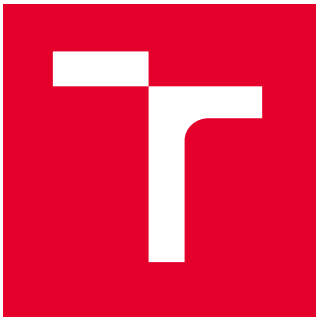


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV VÝROBNÍCH STROJŮ, SYSTÉMŮ A ROBOTIKY

INSTITUTE OF PRODUCTION MACHINES, SYSTEMS AND ROBOTICS

ROBOT PRO SLEDOVÁNÍ ČÁRY ŘEŠENÝ POMOCÍ ROBOTICKÉHO OPERAČNÍHO SYSTÉMU ROS

A LINE FOLLOWING ROBOT SOLVED USING THE ROS ROBOTIC OPERATING SYSTEM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. Jaroslav Svoboda**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Marada, Ph.D.**

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav výrobních strojů, systémů a robotiky
Student:	**Bc. Jaroslav Svoboda**
Studijní program:	Výrobní stroje, systémy a roboty
Studijní obor:	bez specializace
Vedoucí práce:	**Ing. Tomáš Marada, Ph.D.**
Akademický rok:	2024/25

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Robot pro sledování čáry řešený pomocí robotického operačního systému ROS

Stručná charakteristika problematiky úkolu:

V továrnách a skladech se s vozítky bez lidských operátorů potkáváme stále častěji. Pro intralogistiku jsou autonomní vozíky cestou, jak zefektivnit a zrychlit procesy, a samozřejmě možností, jak lidem uvolnit ruce. Cílem práce je návrh a realizace robotu pro sledování čáry. Realizace bude provedena za použití platformy ROS (Robot Operating System). ROS je unikátní platforma, která otevírá dveře do fascinujícího světa robotiky. Nabízí robustní a flexibilní prostředí pro vývoj a nasazení robotických aplikací, ať už se jedná o jednoduché výzkumné projekty nebo komplexní průmyslové roboty. Jsou předpokládány základní znalosti HW a programování.

Cíle diplomové práce:

Rešerše v oblasti existujících konstrukcí robotů pro sledování čáry s jejich přednostmi a slabinami.
Návrh variant konstrukčního řešení, včetně zdůvodnění výběru konkrétního řešení a stanovení jeho technických parametrů.
Popis robotického operačního systému ROS.
Realizace navrženého řešení za použití frameworku ROS a vývojových desek například Raspberry Pi nebo Raspberry Pi Pico.
Vytvoření podrobné dokumentace.
Zhodnocení a otestování realizovaného řešení.
Závěry a doporučení pro praxi.

Seznam doporučené literatury:

ROS2 Documentation. Online. Dostupné z: https://wiki.ros.org/ [cit. 2024-10-23].

ANIS, Koubaa (ed.). Robot Operating System (ROS): The Complete Reference. Netherlands: Springer Nature, 2017. ISBN 3319549278.

SUBRAMANIAN, Rajesh. Build autonomous mobile robot from Scratch using ROS: simulation and hardware. New York: Apress, 2023. ISBN 978-1-4842-9644-8.

KIM, DaeEun. Advanced Mobile Robotics: Volume 1. MDPI - Multidisciplinary Digital Publishing Institute, 2020. ISBN 3039219162. Dostupné z: https://doi.org/10.3390/books978-3-03921-917-9 [cit. 2024-10-23].

PRAKASH, M. Surya; VIGNESH, K. Ajay; SHYAMSUNTHAR, J.; RAMAN, K.; RAJU, J. Senthil et al. Computer Vision Assisted Line Following Robot. Online. In: Procedia Engineering. Elsevier, 2012, s. 1764-1772. ISSN 1877-7058. Dostupné z: https://doi.org/10.1016/j.proeng.2012.06.215. [cit. 2024-10-23].

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2024/25

V Brně, dne

L. S.

doc. Ing. Petr Blecha, Ph.D., FEng. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato diplomová práce se zabývá návrhem a implementací robotu pro sledování čáry s využitím robotického operačního systému ROS2. V rámci práce je vytvořena simulace s modelem robotu s diferenciálním pohonem v prostředí Gazebo Sim, kde robot sleduje čáru a reaguje na překážky pomocí virtuálních senzorů. Celý řídící algoritmus je implementován jako ROS2 node v jazyce Python. Následně je tento algoritmus přenesen na Raspberry Pi 5 a Raspberry Pi Pico, prostřednictvím kterých je robot v simulaci řízen. Zatímco na Raspberry Pi 5 je implementován plnohodnotný ROS2 node, na Raspberry Pi Pico je algoritmus zjednodušen a napsán v jazyce C s využitím micro-ROS frameworku. Součástí řešení je také tvorba Docker kontejneru, které usnadňuje nasazení a sdílení celého projektu. Výsledný systém je navržen jako otevřený a rozšiřitelný nástroj vhodný pro výuku a testování robotických aplikací. Práce také detailně popisuje konfiguraci prostředí ROS2, Gazebo Sim a exportu modelu ve formátu URDF ze softwaru SolidWorks.

ABSTRACT

This diploma thesis focuses on the design and implementation of a line-following robot using the Robot Operating System ROS2. As a part of the project, a simulation with a differential drive robot model was created in Gazebo Sim environment, where the robot follows a line and reacts to obstacles using virtual sensors. The control algorithm was implemented as a ROS2 node in Python. Then, this algorithm was transferred to Raspberry Pi 5 and Raspberry Pi Pico, which were used to controll the robot in simulation. While a full ROS2 node was implemented on the Raspberry Pi 5, the algorithm was simplified and written in C on the Raspberry Pi Pico using the micro-ROS framework. The solution also includes the creation of a Docker container, which makes the deployment and sharing of the entire project much easier. The final system is designed as an open and extensible tool suitable for education and testing robotic applications. The work also describes in detail the configuration of the ROS2 environment, Gazebo Sim and the process of exporting a model in URDF format from SolidWorks.

KLÍČOVÁ SLOVA

ROS2, Gazebo Sim, micro-ROS, Raspberry Pi, sledování čáry

KEYWORDS

ROS2, Gazebo Sim, micro-ROS, Raspberry Pi, line follow

BIBLIOGRAFICKÁ CITACE

SVOBODA, Jaroslav. *Robot pro sledování čáry řešený pomocí robotického operačního systému ROS*. Online, diplomová práce. Tomáš MARADA (vedoucí práce). Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2025. Dostupné z: https://www.vut.cz/studenti/zav-prace/detail/166080. [cit. 2025-05-11].

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat vedoucímu práce Ing. Tomášovi Maradovi, Ph.D. za cenné rady a odbornou pomoc s vypracováním této diplomové práce. Velké poděkovaní pak patří hlavně mé rodině, Honzovi, Jiřímu a mým nejbližším za podporu během celého studia.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. Tomáše Marady Ph.D. a s použitím literatury uvedené v seznamu.

V Brně dne 22.5.2025 ……

Bc. Svoboda Jaroslav

OBSAH

1 ÚVOD 15
2 MOTIVACE 17
3 PŘEHLED SOUČASNÉHO STAVU POZNÁNÍ 19
3.1 Navigační metody mobilních robotů v průmyslu 21
3.2 Techniky a technologie pro sledování čáry 21
3.3 Robot Operating System (ROS2) 25
3.3.1 Klíčové komponenty ROS2 a jejich struktura 25
3.3.2 Unified Robotics Description Format (URDF) 27
3.3.3 Micro-ROS 28
3.4 Gazebo Sim a Rviz 29
3.5 Docker 30
4 VARIANTY ŘEŠENÍ 31
4.1 Způsoby sledování čáry 32
4.2 Princip zvoleného řešení v Gazebo Sim 33
5 ROS2 35
5.1 Instalace ROS2 Jazzy Jalisco 35
5.2 Nové pracovní prostředí a package 36
6 SIMULAČNÍ MODEL ROBOTU V ROS2 A PROSTŘEDÍ SIMULACE 38
6.1 Postup exportu modelu ze SolidWorks 38
6.2 Definování senzorů v Gazebo Sim 42
6.3 Sjednocení souborů pomocí XACRO 43
6.4 Zobrazení a kontrola modelu v RViz a VS Code 44
6.5 Tvorba launch souborů 47
7 SIMULACE A TESTOVÁNÍ 49
7.1 Tvorba prostředí simulace 49
7.2 Gazebo Sim bridge a vizualizace 51
7.3 Tvorba a implementace ovládacích nodů 53
7.4 Ovládání simulace s Raspberry Pi a Pi Pico 57
7.4.1 Integrace skriptu na Raspberry Pi 5 58
7.4.2 Integrace skriptu na Raspberry Pi Pico 59
7.5 Tvorba Docker kontejneru pro projekt a simulaci 60
8 ZHODNOCENÍ A DISKUZE DOSAŽENÝCH VÝSLEDKŮ 63
9 ZÁVĚR 64
10 SEZNAM POUŽITÝCH ZDROJŮ 65
11 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK 69
11.1 Seznam zkratek 69
11.2 Seznam obrázků 69
11.3 Seznam tabulek 70
12 SEZNAM PŘÍLOH 71

1 ÚVOD

Autonomní vozidla AGV pronikají do fabrik, skladů i nemocnic a stává se z nich modelový příklad moderní robotiky. Jejich výzkum ale často brzdí prostá překážka, kterou je nedostatek fyzického hardwaru. Robot nelze rozebrat mezi desítky studentů a testovat nový kód, když hrozí kolize a opotřebení. Omezený přístup ke strojům tak ničí kreativitu a zpomaluje vývoj.

Robot Operating System (ROS2) společně např. se simulátorem Gazebo Sim tuto bariéru z části odstraňuje. Vytváří softwarovou vrstvu, ve které senzory, akční členy i řídící algoritmy používají stejný způsob komunikace bez ohledu na to, zda právě běží ve virtuálním světě, nebo v tom skutečném na reálném stroji. Díky tomu je možné ladit celý systém od regulace motorů po složité plánování trasy, a to vše pouze na obrazovce počítače. Virtuální AGV tak může projíždět simulované skladiště, detekovat překážky a sledovat čáru bez jakéhokoliv rizika, a přitom generovat data srovnatelná s reálným provozem.

Tato diplomová práce si klade za cíl nejprve prozkoumat existující line-follow roboty, včetně porovnání jejich silných a slabých stránek a zmapování různých druhů a způsobů navigace AGV. Na základě rešerše je navržena konstrukce robotu, která je následně převedena do formátu URDF pomocí softwaru SolidWorks a pluginu Solidworks to URDF exporter. Model robotu je poté integrován do prostředí ROS2 a Gazebo Sim ve scénáři sledování čáry, kde jsou zahrnuty i některé bezpečnostní funkce.

Řídící skripty, psané v Pythonu a C, jsou realizovány na deskách Raspberry Pi 5 a Raspberry Pi Pico, které v práci zachovávají roli výpočetních uzlů připojených k simulaci. Zatímco celá simulace běží na PC, Raspberry Pi přes lokální síť ovládá robot v simulaci. Práce detailně popisuje kroky, které jsou nezbytné ke zkompletování celého robotického systému a ukazuje, jak lze AGV navrhnout a oživit bez fyzického hardwaru.

2 MOTIVACE

Téma své diplomové práce jsem si zvolil na základě zkušenosti z programu Erasmus+ na Tampere University ve Finsku. V kurzu Mechatronics and Robot Programming jsem se poprvé setkal s robotickým operačním systémem ROS2 a Gazebo Sim a velmi mě zaujal způsob, jakým lze navrhovat, ladit a simulovat robotické systémy. Možnost okamžitě sledovat dopad rozdílných konfigurací systému a jednotlivých komponent v prostředí Gazebo Sim pro mě byla velkou inspirací. Tato zkušenost mě motivovala pokračovat a dále rozvíjet své znalosti v ROS2 a jsem velmi rád, že jsem tuto možnost na VUT dostal v rámci diplomové práce.

3 PŘEHLED SOUČASNÉHO STAVU POZNÁNÍ

Automaticky řízená vozidla (AGV – Automatic guided vehicles) představují jeden z hlavních prvků moderní logistiky, který umožňuje efektivní a autonomní přepravu materiálu v rámci výrobních a skladových prostorů. Díky jejich implementaci je dosaženo vyšší efektivity, bezpečnosti a snížení provozních nákladů. Příklad takových autonomních vozidel lze vidět na Obr. 1. [1]

Namísto spoléhání se na lidského operátora, využívají kombinace senzorů, softwaru a navigačních technologií k sledování předem stanovených tras. Tyto trasy mohou být tvořeny např. magnetickou páskou nebo jinou reflexní čarou. AGV jsou stále více používané a kvůli tomu se očekává, že do roku 2028 vzroste trh na 3.3 miliardy dolarů z původních 2.4 miliard dolarů zaznamenaných v roce 2023. [1]

AVG se dnes používají v mnoha odvětvích [1]:

- Výrobní sektor – AGV přepravují polotovary, rozpracované díly a hotové produkty mezi jednotlivými pracovními stanicemi. Díky tomu se mohou pracovníci věnovat důležitějším úkonům, než je transport materiálu.
- Skladování a logistika – V dnešní době automatizovaných skladů je na AGV kladen stále větší důraz. Zde se využívají pro inventuru, chystání objednávek a přesun zboží. Např. firma Walmart využívá tyto stroje i k nakládání a vykládání nákladních vozů.
- Automotive – Podobné jako ve výrobě. Přeprava materiálů mezi stanicemi atd.
- Zemědělství – Automatizace úkonů jako je sázení, sklizeň a údržba plodin.

Obr. 1) MasterMover AGV [32]

Kromě automaticky řízených vozidel AGV se také často skloňuje název AMR (automatic mobile robot). AMR představují pokročilejší technologii v automatizaci logistiky. Hlavní rozdíly mezi AGV a AMR jsou uvedeny v Tab. 1. Na Obr. 2a lze poté vidět AGV vezoucí bedny a sledující černou čáru na zeleném povrchu a na Obr. 2b AMR, pohybující se bez nutnosti sledování čáry.

Tab. 1) Rozdíly mezi AGV a AMR [2]

Funkce	AGV	AMR
Navigace	Sledování čáry vyžaduje jednoduchý senzor a malý výpočetní výkon.	Navigace bez předem stanovené dráhy. Zpracovává data z prostředí a počítá trasu onboard.
Vyhnutí se překážce	Ne, AGV zastaví a čeká, než je překážka odstraněna.	Ano, AMR objede překážku a najde nejlepší cestu podle mapy.
Flexibilita	Je složitější upravit předem danou trasu.	Jednoduché přemapování a stanovení nových destinací a cílů.
Cena	AGV je jednodušší, a tedy levnější než AMR.	AMR je dražší vzhledem ke komplexnějším senzorům a kontrolnímu softwaru.
Cena instalace a údržby	Vyšší cena, příprava trvá déle vzhledem k nutnosti instalace cesty (magnetické pásky, drát, reflektory atd.)	Rychlé a snadné na instalaci. Cena instalace je nižší než u AGV.
Spolehlivost	AGV se drží předem stanovené cesty a jsou tedy spolehlivější.	Navigace AMR je více citlivá na změny v prostředí a robot může ztratit pojem o jeho pozici.
Bezpečnost	B56.5-2019 v USA / ISO 3691-4:2023	ANSI/RIA R15.08-1-2020 a R.15.08-2-2023 / ISO 3691-4:2023

2a) AGV

2b) AMR

Obr. 2) Rozdíly mezi AGV a AMR [33]

3.1 Navigační metody mobilních robotů v průmyslu

Mobilní roboty v průmyslu AGV a AMR používají několik druhů senzorů a navigačních metod k uskutečnění pohybu ve výrobních halách a továrnách. Jednotlivé přístupy k navigaci se výrazně liší ve své komplexnosti, přesnosti, nákladech na instalaci a infrastrukturu atd. Volba daného typu navigace a senzorů je tedy závislá na několika faktorech jako jsou požadavky na přesnost, potřebná flexibilita, úroveň dynamiky prostředí (jak často se v prostředí vyskytují změny) atd. Hlavní způsoby navigace jsou popsány níže v Tab. 2 a obvykle se kombinuje více metod najednou. [2; 3]

Tab. 2) Vlastnosti navigačních metod [2; 3; 4]

Metoda	Senzory	Výhody	Nevýhody	Příklad použití
Sledování čáry (optické / magnetické)	Optický nebo magnetický senzor; fyzická čára (barva, páska) nebo vodič v podlaze.	Jednoduchost, nízká cena, vysoká spolehlivost a přesnost na dané trase.	Pevně stanovená trasa, která jde hůř upravit (oproti ostatním možnostem), nutná údržba, neschopnost objet překážku.	Pevné okruhy ve výrobě, prostředí s minimálními změnami layoutu.
Navigace podle značek (QR, RFID)	Kamera (QR kódy) nebo RFID čtečka; značky jako kódy a tagy jsou rozmístěny na trase.	Spolehlivost, snazší rozšíření než u čáry, získání absolutní pozice.	Nutná instalace mnoha bodů v prostoru, stejně jako u čáry pevná síť tras.	Sklady s QR kódy na podlaze, kombinace s páskou pro detekci stanic.
Laserová navigace (reflektory)	LiDAR, pasivní reflektory (odrazky v prostoru).	Vysoká přesnost a spolehlivost, žádné prvky na podlaze, poměrně snadná změna trasy.	Vyšší počáteční náklady (senzor, instalace reflektorů) nutnost mít čisté zorné pole senzoru na odrazku.	Automatizované vysokozdvižné vozíky ve skladech, velké sklady s komplexními trasami.
Přirozená navigace (SLAM) (LiDAR / kamera)	LiDAR nebo kamery, vestavěný počítač pro zpracování mapy.	Velká flexibilita, využívá prostředí tak jak je, možnost objíždět překážky a měnit trasu za běhu.	Vyšší technické nároky, potřeba složitého nastavení a údržby mapy, pohybující se objekty a low-texture environment snižují přesnost.	Moderní automatizované sklady a továrny (průmysl 4.0), dynamické prostředí, kde je potřeba, aby se robot přizpůsobil situaci.
Vision guidance	Kamera (skenuje prostředí a tvoří 3D mapu)	Podobné jako SLAM, flexibilní, nejsou potřeba další investice do prostředí, možnost objíždět překážky atd.	Při horších viditelnostních podmínkách nepředvídatelné chování, vysoká cena, složité nastavení a algoritmy.	Velké sklady, dynamické prostředí náchylné na změny atd.

