnosti manuálního ovládání, nebo chytrý vysavač, schopný samostatně uklidit místnost, vždy se jedná o formu autonomního mobilního robota.

Autonomní mobilní robot je komplexní mechanický systém schopný samostatného pohybu a operování v prostředí, pro který byl specificky předurčen a naprogramován. Základními prvky robota pak jsou senzory, aktuátory a inteligentní řídicí systém, které umožňují robotovi interagovat s okolním prostředím a vykonávat úkoly zadané člověkem nebo počítačovým systémem.

Senzory hrají klíčovou roli u autonomních prostředků tím, že dokážou vnímat okolní podněty okolo vozidla a poskytovat tak zásadní data pro rozhodovací procesy. Aktuátory pak naopak přeměňují elektronické signály vozidla na fyzický pohyb, a tak umožňují pohyb a otáčení, díky nimž je možné dosáhnout přesného polohování v prostoru. Nejdůležitější součástí je však inteligentní systém, který zajišťuje zpracování dat ze senzorů, jejich interpretaci a následné rozhodování. Funkcí systému je pak plánování trasy, vyhýbání se překážkám a reakce jak na aktuální podněty, tak i na dynamicky se měnící podmínky. Vše dle algoritmů a stupně naprogramování. [1]

2 MOBILNÍ ROBOTI

Mobilní roboti, jak už název napovídá, jsou stroje schopné pohybu v prostoru, schopné interakce s okolím a naprogramované pro plnění různých úkolů.

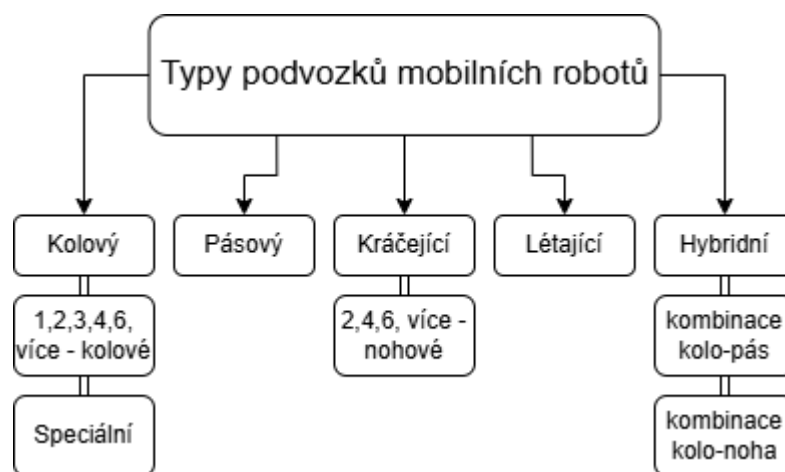
Tyto roboty lze rozdělit do tří hlavních kategorií:

- **Autonomní roboti** vykonávají své úkoly na základě předem zadaných instrukcí zcela samostatně bez nutnosti vnějšího zásahu. Autonomní roboti jsou využíváni v oblastech, kde není možnost nebo efektivní mít neustálý lidský dozor. Využití naleznou například v logistice, průzkumu nebo dopravě.
- **Teleoperovaní roboti** jsou řízeni operátorem, který jim zadává pokyny a má plnou kontrolu nad jejich činností. Tito roboti nemají vlastní inteligenci ani pokročilé algoritmy pro samostatné rozhodování. Nejčastější využití naleznou v nebezpečných a náročných prostředích, nevhodných pro pobyt člověka.
- **Kombinovaní roboti** mají schopnost provádět určité úkoly samostatně, v případě potřeby však mohou být řízeni a ovládáni operátorem. Kombinace výhod autonomního a teleoperovaného robota nalezne využití v prostorech, kde je nutná jak flexibilita a efektivita, tak i možnost ovládání člověkem, pokud je potřeba.

Každá z těchto kategorií má své specifické výhody a je určena pro různé typy úkolů v závislosti na jejich složitosti, požadavcích na autonomii a možnosti lidského dohledu. [2]

2.1 Podvozky mobilních robotů

Při návrhu mobilního robota hraje zásadní roli povaha terénu, na kterém se bude robot pohybovat. Na hladkém nebo mírném povrchu bez zásadních překážek se nejlépe prosadí kolový nebo pásový robot. Naopak v terénu se spoustou překážek je výhodnější využít robota kráčejičího, který si v takovém prostředí poradí lépe. [2]



Obrázek 1 Rozdělení typů podvozků mobilních robotů

(zdroj: vlastní)

Existují i další typy podvozku, tj. např. plazivý, plovoucí, magnetický, šroubovitý, kluzný. Pro tyto podvozky je však využití minimální, skrz svou specifickou konstrukci vhodnou pouze pro omezené aplikace. [3]

Tahle práce je zaměřena na autonomní roboty s kolovým podvozkem, jelikož cílem práce je i takového robota sestavit.

2.2 Kolové podvozky

Nejrozšířenější skupinu mobilních robotů tvoří platformy s kolovými podvozky. Kola lze rozdělit podle pohonu na aktivní poháněná motorem a pasivní, tj. vlečná. Dále je můžeme klasifikovat podle jejich konstrukce a způsobu jejich využití. Mezi nejběžněji využívané typy kol patří: standardní, všesměrové (omniwheel), Mecanum kola, atd. [3]

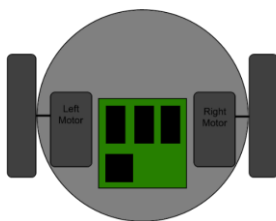
Důležitým kritériem jsou i stupně volnosti a jejich počet. Největší využití u robotů najdou kola s jedním nebo dvěma stupni volnosti. Například u klasického automobilu mají kola u zadní nápravy jeden stupeň volnosti, jelikož rotují pouze kolem své osy. Kola u přední nápravy taktéž rotují kolem své osy, avšak k tomu se otáčí ještě kolem svislé osy, čímž umožňují měnit směr jízdy. Automobily obsahují i další stupně volnosti, jako je volnost ve vertikálním směru, umožňující pohyb tlumiče s pružinou nahoru a dolů. Tohle je však u robotů vzácností. [2]

Podvozky u kolových mobilních robotů pak lze konstruovat různými způsoby. Nejpoužívanější a nejsnáze ovladatelné jsou podvozky diferenciální a všesměrové. Mimo to se setkáváme i s podvozky Ackermannovými (podobné automobilu), synchronními nebo trojkolovými s řízeným předním kolem. [2]

2.2.1 Diferenciální podvozek

U diferenciálního způsobu řízení je podvozek tvořen dvěma koly připevněnými k robotu na opačných stranách. Obě kola jsou aktivně poháněna, nezávisle na sobě.

V případě rotace obou kol ve stejném směru a ve stejné rychlosti, dochází k pohybu robota dopředu nebo dozadu. Pokud jedno kolo stojí a druhé rotuje, robot se začne otáčet okolo stojícího kola. Pokud se však otáčí obě kola stejnou rychlostí, pouze v opačném směru, robot začne rotovat kolem osy nápravy. Diferenciální podvozek bývá většinou doplněn o jedno nebo dvě vlečná kola s více stupni volnosti, která zajistí stabilitu a odstraní nutnost robota balancovat. [4]



Obrázek 2 Diferenciální typ podvozku [4]

2.2.2 Všesměrový podvozek

Roboti se všesměrovým podvozkem mají možnost pohybu ve všech směrech bez omezení. Umožňuje jim to konstrukce jejich podvozku, složená ze Švédských kol (Mecanum), které tvoří série pogumovaných válečků nastavených v úhlu 45° ke kolu. Kombinací rotací jednotlivých kol, které jsou poháněny každé motorem zvlášť, docílí podvozek možnosti pohybu ve všech směrech.

Všesměrový podvozek je hojně využíván v podmínkách, které vyžadují velmi těsnou manévrovatelnost. Použití zahrnuje vysokozdvížné vozíky, autonomní roboty, invalidní vozíky, resp. tam kde je potřeba pohyb do stran bez otáčení. [2]



Obrázek 3 Všesměrový typ podvozku [14]

2.3 Kráčející podvozky

Pohyb po nohách je charakterizován bodovými kontakty se zemí. Kvalita povrchu mezi těmito body není důležitá do doby, dokud robot dokáže udržet dostatečnou světlou výšku nad terénem.

Kráčející podvozky vycházejí z biologicky úspěšných systémů schopných překonávat různé terény a obtížné stoupání. S rostoucími schopnostmi však rostou i nevýhody, kterými jsou vysoká spotřeba energie a mechanická složitost. Nohy kráčejících podvozků, které obsahují několik stupňů volnosti, musí unést hmotnost robota a být schopné jej plynule zvedat a spouštět.

Kráčející mechanismus lze konstruovat pomocí dvou (stejně jako u lidí), čtyř, šesti nebo více nohou (viz obrázky 4 a 5). [2]



Obrázek 4 Robot Atlas [15]

Na obrázku 4 je zobrazen krácející dvounohý humanoidní robot Atlas vyvíjený americkou společností Boston Dynamics s možností kompletních pohybů, jako je skákání, běhání nebo provádění salt.



Obrázek 5 Robot Spot [16]

Krácející čtyřnohý mobilní robot Spot od společnosti Boston Dynamics, ilustrován na obrázku 5, je určený pro průzkumné operace, inspekce, dokumentace a monitorování. Díky své modulární platformě umožňuje množství konfigurací.

2.4 Trajektorie pohybu

Efektivní organizace interní logistiky ve skladech a výrobních zařízeních je pro firmy klíčovým faktorem. Proto se i zde nasazuje automatizace přepravy, která má za úkol optimalizovat přesun materiálu, což vede k urychlení celé výroby a uvolnění lidských zdrojů k činnostem s vyšší hodnotou.

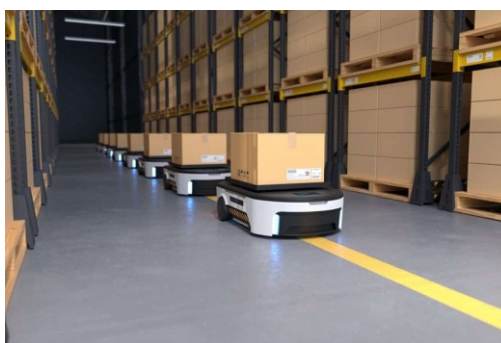
V rámci trajektorie, podle které se automatizovaná vozidla mohou pohybovat, rozlišujeme roboty AGV a AMR. Automatizovaná řízená vozidla (AGV) jsou zařízenými prostředky ve velkých zařízeních, ve kterých je třeba opakovaných a průběžných dodávek

materiálu po prostoru. Avšak v rámci velké firmy je počáteční investice zanedbatelná ve srovnání s objemem práce vykonané optimalizovanými a automatizovanými procesy. Velkým konkurentem pro AGV v dnešní době začínají být, díky své flexibilitě a sofistikovanosti, autonomní mobilní roboti (AMR). Oba způsoby mají za úkol přemístit materiál z jednoho místa na druhé, avšak rozdíly mezi nimi skrývají jejich výhody a nevýhody. [5]

2.4.1 AGV

Vozidla AGV se vyznačují pohybem po pevných trasách, které jsou tvořeny magnetickými pásky, barevnými stopami nebo vodiči vedenými v podlaze na které pak vozidlo zabudovanými senzory reaguje. Tyto trasy musejí být dopředu definované a žádají instalaci před samotným nasazením vozidel. S tím taktéž souvisí problematika změny pevných tras, která si v případě změn vyžádá pozastavení výroby spojenou s nutnou odstávkou vnitřní logistiky.

Díky definovaným trasám však vozidla AGV potřebují jen minimální inteligenci a dají se ovládat jednoduchými pokyny, jež jsou vybaveny, nebo jsou jim zasílány bezdrátově, či v podobě změny průtoku elektrického proudu ve vodičích, které následují. Minimální inteligence a pevné vedení tras má bohužel za následek zastavení přepravy v případě překážky na trase, kterou vozidlo dokáže detekovat, ale nedokáže ji obejít. [5]



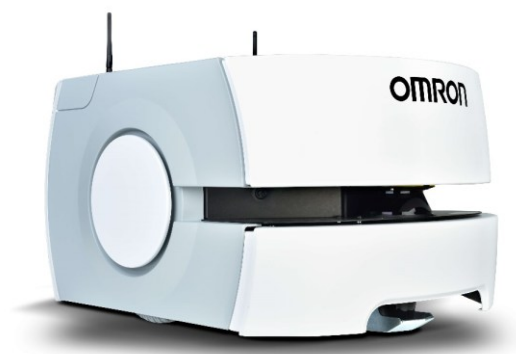
Obrázek 6 AGV Roboti [17]

2.4.2 AMR

Vozidla AMR naproti vozidlům AGV nemusí spoléhat na pevně stanovené trasy. Jejich podstata vychází z možnosti samostatné orientace v prostředí. Jejich sofistikovaný software k mapování a následné navigaci v prostředí využívá kamery, senzory a laserové skenery, díky čemuž dokáže detekovat překážky a efektivně je objet, aniž by došlo k

zastavení logistiky materiálu nebo snížení celkové produktivity. Úkoly robota pak mohou být řízeny pomocí rozhraní, které je dodáváno přímo s robotem. Díky němu pak obsluha nebo servisní technik může upravit konfiguraci softwaru, která upřednostňuje vykonávání zakázek a přiřazuje úkoly robotovi. Tato možnost tedy dovoluje organizacím plně vlastnit a ovládat robota, jelikož mohou kdykoli přemapovat prostředí pracoviště a změnit funkci robota. Tím odpadá nutnost rozsáhlých úprav infrastruktury a nechtěných prodlení ve výrobě s odstavkou spojených.

Nevýhodou AMR je pak možnost nepředvídatelného pohybu v prostředí. Robot se sice dá v softwarovém prostředí konfigurovat k pohybu ve vyznačených oblastech, i tak může nastat situace, kdy nedokáže rozeznat pohyb pracovníků a ostatních vozidel na pracovišti včas a v tu chvíli se stane překážkou. Důležitým krokem společnosti je tedy správné využití těchto prostředků pro nepřetržitý, optimalizovaný, a hlavně bezpečný chod výroby. [5]



Obrázek 7 AMR Omron [18]

3 MIKROPOČÍTAČE

Mikropočítač je název elektronického zařízení, obsahujícího mikroprocesor, který plní funkci hlavní výpočetní jednotky. V jednoduchosti se jedná o počítač malého měřítka, navržený pro provádění úloh vyžadujících nízký výpočetní výkon. Mikropočítač se využívá v různých aplikacích, od osobních počítačů po vestavěné (embedded) systémy, tj. systémy, které jsou součástí přístrojů, vozidel, telefonů apod. [6]

3.1 Fungování mikropočítače

Na desce mikropočítače jsou pak podle použití integrovány, kromě procesoru, i jiné části, jako je paměť programu, paměť dat, sběrnice nebo periferní obvody. [6]

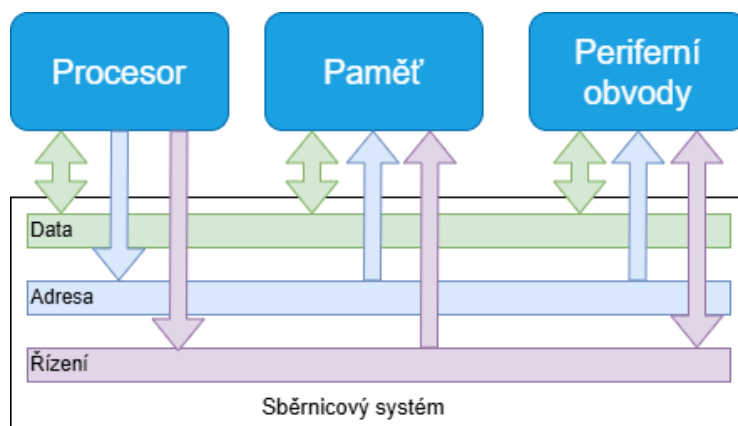
3.1.1 Sběrnice

Propojení jednotlivých součástí mikropočítače je pak zajištěno sběrnici. Sběrnice je soustava vodičů, schopna přenášet elektrické signály, v našem případě informací, mezi částmi počítače. Obecně lze sběrnice rozdělit na sběrnici adresovou, datovou a řídicí.

Adresová sběrnice slouží k adresování jednotlivých obvodů mikropočítače, resp. vybírá, který z obvodů aktuálně komunikuje s procesorem. Dle šíře adresové sběrnice pak rozlišujeme maximální počet obvodů, kterých můžeme na mikropočítač implementovat.

Datová sběrnice slouží k transportu dat mezi jednotlivými obvody s procesorem. Šířku zmíněné sběrnice nejčastěji rozlišujeme v jednotkách bytů.

Řídicí sběrnice pak slouží k řízení aktivity jednotlivých jednotek (tj. zápis, čtení atd.). Řídicí signály nejčastěji generuje procesor, není to však podmínkou. [6]



Obrázek 8 Zjednodušené schéma počítače

(zdroj: vlastní)

3.1.2 Paměť programu

Paměť programu je složená z instrukcí (příkazy sloužící k provádění požadované činnosti počítače) a různých konstant používaných v programu. Daná paměť je nejčastěji realizovaná za pomoci typu ROM (Read Only Memory) nebo typu RAM (Random Access Memory).

Paměť typu ROM obsahuje data, u kterých se nepočítá se změnou, tj. nejdou změnit a povětšinou bývají zapsány již od výroby. V dnešní době jsou nahrazovány paměti typu FLASH, EPROM nebo EEPROM, u kterých lze změna provést, avšak jen několikrát nebo omezeně.

Pokud je programovou paměť potřeba měnit, využívá se paměť typu RAM. I v takovém případě je však nutná paměť typu ROM, která slouží ke spuštění programu a možnosti zápisu do a čtení z programové paměti RAM. [6]

3.1.3 Paměť dat

Paměť dat je prostor pro dočasné uložení dat, např. pro hodnoty vstupů, výstupů, mezivýsledků apod. Bývá realizován pomocí paměti typu RAM.

V závislosti na architektuře adresového prostoru (viz níže), pak při využití typu RAM, jak u paměti dat i paměti programu, může být využita společná sdílená paměťová jednotka. [6]

3.1.4 Periferní obvody

Periferní obvody jsou pak zařízení připojené ke sběrnici mikropočítače, umožňující interakci s okolním světem nebo rozšiřující jeho funkčnost. Tyto obvody mohou být přímo implementované na desce nebo snadno připojitelné pomocí konektorů. Mezi periferní obvody pak můžeme řadit například: [6]

- **Vstupní zařízení** umožňující přísun dat a signálu do počítače, např. klávesnice, dotykové displeje, ovladače ale i snímače vnějších veličin
- **Výstupní zařízení** schopné prezentovat aktuální dění (displeje) a akční členy schopné ovlivňovat okolí (motory, LED diody)
- **Časovací a čítací** obvody sloužící k přesnému měření času a generování signálů
- **Analogové převodníky** sloužící ke zpracování analogového signálu na digitální, a naopak

- **Komunikační obvody** sloužící k transportu dat z mikropočítače do jiných zařízení (UART, USB, SPI, I²C, Ethernet)

3.1.5 Adresový prostor

Paměť mikropočítače si můžeme představit jako jednorozměrné pole míst pro uložení hodnot. Každé z těchto míst pak nabývá jedinečné adresy, která slouží pro jednoznačné adresování procesorem. Množinu těchto adres pak můžeme označovat jako adresový prostor. Počítač pak může mít jeden nebo více paměťových prostorů.

V případě jednoho adresového prostoru lze sdílet programovou, datovou a periferní paměť dohromady. V tomto případě hovoříme o tzv. von Neumanově architektuře.

Naopak při využití vícero paměťových prostorů lze programovou, datovou a periferní paměť oddělit. Takle změna pak procesoru dovolí přistupovat do jednotlivých pamětí současně. Nutností je ale existence vícero sběrnic. Tato architektura se pak označuje jako Harvardská. Architektura tedy ovlivňuje výkon a efektivitu mikropočítačů. [6]

3.2 Platforma Raspberry Pi

Raspberry Pi je řada malých a cenově dostupných jednodeskových počítačů vhodných pro širokou škálu projektů, od domácí automatizace po robotiku. Mikropočítač Raspberry Pi lze srovnat se slabšími stolními počítači, přičemž je zachována velikost o málo větší než platební karta. Díky tomu je snadno přenosný a integrovatelný. Vyvinuto bylo do dnešní doby již několik generací tohoto mikropočítače, které se liší převážně výkonem a použitím. [7]

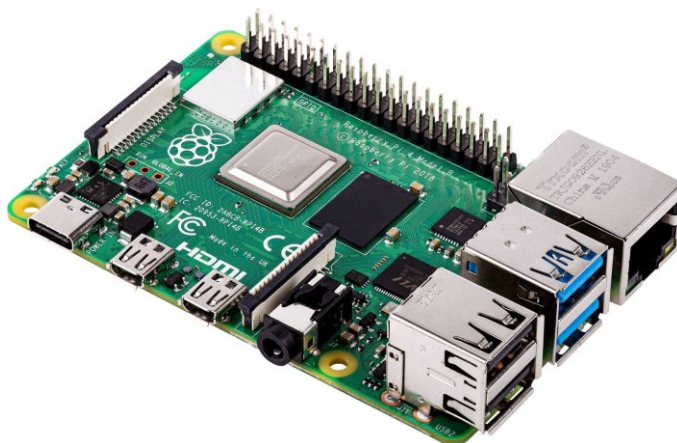
Raspberry Pi je dostatečně výkonný pro běh kompletního operačního systému založeného na Linuxu. Oficiálním operačním systémem je Raspberry Pi OS (vychází z Debianu), není však problém nainstalovat linuxové distribuce Ubuntu, Ubuntu MATE, DietPi, Kali Linux, čistý Debian nebo specifické OS typu Osmc či RetroPi. Srdcem mikropočítače je pak mikroprocesor založený na architektuře ARM, tj. architektura používána například u mobilních telefonů. [8]

Zmíněné mikropočítače obsahují, stejně jako platformy typu Arduino nebo ESP8266, univerzální vstupně-výstupní piny (GPIO), které slouží pro připojení a ovládání různých zařízení. Mimo to však můžeme u většiny modelů narazit na vývod pro připojení monitoru (HDMI nebo mikro HDMI), ethernetový port, zvukový port nebo USB porty pro připojení

standardních zařízení, jako jsou myš, klávesnice, monitor, reproduktory nebo displeje. Výjimkou jsou pak například modely Zero nebo Pico, které Ethernetový nebo standardní USB port nemají. U většiny modelů taktéž nechybí bezdrátové připojení Wi-Fi nebo Bluetooth. [7]

3.2.1 Typy desek Raspberry Pi

Společnost Raspberry Pi k dnešnímu dni nabízí širokou škálu modelů, mezi něž patří univerzální Raspberry Pi, které se v dnešní době chlubí již 5. generací, rozšířeny o varianty A/A+ nebo B/B+. Dále jsou k dispozici kompaktní a úsporné modely Zero, Zero W a Zero 2 W, které jsou určené pro minimalistické aplikace. V neposlední řadě jsou zde modely Pico a Pico W, které jsou mikrokontroléry, nelze na ně nahrát operační systém a jsou určené pro specifické nízkoúrovňové elektronické projekty. Tyhle modely se liší zejména ve výkonech, funkcích, velikosti, portech a spotřebě energie. [8]



Obrázek 9 Typické rozložení desky Raspberry Pi [19]

3.3 Platforma Arduino

Arduino je open-source platforma založená na jednoduchých jednodeskových počítačích a intuitivním vývojovém prostředí, sloužícím k tvorbě softwaru. Jádrem všech vývojových desek Arduino jsou mikrokontroléry ATmega od výrobce Atmel. Tyhle mikrokontroléry lze naprogramovat v softwarové aplikaci (Arduino IDE) za pomoci programovacího jazyka Wiring (podobu frameworku v jazyce C++).

Na rozdíl od platformy Raspberry Pi není Arduino zamýšleno jako plnohodnotný stolní počítač. Řídicí program se tedy musí vyvinout zvlášť na plnohodnotném počítači a do

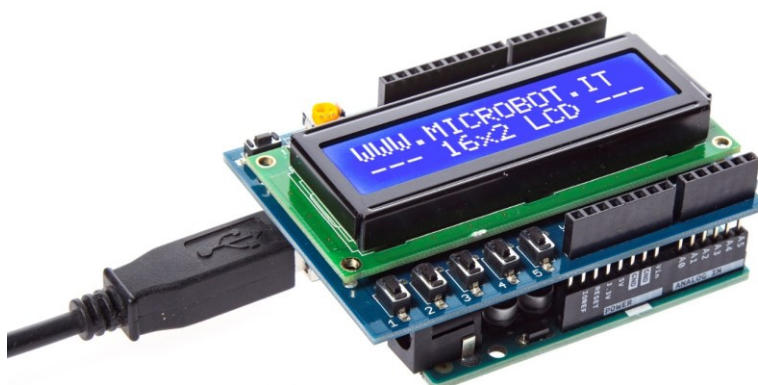
mikropočítače Arduino je posléze nahrán. Po nahrání je následně program spuštěn a obvykle běží ve smyčce, která například neustále kontroluje stav svého okolí a na změny reaguje. [9]

3.3.1 Využití

Původním záměrem vzniku mikropočítače Arduino bylo vytvoření nástroje pro rychlé vytváření prototypů, určeného pro uživatele bez hlubších znalostí programování a elektrotechniky. S postupným nárůstem uživatelů a oblíbenosti se však platforma začala přizpůsobovat a rozšiřovat o desky určené novým potřebám a výzvám. Postupem času tedy šlo narazit na desky určené pro IoT aplikace, robotiku, 3D tisk ale i módní průmysl.

V dnešní době je komunita tvůrců, studentů, programátorů a profesionálů natolik rozsáhlá, že lze jen stěží narazit na aplikaci, která by nebyla na Arduino proveditelná. Vývojové desky jsou standardně vybaveny analogovými a digitálními vstupy a výstupy, které neumožňují připojení monitorů, klávesnic nebo myší, jako jsme zvyklí u klasických počítačů nebo chytrých telefonů. Za to však podporují připojení různých LED diod, displejů, snímačů nebo servomotorů.

S velkou oblibou se taktéž využívají Arduino shiedly, což jsou rozšiřující moduly navržené tak, aby se jednoduše daly připojit na základní desku. Tyto moduly dokážou rozšířit možnosti Arduino desky přidáním specifických funkcí nebo periférií bez nutnosti zbytečného hardwaru a kabeláže. Mezi nejoblíbenější rozšiřující moduly se řadí například motorový, LCD, senzorový, výukový nebo relé shield. Možností je pak i využití prototypového shiedlu s pájivým nebo nepájivým polem. [9]



Obrázek 10 Arduino UNO s připojeným LCD shieldem [20]

3.3.2 Typy desek Arduino

Arduino v dnešní době nabízí širokou škálu vývojových desek, které se liší velikostí, funkcemi, výkonem nebo počtem vstupních a výstupních pinů. V závislosti na typu projektu pak tedy záleží na uživateli, kterou z modelů vybere. K dispozici má originální Arduino nebo lacinější klony od dalších dovozových výrobců. Platí však, že originální modely budou fungovat vždy a za každých okolností, zároveň mají i potřebné certifikáty, které zaručí bezchybné použití i v profesionálních podmínkách. Naopak klony Arduino desek mívají často problémy s nekompatibilitou hardwaru a ovladačů, což způsobuje potíže při nahrávání programu na desku. Je tedy důležité posoudit, jestli je pro náš projekt důležité upřednostnit kvalitu originálního výrobku nebo nižší cenu méně spolehlivějšího klonu.

Nejznámějšími a nejvyžívanějšími deskami jsou Arduino Uno, Nano, Mega, Due nebo Leonardo.

Arduino Uno je jedna z nejpoužívanějších desek v rodině Arduino a díky své univerzálnosti je oblíbená mezi začátečníky i pokročilými uživateli. Je vybavena mikrokontrolérem ATmega328P, který spolu s 32KB Flash pamětí dodává dostatečný výkon pro většinu projektů. Deska obsahuje 6 analogových vstupů pro připojení senzorů a 14 digitálních vstupů a výstupů, ze kterých 6 dokáže podporovat PWM (Pulzně šířková modulace). Pro desku Arduino Uno existuje široká škála připojitelných shieldů, které rozšiřují její možnosti. [9]

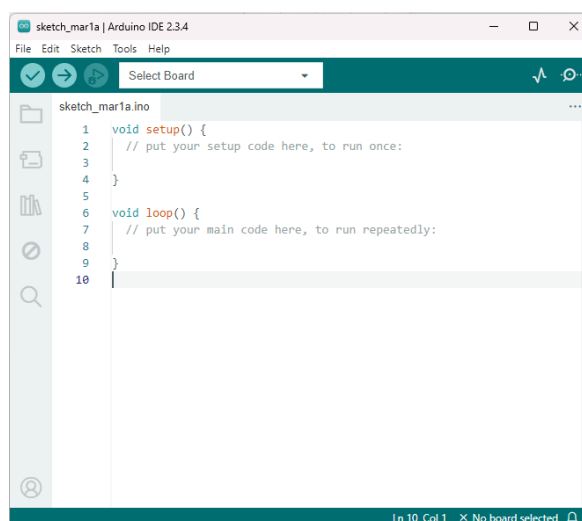


Obrázek 11 Arduino Uno [21]

3.3.3 Vývojové prostředí Arduino IDE

Součástí úspěchu Arduina je i jednoduché vývojové prostředí IDE (Integrated Development Environment = Integrované vývojové prostředí) složené z textového editoru pro psaní kódu, textové konzole s možností sériového portu a lišty s příkazy pro běžné funkce a řadu nabídek.

Programování v oficiálním prostředí Arduino IDE (viz Obrázek 12) se pak nijak výrazně neliší od jiných vývojových prostředí. V textovém editoru uživatel napíše návrh kódu pro desku Arduino. Těmto návrhům se říká „sketche“ a ukládají se s příponou „.ino“. Sketch je následně zkompileován a nahrán do mikropočítače Arduino. O nahrání, či případných chybových hlášeních je uživatel informován v textové konzoli.



Obrázek 12 Vývojové prostředí Arduino IDE

(zdroj: vlastní)

4 ROBOT OPERATING SYSTEM

Robot Operating System (dále jen „ROS“) je open-source vývojová platforma (framework) s definovanou architekturou a funkcionalitou určená pro vývoj robotických aplikací. ROS však nelze považovat za plnohodnotný „operační systém“, tak jako například Windows, macOS nebo Linux. Ve skutečnosti se jedná o sadu nástrojů a knihoven určenou k podpoře vývoje robotického softwaru běžícího na klasickém operačním systému, například na Linuxu. [11]

Flexibilní vlastnosti, např. v počátečních fázích vývoje, však platformu omezují při použití v komerčních a bezpečnostně náročných aplikacích. ROS v důsledku toho tedy není navrženo pro zajištění bezpečnosti, spolehlivosti a reálného času, což jsou klíčové požadavky v těchto oblastech. [4]

ROS obsahuje širokou škálu funkcí, které dokážou vývoj robotických zařízení usnadňovat. Zásadní vlastností je interprocesová komunikace, která slouží ke snadnému vyměňování zpráv mezi jednotlivými procesy. V případě ROS se jedná o uzly (nodes) a témata (topics). Uzel představuje jednotlivý proces, který dokáže odebírat nebo publikovat data – zprávy (messages) skrz témata. Mimo to lze mezi uzly využít možnosti služeb (services) pro synchronní komunikaci a akcí (actions) pro komunikaci asynchronní. Neméně cennými jsou i parametry (parameters) sloužící ke konfiguraci chování jednotlivých uzlů a tím i celé aplikace za běhu. [10]

Velkou výhodou je i podpora více programovacích jazyků, zejména pak oblíbených jazyků C++ a Python, čímž je přístupnější širšímu publiku. To taky souvisí s aktivní komunitou, která poskytuje podporu a sdílení znalostí prostřednictvím oficiálních webových stránek a fór. ROS obsahuje velké množství předpřipravených algoritmů, například SLAM, AMCL, řízení motorů nebo navigace, což ulehčuje práci programátorům a zkracuje čas vývoje. Vývojářům je taky k dispozici několik aplikací a nástrojů. Za zmínku stojí aplikace RViz pro vizualizaci robotických dat, aplikace Gazebo, umožňující realistickou simulaci prostředí, simulující gravitaci, nárazy či tření, nebo rqt GUI, která poskytuje rozhraní pro monitorování a správu různých aspektů robotického systému. [10]

4.1 Architektura ROS

Architektura ROS je založena na distribuovaném systému. To znamená, že jednotlivé části robota (např. senzory, řídicí algoritmy nebo akční členy) představují jednotlivé procesy, tj. uzly, které mezi sebou dokážou komunikovat pomocí komunikačních mechanismů, jako jsou témata, akce a služby. Tento způsob umožňuje snadnou rozšiřitelnost systému o další komponenty bez nutnosti přímé integrace do monolitického programu. [13]

V rámci kontextu architektury je nutné rozlišovat mezi verzemi ROS 1 a ROS 2. Starší verze ROS 1 funguje na principu centrálního komponentu ROS Master, což je v podstatě hlavní uzel spravující registraci ostatních uzlů a ovlivňuje jejich komunikaci. Tento princip je výhodnější pro menší systémy, jelikož v případě selhání hlavního uzlu selže celý systém. ROS 2 řeší tento problém odstraněním ROS Masteru a místo toho používá komunikační standard DDS (Data Distribution Service), který zajišťuje decentralizovanou komunikaci mezi uzly bez nutnosti využití centrálního uzlu ke koordinaci komunikace. Uzly tedy mohou komunikovat napřímo, což vede k větší efektivitě a spolehlivosti. V této práci bude popisován a využíván pouze novější ROS 2. Dále tedy jen jako „ROS“. [12]

Hlavními komponenty architektury ROS jsou:

4.1.1 Uzly a parametry

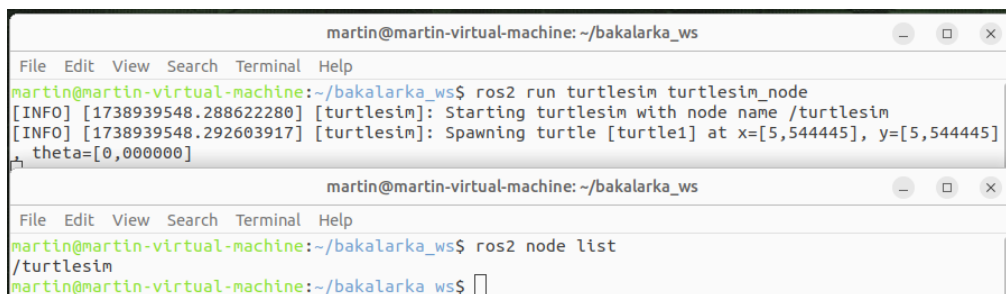
Jak už bylo dříve řečeno, ROS je tvořen několika menšími procesy, schopnými mezi sebou komunikovat, přesněji odebírat nebo publikovat data jiného procesu, tyto procesy nazýváme uzly. Každý uzel si můžeme představit jako malý program vytvořený tak, že může vykonávat specifickou úlohu v systému. Například sbírání dat ze senzorů nebo klávesnice, zasílání signálů do ovladače motorů, výpočet nadcházející trajektorie navigací nebo vizualizace robota v programu.

Pro práci s uzly se v příkazovém řádku využívají příkazy:

```
#Pro spuštění uzlu v příkazovém řádku:  
ros2 run <jméno_balicku> <jméno_uzlu>  
#Pro výpis všech spuštěných uzlů pak:  
ros2 node list
```

Obecně lze říct, že v systému ROS, lze spouštět mnoho jednotlivých uzlů, přičemž nemusí běžet pouze na jednom zařízení. Jednou z výhod ROS je totiž fakt, že uzly mohou být spuštěny na různých zařízeních, které jsou připojené ke stejné síti pomocí routeru nebo

switche, za pomoci ethernetového kabelu nebo Wi-Fi. Uzly na rozlišných zařízeních pak mezi sebou mohou komunikovat.



```

martin@martin-virtual-machine: ~/bakalarka_ws
File Edit View Search Terminal Help
martin@martin-virtual-machine:~/bakalarka_ws$ ros2 run turtlesim turtlesim_node
[INFO] [1738939548.288622280] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1738939548.292603917] [turtlesim]: Spawning turtle [turtle1] at x=[5,544445], y=[5,544445]
theta=[0,000000]

martin@martin-virtual-machine: ~/bakalarka_ws
File Edit View Search Terminal Help
martin@martin-virtual-machine:~/bakalarka_ws$ ros2 node list
/turtlesim
martin@martin-virtual-machine:~/bakalarka_ws$

```

Obrázek 13 Spuštění uzlu ROS systému příkazovým řádkem

(zdroj: vlastní)

Každý z uzlů taktéž obsahuje vlastní sadu parametrů, které ovlivňují jeho chování. Tyhle parametry se dají sledovat a konfigurovat při spouštění či za běhu uzlu v příkazovém řádku, přímo v uzlu prostřednictvím Pythonu, či C++ nebo v parametrických souborech YAML. V závislosti na funkci uzlu se dají měnit například parametry hardwarové, komunikační, uživatelské, grafické nebo dynamické.

Pro konfiguraci parametrů pomocí příkazového řádku lze využít příkazů:

```

#Parametry lze nastavit při spouštění uzlu:
ros2 run <jméno_balicku> <jméno_uzlu> --ros-args -p <parametr>:=hodnota
#Zmena parametru lze provést během běhu uzlu:
ros2 param set /<jmeno_uzlu> <parametr> hodnota
#Parametry, které lze měnit pak zjistíme pomocí:
ros2 param list

```

Při vývoji rozsáhlých systémů obsahujících více uzlů by však spouštění samostatných uzlů, obsahujících přidání parametry, jeden za druhým bylo velice zdlouhavé a bylo by k tomu potřeba spousta otevřených oken příkazových řádků. Proto ROS poskytuje systém zvaný „launching“, který umožňuje nakonfigurovat a spouštět vícero uzlů najednou. Spouštěcí systém může být založený na Python nebo XML souborech a vypadá takto: [13]

```

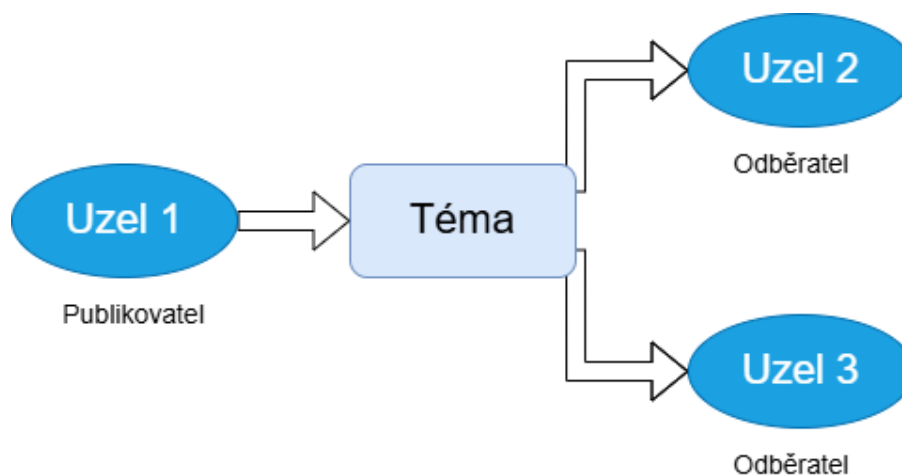
#Pro spouštění z balíčků:
ros2 launch <nazev_balicku> <nazev_spousteciho_balicku>

```

4.1.2 Témata

V prostředí ROS mohou mezi sebou jednotlivé uzly komunikovat prostřednictvím témat a zpráv. Témata jsou hlavním mechanismem pro asynchronní komunikaci mezi uzly, fungující na modelu „publikace-odebírání“ (publish-subscribe). Tento model umožňuje efektivní sdílení dat mezi různými částmi systému bez přímé vazby mezi odesílatelem a příjemcem. Každé téma obsahuje určitý datový typ zpráv, které mohou publikovat jeden nebo více uzlů. Stejně tak může toto téma jeden nebo vícero uzlů odebírat a to tak, aniž by odebírající uzel věděl, který uzel do tématu publikoval. To umožňuje časem přidávat nové uzly, aniž by se na straně ostatních uzlů musely provádět nutné změny.

Témata se nejčastěji využívají pro průběžné sdílení dat, například dat senzorických, ovládacích nebo stavových. I tady platí, že při správné konfiguraci lze komunikovat mezi vícero zařízeními, připojenými ke stejné síti. [13]



Obrázek 14 Komunikace skrz téma (topic) mezi uzly (nodes)

(zdroj: vlastní)

Stejně jako u uzlů, tak i u témat existují určité příkazy, díky kterým lze v terminálu pracovat s tématy:

```
#Seznam dostupných témat lze zobrazit pomocí příkazu:  
ros2 topic list  
  
#Přidáním argumentu „-t“ za příkaz lze dosáhnout zobrazení jednotlivých  
témat spolu s datovými typy zprávy:  
ros2 topic list -t  
  
#K zobrazení dat, která jsou publikována v tématu lze použít:  
ros2 topic echo <jmeno_tematu>
```

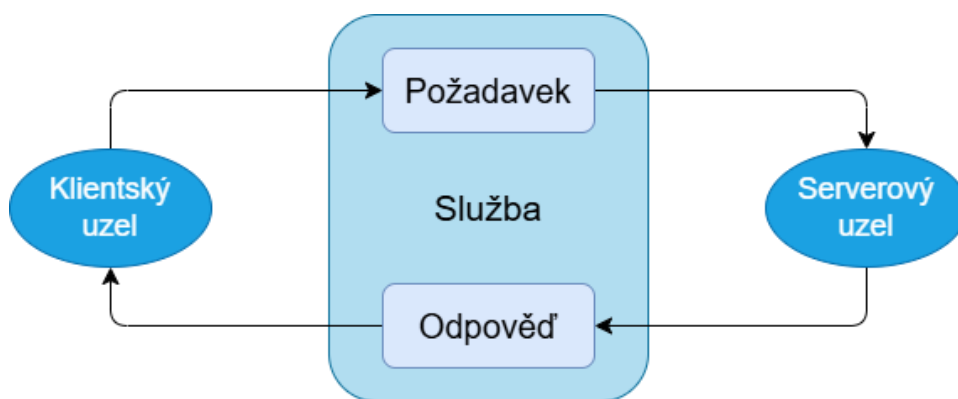
4.1.3 Služby a akce

Služby jsou další z možností komunikačních metod mezi uzly v ROS. Zásadním rozdílem mezi tématy a službami je způsob komunikace mezi jednotlivými uzly, kdy témata fungují na modelu „publikace-odebírání“ a služby na modelu „požadavek-odpověď“ (request-reply). Zatímco tedy témata umožňují nepřetržité sdílení dat mezi uzly, služby poskytnou data, jen pokud si je daný klientský uzel vyžádá.

Mechanismus služeb se zakládá na synchronní komunikaci mezi uzly, přesněji mezi uzlem serverovým a uzlem klientským. Server plní funkci uzlu, nabízejícího určitou službu a plnícího požadavky klientských uzlů, které tyhle požadavky zašlou a čekají na následnou reakci. Využití služeb je vhodné v případech, pokud je potřeba získat jednorázová data (např. aktuální teplotu na senzoru nebo souřadnice v prostoru, na kterých se robot nachází) či provést jednorázovou akci (aktivace senzoru, reset nebo spuštění robota). Ačkoli jsou služby standardně synchronní, tj. čekají na odpověď před tím, než mohou pokračovat, nabízejí i možnost asynchronního chování pomocí speciálních funkcí, které dokážou potlačit blokující chování. [12]

Užitečnými příkazy pro práci se službami v příkazovém řádku jsou:

```
#Seznam dostupných služeb v systému zobrazíme příkazem:  
ros2 service list  
#Zavolat určitou službu můžeme manuálně příkazem:  
ros2 service call /<jmeno_služby> <zpráva_služby>
```



Obrázek 15 Komunikace skrz služby

(zdroj: vlastní)

Kombinací vlastností témat a služeb vzniká další mechanismus sloužící pro komunikaci mezi uzly v prostředí ROS. Tenhle mechanismus se nazývá akce a je určena pro dlouhotrvající úlohy s možností zpětné vazby průběžnou aktualizací stavu a možností úlohu předčasně ukončit. To je zásadní rozdíl oproti službám, které poskytují odpověď okamžitě a neumožňují sledování průběhu úlohy.

Samotná akce je založena na modelu „požadavek-zpětná vazba-výsledek“ a sestává se, obdobně jako služba, z uzlu serverového a klientského. Server v tomto případě přijímá úlohu od klienta, vykonává ji a je zároveň povinen posílat průběžně zpětnou vazbu a finální výsledek. Taktéž musí správně zareagovat v případě požadavku zrušení vykonávané úlohy. Klient je pak uzel, který serveru požadovanou úlohu zasílá a očekává průběžná aktualizovaná data a finální výsledek.