3.2 Techniky a technologie pro sledování čáry

Cílem AGV sledujícího čáru je, aby se co nejvíce držel středu čáry. V závislosti na použitých senzorech a technologiích se může robot na čáře chovat různě. Nejběžnější postupy pro sledování čáry pro AGV jsou následující [5]:

Magnetické navádění

Magnetické senzory patří mezi hlavní technologie v oblasti navigace AGV v průmyslových aplikacích. Nedílnou součástí implementace je magnetická páska. Uvnitř pásky se nacházejí magnetické částice, které vytváří rovnoměrné magnetické pole po celé délce. [6]

Typickým příkladem senzoru je Roboteq MGS1600 (viz Obr. 3), který měří, jak daleko od středu čáry se robot a senzor nachází a podle toho následně rozhoduje mikrokontroler o směru jízdy. Velkou výhodou oproti optickým metodám sledování čáry je odolnost vůči změnám světelných podmínek a postupnému znečištění povrchu dráhy, které může mít negativní vliv na spolehlivost optických senzorů. Lze poměrně snadno měnit dráhu pouhou výměnou pásky, což je oproti indukční metodě také velkou výhodou. [7]

Obr. 3) Magnetický senzor Roboteq MGS1600 [7]

Indukční navádění

Indukční navádění je spolehlivým způsobem pro řízení AGV v průmyslových aplikacích. Technologie využívá vodiče uloženého v podlaze, kterým je veden střídavý proud. V důsledku toho vzniká kolem vodiče elektromagnetické pole, které je následně detekováno indukčními senzory umístěnými na AGV. [8]

Velkou výhodou indukčního navádění oproti optickým metodám je to, že není ovlivněn např. prachem, špínou, opotřebením podlahy atd. Díky tomu se používá i v náročných průmyslových prostředích jako jsou přístavy, tunely nebo výrobní haly. Příkladem senzoru pro technologii indukčního vedení je např. HG G-19370 od firmy Götting KG (viz Obr. 4) [8]

Obr. 4) Indukční senzor HG G-19370 [8]

Computer vision

V oblasti autonomního řízení vozidel jako jsou AGV existují různé způsoby, jak sledovat čáru pomocí počítačového vidění. Níže jsou popsány dva přístupy, přičemž první je příklad profesionálního průmyslového řešení a druhé nízkonákladová alternativa.

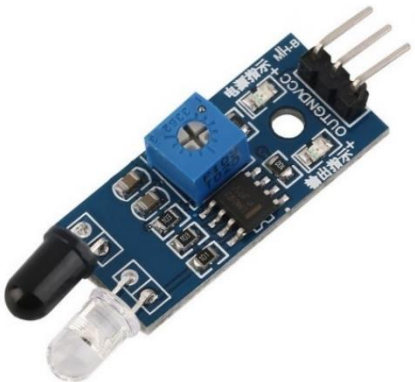
- Profesionální řešení: PGV od Pepperl+Fuchs – Systém PGV (Position Guided Vision) od firmy Pepperl+Fuchs představuje chytré řešení pro sledování čáry a absolutní polohování. Díky kombinaci kamery, osvětlení a dekódovací jednotky v jednom, dokáže tento systém spolehlivě sledovat čáru ve formě barevného pásku a zároveň číst datové kódy na podlaze. Jelikož má PGV velký zorný úhel, neohrozí ho ani poškozené či znečištěné části pásky. Podobu kamer a pásků lze vidět na Obr. 5 níže. [9]

Obr. 5) Kamery a pásky pro PGV od Pepperl+Fuchs [9]

- Alternativou sledování čáry může být i použití běžné USB kamery v kombinaci s knihovnou OpenCV pro zpracování obrazu. Takový postup zahrnuje nejdříve předzpracování obrazu, který se převede do odstínů šedi. Následně je provedena binarizace obrazu pomocí prahování, které ho převede na černobílý, kde je čára reprezentována jednou barvou a pozadí druhou. Poté se pomocí mediánového filtru a morfologických operací odstraní šum a vyčistí kontura čáry. Na takto již zpracovaném obrazu se určuje poloha čáry a počítá se úhlová a boční odchylka od jejího středu, což slouží jako vstup pro řízení robotu. [10]

IR senzor

Spíše u menších robotů případně prototypů a méně náročných aplikací. Obvykle se jedná o černou čáru a robot obsahuje program, který ji dokáže rozeznat od okolního povrchu. Jedná se o detekční systém založený na infračerveném záření, který pracuje na principu proměnlivé



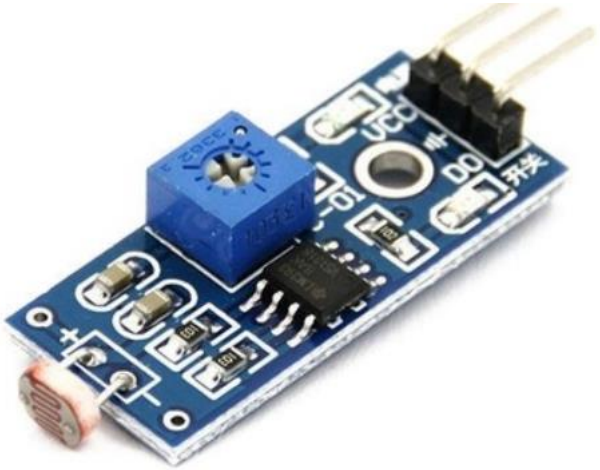
Obr. 6) IR senzor [34]

absorpce IR záření různými povrchy. Zjednodušeně řečeno v příkladu – bílý povrch IR záření odrazí zpět, ale černý ho pohltí. [5; 11]

Infračervený senzor (viz Obr. 6) emituje světlo a detekuje intenzitu jeho odrazu. Výstupní napětí na něm poté závisí na přijímané intenzitě světla a komparátorem se převádí na logické hodnoty 0 a 1. Díky tomu se robot dokáže držet na dané trase např. vypínáním a zapínáním motorů. Při zhoršených a měnících se světelných podmínkách se přesnost infračervených senzorů snižuje. [5; 12]

LDR odporový světelný senzor s bílými LED

Stejně jako klasický IR – využití spíše u menších robotů nebo prototypů. LDR – light dependent resistor pracuje na podobném principu jako IR. LED a LDR jsou umístěny vedle sebe. LED emituje světlo a množství odraženého světla opět závisí na povrchu. Na černé čáře bude více světla pohlceno a odpor na senzoru bude vysoký. Naopak od světlého povrchu se světla do senzoru odrazí více, což na senzoru vede k nižšímu odporu. V závislosti na velikosti odporu na LDR senzoru mikrokontroler rozhodne o pozici robotu a dle toho upraví směr. Jak LDR senzor vypadá lze vidět na Obr. 7. [5; 11]



Obr. 7) LDR senzor [35]

3.3 Robot Operating System (ROS2)

Robot operating system (ROS2) je open-source framework, který nabízí nespočet softwarových knihoven a nástrojů, které pomáhají vývojářům s jejich prací v robotice. Navzdory svému názvu není ROS2 skutečným operačním systémem, ale slouží jako middleware usnadňující komunikaci mezi různými součástmi robotického systému. Nejpodporovanějším operačním systémem pro ROS2 je Ubuntu. [13]

3.3.1 Klíčové komponenty ROS2 a jejich struktura

Předtím než se práce posune do věcí jako jsou Rviz nebo Gazebo (prostředí pro simulace a vizualizace), je nezbytné pochopit, jak vzájemně fungují jednotlivé fundamentální komponenty, aby zajistily efektivní fungování robotických systémů. ROS2 je navržen tak, aby umožňoval komunikaci mezi různými procesy a systémy, což je klíčové pro správnou funkci robotických aplikací. Mezi tyto komponenty patří nodes (uzly), topics (témata), services (služby) a actions (akce). [13]

Node (uzel)

Node je program, který poskytuje funkcionalitu nebo část větší funkcionality díky komunikaci s ostatními uzly. Vezměme kupříkladu robot – ten má obvykle několik nodů, které jsou připojené k síti ROS2. V tomto případě na příklad node 1 přijímá informace (je tedy subscriber) o vzdálenosti mezi robotem a přilehlými překážkami v cestě a rozsvítí LED diodu, pokud se v jeho dostatečné blízkosti jakákoliv překážka nachází. To znamená, že tento node 1 čeká na hodnoty vzdálenosti odeslané z node 2 (publisher), aby případně rozsvítil LED diodu. [14; 15]

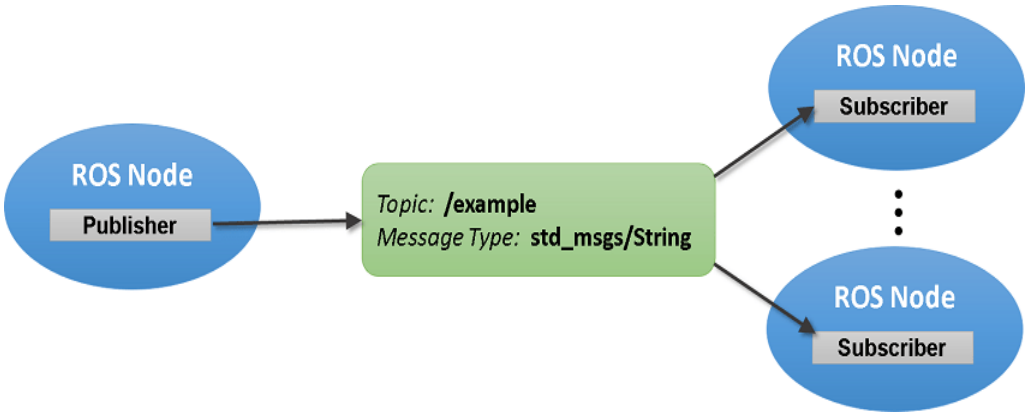
Pro dosažení vyšší funkčnosti, lepší přehlednosti a snadnější škálovatelnosti je vhodnější rozdělit systém na více menších nodů, než soustředit veškeré funkce do jednoho velkého. Tento přístup umožňuje efektivnější správu kódu, usnadňuje testování a ladění jednotlivých částí systému a zvyšuje jeho celkovou spolehlivost. [14; 15]

Centrální node se nazývá master. Jelikož se projekt v ROS2 obvykle skládá z několika různých nodů, musí zde být jeden centrální, který umožňuje všem nodům se navzájem objevovat a komunikovat. Master sbírá informace o názvech, tématech a službách poskytovaných všemi dostupnými nody. [15]

Topics a messages (témata a zprávy)

Předávání informací mezi jednotlivými uzly se provádí prostřednictvím tzv. zpráv. Zpráva obsahuje nadpis (heading), pomocí kterého mohou uzly komunikovat. Tento nadpis se nazývá topic a musí být jedinečný, aby nedošlo ke konfliktu. Například při odesílání e-mailu se přidává předmět a pod něj obsah zprávy. V tomto případě lze tedy topic v ROS2 přirovnat k předmětu e-mailu a text e-mailu k údajům v ROS2 zprávě. [14; 15]

Uzel může tedy publikovat (publish) nebo odebírat (subscribe) zprávy. Node, který zprávu publikuje se nazývá publisher a node, který zprávy odebírá je subscriber. Publisher i subscriber jsou na sobě nezávislé a jeden o druhém neví, dokud je o jejich přítomnosti neinformuje master. Pro zajištění komunikace mezi publisherem a subscriberem je nutné, aby měly zprávy (messages) stejný topic. Jakýkoliv node může publikovat i odebírat libovolný počet topiců. Schéma fungování topiců a nodů lze vidět na Obr. 8. [14; 15]



Obr. 8) Schéma komunikace mezi nody přes topic /example [16]

Služby (services) [16]

Služba je způsob komunikace mezi nody. Zde jsou zprávy mezi nimi provozovány stylem request – reply (požadavek – odpověď). Služby jsou používány v případě, že po odeslání zprávy je potřeba odpověď. Publisher – subscriber postrádá zpětnou vazbu. Např. pokud je potřeba získat potvrzení o tom, že robot pootočil ramenem do pozice, jak mu bylo zprávou přikázáno, bylo by lepší použít službu. [15]

Akce (actions)

Akce je další způsob komunikace mezi nody (schéma viz Obr. 9). Používá se v případě, kdy je na příklad nutné získávat zpětnou vazbu během toho, co robot vykonává nějakou operaci (přejíždí do cílové pozice) a během toho se získávají data o poloze. Akce jsou užitečné zejména u dlouho běžících úloh, protože umožňují aktualizace o průběhu, umožňují sledování aktuálního stavu a poskytují možnost v určitém případě úlohu zrušit. [15]

Obr. 9) Schéma akce [17]

3.3.2 Unified Robotics Description Format (URDF)

Unified Robotics Description Format (URDF) je formát založený na XML, který se používá v robotickém operačním systému (ROS2) k popisu struktury robotů. Umožňuje definovat kinematické a dynamické vlastnosti robotu, jeho vizuál a kolize. [14]

Kromě formátu URDF se běžně používají další tři open-source formáty pro popis objektů. Simulation Description Format (SDF), Universal Scene Description (USD) a MuJoCo File Format (MJCF). Na Obr. 10 je přehled často využívaných nástrojů pro simulaci robotů a ukazuje, které ze zmíněných čtyř formátů jednotlivé open-source nástroje podporují. Z dvanácti simulátorů jich jedenáct podporuje URDF, což svědčí o jeho širokém rozšíření. [18]

Simulation tool	**URDF**	**SDF**	**USD**	**MJCF**
CoppeliaSim	✔	✔	✗	✗
Drake	✔	✔	✗	✔
Gazebo	✔	✔	(✔)	(✔)
MATLAB	✔	✔	✗	✗
MuJoCo	✔	✗	✗	✔
NVidia Isaac	✔	✗	✔	✔
PyBullet	✔	✔	✗	✔
RoboDK	✗	✗	✗	✗
Robotics Toolbox for Python	✔	✗	✗	✗
RViz	✔	✗	✗	✗
Unity	✔	✗	✗	✗
Webots	(✔)	✗	✗	✗
Total 12	11	5	2	5

Obr. 10) Podpora formátů různými softwary pro robotiku [18]

V URDF je mnoho používaných prvků k popisu struktury robotu a zde jsou vypsány ty nejzákladnější [19]:

- <robot> - Základní element, který obsahuje všechny ostatní prvky.
- <link> - Popisuje jednotlivé části robotu a může dále obsahovat:
  - <visual> - Vizuální reprezentace části robotu pro RViz a Gazebo.
  - <collision> - Kolizní model pro detekci případných kolizí v simulacích.
  - <inertial> - Setrvačnost a hmotnost části robotu.
- <joint> - Spojuje dva linky a určuje druh relativního pohybu mezi nimi. Obsahuje:
  - <parent link> a <child link> - Určuje vztah mezi linky a udává jejich propojení.
  - <origin> - Počáteční poloha a orientace.
  - <axis> - Osa rotace nebo posun pro jednotlivé klouby.
  - <limit> - Omezení rozsahu pohybu (např. maximální úhel natočení kloubu).

Co představují linky a jointy na robotickém ramenu lze vidět na Obr. 11. Pokud se jedná o složitější roboty, existuje jednodušší cesta tvorby URDF souboru modelu než ruční psaní kódu. Existuje např. možnost získat URDF model přímo ze softwaru SolidWorks za pomoci doplňku SolidWorks URDF Exporter. Tento doplněk automaticky generuje URDF soubory, což zamezuje chybám vzniklým při manuálním vytváření popisu robotů. Finální model zachová hierarchii sestav, a to včetně definice spojů a os, což umožní snadnou integraci do ROS2. [14]

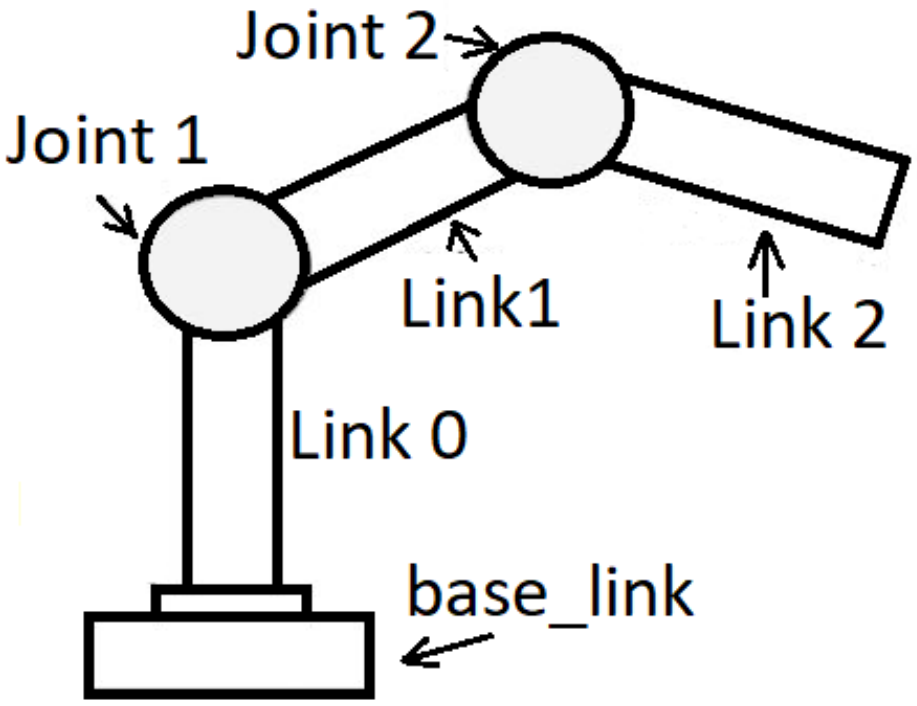
3.3.3 Micro-ROS

Micro-ROS framework rozšiřuje možnosti standardního ROS2 i na mikrokontrolery s omezenými výpočetními možnostmi. Přináší klíčové prvky ROS2 jako jsou nody, topicy, akce, služby, subscribery, publishery atd. na embedded systémy. Tato integrace umožňuje vývojářům pracovat s embedded softwarem skrze známé ROS2 nástroje a API. [20]

Micro-ROS je kompatibilní s širokým množstvím mikrokontrolerů jako je např. Raspberry Pi Pico využívané v praktické části práce. Podporuje open-source real-time operační systémy jako je FreeRTOS, Zephyr a NuttX. Projekt je ve fázi open-source vývoje od roku 2018, kdy začal jako součást EU podporované OFERA (Open Framework for Embedded Robot Applications) činnosti. [20]

Zatímco micro-ROS se snaží o implementaci klíčových funkcionalit ROS2, je důležité zohlednit při tvorbě aplikací několik zásadních rozdílů, které jsou důsledkem zejména omezených hardwarových možností mikrokontrolerů. Hlavní 3 rozdíly jsou [20]:

- Pouze zprávy s fixně danou velikostí – Aby se předešlo dynamické alokaci paměti, musí mít všechna pole (arrays) pevně stanovenou délku nebo alespoň horní limit.
- Žádný launch systém – Micro-ROS neumožňuje uživatelům využívat launch systému tak, jako je známo z klasického ROS2. Naopak nechává spouštění jednotlivých nodů čistě na vývojáři.
- Omezenější výběr pro psaní skriptů – Oficiálně podporuje micro-ROS pouze programování v jazyce C, případně C++.



Obr. 11) Linky a jointy na robotickém ramenu [21]

3.4 Gazebo Sim a Rviz

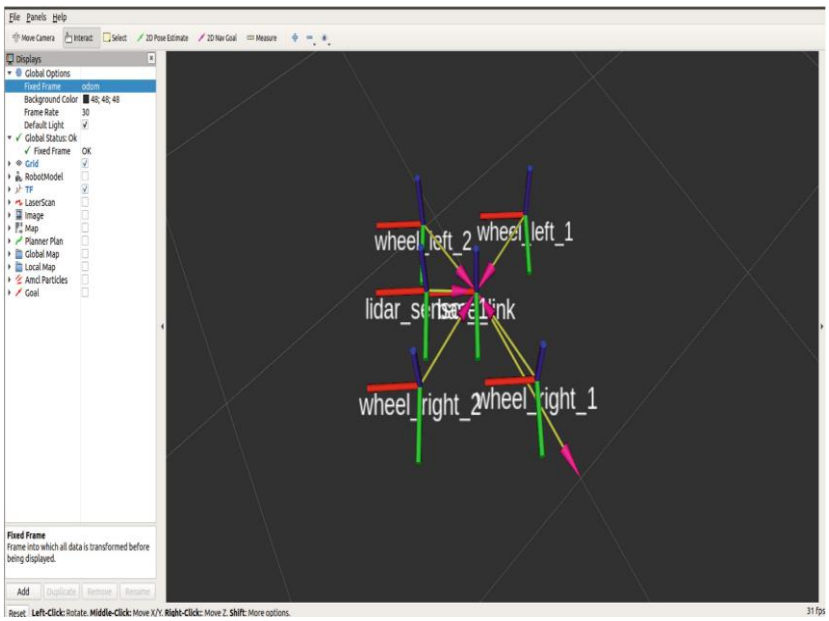
Gazebo je open-source 3D robotický simulátor, který je integrován do robotického operačního systému ROS2 pomocí softwarových balíčků jako např. gazebo_ros_pkgs, díky kterým lze využívat funkce ROS2 jako jsou zprávy, topicy atd. Poskytuje robustní prostředí pro přesné a účinné simulace robotů v komplexních indoor i outdoor situacích. Začleněním fyzikálních enginů, kvalitní grafiky a simulace různých senzorů umožňuje Gazebo vývojářům navrhovat roboty, testovat algoritmy a trénovat AI systémy v bezpečném a kontrolovaném prostředí. Díky tomu nevyžaduje vývoj robotických aplikací fyzický hardware a tím zrychluje pracovní proces a snižuje výslednou cenu produktu. [22; 23]

Hlavní výhody využívání Gazebo Sim [24]:

- Významné snížení času potřebného pro debugging.
- Možnost tvorby specifického prostředí (mapy), kam bude robot vyslán.
- Využití tohoto prostředí pro následné testy robotu.
- Simulace a vizualizace vývoje prostředí, kde bude robot zkoumat, před skutečným nasazením robotu do reálného světa.

RViz (zkratka pro ROS visualization) je open-source vizualizační software. Díky němu získává uživatel přehledné GUI pro vizualizaci přesné pozice a stavu robotu a zároveň i jeho okolí. Umožňuje nejen sledovat data ze senzorů jako jsou kamery, laser atd., ale např. také zadat robotu souřadnice na mapě, na které se má přemístit. [25]

Na Obr. 12 lze vidět uživatelské rozhraní. Panel vlevo se nazývá Display panel a nachází se na něm seznam pluginů. Díky nim lze sledovat již zmiňovaná data ze senzorů a stavy robotu. Co se týče ROS2 balíčků pro RViz, tak za zmínku určitě stojí balíček Transforms (tf). Ten se využívá pro tvorbu souřadnicových systémů, které je možno sledovat a následně s nimi pracovat. Díky němu lze lépe pochopit jak mezi sebou různé články (linky) jednotlivých částí robotu komunikují. To můžeme vidět na Obr. 12, kde jsou zobrazeny jednotlivé souřadnicové systémy čtyř kol (wheel_left:1, wheel_left_2, wheel_right_1, wheel_right_2), lidar senzoru (lidar_sensor) a těla mobilního robotu. [25]



Obr. 12) Prostředí RViz [25]

3.5 Docker

Docker je open-source software (schéma viz Obr. 13), který zabalí aplikaci spolu s potřebnými knihovnami a ostatními závislostmi, které jsou potřeba, do lightweight kontejneru. Ten umožňuje vývojářům spouštět ten stejný obraz (image) beze změn na novém stroji, cloudu atd. Protože kontejnery sdílí hostitelský kernel místo emulování hardwaru, spouští se velmi rychle a na jeden server nebo virtuální stroj jich lze umístit stovky, což je výrazně více oproti klasickým virtual machines. [26; 27]

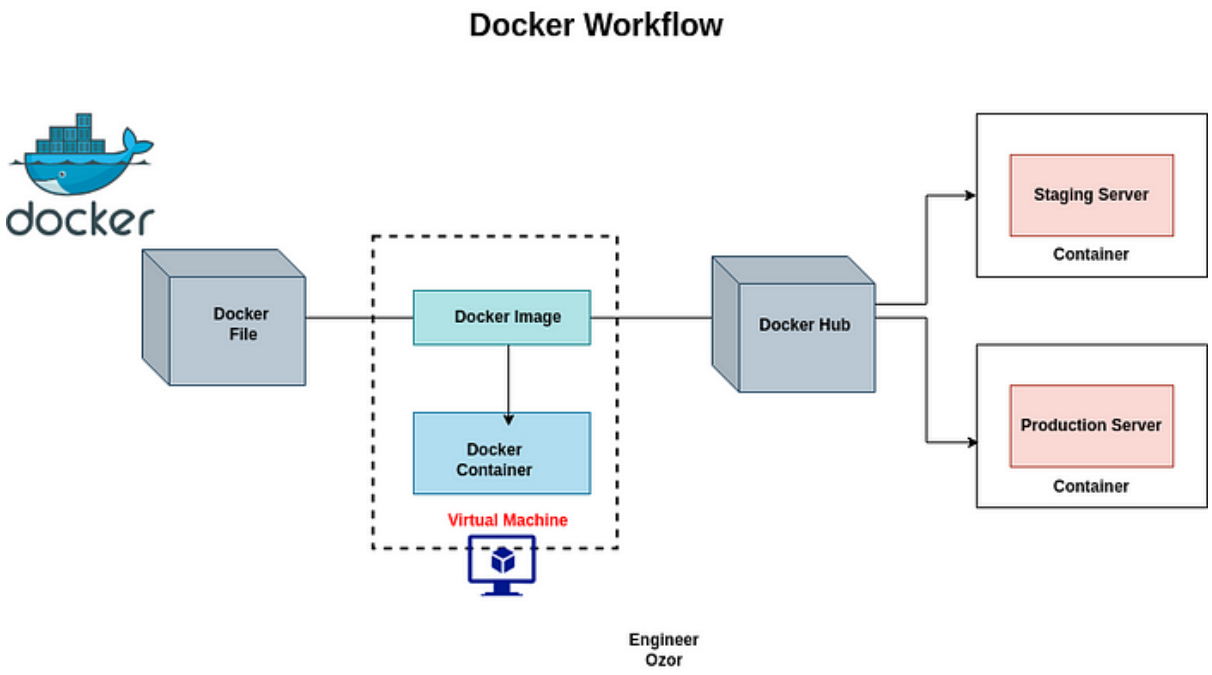
Izolaci kontejnerů v Dockeru zajišťují dvě hlavní linuxové technologie [26]

- Namespaces – Oddělují prostředí jednotlivých kontejnerů – každý kontejner tak vidí jen své vlastní procesy, síť a souborový systém.
- Cgroups – Určují, jaké množství systémových prostředků (CPU, paměť...) má každý z kontejnerů k dispozici.

Souborový systém funguje na principu vrstvení pomocí metody copy-on-write – využívá technologie jako AuFS, Btrfs nebo Device Mapper. To znamená, že Docker ukládá pouze rozdíly oproti základnímu obrazu (podobně jako git commit). [26]

Díky svým vlastnostem docker skvěle zapadá do moderních DevOps postupů. Vývojář může u sebe snadno spustit plnohodnotné prostředí – například LAMP stack (Linux, Apache, MySQL a PHP) pro webové aplikace. Případně může pracovat s mikroslužbami, automaticky testovat kód a poté stejný image přesunout i do stagingu a produkce. [26; 27]

V akademické sféře usnadňuje Docker popis a také sdílení celého analytického prostředí. Pomocí jednoduchého souboru Dockerfile lze tedy přesně definovat všechno – od základní linuxové distribuce až po specializované knihovny a následně uložit finální image do veřejného repozitáře. Ostatní výzkumnící můžou i po delší době snadno reprodukovat výsledky bez klasických problémů se systémovými závislostmi atd. [27]



Obr. 13) Schéma Docker workflow [28]

4 VARIANTY ŘEŠENÍ

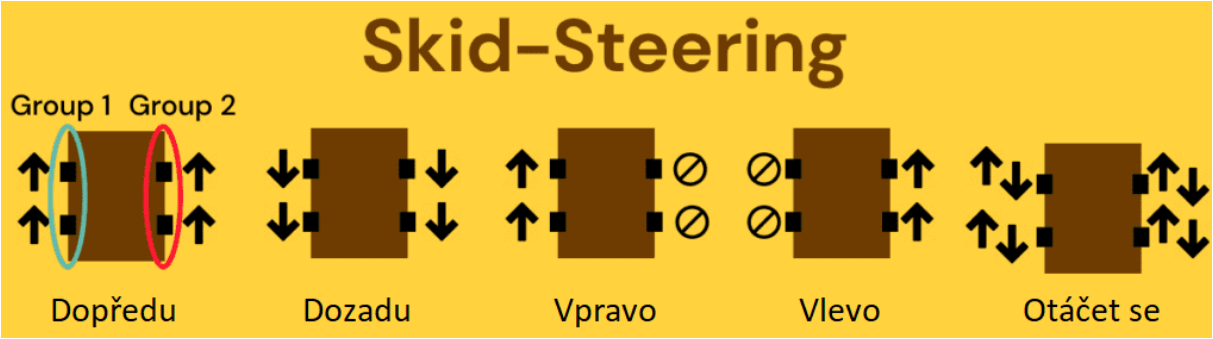
Hlavním cílem této práce není návrh nového robotického systému z pohledu mechanické konstrukce, ale spíše zaměření se na implementaci systému ROS2 a micro-ROS na vývojové desky jako je Raspberry Pi 5 a Raspberry Pi Pico. Kvůli tomu bylo potřeba zvolit cestu, která je schůdná na konstrukci i simulaci v Gazebo Sim a samozřejmě je dostatečně funkční pro implementaci skriptů pro sledování čáry.

V potaz byly brány dvě varianty – diferenciální pohon o dvou kolech (differential wheeled drive) a skid-steer pohon – 4 kola. Obě varianty jsou dobře známé ve vzdělávacích oblastech a výzkumných robotických platformách, a to hlavně kvůli své jednoduché, avšak velmi účinné konstrukci. Hlavní rozdíly jsou vidět v Tab. 3 níže a na Obr. 14 a Obr. 15.

Tab. 3) Vlastnosti verze s diferenciálním pohonem a skid-steer pohonem [29]

Typ pohonu	**Popis**	**Výhody**	**Nevýhody**
Diferenciální pohon	Jedná se o 2 kola poháněná nezávisle na sobě a jedno pasivní pojezdové kolo.	Jednoduchost konstrukce, snadný kinematický model, možnost otáčení na místě, velmi dobrá podpora v prostředí Gazebo a ROS2.	Citlivost na nerovnosti povrchu. Pohyb může být destabilizovaný kvůli pasivnímu kolu. Častý úhyb od přímého směru.
Skid-steer pohon	Systém 4 kol, jež jsou na jedné a druhé straně poháněna společně, zatáčení probíhá rozdílnou rychlostí stran.	Jednoduchá konstrukce. Použitelný v náročnějším terénu oproti první možnosti.	Výrazný prokluz kol při zatáčení. Složitější model pro simulaci pohybu a tření. Náročnější na přesnou simulaci v Gazebo Sim.

Obr. 14) Differential steering princip [36]



Obr. 15) Skid-steering princip [36]

Skid-steer roboty se běžně používají a jedná se o robustní řešení, avšak trpí na výrazný prokluz kol během pohybu. Pro využití této metody v Gazebo Sim by bylo nutné zahrnout pokročilejších modelů tření a kinematiky, což neodpovídá zaměření této práce.

Po důkladném zvážení a porovnání obou možných variant byl zvolen diferenciální pohon se dvěma koly. Nabízí dobrou manévrovatelnost pro princip sledování čáry, umožňuje rychlý vývoj v simulaci, jelikož Gazebo Sim nabízí differential drive plugin pro řízení robotu této konstrukce a je obecně dobře podporován v systému ROS2. Díky tomu se práce může soustředit na hlavní cíle.

4.1 Způsoby sledování čáry

Velká část této práce se věnuje simulaci v Gazebo Sim, a to zejména problematice sledování čáry. Vzhledem k tomu, že Gazebo Sim standardně v knihovně senzorů IR senzory nenabízí, bylo nutné vyhledat jiné řešení, jak k tomuto problému přistupovat. Toto řešení musí umožnit efektivní simulaci těchto senzorů a zajistit realistické výsledky. Existuje několik možných přístupů a každý má své specifické výhody a nevýhody. Níže jsou rozebrány tři zvažované metody:

1. RGB kamera a počítačové vidění – Přístup zahrnuje použití RGB kamery k získávání obrazových dat. Ta jsou následně zpracována pomocí algoritmů počítačového vidění za účelem detekce čáry.

   Hlavní výhody tohoto řešení jsou:

   - Realističnost – Simulace s RGB kamerou by poskytla vizuální data, která by se velmi podobala tomu, co by viděl skutečný robot. To by umožnilo dobré testování počítačového vidění v podmínkách blízkým reálnému užití.
   - Flexibilita – Kamera umožňuje detekci nejen čáry, ale i překážek v cestě, případně jiných objektů. To může být využito pro složitější úlohy pro robot jako je např. rozpoznávání objektů, čtení QR kódů atd.

   Hlavní nevýhody řešení:

   - Výpočetní náročnost – Zpracování obrazu vyžaduje silnější výpočetní zdroje než ostatní jednodušší senzory, což může simulaci zpomalit, a hlavně zvyšuje nároky na výsledný hardware.
   - Složitost implementace – Vývoj a ladění algoritmů pro počítačové vidění může být velmi časově náročné a vyžaduje větší znalosti zpracování obrazu.

2. Použití kontaktních senzorů – Toto řešení spočívá ve využití kontaktních senzorů, které detekují kolizi s předem určeným modelem (funguje podobně jako mechanický switch).

   Hlavní výhody řešení:

   - Jednoduchost – Implementovat kontaktní senzory do simulace je relativně snadné a nevyžaduje žádné složité nastavení ani konfiguraci, pokud se jedná o jednoduchý případ kolize – např. náraz robotu do zdi.
   - Nízké nároky na výpočetní výkon – Detekce kontaktu není náročná a oproti kameře nevyžaduje takový výpočetní výkon, což by mělo umožnit plynulejší simulaci i na slabších strojích.
   - Není třeba dalších převodů výstupu kontaktního senzoru – Výstup senzoru bude vždy buď true nebo false, stejně jako klasický IR senzor.

   Hlavní nevýhody řešení jsou:

   - Signál je při nárazu vyslán pouze jednou, a to v okamžiku nárazu. Pro kontinuální sledování čáry nevhodné.
   - Definice kontaktu není univerzální a je potřeba ji definovat zvlášť pro každý model světa, který má být brán v úvahu pro detekci kolize.

3. LiDAR (ray) senzor měřící vzdálenost – Měření vzdálenosti mezi senzorem a zemí, případně vymodelovanou čarou vloženou do SDF světa Gazebo Sim.

   Hlavní výhody řešení jsou:

   - Přesnost – Tento senzor umožňuje velmi přesné měření vzdálenosti.
   - Nízký potřebný výpočetní výkon – Opět oproti kameře vyžaduje mnohem nižší výpočetní výkon.
   - Jednoduchá implementace – LiDAR senzor je v Gazebo Sim velmi využívaný a použití je dobře zmapované v oficiální dokumentaci.
   - Široké možnosti nastavení – Lze nastavit úhel detekce LiDARu (od 0° do 360˚). Lze také zvolit počet paprsků, vzdálenost, na kterou bude LiDAR měřit atd.

   Hlavní nevýhody jsou:

   - V případě použití pro detekci čáry je nutné převést vzdálenosti na hodnoty true/false.
   - Čára musí být reprezentována jako 3D model.