Akce jsou primárně tedy vhodné pro úlohy, které trvají déle, v případě potřeby mohou být předčasně zrušeny a průběžně zasílají data zpětné vazby. Příkladem využití může být například komunikace mezi řídicím systémem robota a jeho podsystémem zabezpečujícím pohyb. Řídicí systém odešle podsystému požadavek o cílové pozici, podsystém pak průběžně řídicí systém informuje o jeho postupu, a nakonec zašle informaci o případném splnění úkolu. [12]

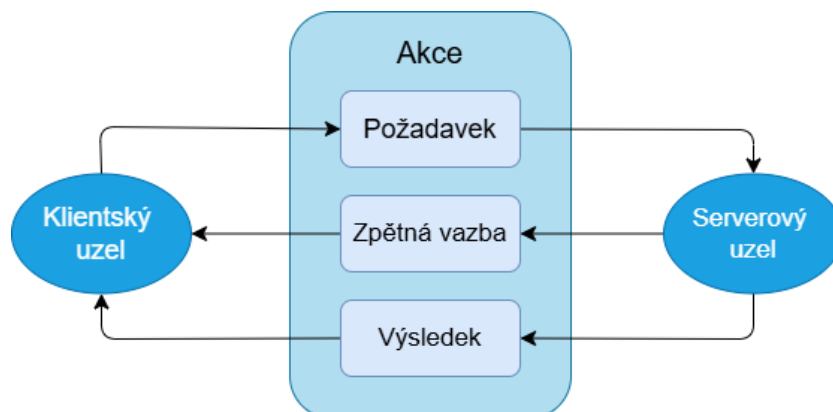
Jednotlivé akce je taky možné ovládat skrze příkazy v terminálu:

```
#Seznam dostupných akcí v systému zobrazíme příkazem:
```

```
ros2 action list
```

```
#Akce se dají zavolat manuálně taktéž v příkazovém řádku, sktruktura je ale dost často na dané službě a jejím typu:
```

```
ros2 action send_goal <action_name> <action_type> <values>
```



Obrázek 16 Komunikace skrz akce

(zdroj: vlastní)

4.1.4 Workspace a balíčky ROS

Z předešlých poznatků je známo, že je systém vývojového prostředí ROS sestaven z několika uzlů, popsanych parametry a navzájem komunikujících metodami, tj. tématy, akcemi a službami. Pro lepší organizaci tohohle softwaru jsou využívány balíčky (packages). Balíček je způsob, jak seskupit související soubory dohromady. Obvykle jsou to skupiny souborů tvořící jeden projekt, obsahující například uzly, parametrické, spouštěcí a konfigurační soubory, modely, softwarové knihovny a seznamy zmíněných souborů. Mimo to mohou obsahovat metadata, instrukce a licence v případě, že jde o balíčky jiného tvůrce. Každý balíček tímto způsobem tvoří samostatnou jednotku, která jde sdílet a použít v jiném zařízení. Jeden, či více balíčků můžeme spravovat uvnitř workspace.

K práci s projekty ROS se využívá workspace, představující pracovní prostor, ve kterém se uchovávají balíčky. Uživatel si může pracovní prostor koncipovat podle libosti, obecně však platí, že každý projekt má vlastní workspace, který může obsahovat více balíčků. Důvodem využití zmíněného prostředí, krom toho, že tvoří přehledný a efektivní prostor pro práci se softwarem, je také nutnost přítomnosti kompilačních souborů. Bez nich by totiž samotný ROS nemohl najít uživatelské balíčky a propojení komponentů systému by bylo složitější. K tomu je určený kompilační nástroj „colcon“, který při kompilaci načte z adresářové složky zdrojová data a zpracuje je na výsledná binární data a knihovny do složky „install“. Podobně pak uloží mezivýsledky do složky „build“. Mimo to je vytvořena složka „log“, do které se při pozdějším běhu uzlů ukládají data vhodné pro ladění chyb. [12]

Kompilací se v adresáři „install“ vytvoří soubor obsahující všechny informace, které ROS potřebuje k bezproblémovému přístupu k balíčkům a případnému následnému spuštění. Načtení tohohle souboru je nutné při každém otevření příkazového řádku a spustí se příkazem:

```
source <workspace>/install/setup.bash
```

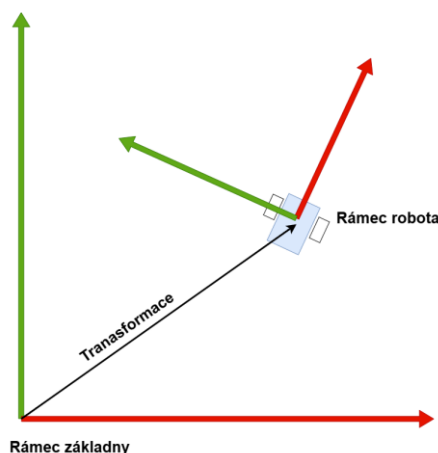
4.2 Prostorové transformace a vizualizace v ROS

V distribuovaném systému postaveném na komunikaci mezi uzly prostřednictvím témat, akcí a služeb vzniká potřeba efektivně reprezentovat a vizualizovat model robota a jeho součástí. ROS nabízí sadu nástrojů a aplikačních rozhraní, které tento úkol usnadňují a umožňují pracovat s prostorovými transformacemi, popisem struktury robota i jeho vizualizací.

Klíčovým prvkem je systém prostorových transformací (tf), jenž umožňuje sledovat vzájemné polohy a orientace jednotlivých souřadnicových systémů (frames) v reálném čase. Tyto dynamické transformace doplňuje formát URDF (Unified Robot Description Format), který slouží k popisu strukturální a kinematické konfigurace robota. Pro vizualizaci modelu robota a jeho prostorových transformací je k dispozici RViz — výkonné grafické rozhraní, které poskytuje intuitivní prostředí pro sledování aktuálního stavu robota, jeho ladění a ověřování chování. Důležitou roli hraje i simulační prostředí Gazebo, které umožňuje testovat robotické modely v realistických podmínkách, včetně interakcí s okolím. [12]

4.2.1 Transformace

Transformační souřadnice (též Transformace) představují nástroj pro správu a sledování souřadnicových systémů v rámci robotického systému. Tento mechanismus dokáže určit a popsat pozici a natočení souřadnicového systému vůči jinému. Tím způsobem pak může vznikat určitá hierarchie na sebe navázaných systémů stromové struktury, zjednodušující umístění jednotlivých bodů v prostoru. Příkladem může být pozice a natočení souřadnic robotického vozidla vůči souřadnicím základny nebo souřadnicím jiného vozidla. [12]



Obrázek 17 Transformace souřadnic

(zdroj: vlastní)

V terminologii vývojového prostředí ROS nazýváme jednotlivé souřadnicové systémy souřadnicovými rámci (frames). Tyhle rámce jsou pak spojeny s určitým objektem nebo částí robota (např. základna robota, základna prostředí, kamera, robotické rameno). Mezi těmito rámci se následně provádí transformace. Jedná se o matematické operace popisující transformaci polohy a natočení jednoho rámce do rámce jiného. K řešení zmíněných transformací je v ROS využívána knihovna „tf2“, která zajišťuje jejich zpracovávání společně s časovou interpolací. Transformace jsou definovány jako zprávy obsahující informace o rámcich, časové razítko a v neposlední řadě translační a rotační hodnoty, reprezentované kvaternionem. [12]

Uzly v ROS využívají knihovnu tf2 pro sdílení dat transformačních mezi souřadnicovými rámci. Tyto data jsou publikována prostřednictvím témat („/tf_static“ a „/tf“), kdy transformace mohou být buď statické nebo dynamické, v závislosti na tom, jestli se v čase mění nebo ne.

Téma „/tf_static“ vzniká při spuštění robotického systému, jakmile je načten URDF model, například prostřednictvím uzlu „robot_state_publisher“. Toto téma obsahuje statické transformace mezi jednotlivými částmi systému, které se nemění během chodu robota – například vztahy mezi pevně připojenými částmi robota. Transformace v tomto tématu jsou publikované nástrojem „StaticTransformBroadcaster“ a zůstávají neměnné, dokud nedojde k přeprogramování systému.

Naopak, téma „/tf“ obsahuje dynamické transformace, které začínají být publikovány nástrojem „TransformBroadcaster“ za předpokladu, že robot obsahuje pohyblivé části, jako jsou lineární nebo rotační stupně volnosti. Tyhle transformace se v průběhu času mění v závislosti na pohybu robota, tzn. v jeho aktuátorech a senzorech, což umožňuje sledování změn v prostoru mezi jednotlivými souřadnicovými rámci. [13]

4.2.2 URDF

Při vývoji robotického systému je mnohdy nezbytné znát i fyzické vlastnosti robota. Pro jednoduchost a efektivnost lze všechny tyhle informace seskupit do jednoho souboru. V ROS k tomuto účelu slouží formát URDF (Unified Robot Description Format), založený na programovacím jazyku XML, sloužící k popisu robota. Tento formát detailně specifikuje strukturu robota, jeho pevné části – linky (links), jejich vzájemné propojení spoji (joints) a fyzikální vlastnosti modelu.

Základním prvkem formátu URDF jsou linky, představující pevné části robota. Každá linka obsahuje informace nezbytné k jejímu popisu, rozdělené do tří hlavních kategorií:

- **Vizuální vlastnosti** specifikující aspekty pevných částí robota, nutné pro samotnou vizualizaci v aplikačních rozhraní. Těmito aspekty jsou geometrické tvary částí a jejich poloha a barva.
- **Kolizní vlastnosti** popisující geometrii používanou v případě detekce kolizí. Takle geometrie bývá často zjednodušená oproti geometrii vizuální.
- **Inerciální vlastnosti** obsahující klíčové vlastnosti ovlivňující simulace a dynamiku robota. Mezi ně patří informace o hmotnosti, její rozložení a setrvačnosti.

Propojení jednotlivých pevných částí následně zaštiťují jednotlivé spoje. Ty definují vztah mezi linkami – například umožňují jejich vzájemný pohyb a tím vytvářejí jednotlivé stupně volnosti (degrees of freedom, DOF). U URDF lze definovat jak rotační, tak i translační spojení (revolute, prismatic, continuous joints). To je klíčové například u robotického ramene, kde linky představují jednotlivé segmenty a spoje vytváří kloubní spojení mezi nimi.

Spoje však nemusí být vždy jen pohyblivé – URDF umožňuje využít i pevné spojení (fixed joints), které propojují dvě linky bez možnosti jakéhokoli pohybu. To se hodí například pro připevnění senzorů nebo jiných komponent k základní části robota a tím definovat jejich pevnou pozici vůči zbytku robota. [13]

4.2.3 RVIZ

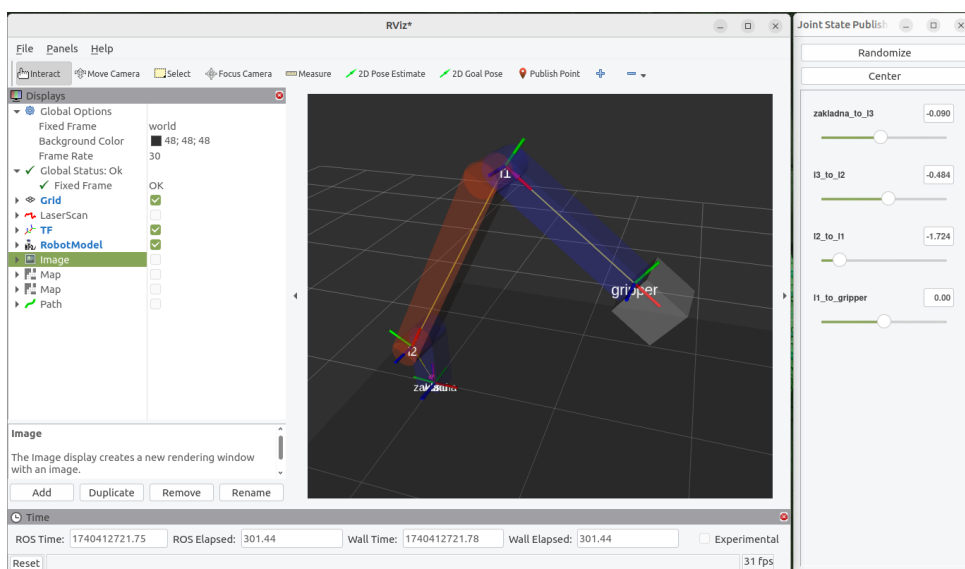
RViz (ROS Visualization) je výkonné grafické prostředí v ROS, sloužící k zobrazení a prozkoumávání dat různých uzlů robotického systému. Umožňuje uživatelům sledovat robotické modely, plánování trajektorie navigačních systémů, data ze senzorů, mapy, transformace a další informace spojené s řešenou problematikou.

Prostředí aplikace RViz je rozdělené do několika částí. V prostřední části se nachází hlavní vizualizační okno, na kterém se zobrazují vybrané prvky scény. Tyhle prvky si může vybrat uživatel v levé části, v záložce „Displays“. Mezi zobrazenými prvky může být dříve zmíněný robotický model URDF, senzorová data z lidarů a kamer, dále různé mapy, mřížky nebo nástroje spojené s navigací. Všechny prvky se pak dají v průběhu využívání

rozhraní upravovat a jsou dostupné v závislosti na funkcích a naprogramování robotického modelu. Podstatnou výhodou je i možnost sledovat prvky transformací mezi jednotlivými souřadnicovými rámci, díky čemuž lze ověřit správné definice vztahů mezi jednotlivými částmi robota a okolí. [12]

U instalačního balíčku ROS, je RViz dostupný pouze v případě plnohodnotné verze (ros-desktop). Spustit se následovně dá v okně příkazového řádku příkazem:

```
ros2 run rviz2 rviz2
#Nebo rovnou zkratkou:
Rviz2
```



Obrázek 18 Vizualizační prostředí RViz

(zdroj: vlastní)

4.2.4 Gazebo

Gazebo je výkonné simulační prostředí určené pro robotické systémy. Jeho funkce spočívá v provádění realistických simulací na importovaném modelu robota v 3D prostředí, zahrnující překážky, kolize a působení gravitace. Díky spolupráci s ROS lze pak robota kontrolovat, tj. odebírat data ze senzorů a posléze ho ovládat, díky čemuž lze pak se simulovaným robotickým systémem pracovat tak, jako by šlo o reálné zařízení.

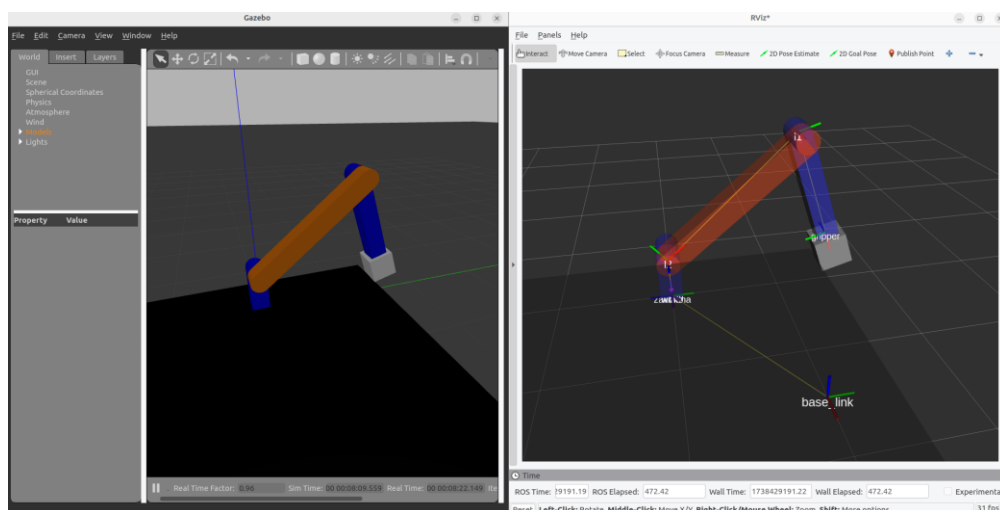
Pro zobrazení robota v prostředí Gazebo je potřeba převést model ve formátu URDF do formátu zvaného SDF (Simulation Description Format). Zatímco URDF dokáže popisovat

pouze samotný model robota, SDF dokáže popisovat i svět, ve kterém se robot nachází, a ostatní modely v něm představující překážky. O tenhle převod se stará samotné Gazebo, které čerpá z dat témat, které publikuje „robot_state_publisher“, načítající soubor URDF.

V současné době existují dvě hlavní verze tohoto simulátoru – Gazebo Classic a Gazebo Ignition (používaným názvem je pouze Gazebo). Gazebo Classic je dlouhodobě používaná verze, která je stále oblíbená komunitou uživatelů ROS, avšak tato starší verze softwaru je podporovaná pouze do verze ROS 2 Humble. Novější verze Gazebo Ignition je pak navržena tak, aby architekturu simulátoru modernizovala, za pomoci pokročilejšího grafického rozhraní a lepších nástrojů pro simulaci složitějších prostředí.

Komunikace mezi ROS a simulátorem Gazebo je založena na speciálních pluginech, sloužících k propojení simulovaných prvků Gazebo s ROS tématy a uzly. Přesný tvar komunikace ale záleží na verzi simulátoru. V Gazebo Classic jsou využívány pro každý typ dat ze senzorů separátní pluginy, které pak publikují data do ROS témat. Naopak Gazebo Ignition využívá systému, který nahrazuje separátní pluginy a publikuje data do Gazebo témat, které jsou za pomoci mostu „ros_gz_bridge“ převedeny do ROS témat. Obdobně pak probíhá ovládání robota, kdy ROS uzly posílají příkazy, které jsou prostřednictvím pluginů předávány simulovanému modelu.

Integrace Gazebo s ROS je klíčová pro vývoj a prototypování robotických systémů, umožňující testování nejrůznějších modelů a jejich řídicích či navigačních algoritmů v bezpečném a kontrolovaném prostředí bez rizika poničení hardwaru či ohrožení osob v reálném světě. [13]



Obrázek 19 Gazebo Classic (vlevo) a RViz (vpravo)

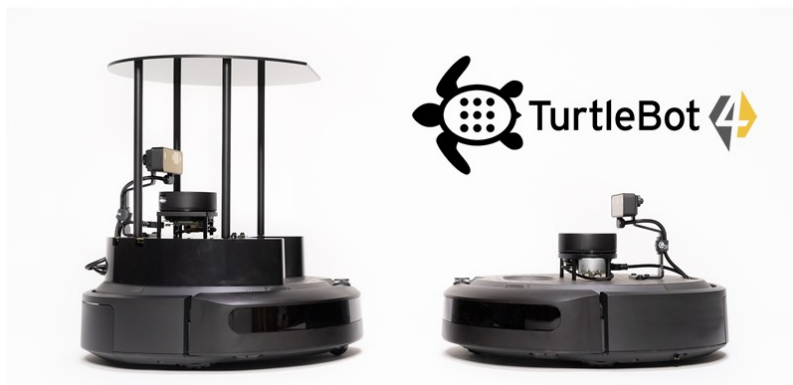
(zdroj: vlastní)

4.3 Alternativy a související projekty

Vývojová platforma ROS je díky svým možnostem vhodným základem pro celou řadu robotických projektů. Na ROS byly k dnešnímu dni postaveny celé platformy, jak komerční, tak i komunitní, sloužící k mnoha účelům. Mimo to však existují i alternativní softwarové systémy pro robotiku, poskytující obdobné řešení.

4.3.1 Robotické platformy založené na ROS

Jednou z nejznámějších komerčních platform postavených s použitím ROS je TurtleBot, původně vyvinutý firmou Willow Garage. Jedná se o platformu vhodnou především pro vývoj a výzkum, s širokou podporou, dokumentací a přístupným softwarem. Nejnovější verze TurtleBot 4 využívá hardwarovou platformu od společnosti iRobot (Roomba). [28]



Obrázek 20 TurtleBot 4 [28]

Vedle komerčních produktů existuje i řada hobby projektů, například Articubot One. Tento projekt je velmi oblíbený komunitou, především mezi začátečníky s malými zkušenostmi v problematice. Výhodou je i konstrukce tohoto projektu, která dokazuje, že realizace není složitá a jsou k ní potřebné běžně dostupné díly. [13]

4.3.2 Frameworky podobné ROS

Vedle ROS existuje několik dalších softwarových platform vhodných k vývoji robotů. Tyto platformy se liší svou architekturou, podporou a použitím. Mezi nejznámější patří například YARP, Microsoft Robotics Developer Studio, MOOS, Player, Orocos nebo CARMEN. Některé z nich již jsou zastaralé, jiné naopak stále naleznou využití ve specifických oblastech, například v námořní nebo průmyslové robotice. Přestože je tedy ROS mezi vývojáři velmi populární, pro některé specializované aplikace nemusí být nejvhodnější volbou. [29]

II PRAKTICKÁ ČÁST

5 KONSTRUKCE AUTONOMNÍHO VOZIDLA

Hlavním cílem této bakalářské práce je návrh a realizace autonomního robota, který bude schopen mapovat okolní prostředí a následně se v něm samostatně pohybovat. Součástí řešení je vytvoření řídicího systému využívajícího nástroje a knihovny vývojového prostředí ROS. Tento systém bude dále rozšířen o komponenty pro mapování a implementaci navigačních funkcí, které umožní přesun robota z výchozí pozice do cílové. Kromě samotného řídicího systému práce zahrnuje také návrh a konstrukci fyzického modelu vozidla, integraci senzorů, akčních členů a řídicích komponent, a jejich uvedení do provozu.

Prvotním záměrem bakalářské práce bylo naprojektovat pásové vozidlo s kovovým šasi. I když by byl takový projekt působivý a realistický, postupem času z něj sešlo. Důvodem opuštění návrhu byly jeho náklady, složitost výroby a hlavně fakt, že pásové vozidlo není efektivní při odometrii. Proces odometrie je založen na principu transformace dat sdílenými enkodéry umístěnými povětšinou na hřídelích pohonných motorů vozidla, sloužících k poskytnutí informací o změně pozice či orientace samotného robota. U pásového vozidla by mohlo docházet k proklouzávání pásu na kluzkých a hladkých površích, čímž by mohlo docházet k razantní chybě odometrie, která je pro můj případ řešení problematiky důležitá, jak pro řízení robota, tak i pro samotné navigování. Řešení chyby odometrie nebo její úplné odstavení pro určité úkony je možné, avšak by způsobovalo jiné nepřesnosti řízení a také by vyžadovalo daleko komplexnější řešení problematiky.

Adekvátní náhradou pásového vozidla se postupem času stalo vozidlo kolové, založené na diferenciálním způsobu řízení, avšak s implementovanými koly s gumovou vrstvou pro styk s podložkou zabraňující prokluzu zmíněných kol. Konstrukce diferenciálního kolového podvozku je obecně jednoduchá. Principem jsou dva na sobě nezávislé hnací motory pohánějící kola na stranách vozidla. Tato kola jsou doplněny jedním nebo více stabilizačními body, realizovanými kuličkami nebo všesměrovými otočnými kolečky.

5.1 Model vozidla

Na trhu je v dnešní době spousta stavebnic, které přímo představují nebo se dá díky nim sestavit diferenciální podvozek robota. V mém případě je snahou tenhle podvozek navrhnout a vyrobit za pomoci technologie 3D tisku, který by měl zajistit vyřešení neduhů zmíněných stavebnic, zajistit robustnost samotného vozidla a v neposlední řadě umožnit vhodné umístění samotného hardwaru tak, aby nedošlo k jeho porušení či zničení.

5.1.1 Návrh modelu

Návrh modelu robotického autonomního vozidla vychází z požadavků na mobilní platformu schopnou navigace a mapování vnitřních prostor. Proto se jedná o model menších rozměrů, čemuž taky přispívá úprava na podvozek s diferenciálním způsobem řízení, který je do takových prostor vhodný svými ovladatelnými vlastnostmi.