4.2 Princip zvoleného řešení v Gazebo Sim

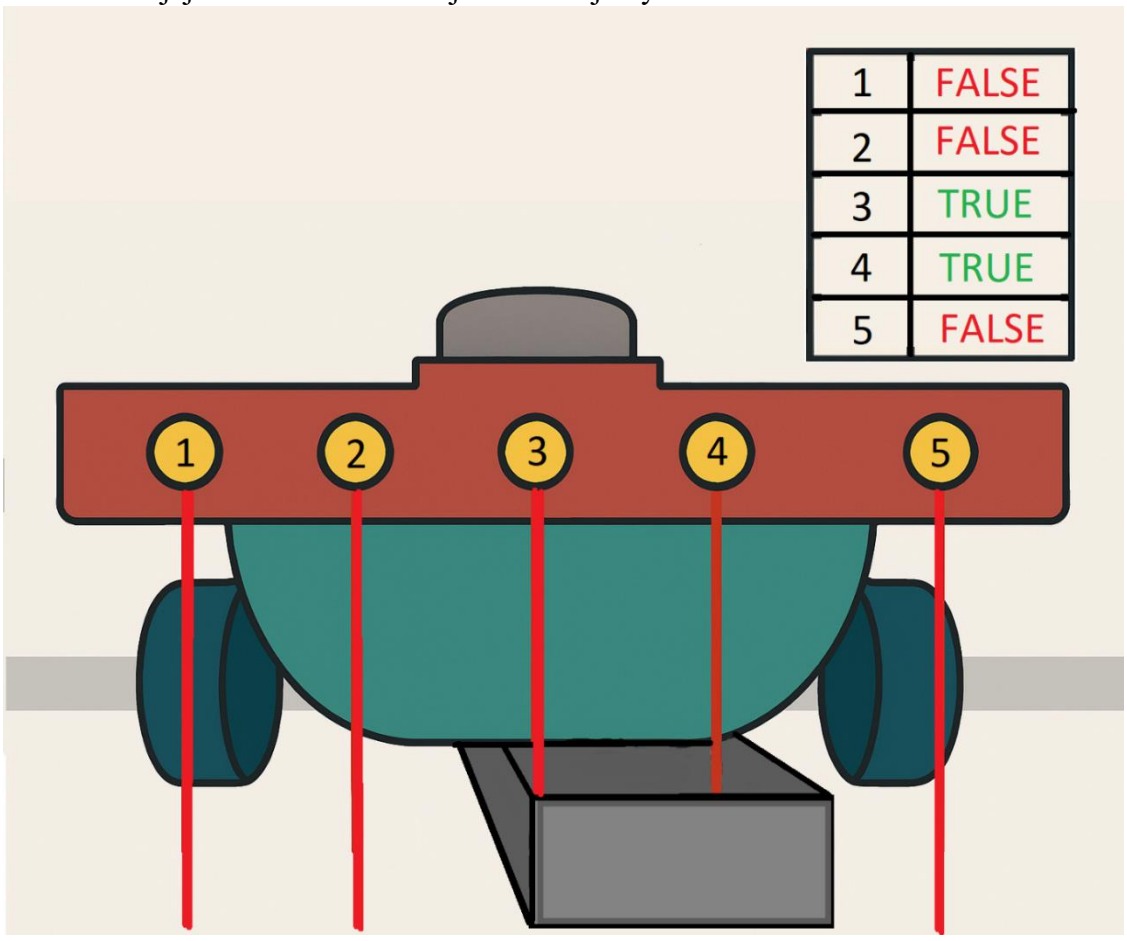
Po zvážení zmíněných postupů a porovnání jejich specifických výhod a nevýhod, bylo rozhodnuto pro detekci čáry v Gazebo Sim využít LiDAR senzorů. Aby mohlo být dosaženo efektivní simulace detekce čáry, byla čára vložena do virtuálního prostředí jako 3D model bez kolizních vlastností. Simulace funguje následujícím způsobem:

- Měření vzdálenosti od povrchu: LiDAR senzory jsou umístěny na přední straně robotu a kontinuálně měří vzdálenost mezi robotem a zemí.
- Rozdíl ve výšce čáry: Pokud je senzor nad čarou, naměřená vzdálenost bude menší než v případě, kdy pod ním žádná čára není.

- Binární výstup: Na základě těchto vzdáleností lze vytvořit jednoduchou logiku, která převádí naměřené hodnoty na binární výstup. Převedení na binární hodnoty ale probíhá až v samotném řídícím nodu (python skriptu) a nelze ho nastavit v konfiguraci senzoru v Gazebo Sim.
- Model čáry bez kolizí – Je klíčové, aby čára byla vytvořena bez kolizních vlastností. V případě, že by kolize měla, by totiž mohlo docházet k fyzické interakci mezi ní a robotem. Robot by na ni mohl najíždět, což by samozřejmě znamenalo změnu vzdáleností a tím pádem špatný výstup senzorů.

Na Obr. 16 lze vidět princip řešení. LiDAR senzory, které v tomto návrhu nahrazují IR senzory, jsou uspořádány v řadě na čelní straně robotu a jsou znázorněny jako žlutá kolečka na červeném těle robotu. Pro představu fungují v simulaci tyto senzory na stejném principu jako ultrazvuková čidla měřící vzdálenost (ultrazvukové senzory stejně jako IR senzory Gazebo Sim bohužel nenabízí). Senzory jsou očíslovány od 1 do 5. Z každého senzoru vychází červená čára, reprezentující měřenou vzdálenost od senzoru k nejbližší překážce (v tom případě k povrchu země nebo vyvýšené čáře). Čára pod robotem je znázorněna jako tmavě šedý kvádr a senzory nad ní detekují menší vzdálenost než senzory mimo ni.

Z tabulky v pravém horním rohu Obr. 16 je zřejmé, jak se tyto rozdíly vzdáleností projeví v binárním výstupu senzorů. Je-li naměřená hodnota menší než určitá prahová hodnota, je výstup ze senzoru TRUE (čára je detekována), v opačném případě je výstup FALSE. Konkrétně zde jsou senzory 3 a 4 TRUE a senzory 1, 2 a 5 FALSE. S těmito daty dále pracuje řídící algoritmus a na jejich základě rozhoduje o směru jízdy.



Obr. 16) Princip náhrady IR senzorů v Gazebo Sim

5 ROS2

Praktická část diplomové práce je již zaměřena na implementaci robotu pro sledování čáry pomocí robotického operačního systému ROS 2. V čase psaní této diplomové práce byla možnost zvolit si ze dvou dlouhodobě podporovaných verzí ROS2– Humble Hawksbill a Jazzy Jalisco (dále jen Jazzy). Vzhledem k možné návaznosti na tuto práci, byla zvolena nejnovější verze Jazzy. Ta byla vydána 23. května 2024 a dle oficiální dokumentace by její podpora měla trvat až do května 2029. Výběr této verze umožní snadnou kompatibilitu a rozšiřitelnost v budoucnu. Nejpoužívanějším a nejvhodnějším operačním systémem pro ROS2 je Ubuntu. V současnosti je jedinou oficiálně podporovanou verzí Ubuntu pro ROS2 Jazzy verze Ubuntu 24.04 (Noble), a proto byla pro práci použita. Dle oficiální dokumentace lze využít i Windows 10 (Visual Studio 2019). [14]

5.1 Instalace ROS2 Jazzy Jalisco

Níže jsou rozepsány příkazy pro instalaci ROS2 Jazzy pro operační systém Ubuntu 24.04 (Noble). [14]

1. Nastavení lokalizace:

```
locale  # check for UTF-8
sudo apt update && sudo apt install locales
sudo locale-gen en_US en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LANG=en_US.UTF-8
locale  # verify settings
```

2. Přidání potřebných repositářů:

```
sudo apt install software-properties-common
sudo add-apt-repository universe
sudo apt update && sudo apt install curl -y
sudo curl -sSL
https://raw.githubusercontent.com/ros/rosdistro/master/ros.key
-o /usr/share/keyrings/ros-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/ros-archive-keyring.gpg]
http://packages.ros.org/ros2/ubuntu $(. /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee
/etc/apt/sources.list.d/ros2.list > /dev/null
```

3. Instalace ROS2:

```
sudo apt update
sudo apt upgrade
sudo apt install ros-jazzy-desktop
```

Proto, abychom mohli v terminálu volat ROS2 příkazy, je nutné v každém nově otevřeném okně nastavit prostředí následujícím způsobem:

```
source /opt/ros/jazzy/setup.bash
```

Tento krok se dá obejít, pokud se příkaz přidá do startup scriptu následovně:

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

Nyní už nebude potřeba, každé nové okno sourcovat.

5.2 Nové pracovní prostředí a package

Pro přehlednost a správnou strukturu celého projektu, je zásadní na úplném začátku vytvořit nový pracovní prostor (workspace), který bude obsahovat všechny balíčky potřebné pro realizaci robotu pro sledování čáry.

Balíčky jsou organizační jednotky pro kód ROS2 (viz Obr. 17). Pokud je potřeba projekt s někým sdílet nebo ho např. instalovat na jiné zařízení, je potřeba ho uspořádat do balíčků. Balíček se následně sestaví pomocí nástroje colcon. Balíčky je možno vytvořit buď pomocí Pythonu či CMake. [14; 30]

Jako první je tedy nutné vytvořit workspace. Název tohoto pracovního prostoru bude *dp2025_ws*. Uvnitř této složky se bude nacházet složka *src*, ve které budou všechny potřebné balíčky. Pomocí terminálu byla vytvořena složka a následné přesunutí se do ní následujícím způsobem:

```
mkdir -p ~/dp2025_ws/src
cd ~/dp2025_ws/src
```

Poté byl uvnitř složky *src* vytvořen balíček s názvem *dp_package* následovně:

```
ros2 pkg create --build-type ament_python --license Apache-2.0
dp_package
```

V balíčku *dp_package* lze ihned po vytvoření najít soubory *package.xml, setup.cfg, setup.py* a složky *resource, test* a *dp_package.*

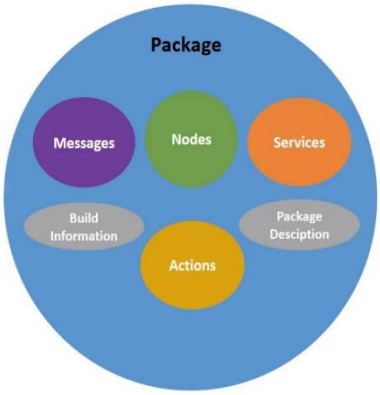
Dobrým zvykem je v souborech *package.xml* vyplnit hodnoty description, maintainer a license. To stejné je dobré vyplnit i v souboru *setup.py.*

Aby bylo možné balíčky používat, je nutné je v pracovním prostoru sestavit, proto je v kořenové složce projektu *dp2025_ws* nutno zavolat příkaz.

```
colcon build --symlink-install
```

Po otevření nového okna terminálu je třeba vždy sourcovat workspace, jinak nebudou nalezeny instalované balíčky (příkaz se provádí ve složce *dp2025_ws*):

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



Obr. 17) Schéma ROS2 balíčku [14]

Princip souboru setup.py

Soubor *setup.py* je skript, který říká Pythonu a nástroji colcon, jak má nainstalovat daný balíček. Obsahuje informace jako název balíčku, verze, co se má kam nainstalovat, závislosti atd. V ROS2 systému je tento soubor využíván při použití příkazu *colcon build* k tomu, aby se dané soubory řádně zkopírovali a zaregistrovali a bylo je možné volat pomocí *ros2 launch, ros2 run*, nebo aby je našel např. RViz.

Při *colcon build* příkazu projde colcon celý workspace, najde všechny balíčky a nainstaluje je právě pomocí skriptu *setup.py* do složky *dp2025_ws*/*install/share/název_balíčku*. Příkazy jako *ros2 launch*, poté potřebné soubory hledají právě v této složce (v tomto případě konkrétně v *dp2025_ws/install/share/dp_package/launch*).

Proto, aby byly všechny potřebné složky obsahující klíčové soubory (např. *urdf, meshes, launch*, atd) správně přemístěny do *install/share/...*, je potřeba definovat je v *setup.py*. Zde je příklad, jak zahrnout do instalace složku *launch* (do proměnné *data_files)*, viz příloha 2 soubor *setup.py* umístěný v *dp2025_ws/src/dp_package*.

```
os.path.join('share/' + package_name + '/launch',
glob('launch/*.launch.py'))
```

V tomto případě příkaz říká colconu, aby přesunul všechny soubory ze složky *launch* s příponou *.launch.py* do složky *install/share/dp_package/launch.* Pokud tedy např. projekt využívá modelu robotu, jehož tělo je zčásti definováno STL soubory, které se nachází ve složce *meshes*, musí být složka *meshes* implementována do *setup.py* stejným způsobem.

6 SIMULAČNÍ MODEL ROBOTU V ROS2 A PROSTŘEDÍ SIMULACE

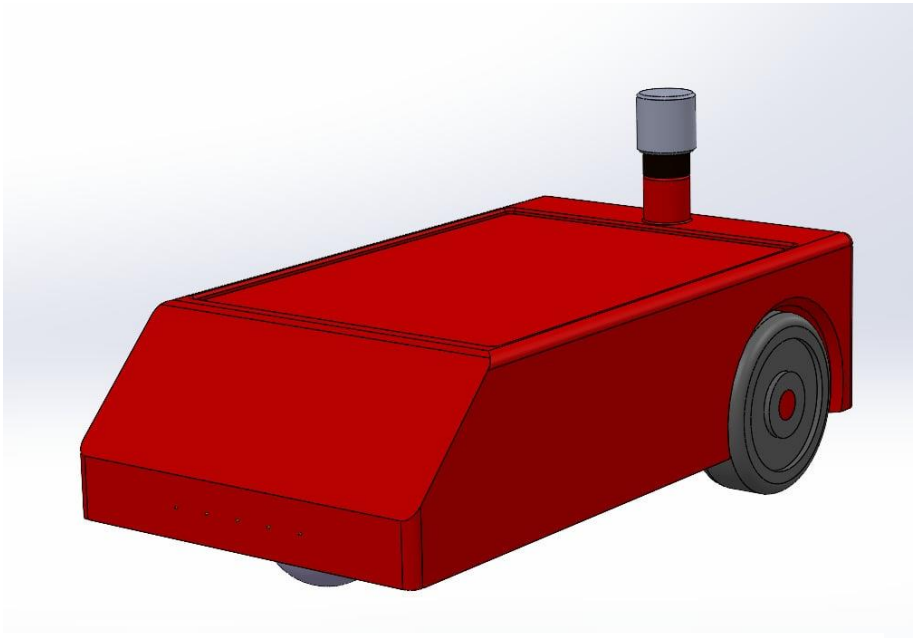
3D model robotu je nezbytný pro simulaci v Gazebo Sim. Zjednodušený model pro sledování čáry byl vytvořen v programu SolidWorks s využitím pluginu SolidWorks to URDF exporter. Tento plugin odstraňuje nutnost ručního psaní XML kódu pro definici geometrie 3D modelu, včetně definice všech nezbytných částí jako jsou linky, jointy a souřadnicové systémy. Díky tomu se eliminuje riziko chybovosti spojené s ručním psaním XML (URDF), což výrazně urychluje celý proces a zároveň dává možnost vytvořit složitější a realističtější modely pro simulaci.

V době psaní této diplomové práce je plugin SolidWorks to URDF exporter dostupný pro verze SolidWorks 2018 až 2021. Pro tento projekt byla zvolena verze SolidWorks 2020, jelikož se jedná o nejnovější verzi, kterou VUT poskytuje studentům v rámci licenčního programu. Nicméně je nutno zdůraznit, že vzhledem ke stáří pluginu jsou vygenerované launch soubory kompatibilní pouze se staršími verzemi ROS2. V rámci této práce budou tedy využity pouze soubory URDF a také STL, které se nachází v exportované složce *meshes*.

6.1 Postup exportu modelu ze SolidWorks

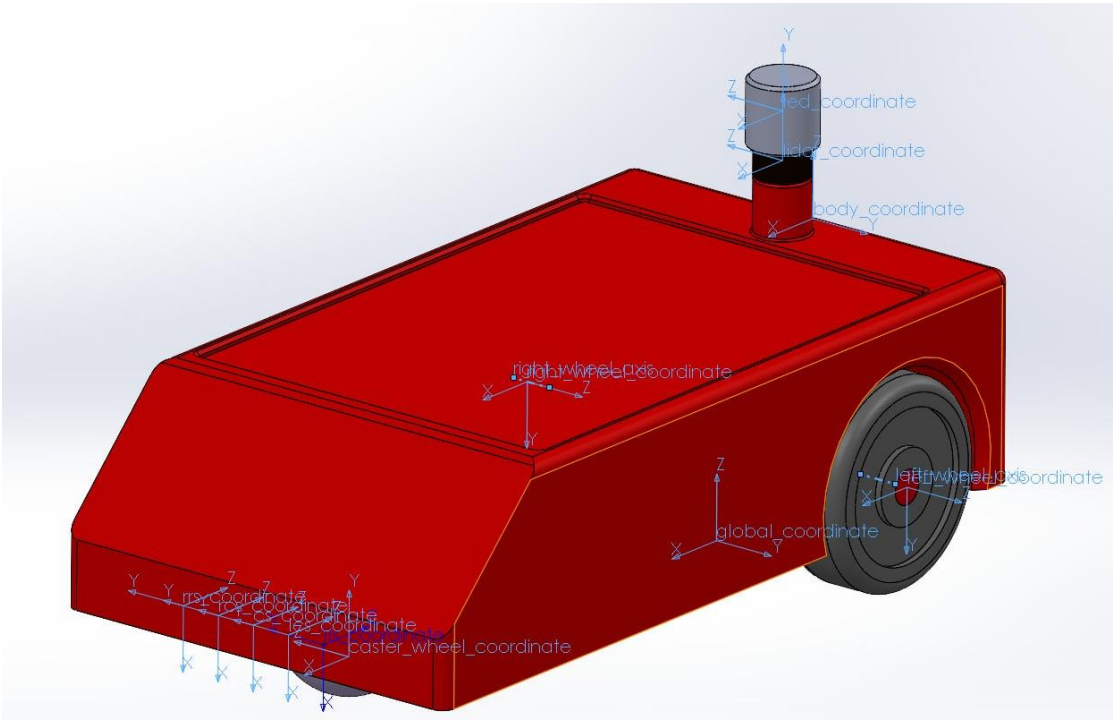
Pro úspěšný export sestavy do URDF je nutné dodržet následující postup:

1. Vytvoření jednotlivých částí robotu – Každá součást robotu musí být vytvořena jako samostatný *SLDPRT* soubor (standardní formát pro 3D model SolidWorks). Součásti lze přiřadit barvy, které se do URDF souboru také zapíšou.
2. Sestavení modelu – Jednotlivé díly je nutno složit do sestavy. Vztahy mezi nimi musí být přesně definovány, stejně jako v případě klasického vytváření sestavy. Využívá se funkce Mate, která umožňuje nastavení vazeb mezi komponentami a zároveň umožňuje případný pohyb mezi nimi. Sestavu lze vidět na Obr. 18.