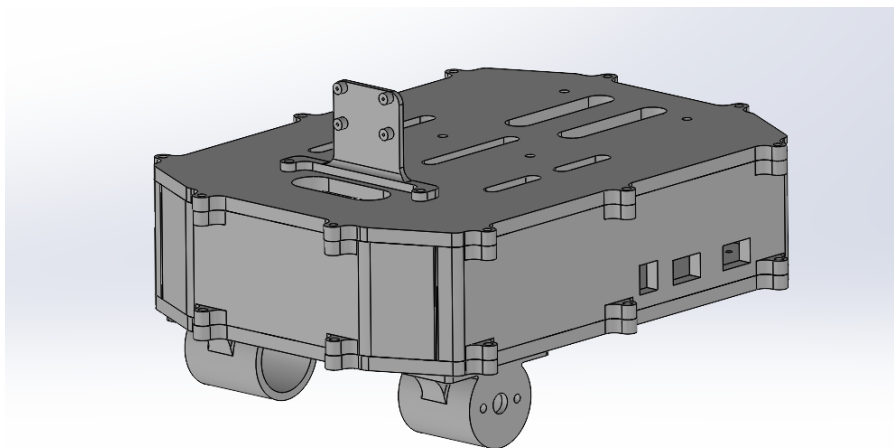
Samotný model je navržen tak, aby byl lehký, robustní a umožňoval snadnou montáž akčních členů, senzorů a ostatních členů elektroniky včetně baterie. Připevnění těchto elektronických součástí je většinou realizováno pomocí šroubů, distančních sloupků nebo vrutů do plastu. K tomu slouží předpřipravené montážní díry v prvcích konstrukce. Hlavními prvky konstrukce jsou:

- **Základní nosná deska** – hlavní část šasi nesoucí veškeré řídicí komponenty elektroniky (Raspberry Pi 4, Arduino UNO, H-bridge, baterie)
- **Držáky motorů** – část zajišťující stabilní uchycení pohonných jednotek k nosné desce
- **Boční panely** – postranní prvky sloužící k ochraně vnitřní elektroniky, spojující nosnou desku s vrchní krycí deskou a dodávající určitou estetiku robotu
- **Vrchní krycí deska** – část sloužící k uchycení lidarového snímače a přední kamery robota
- **Uchycení kamery** – prvek pro uchycení Raspberry Pi Camery, pro případ rozšíření nebo obohacení projektu o zpracování videa

Veškeré prvky modelu robotického vozidla jsou vymodelované do jisté míry modulárně a to tak, aby byla možnost každou část v případě potřeby vyměnit či upravit. Tuto modularitu

umožňují úchyty s průchozími otvory v prvcích konstrukce modelu, sloužící k vzájemnému spojení pomocí šroubů s maticemi.

Model robotického vozidla byl v průběhu vývoje navrhnout ve dvou verzích. Druhá verze vznikla z důvodu pořízení motorů s rozdílnými jmenovitými otáčkami a rozměry. Taktéž byly na druhou verzi modelu osazeny boční panely. Celý návrh byl vymodelován v programu SolidWorks.



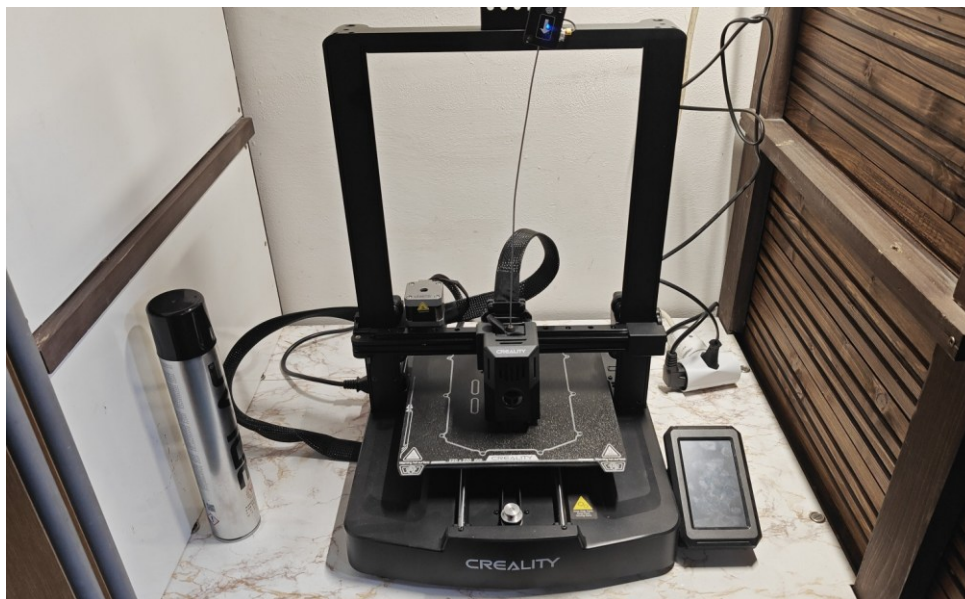
Obrázek 21 Návrh robotického vozidla v prostředí Solidworks

(zdroj: vlastní)

5.1.2 3D tisk

Samotný tisk konstrukce byl realizován na 3D tiskárně Creality Ender-3 V3 KE s použitím filamentu Verbatim PLA v barvě hliníkové-šedé. PLA bylo zvoleno z důvodu jednoduchosti tisku a dostatečné pevnosti pro naše využití. V plánu bylo využít pro tisk i filament PETG, který by měl obecně odolávat i většímu mechanickému zatížení a nárazu. Robotické vozidlo z materiálu PLA však během stavby a využívání nevykazovalo nedostatky a obecně působí robustním dojmem.

Pro přípravu modelu k tisku byl použit slicer, taktéž od Creality, pro nastavení parametrů a převedení formátu přímo pro tiskárnu. Tisk byl realizován ve výšce vrstvy 0,2 mm, s výplní v rozmezí cca. 20–30 %, při teplotě trysky 220 °C a teplotě podložky 50 °C.



Obrázek 22 Tisk modelu na 3D tiskárně

(zdroj: vlastní)

5.2 Hardwarové vybavení robota

Pro sestavení hardwarové architektury autonomního robotického vozidla byly využity prvky známých platforem Raspberry Pi a Arduino Uno, popsané v teoretické části práce. Obě platformy jsou častou volbou pro podobné projekty, a to nejen díky své dostupnosti, ale hlavně široké podpoře komunity a bohaté dokumentaci.

Řídicí funkci v systému robota zajišťuje Raspberry Pi, které se ve spolupráci s klasickým počítačem, skrz bezdrátovou síť Wi-fi, stará o vyšší úroveň řízení. Toto řízení obsahuje zpracování dat, jejich vyhodnocení a následnou kontrolu nižší úrovně. Nižší úroveň je v systému tvořena mikropočítačem Arduino Uno, které slouží k ovládání motorů a k Raspberry Pi je připojeno prostřednictvím USB, přes které také přijímá sériově zasílané příkazy.

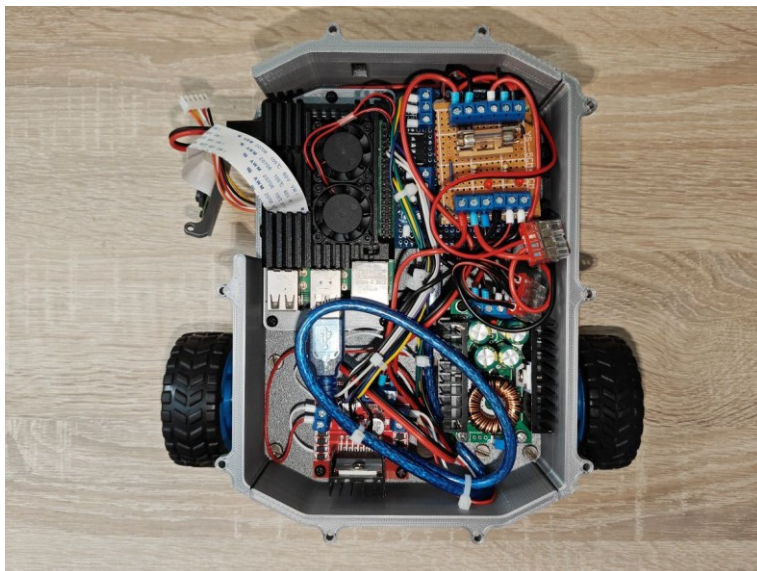
Kromě periférií sloužících k řízení pohybu robota, jsou k Raspberry Pi připojeny i periférie senzorické. Pro tento účel je využit LiDAR připojený za pomoci USB a kamera, která snímá dění před vozidlem, připojená k rozhraní CSI.

5.2.1 Raspberry Pi

Jako mozek celého robota byl zvolen mikropočítač Raspberry Pi 4 s 4GB RAM pamětí. Starší verze mikropočítače byla zvolena i přestože v době koupě byla na trhu dostupná novější verze Raspberry Pi 5 a to z důvodu neznalosti jistých kompatibilit mezi verzemi operačních systému s verzemi ROS a simulačními programy. Raspberry Pi 5 má totiž odlišnou architekturu oproti své starší verzi, což znamená, že nepodporuje některé starší verze LTS operačního systému Ubuntu, například verze 20.04 nebo 22.04.

I když má čtvrtý model mikropočítače horší výkonnostní parametry než model novější, běh systému ROS a dalších potřebných programů zvládá. Nehledě na to, že navigační a lokalizační, případně simulační programy místo něj zařizuje bezdrátově komunikující počítač. Pro jistotu byl na Raspberry Pi implementován hliníkový chladič s ventilátorem (viz Obrázek 23), zajišťující odvod tepla z desky mikropočítače.

Kromě chladiče byla pořízena i originální Raspberry Pi kamera V2. Kamera je připojena skrz CSI rozhraní a umožňuje sledování prostoru před vozidlem. Využití najde při pohybu robota mimo zorné pole uživatele, který může sledovat obraz z kamery v programu RViz na počítači. Implementovaná kamera se pak může hodit při dalších projektech, souvisejících s rozšířením schopností vozidla.



Obrázek 23 Elektronika uvnitř robotického vozidla

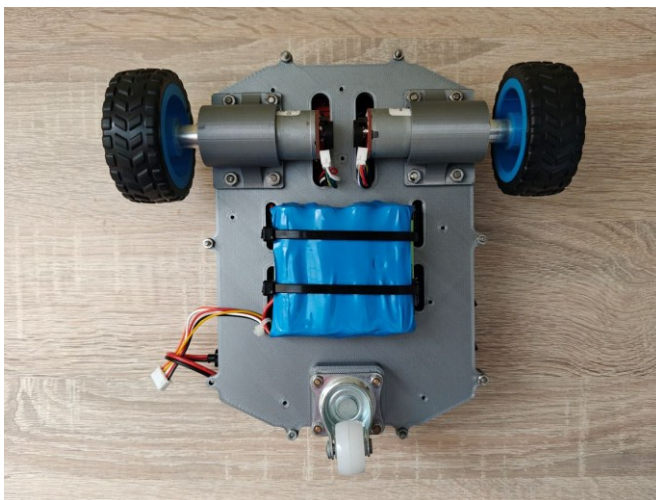
(zdroj: vlastní)

5.2.2 Baterie a napájení

Napájení robotického vozidla zajišťuje baterie složená ze čtyř spojených Li-ion článků typu 18650 s kapacitou 3400 mAh. Bateriový modul, je připevněný k podvozku vozidla (viz Obrázek 24). Napětí baterie při nabití by se mělo pohybovat kolem 16,8 V a při vybití by nemělo klesnout pod 14,4 V. K hlídání napětí na baterii a tím i jejímu nabití slouží k baterii připojený a na vozidle zabudovaný tester baterií, představující malý display s napětím na baterii a ukazatelem nabití, přímo určený pro tenhle typ Li-ion článků. Pro takovou kontrolu stavu nabití předpokládáme, že je uživatel na blízku a stav nabití kontroluje, čímž nedojde k podbití baterie.

Pro nabíjení bateriového modulu je využívána hobby nabíječka iMAX B6. Nabíječka dokáže rovnoměrně dobít jednotlivé články, za pomoci předpřipraveného 5pinového konektoru, vedoucího z bateriového modulu. Takový konektor je v případě balanční nabíječky potřebný, zajišťující dlouhodobou životnost jednotlivých článků rovnoměrným dobíjením.

Pro ochranu komponentů při nechtěném zkratu je v obvodu implementována 4A pojistka, která přivádí elektrickou energii do elektrických komponent skrz dva vypínače, připevněných na bočním panelu vozidla. K jednomu z vypínačů je připojen H-můstek ovládající motory a k druhému z vypínačů je připojený Step-down měnič (viz Obrázek 23). Ten má za úkol snížit a udržovat napětí z baterie na konstantních 5 V, které je následně přivedeno přes konektor USB-C na Raspberry Pi. Schéma zapojení baterie a elektronických modulů lze vidět v příloze P I.



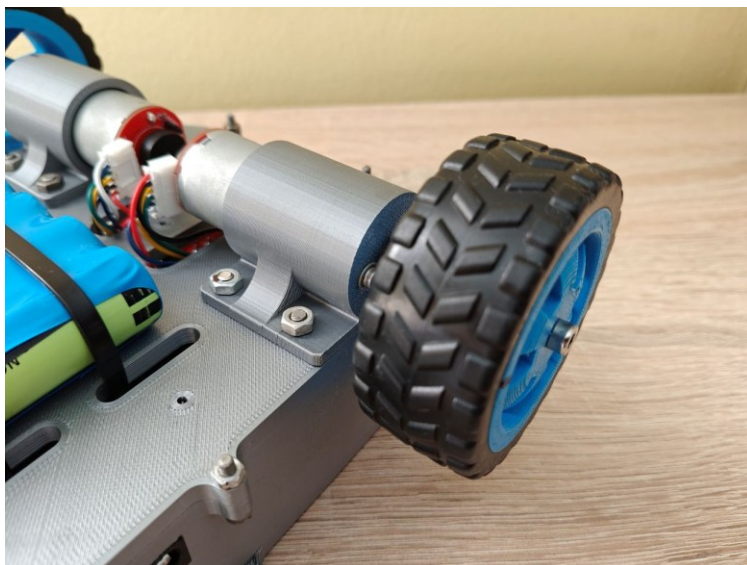
Obrázek 24 Podvozek se zabudovanými motory a baterií

(zdroj: vlastní)

5.2.3 Arduino UNO a pohon kol

Pro nižší úroveň řízení je v robotu osazeno Arduino Uno. Tento mikropočítač slouží ke kontrole či ovládání motorů a je připojen k Raspberry Pi skrz USB. Úkolem desky Arduino je řízení dvou stejnosměrných motorů s převodovkou pro diferenciální způsob řízení, za pomoci H-můstku L298N, v obou směrech a při různých rychlostech. Oba z motorů jsou zpřevodované na 77 RPM a jejich jmenovité napětí je 12 V. Nízké otáčky motorů nejsou na škodu, protože robot se musí pohybovat v uzavřených prostorech pomalu. Díky napětí na baterii, které je v průměru vyšší, než potřebné jmenovité napětí na motorech, lze očekávat i vyšší otáčky. To lze však omezit v programu pro Arduino (omezením PWM signálu na H-můstek) nebo přímo v prostředí ROS.

Motory jsou osazeny koly, která byla vymodelována a vytištěna na 3D tiskárně. Kola byla opatřena gumovou vrstvou pro lepší trakci, která je nutná pro pohyb po kluzkých površích. Motory jsou taktéž osazené enkodéry, každý s dvojicí Hallových sond, díky kterým lze zpětnovazebně oba motory ovládat, resp. sledovat rychlost a směr otáčení hřídele. Motory jsou z důvodu přítomnosti enkodéru vybaveny vícero vodiči – dva z nich slouží pro klasické napájení motorů, dva pro přívod 5V napětí pro enkodéry a poslední dva vedou impulzy z enkodérů do Arduina. Schéma zapojení Arduina s H-můstkem a motory lze vidět v příloze P II.



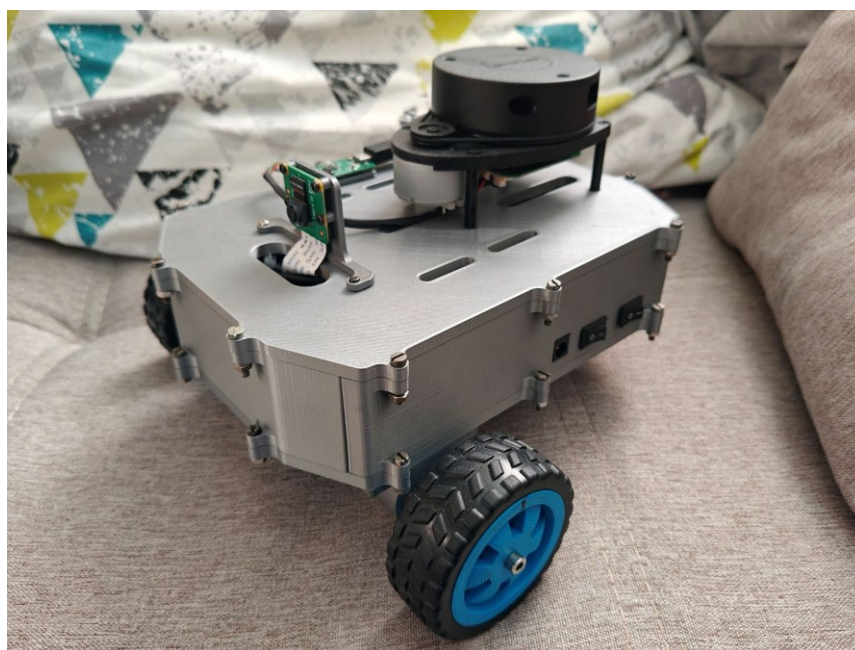
Obrázek 25 Motor robota připevněný k základní nosné desce

(zdroj: vlastní)

5.2.4 Lidar

Důležitou, ne-li nejdůležitější částí autonomního vozidla, je senzor pro snímání blízkého okolí. Z toho důvodu byl robot osazen 360° laserovým skenerem RPLidar A1M8-R6. Tento snímač umožňuje vytvářet 2D mapu prostředí s dosahem až 12 metrů. Fungování tohoto senzoru vychází z principu „Time-of-Flight“. Princip je založený na vysílání a následném odražení laserového paprsku od překážek. Za předpokladu, že známe rychlost, kterou paprsek letí, a dobu za kterou se vrátí, jsme schopni vypočítat vzdálenost senzoru od překážky.

Data o vzdálenosti překážky od senzoru jsou pak skrze USB přenášena do Raspberry Pi, kde jsou následně zpracovávána v rámci systému ROS. Ten je pak dokáže využívat pro algoritmy spojené s mapováním, lokalizací nebo navigací. Primárním úkolem lidarů je tedy detekce překážek, což je důležitým úkolem pro správné fungování autonomního zařízení.



Obrázek 26 Hotové robotické vozidlo se senzory na vrchní krycí desce

(zdroj: vlastní)

6 PROGRAMOVÉ VYBAVENÍ ROBOTA

Po sestavení hardwarové konstrukce robotického vozidla přichází na řadu jeho softwarové zprovoznění, jehož cílem je zajistit funkčnost robota jako celku. Programové vybavení zpracovává senzorická data, zajišťuje systémové řízení procesů a v neposlední řadě umožňuje vhodnou reakci na uživatelské příkazy. K tomu je použit framework (softwarová vrstva) ROS, který zajišťuje vhodné softwarové nástroje a efektivní komunikaci mezi nimi. Na základě výpočtů a rozhodnutí řídicího systému jsou pak generovány výstupy pro akční členy, zajišťující samotný pohyb vozidla.

Prvním krokem tvorby programového vybavení autonomního robotického vozidla je instalace operačního systému na hlavní řídicí jednotky, tj. Raspberry Pi 4 a osobní počítač. V mém případě se bude jednat o notebook, který zajistí flexibilitu celého systému. Díky tomu bude možno systém snadno přesouvat a využívat v různých prostředích. Vzhledem k rozdělení jednotlivých funkcí systému mezi tato zařízení bude taktéž na obou zařízeních nainstalován framework ROS. Mimo komunikační vrstvu, kterou zaštití ROS, je potřeba zajistit i vrstvu síťovou. Obě zařízení budou připojena k routeru přes Wi-Fi a jejich vzájemná komunikace bude umožněna prostřednictvím lokální sítě. Takle strategie nijak neomezuje přenositelnost systému.

Pro správnou funkci robotického vozidla je klíčové vytvořit jeho softwarový model ve formátu URDF. Tento model obsahuje veškeré informace k popisu mechanické struktury robota, včetně senzorů a aktuátorů. Virtuální model je nezbytný jak pro simulaci robota v prostředí Gazebo, tak i pro reálné nasazení. V případě simulace je model URDF využit pro testování vozidla bez fyzického hardwaru, zatímco v reálném nasazení slouží k definici řízení a plánování pohybu.

Dalším krokem je návrh a implementace řídicího systému, který se liší v závislosti na tom, jestli se jedná o robota simulovaného nebo reálného. V případě simulace bude popsán způsob práce s prostředím Gazebo, zatímco u reálného robota se bude práce zabývat spíše nasazením softwaru pro fyzické řízení. Pro obě varianty je v dalších kapitolách popsána implementace mapovacích, lokalizačních a navigačních prvků z balíčků ROS a Nav2.

6.1 Zařízení a jejich umístění v síti

Pro samotné programování či implementaci softwaru autonomního vozidla byla nezbytná instalace operačního systému a následné softwarové vrstvy ROS na zařízení. Komunikace mezi zařízeními na úrovni ROS pak musí být založena na vhodné konfiguraci zařízení v síti a doplněna o možnost vzdáleného přístupu k zařízení Raspberry Pi. To je totiž při provozu odpojené od monitoru či od klávesnice. V neposlední řadě je nutná instalace základních balíčků potřebných k následné práci s tvorbou programů a samotné založení vlastního projektového balíčku.

6.1.1 Instalace Ubuntu a ROS

Pro správné fungování robota je nezbytná instalace OS a ROS, přičemž určitá verze ROS bývá většinou kompatibilní s danou verzí operačního systému. Pro naše zařízení, konkrétně notebooku a Raspberry Pi 4, se jeví adekvátní kombinace Ubuntu 24.04 LTS a ROS Jazzy nebo Ubuntu 22.04 LTS s ROS Humble, přičemž snahou bylo implementovat software novější. Během testování byly odzkoušeny obě varianty a taktéž zjištěny jejich výhody a nevýhody.

Jedním z hlavních rozdílů, který byl v průběhu testování objeven, je fakt, že novější verze ROS již není nadále kompatibilní s oběma verzemi simulačních programů, tj. Gazebo Classic a Gazebo Ignition (dnes už jen jako Gazebo). I přesto, že podpora pro starší verzi Gazebo v tomto roce skončila, je tato verze stále přehlednější a méně náročná na výkon než verze novější. Tento aspekt však bude zmíněn v kapitole 6.3.

Hlavním důvodem pro upřednostnění starší verze operačního systému a ROS byla nefunkčnost jednoho z pluginů pro tvorbu trajektorie navigace (plugin MPPI – Model Predictive Path Integral Controller), který při testování robota nefungoval správně v novější verzi ROS a Ubuntu. I přes snahu o aktualizaci nástrojů, knihoven a úpravu parametrů navigace se problém nepovedlo vyřešit. Po neúspěšném řešení problému bylo rozhodnuto opustit Ubuntu 24.04 a ROS Jazzy a zůstat pouze u starší verze OS.

Celkově lze tedy říct, že kombinace Ubuntu 22.04 LTS s ROS Humble se ukázala jako spolehlivější a prokázala lepší kompatibilitu s nástroji, knihovnami a programy. To potvrzuje i silná uživatelská komunita, která u novějších verzí ještě není tak rozvinutá. Dále je třeba zdůraznit, že Ubuntu 22.04 LTS bude podporováno až do roku 2032, což zajišťuje jeho dlouhodobou stabilitu.