Obr. 18) Sestava robotu

3. Definice všech os otáčení a případně jiného pohybu – V případě modelu použitého v této práci je definice os jednoduchá. Nachází se zde pouze dvě osy otáčení, které jsou umístěny na pravém a levém kole. Tyto osy budou umožňovat otáčení, a tedy i pohyb robotu. Pokud by se však jednalo o složitější model, bylo by nutné označit také např. osy translace (např. pro otevírání gripperu). Správné definování těchto os je zásadní pro správnou funkci a věrnost simulace robotu v Gazebo Sim.
4. Vytvoření souřadnicových systémů jako základu pro jednotlivé komponenty – Každá součást v sestavě robotu musí mít definovaný souřadnicový systém. Ten slouží jako referenční bod pro jeho umístění a pohyb. V pluginu SolidWorks to URDF exporter existuje možnost automatického vytvoření souřadnicového systému, nicméně funkce je velmi nespolehlivá. Tato nespolehlivost se neprojeví až do doby, kdy je třeba model exportovat. V tu chvíli často docházelo k nespecifikovaným chybám a po chvíli bylo zřejmé, že chybu způsobují souřadnicové systémy, které byly automaticky vygenerovány. Z tohoto důvodu je lepší je definovat v SolidWorks sestavě manuálně a těmto chybám se vyhnout. Výsledná sestava se zadefinovanými souřadnicovými systémy a osami je vidět na Obr. 19.
5. Export URDF souboru – Pro export souboru do URDF je v prostředí SolidWorks nutné přejít do záložky *TOOLS → Export as URDF*. Vlevo se zobrazí panel prostředí URDF exporter, viz Obr. 20, kde je nutné pro každou součást modelu provést několik nastavení. Nyní bude uvedeno, co jednotlivé možnosti představují a příklad volby pro tělo robotu.
    - Link name – Název linku, na který se bude URDF soubor odkazovat. V případě těla je název body_link.
    - Joint name – Obdobně jako link name, URDF odkaz na daný joint (body_joint).
    - Reference coordinate system – Z dropdown menu je nutno zvolit příslušný souřadnicový systém.



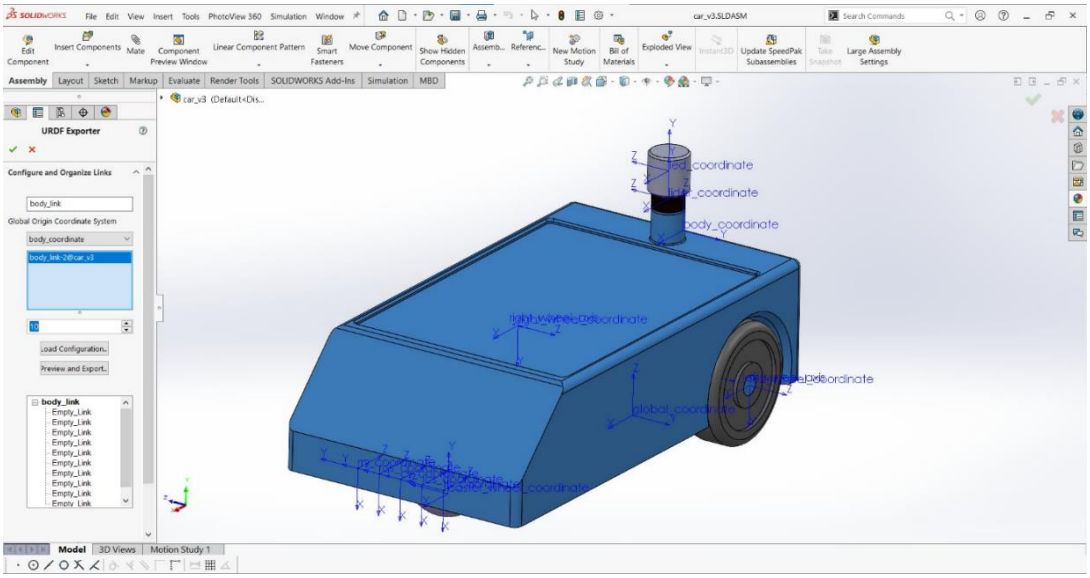
Obr. 19) Sestava s definovanými souřadnými systémy a osami

- Referance axis – Zde je potřeba vybrat osu pro danou součást. V případě body_link se žádná osa nenachází (volba None), pokud by se ale jednalo o např. levé a pravé kolo, je potřeba zvolit správnou osu, okolo které se mají kola otáčet.
- Joint type – Typ jointu, který může být fixed (pevný joint, žadný pohyb), prismatic (translační pohyb), revolute (rotační pohyb, kde je možné limitovat úhel natočení) nebo continuous (rotační bez omezení – používá se např. právě pro kola).
- Number of child links – Počet částí robotu, které jsou podřazené dané části robotu (kolik částí je na danou část přímo napojeno). V tomto případě jsou všechny části robotu přímo spojeny s jeho tělem. Number of child links bude tedy v případě body_link 10 a zbytek součástí bude mít number of child link vždy 0.
- Jako poslední věc je nutno manuálně zvolit část robotu, pro kterou nastavení provádíme. Tzn. v případě těla robotu stačí kliknout na tělo a daná část změní barvu z červené na modrou, což indikuje danou volbu (viz Obr. 20).

Po nastavení všech parametrů je zakliknuto tlačítko *Preview and Export*. V této fázi lze vidět tabulku umožňující naposled zkontrolovat nebo přenastavit vlastnosti všech jointů a linků, včetně jejich dynamických vlastností, textur atd. V této fázi lze také nastavit limity pro pohyb v případě prismatic nebo revolute jointů.

Ve finále už je zvoleno tlačítko *Export URDF and Meshes*, čímž se vygenerují všechny potřebné soubory – URDF soubor robotu a složka *meshes*, která obsahuje jednotlivé součásti robotu ve formátu STL. Všechny zmiňované soubory je poté nutné nakopírovat do ROS2 balíčku a složky zahrnout do souboru *setup.py*, aby byly správně nalezeny při sestavování nástrojem colcon. Stromovou strukturu modelu lze vidět na Obr. 22. Finální soubor je dobré pojmenovat stejně jako balíček, ve kterém má být použit, jelikož se správně zapíšou cesty k daným STL souborům do URDF (není to nutné, ale usnadní to práci).

Po exportu modelu byl nahrazen tvar kolize pojezdového kola z původní geometrie na jednoduchou kouli definovanou přímo v URDF souboru. Během simulace totiž docházelo k neplynulému pohybu a zadrhávání robotu právě o jeho pojezdové kolo, což negativně



Obr. 20) Proces nastavení URDF exportu

ovlivňovalo plynulost a správnost celé simulace. Během zadrhávání totiž robot v přední části mírně poskakoval, a v důsledku toho se měnila vzdálenost senzorů od země, což způsobovalo zkreslené hodnoty měření vzdálenosti. Při pokusu nastavit parametry tření (mu1 a mu2) u původního modelu kola, který byl exportován ze SolidWorks ve formátu STL, však tyto hodnoty simulace ignorovala. Problém byl pravděpodobně způsoben omezenou kompatibilitou mezi vlastnostmi URDF a geometriemi definovanými jako mesh.

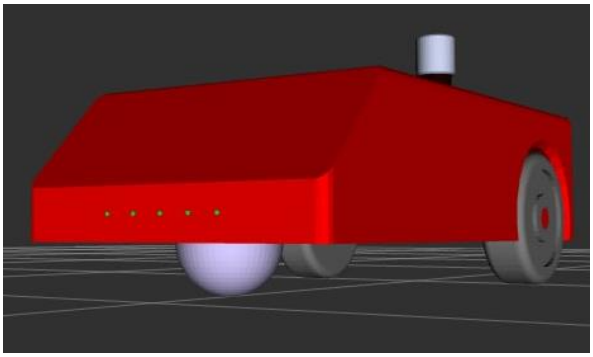
Právě z tohoto důvodu byla použita jednoduchá koule jako kolizní objekt. Koule je pevně umístěna k tělu robotu (neotáčí se) a díky velmi nízkému tření věrně simuluje pojezdové kolo bez nežádoucích interakcí se zemí. Na Obr. 21a lze vidět původní tvar kolize kola a na Obr. 21b je jeho náhrada koulí použitou pro kolizní vlastnosti.

Zde je popsána definice kolize jako koule uvnitř tagů <collision> a nulové tření (mu1 a mu2) uvnitř tagů <gazebo reference>, které se umisťuje mimo caster_wheel_link.

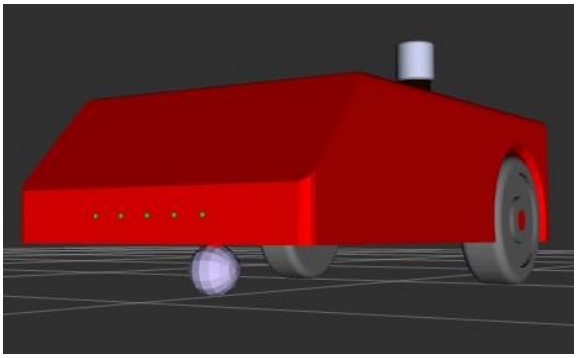
```
<link name=``caster_wheel_link``>
...
<collision>
     <origin xyz="0 0 0" rpy="0 0 0" />
     <geometry>
         <sphere radius="0.15"/>
    </geometry>
</collision>
...
</link>

<gazebo reference="caster_wheel_link">
    <mu1>0.001</mu1>
    <mu2>0.001</mu2>
</gazebo>
```

Poslední nutnou úpravou je přidání tzv. base_link (prázdný nadřazený link) do URDF, který je pro správné zobrazení v RViz nezbytný. Nejjednodušší je přidat ho přímo do URDF modelu robotu a u toho zadefinovat joint body_linku, aby bylo jasné, že je mu base_link nadřazený, viz Obr. 22 následovně:

```
<link name="base_link"/>
  <joint name="body_joint" type="fixed">
    <origin xyz="0 0 0" rpy="0 0 0"/>
    <parent link="base_link"/>
    <child link="body_link"/>
  </joint>
```

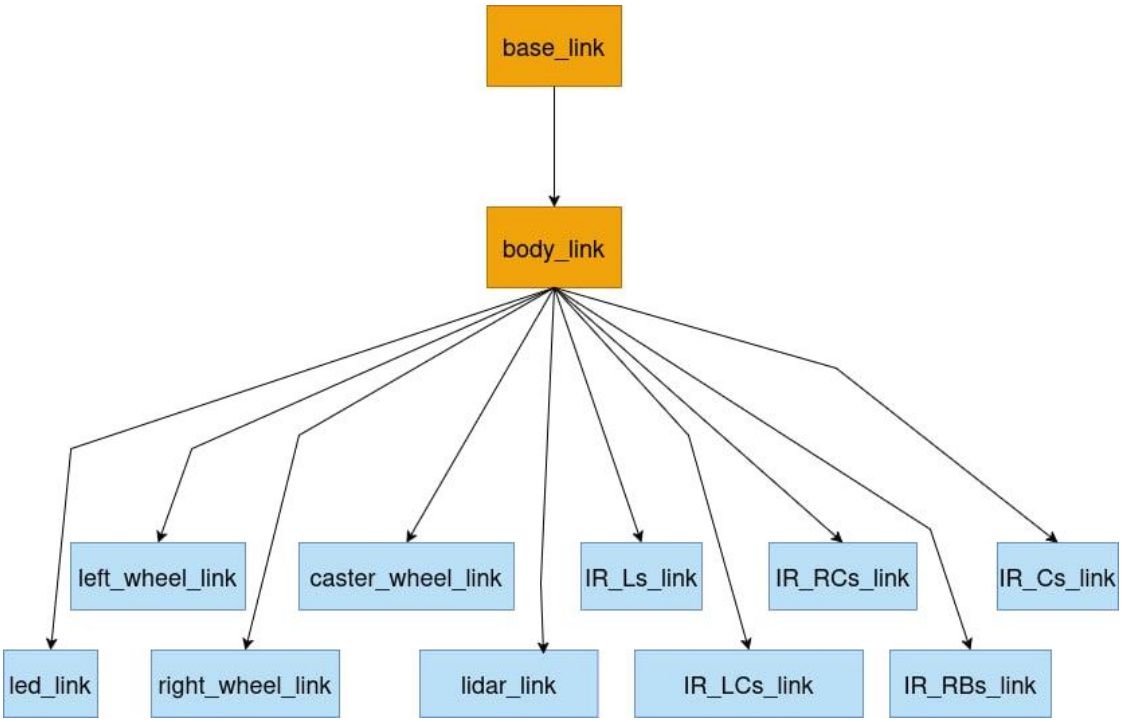


Obr. 21a) Kolize kola z STL souboru



Obr. 21b) Kolize kola definovaná v URDF

Obr. 21) Změna kolize pojezdového kola zobrazená v RViz



Obr. 22) Struktura URDF souboru

6.2 Definování senzorů v Gazebo Sim

Klíčovou součástí Gazebo Sim jsou senzory, které umožňují interakci robotu s virtuálním prostředím. V Gazebo Sim jsou senzory definovány v URDF nebo XACRO souborech (buď jako samostatný soubor nebo přímo v souboru s robotem) a každý senzor je vložen do modelu jako tag <sensor>. Tento tag poté obsahuje všechny vlastnosti a parametry, které popisují, jak se má senzor chovat. <sensor> se definuje v rámci tagu <gazebo> parametrem *gazebo reference*.

Mezi hlavní parametry senzoru patří:

- <name> – Název senzoru.
- <type> – Typ senzoru (např. gpu-lidar pro simulaci LiDARu).
- <pose> – Relativní umístění senzoru.
- <update_rate> – Frekvence aktualizace dat (Hz).
- <topic> – Určuje, na který topic má senzor publikovat data.
- <visualize> – Zda má být senzor v simulaci vidět (např. paprsky LiDARu).
- <Range, scan, resolution>, atd. – Parametry typické pro jednotlivé typy senzorů.

V této práci hlavní LiDAR senzory nahrazující IR, reprezentuje soubor *sensors.xacro* (lze najít v příloze 2 ve složce *dp2025_ws/src/dp_package/urdf)*. Je v něm definovaných všech 5 LiDAR senzorů, jejichž názvy jsou odvozeny dle jejich pozice. (např. CS_IR_link – central sensor link, LCS_IR_link – left central sensor link, LLS_IR_link – left left IR sensor link). Stejně tak názvy topiců na které publikují data (např. lls_ir, lcs_ir, atd.). Kromě 5 senzorů nahrazujících IR, se zde nachází i jeden hlavní LiDAR senzor, který byl umístěn na horní část robotu. Zde je příklad toho, jak vypadá definice hlavního LiDAR senzoru v *sensors.xacro*:

```
<gazebo reference="lidar_link">
  <sensor name="laser" type="gpu_lidar">
    <pose>0 0 0 0 0 0</pose>
    <update_rate>10</update_rate>
    <topic>scan</topic>
    <always_on>true</always_on>
    <gz_frame_id>lidar_link</gz_frame_id>
    <visualize>true</visualize>
    <lidar>
      <scan>
        <horizontal>
          <samples>360</samples>
          <min_angle>-3.14</min_angle>
          <max_angle>3.14</max_angle>
        </horizontal>
      </scan>
      <range>
        <min>0.06</min>
        <max>5.0</max>
        <resolution>0.01</resolution>
      </range>
    </lidar>
  </sensor>
</gazebo>
```

Oproti hlavnímu LiDARu se senzory, které nahrazují IR senzory, liší v několika věcech:

1. <topic> - Hlavní LiDAR publikuje na topic /scan, ovšem náhrady IR publikují na topicy lls_ir, lcs_ir, cs_ir, rcs_ir a rrs_ir (left left sensor IR, left central sensor IR, central sensor IR, right sensor IR, right right sensor IR).
2. <gz_frame_id> - Odkazuje se na link v URDF robotu představující daný senzor.
3. <samples> 360 pro hlavní lidar znamená 1 paprsek LiDARu na 1° pootočení. V případě IR senzorů bude sample pouze 1.
4. <min_angle> a <max_angle> - -3.14 a 3.14 znamenají, že hlavní LiDAR snímá v rozsahu 360°. U IR senzorů je tato hodnota 0, jelikož zde bude pouze jeden paprsek měřící vzdálenost.

6.3 Sjednocení souborů pomocí XACRO

V této fázi je tedy k dispozici model robotu uložený ve formátu URDF, který byl vygenerován pomocí pluginu SolidWorks to URDF exporter a také soubor se senzory *sensors.xacro*. Pro budoucí přehlednost a eliminaci rizika, že jakákoliv změna v jedné části modelu povede k rozsáhlému upravování kódu, bude URDF model robotu převeden do XACRO formátu. Toho lze docílit přidáním hlavičky do URDF souboru - <*robot xmlns:xacro="http://www.ros.org/wiki/xacro"*> a následně přepsat jeho formát na XACRO. Zbytek kódu v souboru se již nemění.