Operační systém Ubuntu 22.04 byl nainstalován za pomoci instalačního média ve formátu ISO získaného z oficiálních stránek Ubuntu. Na notebooku byla instalace provedena ve virtuálním prostředí VMware Workstation 17. V případě Raspberry Pi bylo navíc potřeba instalační médium vhodně převést na SD kartu, k čemuž byl použit nástroj Raspberry Pi Imager. Po vložení SD karty do mikropočítače je pak postup instalace obdobný instalaci na klasický počítač. Nutností je pak, do samotného nainstalování vzdáleného přístupu, připojení periférií, tj. klávesnice, myši a monitoru, ke konfiguraci systému.

Pro instalaci ROS 2 Humble Desktop, na notebook a Raspberry Pi s operačním systémem Ubuntu, bylo použito návodu na oficiálních stránkách ROS [22]. Návod na stránkách obsahuje kroky potřebné ke správné instalaci frameworku, spolu se souvisejícími příkazy pro vložení do příkazového řádku. Instalace je celkově jednoduchá, konkrétně jde o získání klíče pro repozitář ROS, jeho následné přidání do repozitáře systému, a nakonec samotnou instalaci společně s dalšími nástroji a knihovnami.

Pro správné fungování prostředí ROS je však při každém otevření nového terminálu potřeba provést příkaz, který načte všechny potřebné proměnné systému. To umožňuje následně spouštět ROS příkazy, například „ros2 run“ nebo „ros2 launch“. Takle skutečnost lze obejít přidáním příkazu do souboru „.bashrc“, který zařídí provedení příkazu automaticky, při každém novém otevření terminálu. To lze jednoduše provést zápisem do příkazového řádku:

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

6.1.2 Umístění v lokální síti

Síťové spojení systému s robotickým vozidlem je realizováno prostřednictvím samostatné Wi-Fi sítě, vytvořené primárně pro zařízení systému. Komunikaci mezi zařízeními zajišťuje použitý router značky TP-Link. Výhodou tohoto řešení je možnost, kdykoli router odpojit od ethernetového kabelu a přenést jej na jiné místo, kde se zařízení znovu k routeru připojí a mohou pokračovat v komunikaci. ROS totiž ke svému provozu nepotřebuje připojení k internetu, takže provoz robota může probíhat i v prostředích bez přístupu k internetu a kde router pouze plní funkci zprostředkovatele lokální sítě. Nevýhodou je pak omezení v případě aktualizací a instalací nových balíčků a knihoven, s tím se však počítá.

Protože po implementaci Raspberry Pi do vytištěného modelu vozidla nebude připojen monitor či klávesnice, je potřeba k němu zajistit vzdálený přístup. K tomu se dá využít nástroje SSH, ke kterému však potřebujeme znát lokální IP adresu zařízení. Pro potlačení dynamického přidělení IP adresy z routeru je tedy za potřebí na Raspberry Pi nastavit IP adresu statickou. Tenhle krok nám zajistí, že se zařízení po každém zapnutí připojí pod stejnou adresou, díky které pak můžeme zařízení najít pomocí vzdáleného ovládání.

Pro konfiguraci síťového připojení v Ubuntu na Raspberry Pi je využit nástroj Netplan, který funguje jako prostředník mezi uživatelem a NetworkManagerem. Nastavení nástroje je pak uloženo v konfiguračním souboru formátu YAML, uloženém v adresáři „/etc/netplan“. Soubor „network.yaml“ nastavující statickou IP vypadá v mém případě takto:

```
network:
  version: 2
  renderer: NetworkManager

  wifis:
    wlan0:
      dhcp4: no
      addresses: [192.168.2.102/24]
      gateway4: 192.168.2.1
      nameservers:
        addresses: [192.168.2.1]
      access-points:
        "Martin":
          password: "homer123"
```

Soubor obsahuje parametry popisující připojení k Wi-Fi síti, například potlačuje DHCP, což zamezí dynamickému přidělení adresy z routeru, nastavuje statickou IP adresu zařízení, specifikuje IP adresu routeru a v neposlední řadě specifikuje jméno a heslo sítě, ke které se připojuje. Konfiguraci síťového nastavení je pak nutné aktivovat pomocí příkazů v terminálu:

```
sudo netplan generate
sudo netplan apply
```

Po nastavení Netplanu je již IP adresa zařízení známá a můžeme se na ni dotazovat. Pro pohodlnou správu využíváme nástroje SSH (Secure Shell), umožňující vzdálené připojení k terminálu Raspberry Pi z notebooku nebo jiného zařízení. Primárně budeme skrz SSH spouštět launcher soubor na robotickém vozidle, který dále zajistí spuštění vícero ROS uzlů najednou. Další možností je pak využití SSH u programu Visual Studio Code, díky čemuž lze upravovat jednotlivé soubory na Raspberry Pi, obdobně z jiného zařízení.

Nainstalovat SSH se dá skrz příkazový řádek příkazem:

```
sudo apt install openssh-server
```

Pro připojení k jinému zařízení, v mém případě Raspberry Pi, je potřeba znát jméno zařízení a IP adresu, zápis vypadá následovně:

```
ssh pi@192.168.2.102
```

Po zadání příkazu je následně potřeba zadat heslo žádaného zařízení, pro zajištění ochrany. Nutnost zadávání hesla jde obejít, pro to je potřeba však vygenerovat a uložit SSH klíč.

6.1.3 Tvorba workspace a balíčku

Pro vývoj a uložení softwaru v prostředí ROS byl vytvořen workspace (pracovní prostor). Ten je založen v adresáři „/bakalarka_ws/src“. V tomto pracovním prostoru byl následně vytvořen ROS balíček „autobot“ pomocí příkazu provedeném v terminálu:

```
ros2 pkg create --build-type ament_cmake autobot
```

Balíček je vytvořen s využitím build systému „ament_cmake“, který umožňuje efektivní správu závislostí a kompilací zdrojového kódu. Po vytvoření balíčku byly v jeho adresáři automaticky vygenerovány základní soubory, které byly následně doplněny o další podadresáře, které jsou určeny pro určité části projektu:

- **launch** – složka obsahující spouštěcí skripty jednotlivých ROS uzlů robotického systému
- **description** – složka určená pro soubory formátu URDF, který definuje jeho strukturu, senzory a pohyblivé části
- **params** – složka obsahující konfigurační soubory ve formátu YAML s uloženými parametry uzlů a nastavení
- **worlds** – složka obsahující soubory pro základní definici prostředí simulačního programu Gazebo

Pro správnou kompilaci a správu pracovního prostoru byl nainstalován nástroj „colcon“, umožňující sestavení celého workspace. Colcon byl nainstalován pomocí příkazu v terminálu:

```
sudo apt install python3-colcon-common-extensions
```

Provedení kompilace a vytvoření nového buildu zdrojového kódu se provádí za pomoci příkazu:

```
colcon build
```

Provést kompilaci je pak potřeba při každé úpravě souborů, především při vytváření či odstraňování souborů nebo jejich přejmenovávání. Kompilací se vytvoří soubor obsahující veškeré nutné informace k přístupu k balíčkůům a jejich spuštění. Tento soubor je nutné zavolat při každém novém otevření terminálu příkazem (v opačném případě by spouštěcí ROS příkaz balíček nenašel):

```
source install/setup.bash
```

6.2 Tvorba softwarového modelu

Pro správnou funkci robotického vozidla, ať už v simulaci či při reálném nasazení, je potřeba zajistit přesný popis jeho konstrukce a kinematiky v prostředí ROS. K tomu je v mé práci využit formát URDF, který definuje strukturu robota, včetně jeho částí a jejich vazeb. Pro efektivní práci s tímto modelem je využit uzel „Robot State Publisher“, jehož úkolem je zpracovat URDF popis robota a publikovat informace o něm do příslušných ROS témat.

6.2.1 Robot State Publisher

Uzel „robot_state_publisher“ (dále jen RSP) je důležitým prvkem pro samotné fungování řídicího systému robota. Jeho hlavní úlohou je na základě popisu robota ve formátu URDF generovat a publikovat transformační vztahy (transformace) mezi jednotlivými částmi robota, které jsou interpretované souřadnicovými systémy (rámcí). Tyto informace jsou pak důležité pro ostatní softwarové komponenty, jako jsou třeba lokalizační a navigační algoritmy.



Obrázek 27 Schéma funkce Robot State Publisher

(zdroj: vlastní)

V rámci implementace tohoto řešení je v mé práci využit soubor „**rsp.launch.py**“. Jeho fungování lze popsat pomocí Obrázku 27. Celý proces lze pak rozdělit na jednotlivé části:

- **Načtení URDF souborů** – soubory definující robota jsou pomocí nástroje Xacro zkompletovány do jednoho výsledného souboru URDF, který je následně zpracován uzlem.
- **Publikace ROS témat** – uzel publikuje kinematické transformace do tématu „/tf“ a textový popis (kopii URDF) do tématu „/robot_description“.
- **Odběr tématu ROS** – uzel naslouchá tématu „/joint_states“, představující data ze senzorů nebo simulace, informující o aktuálních hodnotách týkajících se robota a podle toho upravuje témata výstupní.

6.2.2 Struktura URDF modelu

K popisu konstrukce a kinematiky modelu vozidla jsou určeny soubory URDF. Ty jsou umístěny v balíčku autobot v adresáři „description“ a jsou napsány v jazyce XML. Díky tomuhle formátu byly definované jednotlivé díly robota, jejich geometrie, vazby mezi nimi a další důležité parametry. Například hmotnost, barva dílů nebo vlastnosti kolizí či rozložení váhy.

Pro zvýšení flexibility a modularity bylo využito rozšíření URDF zvané Xacro (XML Xacro). Xacro umožňuje v rámci souborů definovat makra, proměnné a podmínky, čímž usnadňuje strukturování a přehlednost rozsáhlých modelů. V mém případě je tedy model robotického vozidla rozdělen do vícero souborů, které jsou za pomoci Xacra v hlavním souboru načteny. Rozšíření je nutno nainstalovat na obě zařízení pomocí příkazu:

```
sudo apt install ros-humble-xacro
```

Hlavním souborem popisu robota je soubor „robot_main.xacro“. Ten slouží jako vstupní bod pro uzel „Robot State Publisher“. Jeho funkcí je definice všech Xacro souborů:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro" name="robot">
  <xacro:arg name="sim_mode" default="false"/>
  <xacro:include filename="robot_base.xacro" />
  <xacro:include filename="lidar.xacro" />
  <xacro:include filename="camera.xacro" />
  <xacro:include filename="ros2_control.xacro" />
</robot>
```

Každý ze souborů pak odkazuje na specifickou část modelu. Postupně tedy na část základny, lidarů a kamery. Poslední je pak soubor nastavující parametry řízení, tj. obsahující parametry pro konfiguraci části kontroléru zajišťující řízení reálného a simulovaného robota.

Hlavní část těla robota je pak definována v souboru „robot_base.xacro“. Soubor obsahuje proměnné popisující vlastnosti robota, jako jsou rozměry, hmotnost, barva:

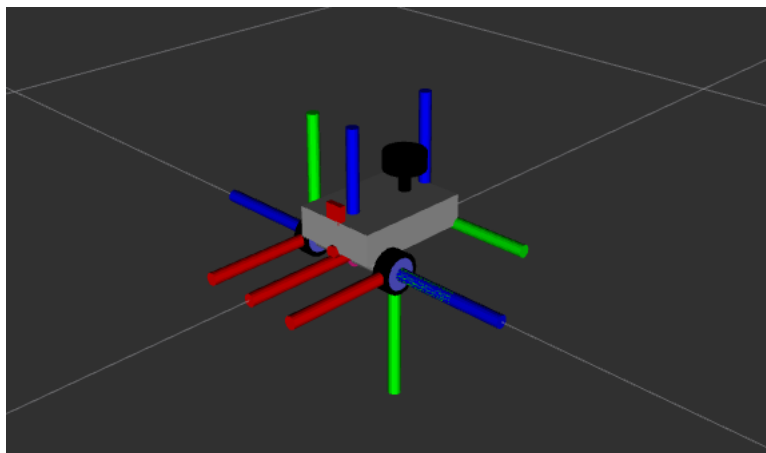
```
<xacro:property name="chassis_length" value="0.195"/>
<xacro:property name="chassis_width" value="0.140"/>
<xacro:property name="chassis_height" value="0.05"/>
<xacro:property name="chassis_mass" value="1.5"/>
```

Pomocí těchto proměnných jsou dále definovány pevné části robota (links), které jsou pospojovány spoji (joints). A právě tyto části a spoje jsou později, po spuštění uzlu „Robot State Publisher“ publikovány do ROS témat v podobě rámců a transformací. Příkladem pevné části a spoje může být hnací kolo vozidla:

```
<joint name="left_wheel_joint" type="continuous">
  <parent link="base_link"/>
  <child link="left_wheel"/>
  <origin xyz="0 ${main_wheel_offset_y} 0" rpy="-${pi/2} 0 0" />
  <axis xyz="0 0 1"/>
</joint>

<link name="left_wheel">
  <visual>
    <geometry>
      <cylinder radius="${main_wheel_radius}"
        length="${main_wheel_thickness}" />
    </geometry>
    <material name="black"/>
  </visual>
```

Správně definovaný model robota lze zobrazit v aplikaci RViz. Model robota je zde vykreslen z tématu „/robot_description“. Současně lze spustit i vizualizace tématu „/tf“, který zobrazuje jednotlivé rámce a transformace mezi nimi (viz Obrázek 28).



Obrázek 28 RViz – vizualizace modelu robota

(zdroj: vlastní)

6.3 Software simulovaného modelu

Hlavním cílem této práce je sestavení a naprogramování reálného robotického vozidla, přičemž simulace není nezbytnou součástí. I přesto nabízí zajímavou možnost, jak si některé aspekty softwaru robota vyzkoušet v kontrolovaném simulovaném prostředí. Simulace tedy slouží jako takový doplňkový nástroj, který určité prvky řídicího systému sdílí se systémem reálného robota. **Z toho důvodu budou některé z těchto prvků vysvětleny zde a v následující kapitole se softwarem reálného robota budou pouze zmíněny.**

V téhle kapitole bude uveden postup instalace simulačního prostředí Gazebo a k němu souvisejících balíčků, sloužících k vzájemné komunikaci s prostředím ROS. Dále pak nezbytné kroky ke spuštění simulace, vložení virtuálního modelu robota z uzlu „Robot State Publisher“ do aplikace a kontrolu implementovaných kontrolérů zajišťujících pohyb v simulaci a následnou zpětnou vazbu. V neposlední řadě pak i vytvoření jednoduchého programu vhodného pro teleoperování vozidla v případě mapování prostředí.

6.3.1 Teleoperování robotického vozidla

Pro možnost manuálního řízení robota v simulaci ale i reálném prostředí, například při vytváření mapy prostředí mapováním či nutné úpravě pozice vozidla, je nutné přidat možnost teleoperování. K tomu lze využít klávesnici notebooku, popřípadě jako v mém případě herní ovladač s tlačítky a joysticky. Pro zpracování signálu z joysticku, publikaci do

témata nesoucího informaci o rychlosti a vhodnou úpravu pro řídicí systém jsou vyhrazeny uzly umístěné ve skriptu „joystick.launch.py“. Těmito klíčovými uzly jsou:

- **joy_node** – Zajišťující čtení vstupu z herního ovladače připojeného k počítači za pomoci USB kabelu.
- **teleop_node** – Převádějící vstupy z ovladače na zprávy typu „Twist“, obsahující data ve dvou vektorech nesoucí informace o požadované lineární a úhlové rychlosti robota, publikované do tématu „cmd_vel joystick“.
- **twist_mux** – Určený pro sloučení vstupních témat nesoucích data o rychlosti (z joysticku a navigačního systému) s připravenou prioritou ve prospěch herního ovladače, pro zamezení nechtěné situace v případě nekontrolovaného pohybu zapříčiněného navigací. Výstupním tématem je „/diff_cont/cmd_vel_unstamped“.
- **twist_stamper** – Přidávající k rychlostním příkazům předchozího tématu časová razítka, důležité pro synchronizaci. Výsledkem je téma „/diff_cont/cmd_vel“, které je spolu s neoraženým tématem posíláno do řídicího kontroléru zajišťující zasílání povelů pro motory.

Konfigurace jednotlivých uzlů jsou pak uloženy v parametrických souborech „twist_mux.yaml“, obsahující prioritizaci témat s daty o rychlostech, a „joystick.yaml“, uchovávající nastavení tlačítek a os joysticku herního ovladače.

Pro správnou funkčnost je nutné nainstalovat balíčky obsahující některé ze zmíněných uzlů. Nainstalovat se dají podle následujících příkazů (postačí instalace na PC):

```
sudo apt install ros-humble-twist-mux
sudo apt install ros-humble-twist-stamper
```

6.3.2 Instalace aplikace Gazebo

Pro simulaci robotického vozidla byly odzkoušeny dvě verze simulátoru, Gazebo Classic a Gazebo Ignition. Dnes už se novější verze označuje pouze jako Gazebo, nýbrž verze mnou odzkoušená nese ještě název Ignition. Přestože podpora Gazebo Classic skončila v tomto roce, nadále se z mého pohledu jeví jako lépe optimalizovaná a jednodušší na použití. Novější Ignition sice přináší modernější vzhled a větší možnosti využití, to však na úkor komplikací spojených s náročnější integrací do ROS. Například starší verze Gazebo dokáže

s ROS komunikovat napřímo, zatímco verze novější potřebuje k fungování „bridge“, který převádí témata mezi simulací a ROS.

Obě verze byly nainstalovány rozlišným způsobem:

```
# Pro instalaci Gazebo Classic s potřebnými balíčky ROS balíčky
sudo apt install gazebo
sudo apt install ros-humble-gazebo-ros-pkgs
# Pro instalaci Gazebo Ignition
sudo apt install ros-humble-ros-ign
```

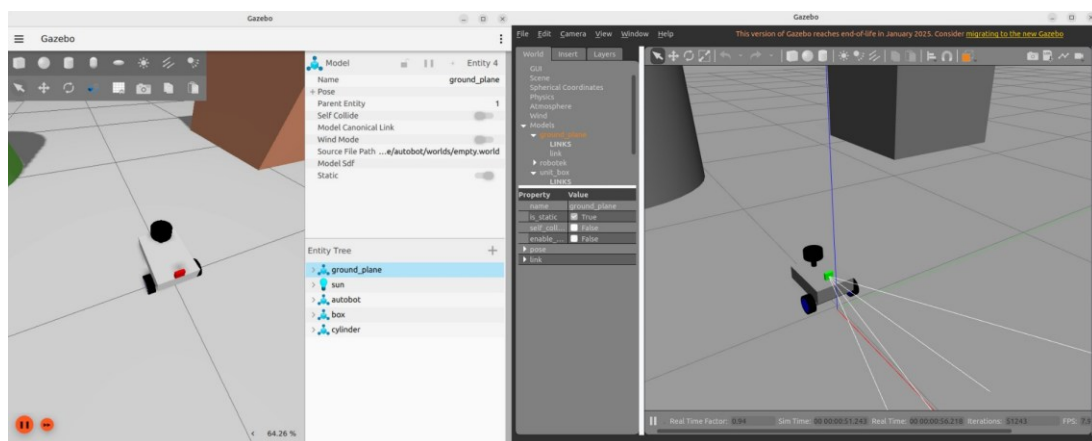
Instalace byla na PC provedená bez problémů, při pokusu spuštění verze Ignition na virtuálním stroji však nastaly komplikace se správným načítáním. Prostředí se sice načetlo, avšak renderování světa bylo nefunkční, obraz stále blikal a celý simulátor se výrazně zpomaloval, až se stal nepoužitelným. Po analýze problému bylo zjištěno, že příčinou je nesprávná detekce grafických ovladačů ve virtuálním prostředí. Gazebo Ignition totiž využívá modernější grafický backend, než starší verze Classic, a spoléhá na hardwarovou akceleraci, která ve virtuálním prostředí nemůže fungovat správně. Provizorním řešením problému bylo zakázat některé grafické drivery v nastavení, čímž virtuální stroj přešel na softwarové vykreslování. Po téhle úpravě se již aplikace načetla správně, i tak však v dosti omezeném režimu, který neumožňoval adekvátní využívání aplikace. To bylo také následně podnětem pro opuštění ROS Jazzy, kde již Gazebo Classic nefunguje a které bylo, z důvodu nízkých nároků na výkon, možné používat alespoň na ROS Humble.

Krom rozdílné instalace a dříve zmíněného přenosu mezi tématy se dále aplikace liší v rozdílném systému pluginu, struktuře a syntaxi konfiguračních souborů a spouštění verzí:

```
# Gazebo Ignition
ros2 launch ros_ign_gazebo ign_gazebo.launch.py
ign gazebo #zkrácená verze
# Gazebo Classic
ros2 launch gazebo_ros gazebo.launch.py
gazebo #zkrácená verze
```

I když simulace v mém případě nesloužila k velkému testování navigačních algoritmů, využíval jsem ji především k úpravě parametrů a občasnému ověření chování vozidla. Bylo to užitečné zejména v situacích, kdy nebylo praktické zapínat a připojovat se k reálnému

vozidlu. Například když bylo právě na nabíječce nebo jsem potřeboval rychle vyzkoušet drobnou úpravu bez fyzického zásahu. Simulace tak poskytla pohodlný způsob, jak si některé aspekty ověřit, aniž by bylo nutné pokaždé pracovat s reálným hardwarem. I když je Gazebo Classic jednodušší, výkonně nenáročnou volbou a pro můj účel vystačila, ostatním uživatelům bych doporučil novější verzi Gazeba. V případě plnohodnotného počítače totiž může být daleko přínosnější a umožňovat rozsáhlejší využití. Z toho důvodu je zbytek této kapitoly zaměřený spíše na novější verzi Gazeba (Gazebo Ignition), práci s ní a její začlenění do ROS.



Obrázek 29 Gazebo Ignition (vlevo) a Gazebo Classic (vpravo)

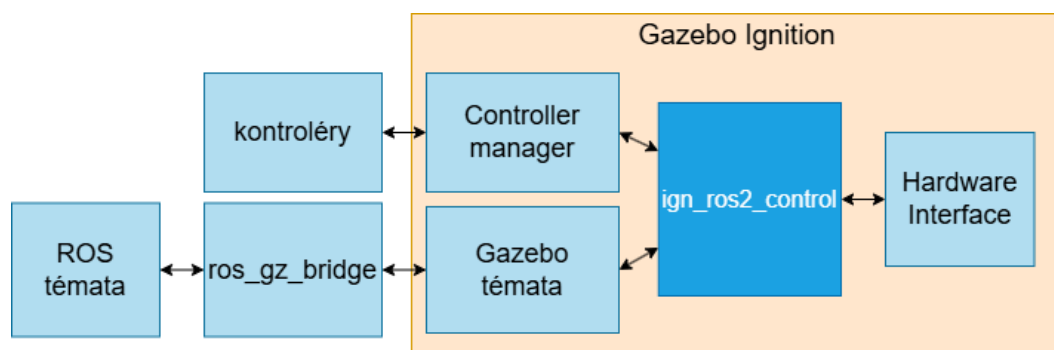
(zdroj: vlastní)

6.3.3 Řídicí systém pro simulované prostředí

Součástí řídicího systému robota, v simulaci i v reálném prostředí, je softwarový rámec „ros2_control“, který v systému ROS představuje modulární nástroj, pro řízení jednotlivých pohyblivých částí robota. Výhodou tohoto rámce je, že spojuje řídicí algoritmy s konkrétní implementací hardwaru, tj. převádí data z témat o rychlosti na signály vhodné pro ovladače a opačně publikuje témata o pozici či natočení spojů (links).

Architektura „ros2_control“ je tvořena několika komponenty:

- **Controller Manager** – Centrální uzel zajišťující spouštění a správu jednotlivých kontrolérů. V simulaci je realizován pluginem programu Gazebo, tj. není jej potřeba spouštět samostatně.
- **Hardware Interface** – Zajišťuje a definuje způsob, jakým rámec komunikuje se simulovaným nebo reálným hardwarem robota. V případě simulace v programu Gazebo Ignition je to plugin „ign_ros2_control/IgnitionSystem“, který i jednotlivé aktuátory a senzory simuluje.
- **Resource Manager** – Část Controller Manageru, která definuje a spravuje Hardware Interface a tím i ovládané spoje (levé a pravé kolo robota) a jejich stavy.
- **Kontroléry (Controllers)** – Řídicí prvky, které zpracovávají příchozí příkazy z témat (například „/diff_cont/cmd_vel“) a předávají konkrétní pokyny aktuátorům skrz Hardware Interface. Opačně směrem pak publikují stavové informace do témat „/odom“ nebo „/joint_states“.



Obrázek 30 Schéma řídicího systému simulace

(zdroj: vlastní)

Pro spuštění a definování zmíněných komponentů je vytvořen soubor „ros2_control.xacro“, který je načten spolu s ostatními definičními soubory při spuštění uzlu „Robot State Publisher“. Obsahem tohoto souboru je definice Hardware Interface, kontrolovaných spojů s parametry a v neposlední řadě pluginu, který zajišťuje simulaci Controller Manageru spolu s informacemi o používaných kontrolérech.

Pro integraci „ros2_control“ do simulace v prostředí **Gazebo Ignition** je v souboru xacro zanesena poznámka o využívání pluginu „ign_ros2_control“ hned dvakrát.

V prvním případě jde o značku pro Gazebo, která jej informuje o použití pluginu pro zajištění simulace Controller Manageru, který bude fungovat jako most mezi kontroléry a simulovaným hardwarem:

```
<gazebo>
  <plugin filename="libign_ros2_control-system.so"
    name="ign_ros2_control::IgnitionROS2ControlPlugin">
    <parameters>$(find
      autobot)/params/controllers_params.yaml</parameters>
  </plugin>
</gazebo>
```

Kromě definování pluginu je přiložen i odkaz na parametrický soubor „controllers_params.yaml“, obsahující informace o používaných kontrolérech, jejich nastavení a ovládanými spoji (koly):

- **Joint_state_broadcaster** – kontrolér publikující stavy natočení kol do tématu „/joint_states“
- **Diff_drive_controller** – kontrolér pro diferenciální způsob řízení, převádějící rychlostní příkazy na signály vhodné pro Hardware Interface

V druhém případě je plugin zmíněný v části definující Hardware Interface simulace, ovládané komponenty a jejich rozhraní (rychlost, pozice):

```
<ros2_control name="IgnitionSystem" type="system">
  <hardware>
    <plugin>ign_ros2_control/IgnitionSystem</plugin>
  </hardware>
  <joint name="left_wheel_joint">
    <command_interface name="velocity">
      <param name="min">-10</param>
      <param name="max">10</param>
    </command_interface>
```

```
<state_interface name="velocity"/>
<state_interface name="position"/>
</joint>
</ros2_control>
```

Nutností pro práci s rámcem „ros2_control“ společně se simulátorem a kontroléry je instalace ROS balíčku pomocí následujících příkazů:

```
sudo apt install ros-humble-ros-ign
sudo apt install ros-humble-ign-ros2-control
sudo apt install ros-humble-ros2-control
sudo apt install ros-humble-ros2-controllers
```

6.3.4 Spuštění simulace robota v prostředí Gazebo Ignition

Pro simulaci robotického vozidla je v adresáři „./launch“ vytvořen spouštěcí python skript, obsahující důležité ROS komponenty a samotný simulátor. Spuštěním tohoto skriptu se automaticky spustí prostředí, teleoperace a publikuje se stav modelu robota, který se nahraje do simulátoru Gazebo. Tenhle postup zamezí nutnému spouštění mnoha skriptů samostatně v terminálech.

Jako první se spouští uzly definované v souboru „joystick.launch.py“, který zajistí ovládání vozidla pomocí herního ovladače připojeného přes USB. Obsahem tohoto souboru je i uzel, které slouží jako multiplexer pro sloučení rychlostních příkazů z joysticku a navigace. Bez jeho spuštění by nebylo později možné ovládat robota pomocí navigace.

Dále je spuštěn soubor „rsp.launch.py“, který publikuje transformace a stav virtuálního modelu na základě URDF modelu. Teď už i doplněného o soubor zahrnující „ros2_control“.

Samotný simulační program je spuštěn s definovaným výchozím světem, obsahujícím pouze základní fyziku a slunce. Případné překážky pak lze jednoduše přidat v aplikaci. Následně je do simulace přidán pomocí uzlu „create“ samotný robot. Ten je načten z ROS tématu „./robot_description“, publikovaným pomocí uzlu „Robot State Publisher“.

Dále se spustí dva řídicí kontroléry (joint_state_broadcaster a diff_drive_controller) pomocí uzlu „spawner“. Ty pak komunikují s Controller Managerem, který je simulovaný aplikací Gazebo Ignition.

Nakonec je ještě aktivovaný uzel „ros_gz_bridge“, který tvoří most mezi ROS a Gazebo tématy. Témata, která jsou tímto stylem transformována, jsou definována v parametrickém souboru „gz_bridge.yaml“. Mezi tyto témata patří data směřovaná do a z kontrolérů nesoucí zprávy o požadované rychlosti, pozici robota nebo natočení spojů motorů, tj. „/diff_cont/cmd_vel“, „/odom“ a „/joint_states“.

Tento spouštěcí skript je centrálním bodem celé simulace, čekající pouze na následné algoritmy mapování, lokalizace, navigace nebo pouhé teleoperování uživatelem. Spustit se dá jednoduše příkazem:

```
ros2 launch autobot sim.launch.py
```

6.4 Software reálného robota

Tato kapitola přímo navazuje na předchozí část věnovanou softwaru simulovaného robota. I v případě reálného robota zůstává mnohé zachováno, tj. nadále se využívá rámec „ros2_control“, který umožňuje oddělit řídicí logiku ROS od reálných komponent hardwaru. Zatímco v případě simulovaného vozidla se o kontrolní činnosti a komunikaci se simulovanými komponenty starala aplikace Gazebo spolu s pluginy, v případě reálného vozidla musí být však tyto prvky zajištěny uživatelem.

Rozdíl spočívá v komponentech, které již nadále nejsou virtuální a představují reálné zařízení (lidar, kamera, motory). U těchto zařízení je zapotřebí zajistit jejich správné fungování nakonfigurováním uzlů, nastavením parametrů a zajištěním komunikace s Raspberry Pi.

Změna je taky patrná u řídicího systému, který je doplněn o uzel Controller Manageru. Tento uzel již nadále není simulovaný Gazebem a musí být spuštěn samostatně. Další změna se pak týká Hardware Interface, který musí být implementovaný na míru k použitým komponentům zajišťující pohyb.

6.4.1 Lidar

V rámci reálného modelu robota je využíván skener RPLidar A1M8-R6, který je připojený k mikropočítači Raspberry Pi skrz USB. Pro správnou komunikaci mezi lidarem a ROS je použitý balíček „rplidar_ros“, zajišťující připojení a čtení dat ze senzoru. Nainstalovat se dá pomocí příkazu:

```
sudo apt install ros-humble-rplidar-ros
```

Pro spuštění uzlu ze zmíněného balíčku je vytvořen python skript „rplidar.launch.py“, který má za úkol z lidaru publikovat data do tématu „/scan“. Data jsou v podobě zpráv typu „LaserScan“ a obsahují informace o vzdálenosti překážek detekovaných kolem obvodu celého senzoru.

Součástí skriptu jsou i parametry specifikující připojení a komunikaci:

- **serial-port** – Parametr určující cestu k sériovému portu Raspberry Pi, ke kterému je lidar připojený. Aby nedošlo ke zmatkům ohledně záměny fyzického portu, je tahle cesta specifikována adresářem určující cestu ke konkrétnímu USB zařízení, tj. „/dev/serial/by-path/platform-fd500000.pcie-pci-0000:01:00.0-usb-0:1.3:1.0-port0“
- **serial-baudrate** – Parametr určující rychlost sériového přenosu dat mezi lidarem a zařízením. Pro použitý lidar je doporučená rychlost komunikace 115200 bps.
- **frame_id** – Název souřadnicového rámce, ke kterému jsou data z lidaru vztaženy.

6.4.2 Kamera

Model reálného robota byl taktéž osazen kamerou RPi V2, připojenou pomocí CSI portu na desce Raspberry Pi. Pro integraci s ROS byl využit balíček „v4l2_camera“, který umožňuje přístup ke kameře přes rozhraní Video4Linux2. Uzel z tohoto balíčku se dá, obdobně jako u lidaru, spustit pomocí skriptu „camera.launch.py“.

Součástí skriptu jsou samozřejmě i parametry:

- **image_size** – rozlišení výstupního obrazu (šířka x výška)
- **Time_per_frame** – doba mezi jednotlivými snímky, odpovídající obnovovací frekvenci obrazu
- **camera_frame_id** – název souřadnicového rámce, ke kterému jsou data z kamery vztaženy

Výstupem z uzlu jsou data publikované ve zprávách typu „Image“ v tématu „/camera/image_raw“. Ty lze následně zobrazit například v aplikaci RViz nebo rqt.

6.4.3 Arduino

Arduino Uno slouží v rámci systému robotického vozidla jako nízko-úrovňová řídicí jednotka pro kontrolu a ovládání motorů v diferenciálním způsobu řízení. Mikropočítač je připojen k Raspberry Pi pomocí USB rozhraní a jeho funkcí je zajistit přesné řízení otáček podle příkazů přijatých ze sériové linky.

Vývoj vlastního firmwaru pro Arduino by byl časově náročný a vzhledem k dostupnosti již hotových ověřených řešení i zbytečný. Proto bylo rozhodnuto pro využití open-source projektu „ros_arduino_bridge“ [23]. Tento projekt poskytuje spolehlivý software, který implementuje potřebné funkce pro komunikaci s počítačem a řízení motorů pomocí H-můstku. V mém případě podporuje i využitý typ L298N a umožňuje tak i jeho bezproblémovou obsluhu bez nutných úprav kódu.

Firmware byl do mikropočítače nahrán pomocí notebooku, možností je pak i úprava kódu přímo z Raspberry Pi nebo vzdáleně, pomocí doplňkových nástrojů Visual Studio přes SSH. V rámci firmwaru razantní změny provedeny nebyly. Jedinou úpravou byla konfigurace pinů, aby odpovídaly konkrétnímu zapojení použitého hardwaru. Možností je pak samozřejmě úprava PID parametrů regulace, což je taky principem, na kterém řízení funguje. Úprava však provedena nebyla, jelikož regulace v případě implementovaných motorů funguje, již od začátku, bez problému.

Komunikace mezi Arduinem a Raspberry Pi probíhá prostřednictvím sériové asynchronní komunikace s přenosem dat o rychlosti 57600 bps. Arduino pak čeká na předem definované příkazy, například:

```
#define MOTOR_SPEEDS    'm' /*closed-loop řízení*/  
#define MOTOR_RAW_PWM  'o' /*open-loop řízení*/
```

Pro ovládání obou motorů pak příkazy vypadají například takto:

```
m 100 100 /*typ příkazu - 1. motor - 2. motor*/
```

Nutností firmwaru pro Arduino je i využívání zpětné vazby z enkodérů motorů. Díky této zpětné vazbě, v porovnání s požadovanou rychlostí, může PID regulace zajistit přesné řízení pohybu vozidla. Bez ní by nebylo možné zajistit pohyb vozidla v nízkých rychlostech, tj. v otevřené smyčce se by se mohlo stát, že by dodávaná elektrická energie nebyla dostatečně vysoká, aby se vozidlo vůbec rozjelo.

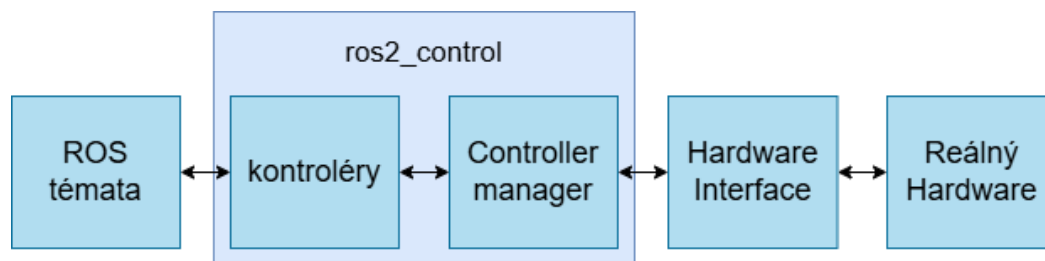
6.4.4 Řídicí systém reálného robotického vozidla

Pro řízení reálného robotického vozidla je, stejně jako u simulované verze, použit balíček „ros2_control“. Na rozdíl od simulace však již systém není propojen s prostředím Gazebo a jeho pluginy. Proto je nutné některé komponenty, které u simulované verze zařizoval simulátor, nahradit jinými a spouštět je samostatně.

Základem řídicího systému jsou následující komponenty:

- **Controller Manager** – Musí být spuštěn jako samostatný uzel, jelikož už nadále neexistuje žádný plugin, který by zajišťoval jeho simulaci. Zajišťuje správu a kontroluje kontroléry.
- **Hardware Interface** – Plugin, který zajišťuje propojení mezi „ros2_control“ a fyzickým hardwarem, konkrétně Arduinem a motory.
- **Resource Manager** – Součást Controller Manageru, která definuje, obdobně jako u simulované verze, jaký plugin používáme a jaké spojení (jointy) sledujeme a ovládáme.
- **Kontroléry** – Stejně řídicí prvky jako u simulace, tj. „diff_drive_controller“ a „joint_state_broadcaster“

Následující obrázek vystihuje schéma řídicího systému:



Obrázek 31 Schéma řídicího systému reálného robota

(zdroj: vlastní)

Jako prostředek komunikace mezi „ros2_control“ a Arduinem byl použit Hardware Interface plugin „diff_drive_arduino“, dostupný jako open-source projekt [24]. Tento plugin nebyl mnou vytvořen, ale převzat a umístěn do mého pracovního prostoru (workspace) v uložišti Raspberry Pi, kde se nachází vedle ROS balíčku. Plugin zajišťuje převod příkazů z kontrolérů na sériovou komunikaci s Arduinem Uno. Díky tomu je zajištěno splnění

příkazů na požadovanou rychlost otáčení kol a zároveň se do systému ROS vrací zpětná vazba o natočení kol a transformaci polohy robota.

Parametrické informace Resource Manageru byly, obdobně jako u simulované verze, zasazeny do souboru „ros2_control.xacro“.

```
<ros2_control name="RealRobot" type="system">
  <hardware>
    <plugin>diffdrive_arduino/DiffDriveArduino</plugin>
    <param name="left_wheel_name">left_wheel_joint</param>
    <param name="right_wheel_name">right_wheel_joint</param>
    <param name="loop_rate">30</param>
    <param name="device">/dev/serial/by-path/platform-fd500000.pcie-
pci-0000:01:00.0-usb-0:1.1:1.0-port0</param>
    <param name="baud_rate">57600</param>
    <param name="timeout">1000</param>
    <param name="enc_counts_per_rev">3270</param>
  </hardware>
  <joint name="left_wheel_joint">
    <command_interface name="velocity">
      <param name="min">-10</param>
      <param name="max">10</param>
    </command_interface>
    <state_interface name="velocity"/>
    <state_interface name="position"/>
  </joint>
</ros2_control>
```

Tyto parametry zajišťují přiřazení konkrétních spojů kol („left_wheel_joint“, „right_wheel_joint“) a jejich rozhraní (rychlost, pozice). Dále pak popisují použitý Hardware Interface plugin, komunikační parametry pro připojené Arduino (adresář cesty ke konkrétnímu portu USB, rychlost přenosu dat) a v neposlední řadě počet enkodrových pulsů, odpovídajících jedné otočce kola použitých motorů.

Pro rozlišení parametrů, reálného robota od simulovaného, je v souboru aktivační podmínka o načtení části kódu, odpovídající pouze příslušnému spouštěcímu skriptu.

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

  <xacro:unless value="$ (arg sim_mode)">
    <ros2_control name="RealRobot" type="system">
```

```

        </ros2_control>
    </xacro:unless>

    <xacro:if value="$ (arg sim_mode) ">
        <ros2_control name="IgnitionSystem" type="system">
            </ros2_control>
        </xacro:if>

    </robot>

```

6.4.5 Spuštění reálného robota

Pro úspěšné spuštění reálného robotického vozidla je nutné, aby byly předem zapnuty a napájeny všechny základní komponenty, tj. Raspberry Pi, Arduino Uno a H-můstek s motory. Současně je potřeba mít zapnutý i počítač, který slouží pro kontrolu robota skrz SSH a případně pro běh dalších nadřazených komponent. Těmi jsou ovládání pomocí herního ovladače či případně navigace.

Na straně Raspberry Pi se celý potřebný systém spouští pomocí launch souboru „robot.launch.py“, který se nachází v mém ROS balíčku. Připravený ROS balíček je nutné mít v tuto chvíli umístěný, jak v počítači, tak i v Raspberry Pi. Tento skript postupně spouští všechny potřebné uzly a jejich závislosti. Mezi ně patří:

- **rps.launch.py** – Skript vytvářející virtuální model robota a publikující popis robota v „robot_description“. Zároveň jsou vloženým uzlem aktivované parametry „ros2_control“.
- **rplidar.launch.py** – Spouští senzor lidar a publikuje z něj data.
- **camera.launch.py** – Spouští kameru a publikuje data.
- **Controller Manager** – Hlavní uzel „ros2_control“, načítající parametry z „robot_description“ a konfigurační soubor „controllers_params.yaml“ s nastaveními pro kontroléry.
- **Kontroléry** – Spuštění uzlů kontrolérů pro diferenciální řízení a publikaci stavu kol.

Tento jediný skript spouští všechno potřebné spojené s robotem. Dá se zavolat příkazem v terminálu, například pomocí SSH, následovně:

```
ros2 launch autobot robot.launch.py
```

Na straně počítače je potřeba samostatný, v tomto případě neimplementovaný, launch soubor „joystick.launch.py“, který pro robota zpřístupní témata se zprávami o rychlosti. Důvodem pro tohle rozčlenění je herní ovladač, který zůstává k počítači připojený, jako v případě simulace. Bez jeho spuštění není možné vozidlo ovládat dálkově ani využívat navigaci.

6.5 Mapování / lokalizace / navigace

Tato kapitola plynule navazuje na předchozí část, ve které bylo popsáno robotické vozidlo, jak po stránce hardwaru, tak i softwaru. Zároveň je reálná i simulovaná verze připravena na implementaci systémů vyššího stupně řízení. Tyto algoritmy již zajišťují schopnosti robota vnímat okolí, eliminovat chybu v odometrii, a samostatně se pohybovat v prostoru.

Těmito systémy jsou:

- **Mapování** probíhá v našem případě v 2D prostoru a jejím úkolem je zaznamenávat informace o obsazenosti prostoru, tj. rozpoznávání překážek a bezpečného volného prostoru.
- **Lokalizace** umožňuje robotu určovat svou vlastní polohu vůči překážkám, zejména podle dat z odometrie a z lidarů.
- **Navigace** zajišťuje samotný pohyb robota do cílového bodu, včetně plánování trasy a vyhýbání se překážkám.

Kombinací těchto tří systémů vzniká autonomní chování, díky kterým robot dokáže samostatně zpracovat okolní prostředí a efektivně se v něm pohybovat.

Součástí této kapitoly je také popis SLAM-u (Simultaneous Localization and Mapping), který dokáže současně vytvářet mapu a odhadovat svou vlastní pozici v ní. V rámci této práce jsem narazil na dvě možnosti využití SLAMu.

V prvním případě lze použít SLAM přímo s navigací. V tomto případě je SLAM aktivní po celou dobu provozu. To znamená, že vozidlo může být vysláno i do dosud nenamapované oblasti, tj. za jízdy bude zároveň mapovat prostředí, lokalizovat se a pokoušet se splnit navigační cíl. Díky tomu je možné robotu zadat cíl v neznámém prostředí. SLAM zajistí průběžnou korekci pozice, čímž eliminuje chyby dané odometrií. V ideálním případě

vozidlo dorazí až do cíle. Pokud je však cíl umístěn například v oblasti, kde se nachází překážka, robot se zastaví a navigování nedokončí. Dále platí, že pokud SLAM již jednou překážku rozpoznal a zaznamenal ji do mapy, zůstává tam již vyznačená. V případě fyzického odstranění překážky, ještě nějakou dobu trvá, než se odebere i z mapy. V „očích“ robota tedy místo zůstává neprůjezdné. To může v některých případech omezit další pohyb a plánování tras.

V druhém případě lze SLAM použít pouze jako prostředek pro vytvoření mapy. V tomto případě je robot ovládán ručně, například klávesnicí nebo herním ovladačem, a SLAM slouží pro zaznamenávání prostředí. Jakmile je mapování dokončeno, tak se mapa uloží a SLAM se vypne. S touto „statickou“ mapou nadále pracují samostatné lokalizační (již ne SLAM) a navigační systémy, které již nepracují s obměňující se mapou ale s mapou fixní. V tomto případě jsou nové překážky nadále zaznamenávané senzory, avšak již nezaneseny do mapy a po fyzickém odebrání z prostředí ihned zmizí i z „očí“ robota.

Nevýhodou tohoto řešení je nutnost manuálního namapování prostoru předem. Výhodou pak je, že robot může být navigován do předem známých míst. Tento přístup je zároveň vhodnější pro některé typy navigačních kontrolérů, které předpokládají práci se statickou mapou.

6.5.1 SLAM

Pro vytvoření mapy okolního prostředí robotického vozidla je využívána knihovna SLAM Toolbox [25], která byla nainstalována ve formě předpřipraveného balíčku pomocí příkazu:

```
ros2 apt install ros-humble-slam-toolbox
```

Tento balíček implementuje algoritmus SLAM (Simultaneous Localization and Mapping), jehož úkolem je současně lokalizovat robota v neznámém prostředí a zároveň toto prostředí mapovat. Algoritmus tedy po celý čas používání neustále odhaduje správnou pozici robota a vytváří pomocí detekce překážek mapu. Tahle mapa je ve formě 2D a obsahuje mřížku tvořenou pixely, které nesou informaci o obsazenosti. Každý namapovaný pixel může nést tmavou nebo světlou barvu v závislosti na tom, jestli je dané místo tvořeno překážkou nebo ne (viz Obrázek 32).



Obrázek 32 2D mapa vytvořená pomocí SLAM Toolbox

(zdroj: vlastní)

SLAM Toolbox zpracovává data ze senzoru Lidar, publikující do tématu „/scan“, a informace o odometrii robota. Z těchto dat pak algoritmus vypočítává transformaci mezi souřadnicovými rámci „/map“ a „/odom“, čímž kompenzuje chyby vzniklé vlivem nepřesné odometrie. Výsledkem je to, že pohyb robota po mapě odpovídá pohybu robota v reálném prostředí a vzniklá mapa odpovídá realitě.

Pro spuštění SLAM systému byl z knihovny přidán do vlastního balíčku spouštěcí skript „slam.launch.py“, který lze zavolat příkazem v terminálu:

```
ros2 launch autobot slam.launch.py
```

Tento skript zajistí spuštění potřebného uzlu s parametry, které jsou definované v konfiguračním souboru „slam_params.yaml“. Tento soubor obsahuje definici používaných rámců (například „/odom“ nebo „/base_footprint“) a nastavuje parametry zpracování dat z lidarů.

V této práci je SLAM Toolbox využíván především pro vytváření mapy, která je následně použita při lokalizaci a navigaci, bez dalšího využívání SLAM-u. Tento postup je vhodnější pro složitější překážkové dráhy (bludiště) a prostředí s dočasnými překážkami, u kterých nestojíme o namapování.

6.5.2 Lokalizace

Pro lokalizaci robota během navigování je použit jiný systém než při mapování pomocí SLAM. Místo něj je v tomto případě implementován lokalizační systém z knihovny Nav2 [26], která byla nainstalována pomocí příkazů:

```
ros2 apt install ros-humble-navigation2
```

```
ros2 apt install ros-humble-nav2-bringup
```

Lokalizační systém využívá metodu AMCL (Adaptive Monte Carlo Localization). Jedná se o pravděpodobnostní metodu lokalizace, které používá tzv. částicový filtr. Tento filtr udržuje množinu hypotéz o pozici robota (tzv. částice), přičemž každá částice reprezentuje jednu možnou pozici robota na mapě. Po každém obdržení aktualizovaných senzorických dat z lidarů dochází k přehodnocení částic a s tím i pravděpodobnosti o pozici robota na mapě. [27]

Spouštěcí skript lokalizace byl převzat z oficiálního balíčku knihovny a byl umístěn do balíčku vlastního. Ten lze spustit příkazem:

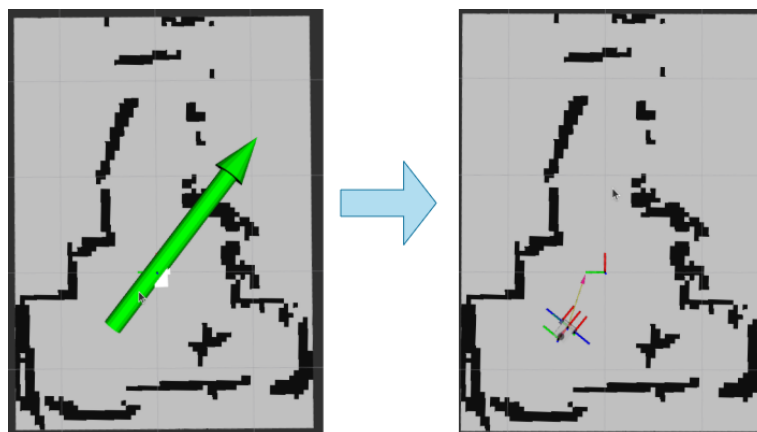
```
ros2 launch autobot localization_launch.py map:={nazev_mapy}.yaml
```

Tento python skript načítá konfiguraci z upraveného parametrického souboru „nav2_params.yaml“, který je sdílený i se systémem navigace. Skript lokalizace spouští dohromady tři uzly:

- **map_server** – Zajišťující načtení vytvořené mapy (pomocí SLAMu) ze souboru. Mapa je přiložena pomocí parametru v příkazovém řádku a publikována do ROS tématu „/map“, které lze zobrazit v RViz.
- **amcl** – Zajišťuje lokalizaci robotického vozidla. Porovnává data z tématu „/scan“ s mapou „map“ a za pomoci metody AMCL vytváří transformaci mezi rámci „/map“ a „/odom“. Tím kompenzuje chybu odometrie a výsledkem je pozice virtuálního robota na mapě, odpovídající poloze robota ve skutečnosti.
- **lifecycle_manager** – Stará se o automatizované řízení životního cyklu ostatních uzlů.

Parametry AMCL jsou nakonfigurovány v parametrickém souboru a nesou informace primárně o typu modelu vyhodnocování shody mezi skenem a mapou, adaptivního rozmezí počtu používaných částic, názvu souřadnicových rámců a v neposlední řadě minimálního nutného pohybu robota pro aktualizaci lokalizačního odhadu. Ten je v mém případě nastavený na 0.03 m (u vzdálenosti) a 0.03 rad (u úhlu), z důvodu pomalého pohybu v úzkých prostorech při testování.

Při každém spouštění lokalizačního systému v nástroji RViz je zobrazena mapa. Do ní je však potřeba ručně vložit pomocí nástroje „2D Pose Estimate“ počáteční polohu robota (viz Obrázek 33). Tím se inicializuje poloha robota, který se zobrazí v adekvátní pozici, a lokalizace začne fungovat. S funkční lokalizací lze následně spustit i systém navigace.



Obrázek 33 Inicializace počáteční polohy lokalizace

(zdroj: vlastní)

6.5.3 Navigace

Navigation system of the robotic vehicle is implemented using the Nav2 [26] library, providing a set of tools and nodes enabling autonomous movement of the robot from the starting position to the goal. Prerequisite for correct navigation is knowledge of the robot's current position, which is handled by the localization system AMCL or SLAM.

The navigation system connects several processes (nodes), which cooperate together to ensure that the robot reaches the goal autonomously. The system works in the coordinate frame „/odom“, „/map“ and „base_footprint“. It also utilizes data from the lidar through the „/scan“ topic. The result is the „/cmd_vel“ output topic, which contains velocity commands for the robot, processed by the motion controller.

Running navigation is ensured by a script, which is again taken from the official Nav2 package, modified for loading custom parameters and saved in a custom package. This is done using the command:

```
ros2 launch autobot navigation_launch.py
```

This script loads the configuration of individual nodes from the „nav2_params.yaml“ file, which is shared with the localization script. In the file, parameters and plugins of the mentioned nodes are specified, adapted to specific requirements of the environment and situation. Specific settings will be mentioned in the following chapter focused on robot testing.

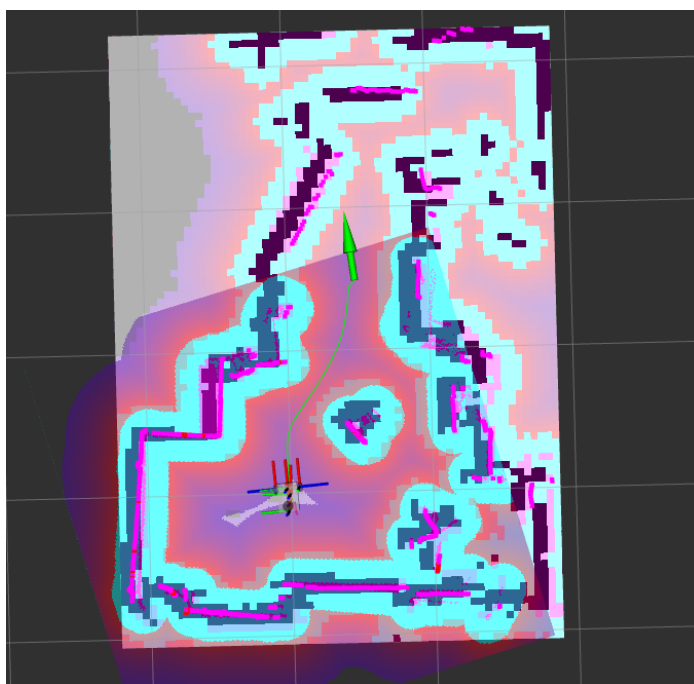
Nodes forming navigation:

- **Nav2_planner** – Uzel vytvářející globální cestu z pozice robota do cíle. Plánování probíhá na základě globální costmapy, reprezentující překážky a volný prostor v celém rozsahu mapy (globální plánování). Mezi používané pluginy patří například NavFn, Smac nebo Theta star.
- **Nav2_controller** – Uzel převádějící plánování na konkrétní pohyby robota. Zohledňuje globální cestu ale i aktuální překážky dle dat z lidarů, v závislosti na použitém pluginu (lokální plánování). Mezi lokální plánovače patří DWB, Model Predictive Path Integral (MPPI) nebo Regulated Pure Pursuit (RPP).
- **Nav2_vt_navigator** – Realizuje logiku celého navigačního procesu pomocí behaviorálních stromů (behavior trees). Koordinuje plánování, řízení, opakování cílů, zotavení z chyb atd.
- **Nav2_behaviours** – Obsahuje „recovery behavior“ plugin, které se využívají v případě, že se robot nedokáže dostat dál nebo zabloudí. Například otočení na místě, couvání, čekání.
- **Nav2_smoother** – Slouží k vyhlazení globální cesty, vedoucí k hladší a plynulejší trajektorii robota.
- **Nav2_velocity_smoother** – Vyhlazuje příkazy v tématu „/cmd_vel“, což vede k plynulejší jízdě a zamezí náhlým změnám akcelerace.
- **Nav2_waypoint_follower** – Uzel umožňující definovat vícero cílů, kterými robot projíždí postupně.
- **Nav2_lifecycle_manager** – Spravuje životní cyklus všech uvedených uzlů. Zajišťuje jejich inicializaci, spuštění a restart v případě selhání.

Pro reprezentaci okolního prostředí robota z pohledu plánování tras a vyhýbání se překážkám používáme tzv. costmapy. Costmapa je 2D mřížka, ve které každá buňka (pixel) označuje, jak moc nebezpečné pro robota je, aby v daném místě projel. Rozlišuje dva typy těchto map:

- **Global Costmap** – Mřížka zahrnující celé prostředí známé robotovi (2D mapa vytvořená SLAMem). Obsahuje všechny překážky, objekty a volný prostor.
- **Local Costmap** – Mřížka vytvářená v bezprostředním okolí robota a sloužící k podpoře lokálního plánovače. Je aktualizována v reálném čase z dat posbíraných lidarem. Využívána pro detekci dynamických překážek.

Obě mapy lze vizualizovat v prostředí RViz pro ladění a sledování navigace (viz Obrázek 34). Cíle navigace lze systému předávat pomocí příkazového řádku nebo nástroje „2D Goal Pose“. Možností je pak také spuštění speciálního okna navigace a zadání více cílů najednou, jimiž bude vozidlo následně projíždět.



Obrázek 34 Zobrazení Costmap v prostředí RViz

(zdroj: vlastní)

7 TESTOVÁNÍ AUTONOMNÍHO VOZIDLA

Autonomní vozidlo bylo testováno ve dvou případech, tj. se statickými a dynamickými překážkami. Nejprve bylo chování vozidla ověřeno v bludišti, tvořeného pouze ze statických překážek. Následně pak bylo otestováno v situaci, kdy se v jeho trase objevily dynamické překážky, na které muselo v reálném čase reagovat. Pro oba typy případů byla použita odlišná nastavení parametrů a pluginů navigačního systému. Konkrétní úpravy jsou popsány v následujících podkapitolách.

Při testování byl SLAM použit pouze pro vytváření mapy prostředí. Robotické vozidlo bylo nejprve manuálně ovládáno, aby zaznamenalo okolí a po dokončení mapování byla vygenerovaná mapa exportována. Následně byl SLAM vypnut a mapa využívána pro samotnou lokalizaci a navigační systém.

Tento přístup byl zvolen zejména kvůli testování ve složitějších prostředích, například v bludišti, kde je potřeba přesné a neměnné prostředí. Při testování s dynamickými překážkami je tento způsob výhodný také, jelikož statické překážky zůstávají pevně zaneseny v mapě. Zatímco dynamické objekty, jako jsou pohybující se osoby nebo předměty, jsou detekovány v reálném čase. Tyto pohyblivé překážky se nezapisují do uložené mapy, ale jsou pouze dočasně promítnuty do costmapy, se kterou navigační systém pracuje.

7.1 Testování na statických překážkách

V prvním testovacím scénáři bylo robotické vozidlo nasazeno do prostředí se statickými překážkami. Testovací prostředí bylo tvořeno bludištěm složeným z 5 x 4 segmentů, přičemž délka jednoho segmentu měřila 56 cm a výška 22 cm.



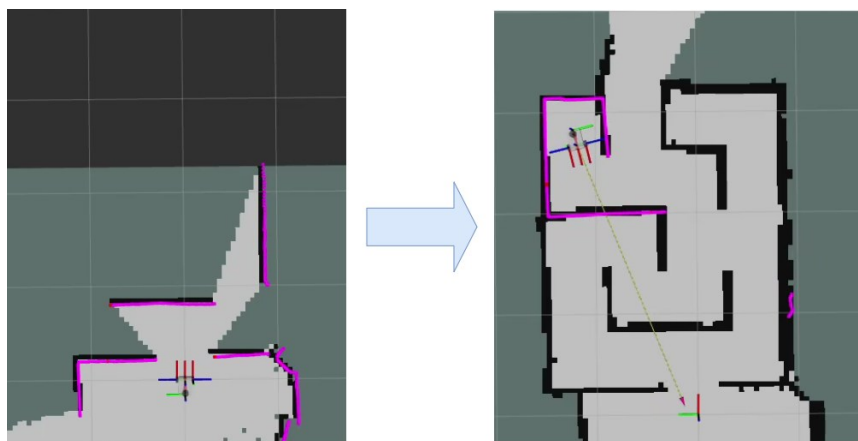
Obrázek 35 Překážková dráha – bludiště

(zdroj: vlastní)

7.1.1 Mapování

Teleoperované robotické vozidlo bylo ovládáno tak, aby postupně projelo celou překážkovou dráhou. Při průjezdu se v prostředí RViz průběžně vykreslovaly všechny segmenty a překážky bludiště.

Jakmile byly všechny části dráhy zaznamenány, mapa byla exportovaná do souboru ve formátu „.yaml“. Tento soubor slouží následně jako podklad pro lokalizační algoritmus.



Obrázek 36 Zaznamenávání překážek v prostředí RViz

(zdroj: vlastní)

7.1.2 Nastavení navigace

Pro správné fungování navigace v bludišti bylo nutné provést vhodné nastavení parametrů vybraných uzlů v souboru „nav2_params.yaml“.

Navigační uzel „nav2_controller“, který zajišťuje lokální plánování pohybu robota (lokální plánovač), byl vybaven pluginem RPP. Tento plugin je určen pro přesné sledování předem vygenerované trasy. Požadovaná maximální rychlost pohybu robota byla nastavena na 0.1 m/s. RPP pracuje na principu „virtuální mrkve“ (neviditelný bod), kterou vozidlo následuje s odstupem 0.17-0.18 m.

Pro generování globální trasy, kterou vozidlo sleduje, byl použit plugin Smac Planner v rámci uzlu „nav2_planner“. Tento globální plánovač byl zvolen z důvodu schopnosti vytvářet plynulé a realistické trajektorie. Důležitým parametrem ovlivňujícím generovanou trasu bylo nastavení „cost_travel_multiplier“ na vyšší hodnotu (např. 10). To vedlo ke generování tras s vyšším odstupem od překážek, díky čemuž robot projížděl zatáčky bezpečněji s menším rizikem kolize.

V rámci konfigurace globální a lokální costmapy byl nastaven parametr „robot_radius“ na 0,15 m. Udává velikost robota a slouží k vymezení ochranné zóny kolem překážek, do které by robot neměl vjíždět kvůli riziku kolize. Dále byly nastaveny parametry „inflation_radius“, který definuje vzdálenost, do které se kolem překážky vytváří bezpečná zóna a „cost_scaling_factor“, ovlivňující, jak rychle klesá rizikovost dané zóny. V lokální costmapě byly tyto parametry nastaveny na 0,55 m a 4,0, v globální pak byly zvoleny hodnoty 0,30 m a 10,0.

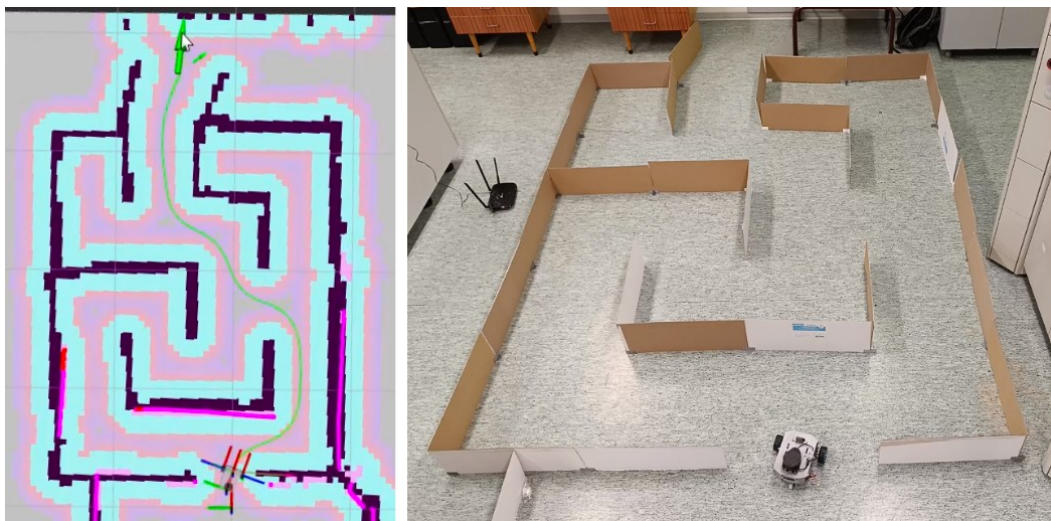
Ostatní parametry byly v průběhu testování upraveny na hodnoty, které vedly k nejstabilnějšímu a nejbezpečnějšímu chování.

7.1.3 Statické překážky – navigace

Pro umožnění autonomního pohybu robota v bludišti byl nejprve spuštěn lokalizační systém s vyexportovanou mapou. Dále byla nastavena počáteční pozice virtuálního robota v RViz tak, aby odpovídala skutečné poloze reálného robota v prostoru.

Po úspěšném spuštění lokalizace byl taktéž spuštěn navigační systém, čímž se v RViz zobrazili costmapy (za předpokladu, že jsou aktivovány v okně „Displays“).

Pomocí nástroje „2D Goal Pose“ v RViz byl definovaný cíl, ke kterému se mělo vozidlo autonomně přesunout. Po zadání cílové pozice se robot rozjel a začal sledovat vygenerovanou trasu k cíli. Cílový bod označený šipkou a naplánovaná trajektorie jsou označeny zelenou barvou (viz Obrázek 37).

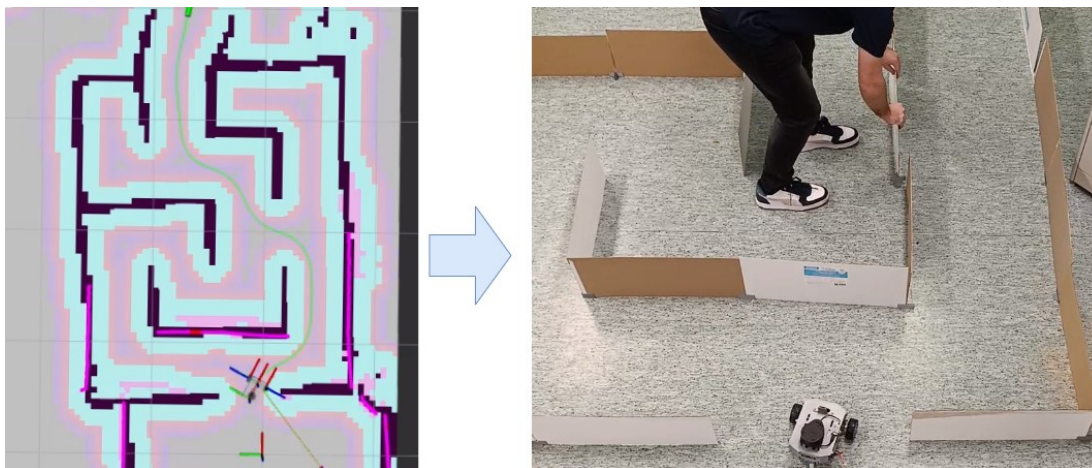


Obrázek 37 Navigace k bludišti

(zdroj: vlastní)

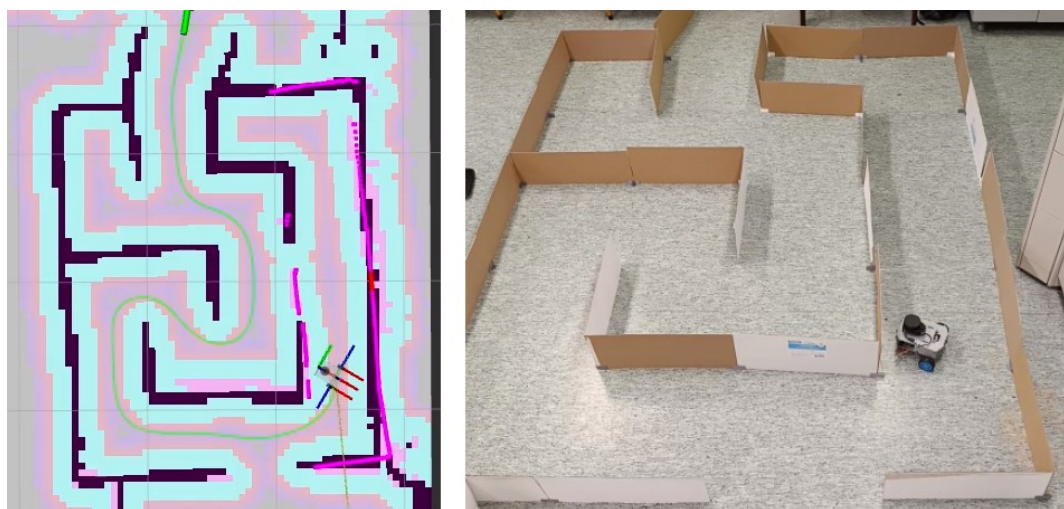
Po úspěšném dosažení cíle byla situace zopakovaná. Robotické vozidlo bylo přemístěno zpět na počáteční pozici a byl znovu definován cíl pomocí nástroje „2D Goal Pose“. V tomto opakovaném scénáři byla však testovací situace rozšířena o novou překážku, která byla dodatečně umístěna do dráhy robota (viz Obrázek 38). Překážkou byl segment přidaný po startu navigace, který znemožnil vozidlu pokračovat k cíli vygenerovanou kratší cestou/trasou bludištěm.

Navigační systém detekoval změnu v prostředí na základě senzorických dat, aktualizoval globální plán a vygeneroval novou trasu k cíli. Tato trasa umožnila vozidlu bezpečně obejít překážku a dostat se k cíli po delší, ale schůdné cestě (viz Obrázek 39).