Jednou z nejsilnější funkcí XACRO oproti URDF je možnost definovat makra. Jedná se o znovupoužitelný kód, jehož příkladem může být opakované vytváření senzorů, výpočet těžiště tělesa atd. V případě, že je soubor vyexportován ze SolidWorks jsou tyto přepočty

provedeny automaticky, ovšem pokud je model robotu tvořen ručně psaním kódu, je třeba tyto věci definovat a do xacro souboru je zahrnout. Následně byl vytvořen soubor s názvem *robot_core.xacro* s následujícím obsahem, který představuje kompletní model robotu:

```
<?xml version=''1.0''?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
     <xacro:include filename="robot_model.xacro"/>
     <xacro:include filename="gz_control.xacro"/>
     <xacro:include filename="sensors.xacro"/>
     <xacro:include filename="led.xacro"/>
</robot>
```

Robot_model.xacro obsahuje 3D model robotu, *gz_control.xacro* reprezentuje plugin pro ovládání robotu pomocí klávesnice, *sensors.xacro* představuje soubor s LiDAR senzory a *led.xacro* představuje nastavení LED světla. Všechny soubory lze najít v příloze 2 v *dp2025_ws/src/dp_package/urdf*. Díky *robot_core.xacro* souboru je nyní možné přidávat nové xacro soubory a zahrnout je do modelu, aniž by bylo nutné upravovat již fungující stávající kód. Pokud by se tedy teď např. přidávala do robotu kamera, je možné vytvořit ji v novém xacro souboru např. *camera.xacro* a následně ji pouze zahrnout do *robot_core.xacro* takto:

```
<?xml version=''1.0''?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
     <xacro:include filename="robot_model.xacro"/>
     <xacro:include filename="gz_control.xacro"/>
     <xacro:include filename="sensors.xacro"/>
     <xacro:include filename="led.xacro"/>
     <xacro:include filename="camera.xacro''/>
</robot>
```

Je důležité všechny tyto soubory uchovávat na jednom místě, a to v tomto případě ve složce s názvem *urdf* (název složky je libovolný) uvnitř vytvořeného balíčku *dp_package*. Finální struktura vypadá následovně:

```
dp_package /
└── urdf/
    ├── robot_core.xacro
    ├── robot_model.xacro
    ├── sensors.xacro
    ├── gz_control.xacro
    ├── led.xacro
```

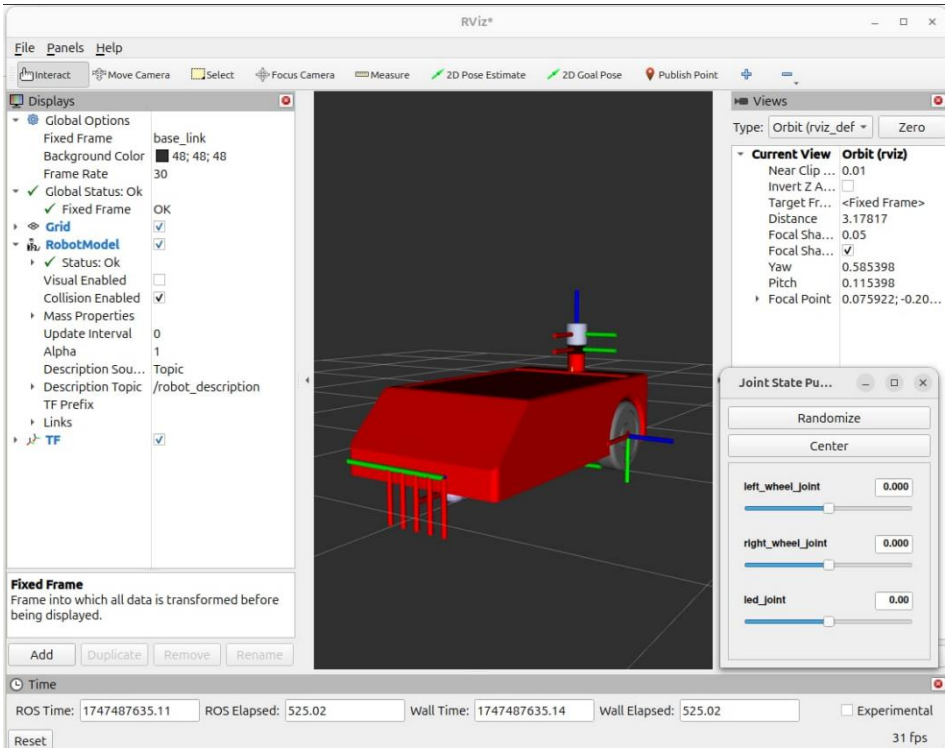
6.4 Zobrazení a kontrola modelu v RViz a VS Code

Zobrazení v RViz

Zda se model vyexportoval správně a všechny jeho linky a jointy jsou správně zadefinované, lze zjistit v již zmiňovaném RViz. K tomu, aby byl model ve vizualizaci vidět, je potřeba využít balíčku *robot state publisher*, který obsahuje node *robot_state_publisher*. Tento node publikuje informace o jednotlivých (statických) částech robotu na topic */robot_description* (typ zprávy std_msgs/msg/String). Pro zobrazení pohyblivých kloubů (v tomto případě obě kola) je zapotřebí také spustit node *joint_state_publisher*.

Pro spuštění *robot_state_publisher* a *joint_state_publisher* byl napsán python skript s názvem *rsp.launch.py* (příloha 2 v *dp2025_ws/src/dp_package/launch*). Ve skriptu je obsažena cesta k souboru daného modelu robotu, zpracování xacro souboru a poté spuštění

robot_state_publisher a *joint_state_publisher*. Pro kontrolu, že oba nody běží správně lze využít příkaz v terminálu *ros2 node list.* Ve správném případě by output měl být */robot_state_publisher* a */joint_state_publisher*. Více o launch souborech je zmíněno v kapitole 6.5. Následně lze v novém okně terminálu zadat příkaz *rviz2* a otevře se okno s vizualizací (i toto okno je třeba sourcovat příkazem– *source install/setup.bash*). Nyní je potřeba nastavit Fixed Frame na *base_link*. Žádný model nejprve vidět nejde. To protože zatím není dáno, jaký topic je potřeba odebírat a přidat RobotModel, který lze získat kliknutím na tlačítko Add v levém dolním rohu a výběrem příslušné kolonky RobotModel. V kolonce Description Topic je třeba zvolit */robot_description* a v tu samou chvíli by se již měl model ve vizualizaci objevit viz Obr. 23.



Obr. 23) RViz a joint_state_publisher_gui slider

Zda continuous joints fungují, jak mají, a tedy kola se otáčí, lze zjistit zadáním příkazu *ros2 run joint_state_publisher_gui joint_state_publisher_gui* do nového okna terminálu. Objeví se slider (viz pravá dolní strana Obr. 23), pomocí kterého lze otáčet obě kola. Pokud se kola otáčejí opačně, stačí v *robot_model.xacro* souboru v *left_wheel_joint* a *right_wheel joint* najít <axis xyz="0 0 1" /> a 1 změnit na -1 nebo naopak. Jak příkaz r*viz2*, tak *ros2 run joint_state_publisher_gui joint_state_publisher_gui* lze tak zakomponovat do launch souboru a spouštět je ihned, zároveň s oběma nody.

Zobrazení ve VS Code

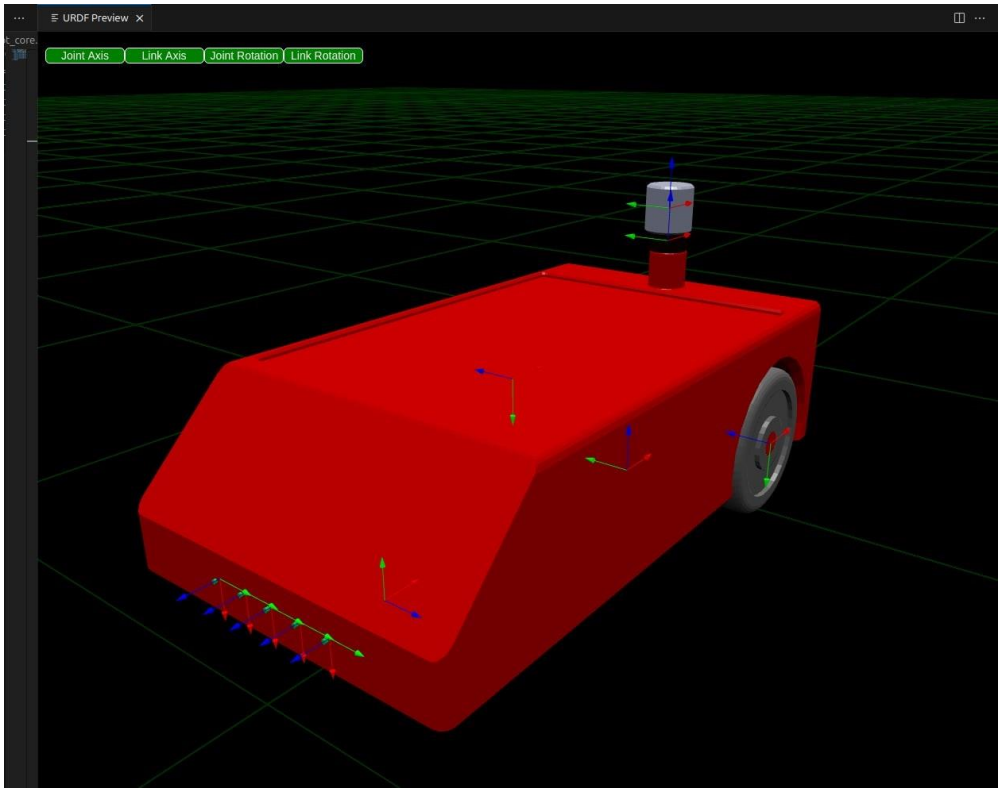
Další možností, kde zkontrolovat správnost modelu, je přímo VS Code, konkrétně s pomocí doplňku ROS. Tento doplněk umožňuje zobrazovat XACRO a URDF soubory přímo v okně Visual Studio Code a může tak výrazně urychlit vývojový proces. Pokud je opominut zde zmíněný SolidWorks to URDF plugin, je normálně URDF model tvořen zapisováním kódu „naslepo“ a následně je výsledek ověřen až v RViz. Tento proces je zdlouhavý, jelikož je potřeba napsat nejen kód modelu, ale také launch file pro jeho zobrazení.

Díky rozšíření ROS Extension pro VS Code se dá tento proces zefektivnit. Pro práci s novějšími verzemi VS Code je doporučeno pracovat s pre-release verzí ROS Extension, konkrétně s verzí 0.9.6, jelikož v předchozích verzích se nemusí URDF soubory zobrazovat správně. Doplněk obsahuje funkci pro živý 3D náhled URDF a XACRO souborů. Je možné tedy sledovat změny v modelu při psaní kódu okamžitě, což je oproti zobrazování v RViz velká výhoda.

Rozšíření se ukazuje jako velmi přínosné i při práci s pluginem SolidWorks to URDF Exporter. V případě, kdy je potřeba dodatečně doplnit např. chybějící senzor, není nutné zasahovat přímo do sestavy v prostředí SolidWorks, přidávat nové těleso a celé URDF včetně STL souborů znovu exportovat. Namísto toho lze chybějící prvek doplnit přímo v XACRO/URDF souboru a ihned si v živém náhledu ověřit jeho umístění a orientaci (včetně os a souřadných systémů).

Postup pro nastavení náhledu ve VS Code je následující:

1. Instalace ROS Extension – Ve VS Code nainstalovat rozšíření ROS. Doporučená verze je 0.9.6 (v době psaní práce se jedná o pre-release verzi).
2. Otevření složky s ROS workspace – Ve VS Code je zejména pro správné načtení mesh souborů (STL) důležité být ve správné složce, a to v kořenové složce ROS2 workspace. V případě této práce se jedná o *dp2025_ws*.
3. Otevření XACRO/URDF souboru – Otevřít hlavní soubor robotu, zde konkrétně *robot_core.xacro.*
4. Spuštění URDF preview – Použitím Ctrl + Shift + P lze vyhledat ROS: Preview URDF.
5. Zobrazení modelu – Okno s 3D náhledem je otevřeno viz Obr. 24.



Obr. 24 Zobrazení modelu ve VS Code

6.5 Tvorba launch souborů

V prostředí ROS2 je obvyklé, že je potřeba spouštět několik nodů zároveň. Místo toho, aby byl každý node spouštěň zvlášť v několika oknech terminálu, je možné vytvořit tzv. launch soubory. Jedná se o elegantní řešení, díky kterému je možné spustit jedním příkazem libovolný počet nodů. Těm lze jednotlivě definovat parametry, pořadí ve kterém se mají spouštět atd.

Launch soubory se obvykle tvoří v programovacím jazyku Python, kde využívají *launch_ros* knihovnu (lze je také tvořit v XML nebo YAML) a spouští se příkazem *ros2 launch název_balíčku název_launch_souboru*. Obvykle každý launch soubor obsahuje funkci *generate_launch_description()*, která vrací instanci třídy LaunchDescription s výčtem všech požadovaných akcí – spuštění nodu, nastavování proměnných, předávání parametrů apod.

V textu práce níže je zobrazen jednoduchý launch soubor s názvem *rsp.launch.py*, který byl vytvořen za účelem načtení a publikování dat modelu robotu pomocí *robot_state_publisher* a *joint_state_publisher* nodů. Hlavním cílem tohoto souboru je příprava prostředí pro zobrazení robotu v RViz a umožnění zkontrolovat jeho jednotlivé části viz předchozí kapitola.

```
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import
get_package_share_directory
import os
import xacro

def generate_launch_description():
    package_name = "dp_package"
    pkg_share = get_package_share_directory(package_name)
    # Cesta k hlavnímu XACRO souboru
    xacro_file = os.path.join(pkg_share, 'urdf',
'robot_core.xacro')

    # Zpracování XACRO do URDF
    robot_description_config = xacro.process_file(xacro_file)
    robot_description = {'robot_description':
robot_description_config.toxml()}

    return LaunchDescription([
        # Spuštění robot_state_publisher
        Node(
            package='robot_state_publisher',
            executable='robot_state_publisher',
            name='robot_state_publisher',
            output='screen',
            parameters=[robot_description]
        ),
        # Spuštění joint_state_publisher
        Node(
            package='joint_state_publisher',
            executable='joint_state_publisher',
            name='joint_state_publisher',
            parameters=[robot_description]
        ),
    ])
```

Zde jsou popsány jednotlivé části launch souboru:

1. Získání cesty k XACRO souboru – Soubor *robot_core.xacro*, který reprezentuje celý model robotu, se nachází ve složce *urdf* ROS2 balíčku *dp_package*. Díky funkci *get_package_share_directory()* je nalezena cesta k balíčku a poté k souboru.
2. Zpracování XACRO do URDF – Funkce *xacro.process_file()* načte soubor a převede ho do běžného URDF formátu. Výsledné hodnoty ze souboru jsou uloženy do proměnné *robot_description*, která bude předána jako parametr oběma nodům.
3. Spuštění nodů r*obot_state_publisher* a *joint_state_publisher* – Tyto dva nody jsou klíčové pro vizualizaci modelu robotu v RViz.
    - *robot_state_publisher* čte URDF popis robotu a publikuje transformační framy (TF) mezi jednotlivými částmi.
    - *joint_state_publisher* vytváří a publikuje informace o stavu kloubů robotu. V případě simulace bez reálného hardware se zde využívají posuvníky k simulaci pohybu (např. kol).

Po zajištění, že je balíček správně nainstalován (colcon build) a zdrojován (*source install/setup.bash*), lze soubor spustit příkazem:

```
ros2 launch dp_package rsp.launch.py
```

7 SIMULACE A TESTOVÁNÍ

Po úspěšném vytvoření modelu robotu, jeho integraci s ROS2 a implementaci všech potřebných senzorů je potřeba vše řádně simulovat, rozchodit a otestovat. Tato část práce se zabývá ověřením funkčnosti celého systému bez využití reálného hardwaru. Díky simulaci lze nejen ladit pohybové algoritmy, ale také testovat, zda senzory fungují správně a robot správně reaguje na jejich výstup.

Simulace je provedena v Gazebo Sim, které obsahuje realistický fyzikální engine a je možné vizualizovat všechny relevantní prvky robotu. Pomocí Gazebo bylo vytvořeno virtuální prostředí, do kterého byl robot umístěn s cílem sledovat pomocí jeho senzorů čáru a pohybovat se po ní. Zahrnuty jsou také některé bezpečnostní prvky robotu.

V této části je také řešena samotná logika robotu, která je pro správný chod simulace klíčová. Logika je implementována pomocí ROS2 nodů. Ty fungují pro zpracování dat ze senzorů, rozhodují o směru pohybu robotu atd. Cílem této kapitoly je vytvoření a simulace určitého scénáře pro robot, demonstrace správnosti funkce jednotlivých systémů a ověření, zda se robot chová, jak má.

7.1 Tvorba prostředí simulace

První věc, která je potřebná pro simulaci je její prostředí. Světy v Gazebu mají příponu SDF (structured data file) a opět se píšou jako XML. Pro účely simulace byl použit prázdný svět, kde se vyskytuje pouze robot a model čáry, kterou má následovat, pár překážek a actor. Model se do světa přidává mezi tag <world> v tomto případě následovně (viz v příloze 2 *track.sdf* ve složce *dp2025_ws/src/dp_package/worlds*):

```
<model name="track_model">
    <static>true</static>
    <link name="link">
        <pose>0 0 -0.13 1.5708 0 1.5708</pose>
        <visual name="visual">
        <geometry>
            <mesh>
              <uri>package://meshes/path.STL</uri>
               <scale>0.001 0.001 0.001</scale>
             </mesh>
        </geometry>
            <material>
              ...
            </material>
        </visual>
    </link>
</model>
```

Základní jednotky jsou zde metry. Pokud se v SolidWorks pracuje v milimetrech, což je samozřejmě standard, je potřeba v <mesh> tagu použít také tag <scale> a dát mu hodnotu 0.001 a tak převést jednotky z metrů na milimetry. V tomto případě se jedná o model uložený lokálně a z toho vyplývá odkaz na něj v <uri> tagu.

Další možností, jak do Gazebo světa vložit model je využít Gazebo fuel a stáhnout již předem vytvořený model. Na oficiálních webových stránkách je k dispozici ke stažení a implementaci nespočet modelů. K dispozici jsou modely robotů, různých komponent, stromů,

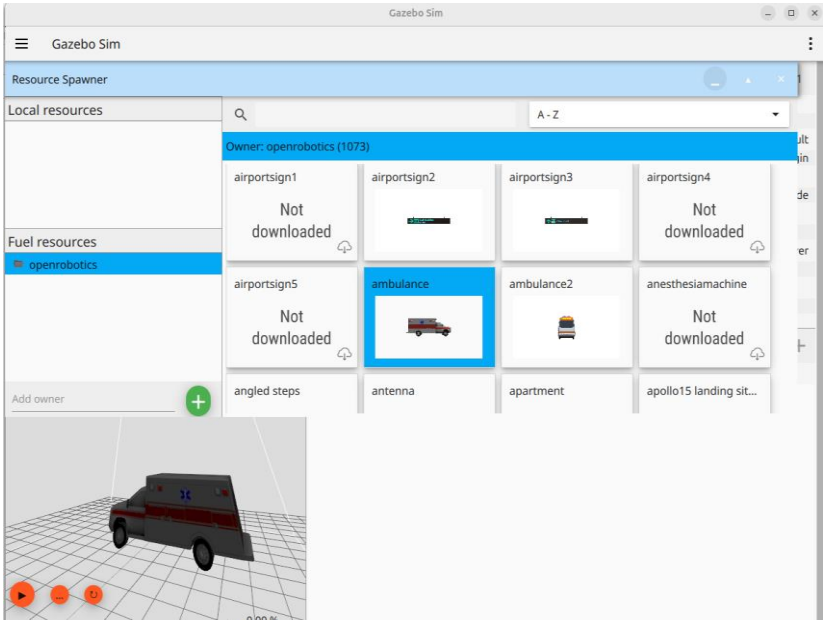
nábytku atd. Není ale třeba zůstávat jen u modelů, jelikož k dispozici ke stažení jsou i celé SDF světy, které je možné rychle implementovat do projektu a pracovat s nimi.