Obrázek 38 Přidání překážky do bludiště

(zdroj: vlastní)



Obrázek 39 Navigace bludištěm alternativní trasou

(zdroj: vlastní)

7.2 Testování na dynamických překážkách

Ve druhém testovacím scénáři bylo robotické vozidlo nasazeno v uzavřeném prostoru. Do něj byly následně náhle po startu navigace vloženy překážky, jejichž výskyt nebyl vozidlu dopředu znám. Testovací prostor byl ohraničen barierou složenou z 6 x 2 segmentů, která sloužila k omezení pohybové oblasti vozidla. Důvodem pro to bylo zabránění tomu, aby vozidlo jelo mimo požadovaný prostor. To by mohlo vést ke zkreslení výsledků testů tím, že by se vozidlo překážkám výrazně vyhýbalo.

Ohraničený prostor byl před samotným testováním navigace zmapován a výsledná mapa exportována, pro použití lokalizace a navigace. Bariéru bylo tedy možné označit za statickou překážku.

7.2.1 Nastavení navigace

Pro potřeby tohoto scénáře byly opět poupraveny některé parametry v konfiguračním souboru „nav2_params.yaml“, aby lépe odpovídali prostředí s nepředvídatelnými překážkami.

Navigační uzel „nav2_planner“ byl v tomto případě vybaven pluginem MPPI. Tento lokální plánovač pracuje na principu vzorkování možných trajektorií, které následně vyhodnocuje a optimalizuje. Výhodou MPPI je schopnost zohledňovat nejen globálně naplánovanou trasu, ale také rychle reagovat na aktuální situaci v bezprostředním okolí robota. Pokud to situace vyžaduje, robot není nucen následovat původní trasu a dokáže ji opustit, což přispívá k plynulejší a bezpečnější jízdě k cíli. Tenhle postup však není vhodný například pro scénář s bludištěm, kde je nutné sledovat naplánovanou trasu a kde přímá cesta vzdušnou čarou často vede skrz překážky. MPPI v takovém případě může vyhodnocovat kolizní trasu jako nejvýhodnější, ale ostatní subsystémy ji zabrání, takže robot zůstane stát a není schopen pokračovat.

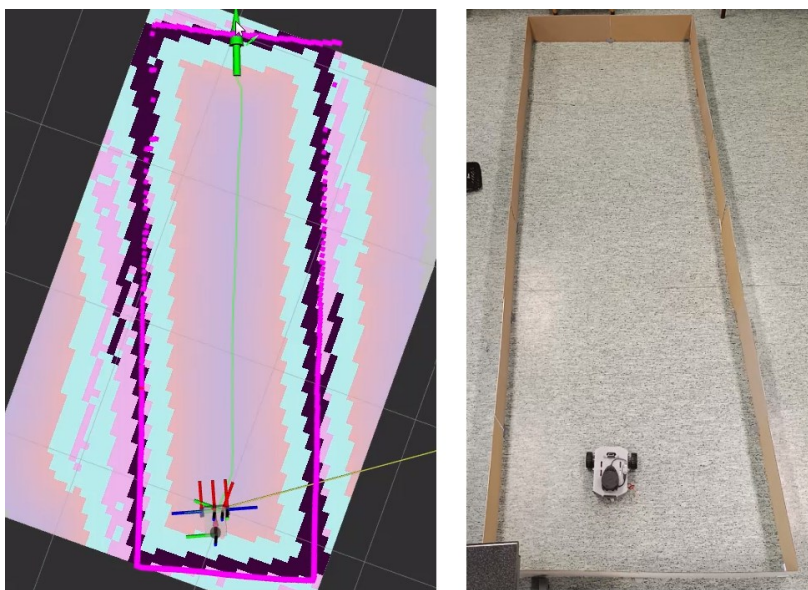
V rámci konfigurace lokální a globální costmapy byl stanoven parametr poloměru robota (robot_radius) opět na 0.15 m, což opět vytvořilo bezpečnou zónu okolo překážek. Hodnoty parametrů „cost_scaling_factor“ a „inflation radius“ byly v případě obou costmap nastaveny na 3.0 a 0.55 m.

Zbýlé parametry byly nastaveny tak, aby bylo dosaženo co nejstabilnějšího chování při testování. Případně pak byly zachovány výchozí parametry z původního souboru knihovny Nav2 pro ROS 2 Humble.

7.2.2 Dynamické překážky – navigace

Pro zajištění autonomní navigace byl opět nejprve spuštěn lokalizační systém, využívající dříve vytvořenou mapu. Do mapy byla v prostředí RViz vložena aktuální poloha robota, čímž se aktivovala lokalizace.

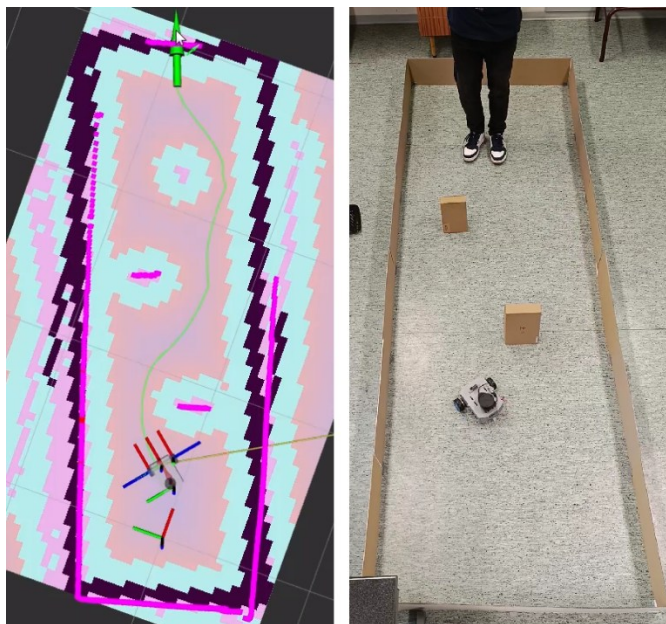
Následně pak byl spuštěn navigační systém a pomocí nástroje „2D Goal Pose“ byla zadána cílová pozice, ke které má robotické vozidlo samostatně dorazit. Na základě této pozice byla vygenerována globální trasa, s její pomocí se robot vydal k cíli (viz Obrázek 40).



Obrázek 40 Uzavřený testovací prostor

(zdroj: vlastní)

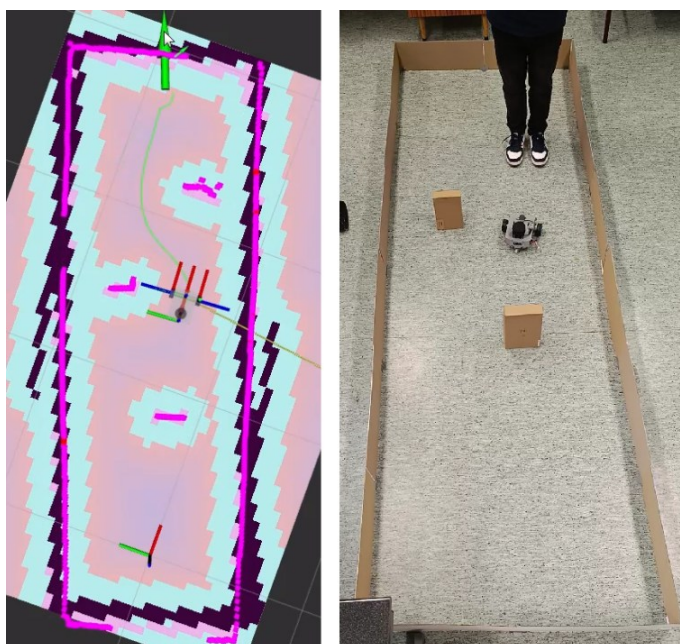
Během pohybu k cílovému bodu byly do plánované trasy vozidla náhle umístěny překážky, konkrétně dvě kartonové krabice a člověk. Robotické vozidlo tyto objekty samostatně detekovalo a okamžitě upravilo svou trajektorii pohybu tak, aby se překážkám vyhnulo (viz Obrázek 41).



Obrázek 41 Detekce nových překážek

(zdroj: vlastní)

V průběhu navigace došlo k další změně situace, kdy člověk náhle před vozidlem ustoupil do strany (viz Obrázek 42). Vozidlo i na tuto změnu zareagovalo, přepočítalo trasu a bezpečně pokračovalo v jízdě. Navigační systém tak prokázal schopnost reagovat na dynamicky měnící se prostředí a posléze se dostat do předem stanoveného cíle, bez potřeby manuálního zásahu.



Obrázek 42 Detekce přemístění člověka

(zdroj: vlastní)

ZÁVĚR

Cílem této bakalářské práce bylo navrhnout a realizovat autonomní robotické vozidlo s využitím ROS (Robot Operating System), které bude schopné samostatného pohybu v prostředí s překážkami. V úvodní části práce byly prostudovány základní principy fungování autonomních vozidel a architektura platformy ROS. Získané poznatky následně sloužily jako východisko pro hardwarové i softwarové řešení vozidla.

V průběhu práce byl kompletně sestrojen model vozidla, jehož konstrukce vychází z vlastního návrhu. Následně byla konstrukce vytištěna na 3D tiskárně. Pro diferenciální řízení motorů bylo použito Arduino, přičemž každý motor je vybaven enkodérem pro snímání otáček a odhad ujeté vzdálenosti. Hlavním výpočetním prvkem, instalovaným na vozidle, byl použit mikropočítač Raspberry Pi s nainstalovaným frameworkem ROS. Pro snímání okolí byl využit 2D LiDAR, který slouží k detekci překážek.

Byl vytvořen řídicí systém, umožňující komunikaci mezi jednotlivými komponenty vozidla a realizována implementace podstatných funkcí jako je SLAM, následná lokalizace v již známém prostředí a samotná autonomní navigace. Vozidlo je schopné reagovat jak na statické, tak i dynamické překážky, což bylo ověřeno v testovacích scénářích, včetně navigování v modulárním bludišti.

Tato práce demonstruje, že je možné navrhnout a realizovat funkční autonomní vozidlo s využitím open-source softwaru a cenově dostupného hardwaru. Výsledné řešení tak může sloužit jako základ pro další vývoj nebo výukové projekty.

SEZNAM POUŽITÉ LITERATURY

- [1] PRITZ, Helmut. Autonomní řízení. Online. MD ELEKTRONIK. 2023. Dostupné z: <https://www.md-elektronik.com/cs/autonomni-rizeni/>. [cit. 2024-12-29].
- [2] SIEGWART, Roland; NOURBAKHSH, Illah Reza a SCARAMUZZA, Davide. *Introduction to autonomous mobile robots*. 2nd ed. Cambridge, Mass: MIT Press, 2011. ISBN 9780262295321.
- [3] MEJZLÍK, Bohumír. *Všesměrový synchronní podvozek*. Online, Diplomová práce, vedoucí František Šolc. Brno: Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav automatizace a měřicí techniky, 2009. Dostupné z: <http://hdl.handle.net/11012/2199>. [cit. 2024-12-29].
- [4] JOSEPH, Lentin a JOHNY, Aleena. *Robot operating system (ROS) for absolute beginners*. 2nd ed. New York: NY : Apress, 2022. ISBN 9781484277508.
- [5] *AGV vs. AMR - jaký je mezi nimi rozdíl?* Online. Mobile Industrial Robots. 2024. Dostupné z: <https://mobile-industrial-robots.com/cs/blog/agv-vs-amr>. [cit. 2024-12-29].
- [6] PINKER, Jiří. *Mikroprocesory a mikropočítače*. Praha: BEN - technická literatura, 2004. ISBN 9788073001100.
- [7] STAPLE, Danny. *Learn robotics programming*. 2nd ed. Birmingham: Packt Publishing, 2021. ISBN 9781839217869.
- [8] *Raspberry Pi*. Online. In: Wikipedia: the free encyclopedia. San Francisco (CA): Wikimedia Foundation, 2024. Dostupné z: https://cs.wikipedia.org/wiki/Raspberry_Pi. [cit. 2024-12-30].
- [9] *What is Arduino?* Online. Arduino. 2018. Dostupné z: <https://www.arduino.cc/en/Guide/Introduction>. [cit. 2024-12-29].
- [10] JOSEPH, Lentin a CACACE, Jonathan. *Mastering ROS for robotics programming*. 3rd ed. Birmingham: Packt Publishing, 2021. ISBN 9781801079464.
- [11] MACENSKI, S., FOOTE, T., GERKEY, B., LALANCETTE, C. a WOODALL, W. *Robot Operating System 2: Design, architecture, and uses in the wild*. Online. *Science Robotics*. 2022, vol. 7. Dostupné z: <https://doi.org/10.1126/scirobotics.abm6074>. [cit. 2025-03-17].

- [12] OPEN ROBOTICS. ROS 2 *Humble documentation*. Online. 2025 Dostupné z: <https://docs.ros.org/en/humble/index.html>. [cit. 2025-03-17].
- [13] NEWANS, Josh. *Articulated Robotics*. Online. 2025. Dostupné z: <https://articulatedrobotics.xyz/>. [cit. 2025-03-17].
- [14] Stavebnice podvozku robota 4 všesměrová kola [obrázek]. Online. Dostupné z: <https://www.postavrobota.cz/Stavebnice-podvozku-robota-4-vsesmerove-kola-d2197.htm>. [cit. 2025-04-02].
- [15] ATLAS 2016. Robotsguide [obrázek]. Online. Dostupné z: <https://robotsguide.com/robots/atlas2016>. [cit. 2025-04-02].
- [16] Boston Dynamics. Spot Robot [obrázek]. Online. Dostupné z: <https://bostondynamics.com/news/boston-dynamics-expands-global-sales-of-spot-robot/>. [cit. 2025-04-02].
- [17] TechDesign. Automated Guided Vehicles (AGVs): Types, Definition, Solutions [obrázek]. Online. Dostupné z: <https://blog.techdesign.com/automated-guided-vehicles-agvs-types-definition-solutions/>. [cit. 2025-04-02].
- [18] Omron. Autonomous Mobile Robot [obrázek]. Online. Dostupné z: <https://industrial.omron.cz/cs/products/autonomous-mobile-robot>. [cit. 2025-04-02].
- [19] RASPBERRY PI 4 B [obrázek]. Online. Dostupné z: <https://www.conrad.cz/cs/p/raspberry-pi-4-b-4-gb-4-x-1-5-ghz-raspberry-pi-2138865.html>. [cit. 2025-04-02].
- [20] LCD Shield for Arduino 16x2, Blue LED Backlight [obrázek]. Online Dostupné z: <https://www.microbot.it/en/product/160/LCD-Shield-for-Arduino-16x2-Blue-LED-Backlight.html>. [cit. 2025-04-02].
- [21] ARDUINO UNO R3 [obrázek]. Online. Dostupné z: <https://rpishop.cz/arduino-uno/1356-arduino-uno-r3.html>. [cit. 2025-04-02].
- [22] Open Robotics. Ubuntu Install (binary) — ROS 2 Humble documentation. Online. Dostupné z: <https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debs.html>. [cit. 2025-04-02].
- [23] GOEBEL, Patrick. `ros_arduino_bridge` Online. GitHub, 2012. Dostupné z: https://github.com/hbrobotics/ros_arduino_bridge. [cit. 2025-04-06].

- [24]NEWANS, Josh. diffdrive_arduino Online. GitHub, 2020. Dostupné z: https://github.com/joshnewans/diffdrive_arduino. [cit. 2025-04-06].
- [25]MACENSKI, S. a JAMBRECIC, I. SLAM Toolbox: SLAM for the dynamic world. Journal of Open Source Software Online. 2021, 6(61), 2783. ISSN 2475-9066. Dostupné z: <https://doi.org/10.21105/joss.02783>. [cit. 2025-04-06].
- [26]MACENSKI, Steve, Francisco MARTÍN, Ruffin WHITE a Jonatan GINÉS CLAVERO. The Marathon 2: A Navigation System. In: 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020. Dostupné z: <https://arxiv.org/abs/2003.00368>
- [27]Adaptive Monte Carlo Localization – Robotics Knowledgebase. Online Dostupné z: <https://roboticsknowledgebase.com/wiki/state-estimation/adaptive-monte-carlo-localization/>. [cit. 2025-04-30].
- [28]Open Robotics; Clearpath Robotics. TurtleBot. Online. Dostupné z: <https://www.turtlebot.com/>. [cit. 2025-05-05].
- [29]NATO Science and Technology Organization. STO-EN-SCI-271-06: Advanced Materials for Defense Applications [online]. Neuilly-sur-Seine: NATO STO, 2013. Dostupné z: <https://www.sto.nato.int/publications/STO%20Educational%20Notes/STO-EN-SCI-271/EN-SCI-271-06.pdf>. [cit. 2025-05-05].

SEZNAM OBRÁZKŮ

Obrázek 1 Rozdělení typů podvozků mobilních robotů	17
Obrázek 2 Diferenciální typ podvozku [4]	18
Obrázek 3 Všesměrový typ podvozku [14].....	19
Obrázek 4 Robot Atlas [15]	20
Obrázek 5 Robot Spot [16]	20
Obrázek 6 AGV Roboti [17].....	21
Obrázek 7 AMR Omron [18].....	22
Obrázek 8 Zjednodušené schéma počítače	23
Obrázek 9 Typické rozložení desky Raspberry Pi [19]	26
Obrázek 10 Arduino UNO s připojeným LCD shieldem [20].....	27
Obrázek 11 Arduino Uno [21]	28
Obrázek 12 Vývojové prostředí Arduino IDE	29
Obrázek 13 Spuštění uzlu ROS systému příkazovým řádkem	32
Obrázek 14 Komunikace skrz téma (topic) mezi uzly (nodes).....	33
Obrázek 15 Komunikace skrz služby.....	34
Obrázek 16 Komunikace skrz akce.....	35
Obrázek 17 Transformace souřadnic	37
Obrázek 18 Vizualizační prostředí RViz	40
Obrázek 19 Gazebo Classic (vlevo) a RViz (vpravo).....	41
Obrázek 20 TurtleBot 4 [28]	42
Obrázek 21 Návrh robotického vozidla v prostředí Solidworks.....	46
Obrázek 22 Tisk modelu na 3D tiskárně.....	47
Obrázek 23 Elektronika uvnitř robotického vozidla	48
Obrázek 24 Podvozek se zabudovanými motory a baterií	49
Obrázek 25 Motor robota připevněný k základní nosné desce	50
Obrázek 26 Hotové robotické vozidlo se senzory na vrchní krycí desce	51
Obrázek 27 Schéma funkce Robot State Publisher.....	58
Obrázek 28 RViz – vizualizace modelu robota	60
Obrázek 29 Gazebo Ignition (vlevo) a Gazebo Classic (vpravo)	63
Obrázek 30 Schéma řídicího systému simulace.....	64
Obrázek 31 Schéma řídicího systému reálného robota.....	70
Obrázek 32 2D mapa vytvořená pomocí SLAM Toolbox.....	75
Obrázek 33 Inicializace počáteční polohy lokalizace	77
Obrázek 34 Zobrazení Costmap v prostředí RViz.....	79

Obrázek 35 Překážková dráha – bludiště.....	81
Obrázek 36 Zaznamenávání překážek v prostředí RViz	82
Obrázek 37 Navigace k bludišti.....	83
Obrázek 38 Přidání překážky do bludiště	84
Obrázek 39 Navigace bludištěm alternativní trasou	84
Obrázek 40 Uzavřený testovací prostor.....	86
Obrázek 41 Detekce nových překážek.....	87
Obrázek 42 Detekce přemístění člověka	87

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

ROS	Robot Operating System
SLAM	Simultaneous Localization And Mapping
URDF	Unified Robot Description Format
AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robots
AMCL	Adaptive Monte Carlo Localization
GUI	Graphical User Interface
DDS	Data Distribution Service
DOF	Degree Of Freedom
LTS	Long Term Support
PWM	Pulse Width Modulation
USB	Universal Serial Bus
MPPI	Model Predictive Path Integral
RPP	Regulated Pure Pursuit
RSP	Robot State Publisher
SSH	Secure Shell
DHCP	Dynamic Host Configuration Protocol
CSI	Camera Serial Interface
IDE	Integrated Development Environment
OS	Operating System

SEZNAM PŘÍLOH

Příloha P I: Schéma baterie a rozvedení elektrické energie do elektroniky robota

Příloha P II: Schéma připojení motorů k Arduino Uno

**PŘÍLOHA P I: SCHÉMA BATERIE A ROZVEDENÍ ELEKTRICKÉ
ENERGIE DO ELEKTRONIKY ROBOTA**

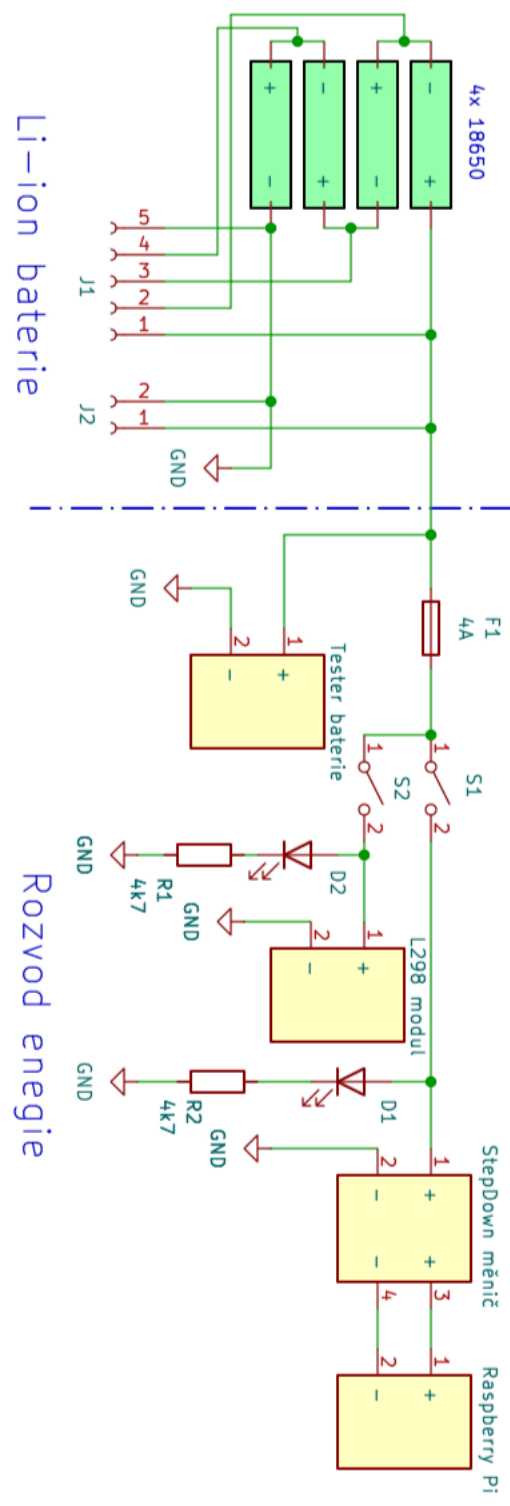


Schéma zapojení baterie a elektroniky

PŘÍLOHA P II: SCHÉMA PŘIPOJENÍ MOTORŮ K ARDUINU UNO



Schéma připojení motorů k Arduino UNO skrz H-můstek L298N