Způsoby, jak model z Gazebo fuel přidat jsou dva:

1. Stažení SDF modelu z webu Gazebo Fuel – Nejprve je možné navštívit oficiální webové stránky Gazebo Sim, kde lze najít spoustu modelů, které jsou volně ke stažení. Výhoda oproti Resource Spawner funkce přímo v Gazebo je ta, že webové rozhraní nabízí prohlížení náhledů modelů ještě před jejich stažením. Po nalezení příslušného modelu ho lze stáhnout ve formátu SDF včetně složky s potřebnými daty (mesh, textury apod.).

2. Implementace modelu přímo v Gazebo Sim pomocí Resource Spawner – Druhou možností je model přidat bez nutnosti navštěvovat web, ale využít pluginu přímo v Gazebo Sim. Kliknutím na možnosti v pravém horním rohu (reprezentováno třemi tečkami) se rozbalí několik možností. Po zvolení Resource Spawner se objeví okno s modely ke stažení. V kolonce Fuel resources je zprvu vidět pouze *openrobotics*, což jsou modely od vydavatelů samotného Gazeba. Je ale možné přidat další autory modelů. Pro vložení modelu stačí kliknout na vybraný model, ten se vzápětí stáhne a poté již stačí kliknout na místo, kam je model potřeba vložit. V tomto případě je pro ukázku do světa vkládán model s názvem ambulance2, a to kliknutím na příslušné místo v simulačním okně Gazebo Sim, jak lze vidět na Obr. 25.

   Nevýhoda vkládání modelů přes Resource Spawner oproti webu spočívá v tom, že se nezobrazují náhledy modelů, které ještě nejsou lokálně stažené (viz Obr. 25, obrázky s tagem Not Downloaded). Pokud tedy není hledán konkrétní model, je lepší využít webových stránek, kde náhledy vidět jsou a poté např. dle názvu modelu využít již zmiňovaný Resource Spawner.



Obr. 25) Gazebo Sim a výběr modelu z Fuel

7.2 Gazebo Sim bridge a vizualizace

Jakmile je k dispozici SDF, je možné do něj vložit robot. Pro účely otestování senzorů a zobrazení modelu byl vytvořen soubor světa *my_world.SDF* (structured data file), který udává prázdný svět, jednoduché kolize (ground plane, aby se model robotu nepropadl do země), fyzikální pluginy a plugin pro LiDAR. Opět byl vytvořen launch file (*gazebo_simulace.launch.py* lze najít v příloze 2 ve složce *dp2025_ws/src/dp_package/launch*) pro snadnější spouštění všech potřebných souborů. Obsahuje spuštění *rsp.launch.py*, nalezení modelů robotu a SDF světa, nakonec samotného Gazebo Sim a spawn robotu.

Co se týče Gazeba a dat simulovaných jednotlivými senzory je nutno zmínit jednu klíčovou věc. Data ze senzorů nejsou automaticky publishována na ROS2 topics. Proto pokud např. simulovaný LiDAR posílá data na topic */lidar* a je použit příkaz *ros2 topic list* na výpis aktivních topiců, */lidar* zde není k nalezení. Tento topic je nutno hledat příkazem *gz topic -l*, což je Gazebo verze *ros2 topic list*. Pro účel využití dat ze senzorů v Gazebo a jejich následné zužitkování v prostředí ROS2 je nutné data přemostit z Gazebo Sim do ROS2. V případě této práce publishuje LiDAR mimo jiné i na topic */cs_ir*. Vytvořit bridge pro jeden topic lze i v okně terminálu následujícím příkazem:

```
ros2 run ros_gz_bridge parameter_bridge
/cs_ir@sensor_msgs/msg/LaserScan@gz.msgs.LaserScan
```

Rozpis jednotlivých parametrů příkazu:

- *ros2 run* – Syntax pro spouštení node,
- *ros_gz_bridge* – package umožňující přemostění mezi Gazebo a ROS2,
- *parameter_bridge* – spouštěný program v rámci ros_gz_bridge,
- */cs_ir* – topic, který má být přemostěn,
- *sensor_msgs/msg/LaserScan* – typ zprávy v ROS2,
- *gz.msgs.LaserScan* – typ zprávy v Gazebo Sim.

Pokud je potřeba přemostit více topiců, stejně jako v případě této práce, je vhodné vytvořit pro ně bridge najednou. Toho lze docílit tvorbou config souboru pro bridge. Ten byl vytvořen pod názvem *config_sensors.YAML* (lze ho najít v příloze 2 ve složce *dp2025_ws/src/dp_package/config*). Obsahuje parametry pro přemostění 6 LiDAR topiců, *cmd_vel* topicu a *led* topicu pro změnu barvy LED. Následně byl vytvořen launch soubor pro spuštění bridge nodu, který lze spustit příkazem: *ros2 launch dp_package bridge.launch.py*

Po spuštění bridge nodu už by měly být všechny topicy viditelné i po zavolání *ros2 topic list*. Zde je uveden příklad, jak vypadá přemostění topicu */cs_ir* v *config_sensors.YAML* souboru (pro ostatní topicy platí stejný syntax):

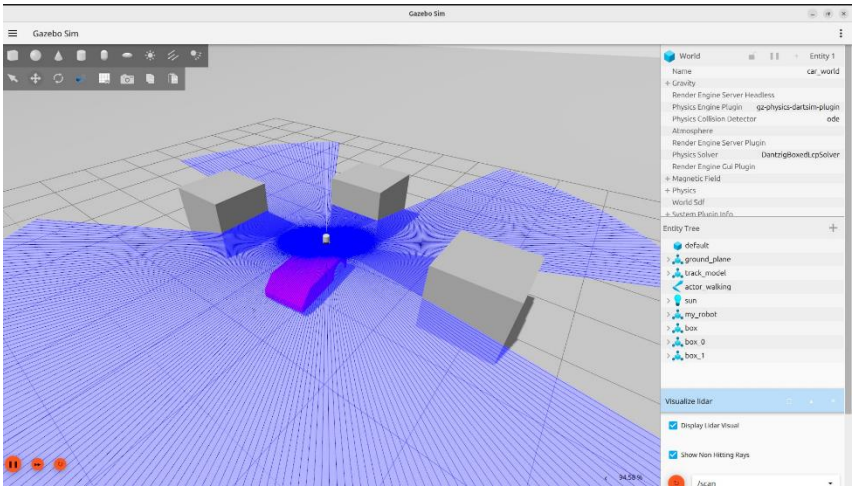
```
- ros_topic_name: "cs_ir"
  gz_topic_name: "/cs_ir"
  ros_type_name: "sensor_msgs/msg/LaserScan"
  gz_type_name: "gz.msgs.LaserScan"
  direction: BIDIRECTIONAL
```

Pro zjištění typu zprávy (message type) ROS2 a Gazebo pro některý topic lze je buď dohledat v dokumentech nebo využít příkazů:

```
ros2 topic info název_topicu
gz topic -i -t název_topicu
```

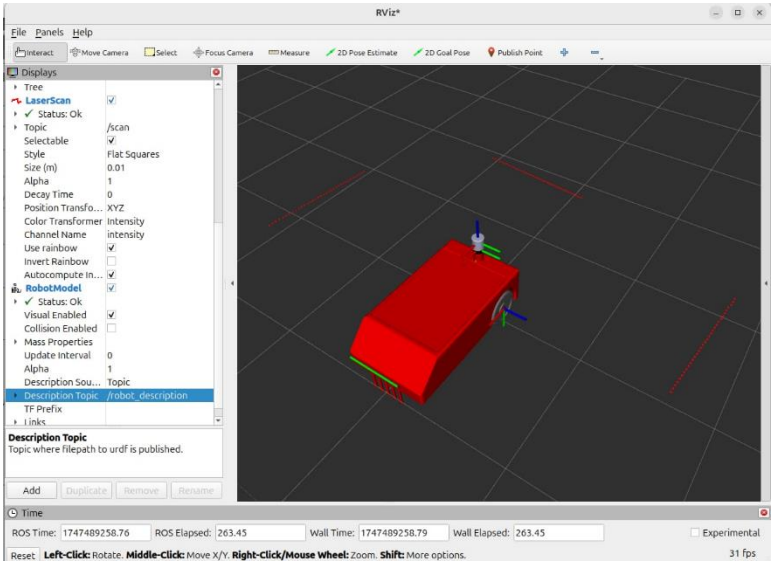
Po přemostění všech žádaných topiců, je možné začít testovat vizualizaci v Gazebo Sim. Je třeba zjistit, zda jsou LiDAR senzory správně zadefinované a fungují, jak mají. Je také důležité ověřit, zda se robot v simulaci pohybuje správně a je kompletní.

Příkazem do terminálu *ros2 launch dp_package gazebo_simulace.launch.py* se spustí Gazebo Sim s předem vytvořeným světem, uprostřed něhož lze vidět vytvořený model robotu (viz Obr. 26). Pro otestování, zda se robot pohybuje správně lze využít diff drive pluginu. V pravém horním rohu je třeba zvolit tlačítko možnosti (reprezentováno 3 tečkami) a vyhledat možnost *Key Publisher*. Po spuštění simulace tlačítkem Start v levém dolním rohu je možné ovládat pohyb robotu pomocí šipek na klávesnici. Správnou funkčnost LiDARu lze ověřit spuštěním pluginu *Visualize Lidar* – kliknutím na 3 tečky v pravém horním rohu a zakliknutím daného pluginu. Do světa byly pro účel demonstrace funkčnosti LiDARu přidány 3 kvádry. Laserové paprsky LiDARu simulované modrými čarami neprojdou kvádry a senzor tedy funguje správně.



Obr. 26) Test LiDAR senzorů v Gazebo Sim

Z pohledu robotu lze tento jev sledovat v RViz (viz Obr. 27), kde jsou kolem robotu vidět 3 červené „čáry“, tvořené body, kam paprsky laseru dopadají a robot tedy ví, že se v jeho blízkosti něco nachází. Pro zobrazení těchto bodů je nutné v RViz přidat *LaserScan* pod tlačítkem Add a zvolení správného topicu – v tomto případě */scan*.



Obr. 27) RViz vizualizace outputu z LiDAR senzorů

7.3 Tvorba a implementace ovládacích nodů

Nyní je potřeba objasnit, jak implementovat řídící logiku pro daný robot. Jak bylo zmíněno v teoretické části práce, pro implementaci logiky a ovládání robotu byl vytvořen node s názvem line_follower, který obsahuje nejen veškerou logiku sledování čáry, ale i tvorbu všech potřebných subscriberů a publisherů. Tento node je obsažen v souboru s názvem *pid_5sensors.py* a lze ho opět najít v příloze 2 práce (ve složce *dp2025_ws/src/dp_package/scripts*). Tento soubor není záměrně integrován v launch filech práce, kvůli následné implementaci na Raspberry Pi 5.

Sběr dat ze senzorů – Node line_follower v první řadě inicializuje čtení dat z 5 LiDAR senzorů, které slouží jako náhrada IR senzorů v případě reálného nasazení. K tomuto účelu tedy spustí 5 subscriberů na topicy *lls_ir, lcs_ir, cs_ir, rcs_ir a rrs_ir (*označující postupně senzory zleva doprava). Na tyto topicy jsou odesílána data ze senzorů ze simulace v Gazebo Sim ve formátu zprávy *sensor_msgs/msg/LaserScan*. V příslušné callback funkci pro všechny senzory je poté zaznamenána hodnota vzdálenosti ze senzoru. Ta je dále uložena do pole s názvem *sensor_values*, ve kterém každá pozice odpovídá jednomu senzoru. Díky tomu lze sledovat informace stavu všech 5 senzorů na jednom místě.

Řízení pohybu a PID regulátor – Co se týče pohybu robotu, ten je řízen publikováním zpráv typu *geometry_msgs/msg/Twist* na topic cmd_vel. Díky PID regulaci je v tomto případě upravována úhlová rychlost robotu v závislosti na odchylce od ideálního směru. Vzdálenosti senzorů jsou převedeny na binární hodnoty a jsou uloženy do pole s názvem *sensor_activations*, kde 0 značí absenci čáry a 1 detekci čáry. Pro výpočet odchylky byla každému senzoru přiřazena váha v závislosti na jeho pozici. Krajní senzory mají hodnoty -2 a 2, levý střední a pravý střední senzor mají hodnoty -1 a 1 a prostřední senzor 0. Jedná-li se jednoduchý případ, kdy je aktivní pouze jeden senzor, jsou v logice nastaveny pevné hodnoty odchylky. Např. -2 ostře vlevo, případně 2 ostře vpravo. V ostatních případech je odchylka vypočítána jako vážený průměr hodnot senzorů.

Chyba (error) je následně využita pro výpočet výsledné korekce pohybu robotu:

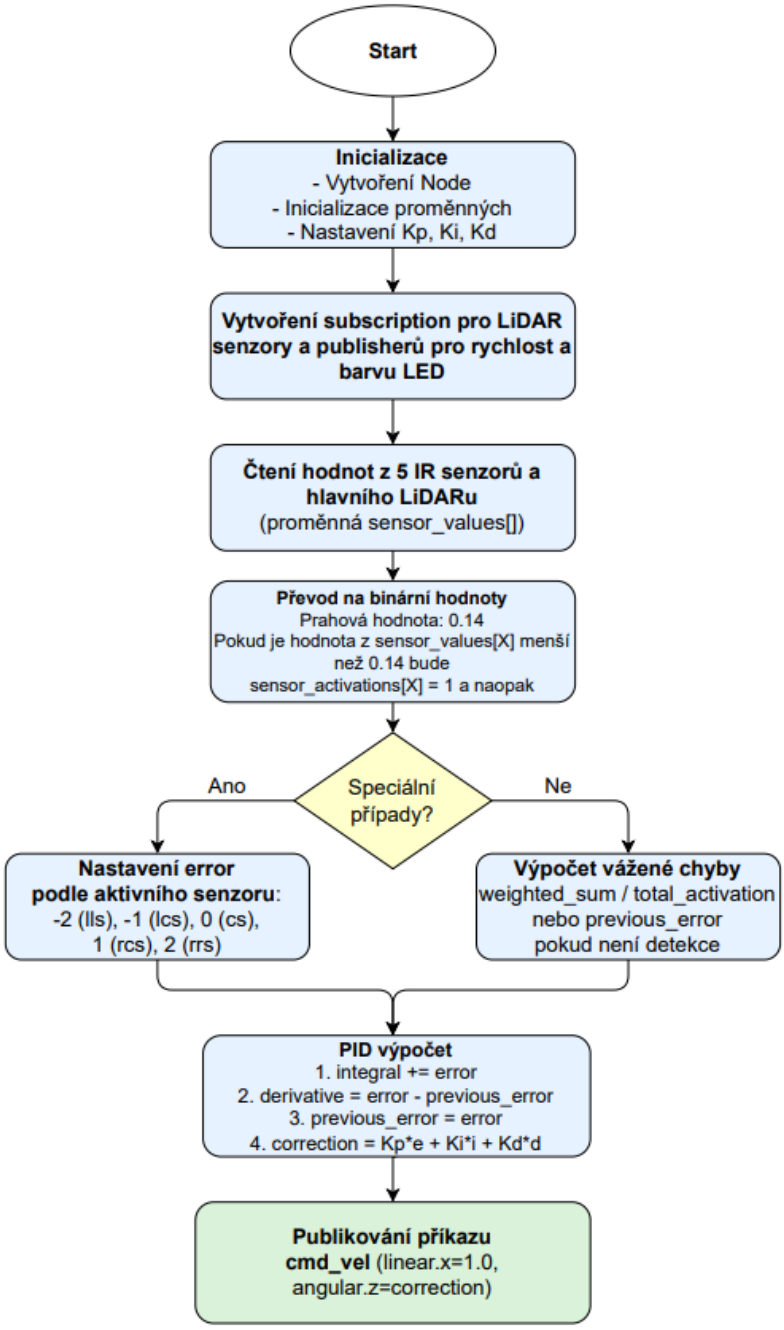
$$correction = Kp * error + Ki * integral + Kd * derivative$$

Kde:

- Kp je proporcionální složka,
- Ki je integrační složka, – integral += error
- Kd je derivační složka. – derivative += error – previous_error

Výsledná korekce je poté použita jako úhlová rychlost robotu. Lineární rychlost zůstává konstantní. Jednoduchý vývojový diagram lze vidět níže na Obr. 28, kde:

- Inicializace – Tvorba subscriberů,
- Binární aktivace – převod vzdáleností LiDAR senzorů na binární hodnoty,
- Speciální případ – zda je pouze jeden aktivní senzor.



Obr. 28) Vývojový diagram logiky sledování čáry

Pro účely implementace hlavního bezpečnostního LiDARu byl vytvořen také node s názvem line_follower_with_safety, který byl obsažen v souboru *safety_pid.py* a lze ho najít ve složce *dp2025_ws/src/dp_package/scritps* v příloze 2 a jeho vývojový diagram v příloze 1. Jedná se o totožnou logiku sledování čáry, avšak s přidanými bezpečnostními prvky, které budou představeny dále.

Spuštění finální simulace pro sledování čáry s řídící logikou

Po úspěšném vytvoření hlavního řídícího nodu následuje jeho integrace do samotné simulace v Gazebo Sim. Jedná se o důležitou část projektu a umožňuje robotu reagovat na data z virtuálních senzorů a vykonávat požadované manévry v závislosti na okolním prostředí.

Byl vytvořen hlavní launch soubor *allinone.launch.py* (příloha 2, *launch*), který zajišťuje:

- Start simulace v Gazebo Sim – Spustí se se světem track.sdf, který obsahuje trasu robotu ve formě 3D modelu, pohybujícího se člověka atd.
- Spawn robotu do světa simulace – Robot je umístěn na začátku trasy.
- Spuštění bridge – Launch file také spouští obousměrný bridge mezi Gazebo Sim a ROS2, který byl popsán v kapitole 7.2.

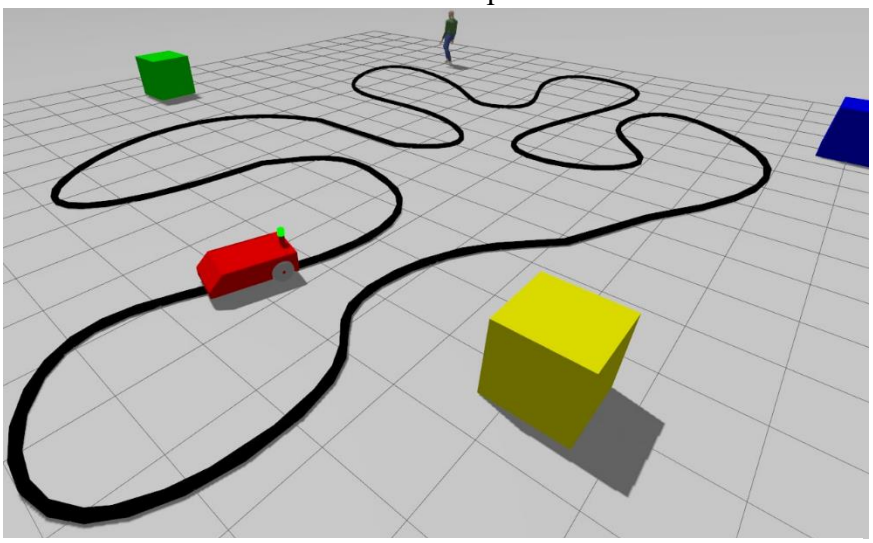
Jako první je tedy inicializována simulace v Gazebo Sim následovně:

```
ros2 launch dp_package allinone.launch.py
```

Tímto příkazem se otevře okno s kompletní a připravenou simulací. Dalším krokem je spuštění řídícího nodu line_follower_with_safety následovně:

```
python3 dp2025_ws/src/dp_package/scripts/safety_pid.py
```

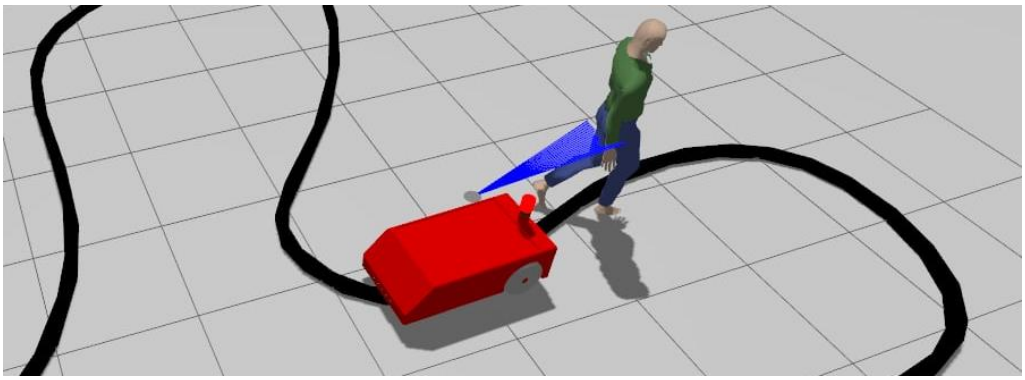
V tuto chvíli je vše připraveno a lze spustit simulaci tlačítkem Play, které se nachází v levém dolním rohu okna Gazebo Sim. Po stisknutí tlačítka se robot rozjíždí a lze pozorovat, že se celou dobu drží na čáře viz Obr. 29 a video v příloze 4.



Obr. 29) Robot sledující čáru

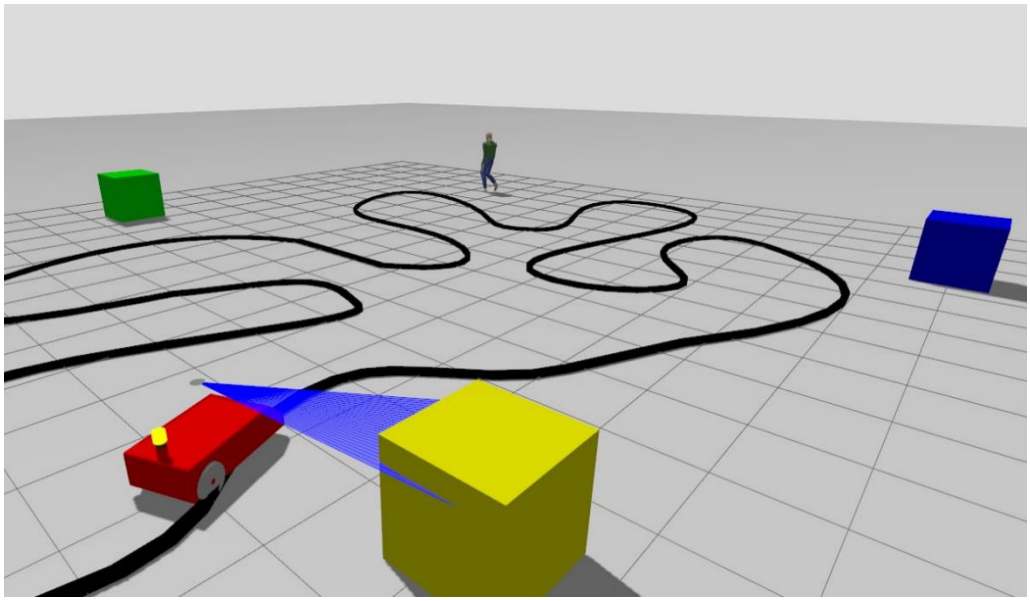
Funkce spojené s LiDARem na vrchní části robotu (funkce navíc oproti *pid_5sensors.py*):

- Úplné zastavení v těsné blízkosti překážky – Pokud LiDAR detekuje překážku, člověka nebo cokoliv v blízkosti menší než 1.5 m, rozsvítí červenou LED a okamžitě zastaví pohyb viz Obr. 30.



Obr. 30) Robot stojí a LED svítí červeně z důvodu bezpečnosti

- Zpomalení při detekci překážky/člověka – Detekuje-li LiDAR cokoliv ve vzdálenosti menší než 3 m, LED světlo se rozsvítí žlutou barvou a robot zpomalí na polovinu své původní rychlosti viz Obr. 31.

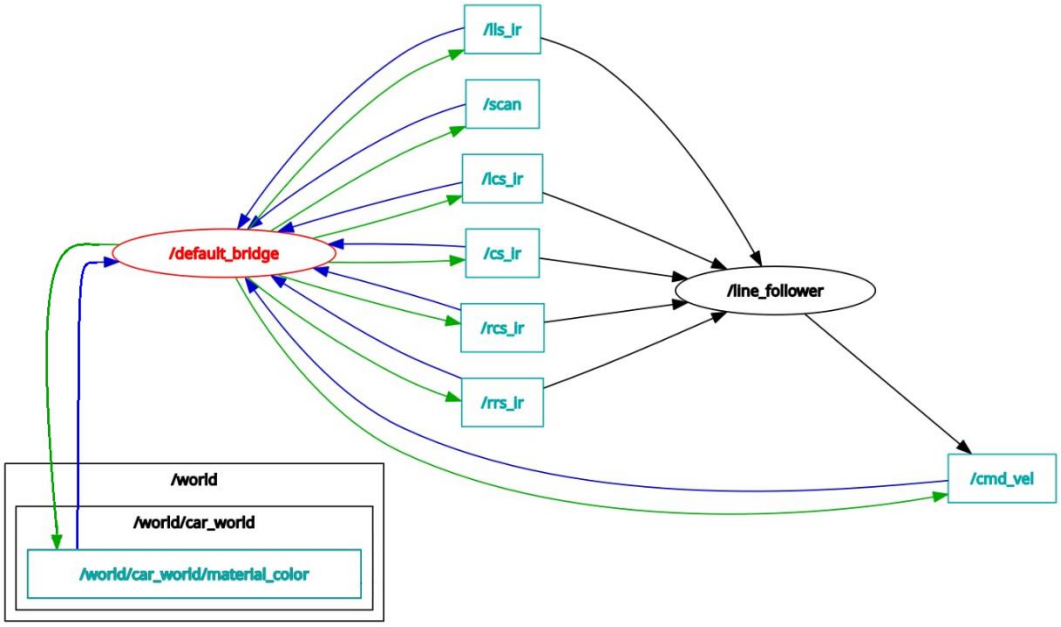


Obr. 31) LIDAR detekující kvádr v blízkosti robotu a žlutá LED

- Signalizace rozjezdu – Pokud robot zastaví kvůli překážce v jeho blízkosti a překážka zmizí, rozsvítí žlutou LED pro signalizaci rozjezdu, počká 2 sekundy a až poté se robot rozjede plnou rychlostí a LED světlo rozsvítí zelenou barvou.

Pro lepší pochopení celého systému nebo případný troubleshooting je možné v ROS2 využít příkazu *rqt_graph*, který umožňuje nahlédnout na grafické zpracování a propojení celého systému. To lze vidět na Obr. 32. Jednotlivé prvky jsou od sebe graficky odděleny jak barvou, tak tvarem – nody jsou elipsy, topicy jsou znázorněny obdélníky, modré šipky znázorňují subscribery a zelené Publishery (řídící line_follower node je černý 5x subscriber 1x publisher).

1. /line_follower_with_safety node (černá elipsa) – Popsán výše.
2. /default_bridge node (červená elipsa) – V tomto případě se node /default_bridge chová jako most mezi daty z Gazebo Sim a ROS2. Přenáší data z Gazebo Sim do ROS2, a naopak z ROS2 do Gazebo Sim.
3. Data ze senzorů lls_ir, rrs_ir, cs_ir, lcs_ir, rcs_ir a scan – Přijímané hodnoty ze senzorů jsou pomocí /default_bridge převedeny na hodnoty čitelné pro ROS2 a následně jsou zpracovány v /line_follower nodu. Všechny senzory pracují v Gazebo Sim s frekvencí 10 Hz a jejich message type je sensor_msgs/LaserScan.
4. /cmd_vel topic – Topic který představuje výsledný pohyb robotu. Na tento topic publishuje /line_follower_with_safety node výsledné hodnoty rychlostí, podle kterých /cmd_vel řídí robot a sleduje čáru. Co se týče formátu zprávy, používá message type geometry_msgs/Twist.
5. /world/car_world/ – Reprezentuje simulaci světa v Gazebo Sim.
6. world/car_world/material_color – Topic, který ovládá barvu LED diody.



Obr. 32) Výstup z rqt_graph s řídícím skriptem pid_5sensors.py

Jak lze tedy vidět, rqt_graph poskytuje vhled do fungování systému v reálném čase. Díky němu je možno rychle identifikovat problémy v komunikaci mezi jednotlivými komponentami. Např. pokud některý z topiců nepřijímá data, jak by měl, je tohle velmi jednoduchý a účinný způsob, jak odhalit, zda je s ostatními nody spojen správně.

7.4 Ovládání simulace s Raspberry Pi a Pi Pico

Jednou ze součástí návrhu systému v této práci byla integrace vývojových desek. Pro tento účel bylo zvoleno Raspberry Pi 5 a později Raspberry Pi Pico. V obou případech byl záměr docílit přímého ovládání simulace v prostředí Gazebo Sim pomocí těchto vývojových desek, což umožní testovat algoritmy a další nastavení bez nutnosti vlastnit fyzický hardware. V ROS2 je možné provozovat část řídící logiky na odděleném zařízení, které komunikuje se simulací, která běží na výkonnějším hostitelském počítači. ROS2 je pro tento případ skvěle vybaven a bez větších potíží toho lze dosáhnout pomocí připojení obou zařízení na lokální síti. Jednoduché schéma lze vidět na Obr. 33.

ROS2 umožňuje jednotlivým nodům a topicům rozprostřít se mezi více zařízení v rámci jedné lokální sítě, a to i bez složitější konfigurace. Každé zařízení, jež má správně nastavené ROS2 prostředí a je připojeno k lokální síti, automaticky sdílí všechny topicy a nody s ostatními zařízeními v dané síti. V případě této práce tedy na PC běží simulace v Gazebo Sim, ze které jsou data ze senzorů publishována na topicy, které byly již zmíněny ve dřívějších kapitolách a node pro účely sledování čáry (line_follower/line_follower_with_safety) je spuštěn na Raspberry Pi 5. Node sbírá data ze senzorů, následně je vyhodnotí a nakonec posílá zpět výslednou rychlost robotu na topic cmd_vel z Gazebo Sim, které běží na hostitelském PC.

Pro tento účel bylo nutné nainstalovat na Raspberry Pi 5 distribuci ROS2, která je shodná s verzí ROS2 na PC – tedy Jazzy Jalisco. Na Raspberry není nutné instalovat plnou verzi ROS2. Pro tyto případy je možné nainstalovat tzv. bare bones verzi. Funkčnost tohoto prostředí je úplně stejná jako v klasické verzi, ale není možné využívat GUI nástrojů jako je např. RViz. Instalace plné verze na RPI je také možná, avšak simulace v Gazebo Sim není reálná, jelikož je aplikace velmi náročná.

7.4.1 Integrace skriptu na Raspberry Pi 5

1. Instalace OS na Raspberry Pi 5 – Instalace Ubuntu 24.04. Pouze headless verze, jelikož instalovaná verze ROS2 je bez GUI.
2. Připojení RPI ke stejné místní síti jako host PC.
3. Instalace ROS2 Jazzy Jalisco – Lze využít návodu v úvodu praktické části práce, a podle oficiálních dokumentů nainstalovat pouze bare bones verzi.
4. Inicializovat ROS2 prostředí na RPI pomocí terminálu:

```
source /opt/ros/jazzy/setup.bash
export ROS_DOMAIN_ID=0
```

   ROS_DOMAIN_ID umožňuje rozlišit více zařízení na jedné síti. V síti se vidí pouze zařízení se stejným *ROS_DOMAIN_ID* (to stejné je tedy nutné udělat na PC).
5. Tvorba workspace a package – postup shodný jako klasický ROS2.
6. Upload řídícího skriptu – umístit do např. *dp2025_ws/src/rpi_package/scripts* a následná kompilace workspace pomocí *colcon build.*
7. Spuštění ROS2 nodu:

```
python3 src/rpi_package/safety_pid.py
```

8. Na PC spuštění Gazebo simulace pomocí launch file:

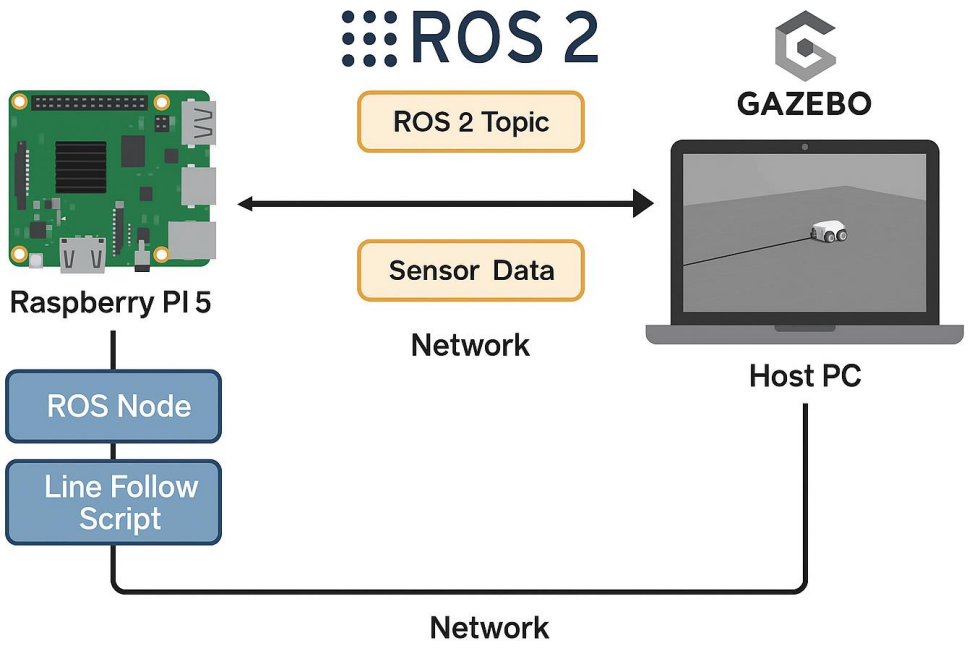
```
ros2 launch dp_package allinone.launch.py
```

9. Spustit bridge Gazebo←→ROS2 (pokud není zahrnut v jiném launch file, zde už je).
10. Test komunikace – ověření, že lze vidět nody a topicy i na PC:

```
ros2 topic list
ros2 topic echo /nazev_topicu
```

    První příkaz vypíše všechny topicy a druhý začne vypisovat data přijímaná na vyžádaný topic.

11. Pokud je vše správně, node běžící na Raspberry Pi 5 nyní ovládá robot v Gazebo Sim.



Obr. 33) Schéma fungování Raspberry Pi s ROS2 a Gazebo

7.4.2 Integrace skriptu na Raspberry Pi Pico

V další fázi bylo využito levnějšího a kompaktnějšího mikrokontroleru Raspberry Pi Pico ve spojení s micro-ROS. Ten umožňuje využívat ROS2 funkcionalitu i na zařízeních s velmi omezenými výpočetními prostředky. I tak lze ale využívat základní funkce ROS2 jako jsou zprávy, nody, topicy atd. Implementace micro-ROS na Pi Pico je oproti RPI 5 náročnější. Velkou nevýhodou je, že vývoj nelze provádět v jazyce Python tak, jako je tomu v ROS2. Veškerou logiku je nutno psát v jazyce C nebo C++, což může být z hlediska akademického využití komplikovanější než v případě ROS2 na RPI5.

Pro integraci skriptu a celkovou přípravu Pi Pico bylo potřeba postupovat následovně:

1. Vytvoření nového pracovního prostředí pro micro-ROS – nová složka např. s názvem *micro_ros_project*.
2. Instalace micro-ROS prostředí na PC – Je třeba následovat postup, který je dán popsán na GitHubu micro-ROS knihovny [31].
3. Tvorba nového programu *pico_micro_ros_example.c* v nově vytvořené složce (tento soubor je součástí přílohy 3).
4. Konfigurace pomocí CMake (*cmakelists.txt* -příloha 3) – příprava souborů pro build.
5. Kompilace projektu a vygenerování UF2 souboru:

```
cd micro_ros_project
mkdir build
cd build
cmake ..
make
```

6. Připojení Raspberry Pi Pico do PC v BOOTSEL módu – během připojování k PC pomocí micro USB je nutné držet tlačítko BOOTSEL.
7. Nahrání UF2 souboru do Pi Pico – Ve složce build je nyní nový soubor s názvem *pico_micro_ros_example.uf2,* který stačí přesunout do RPI Pico. To by se po přesunutí souboru do něj mělo automaticky odpojit od PC.
8. Spuštění micro-ROS klienta – Nejjednodušším způsobem je spuštění pomocí docker kontejneru. Do terminálu zadat následující příkaz a připojit Pi Pico.

```
docker run -it --rm -v /dev:/dev --privileged --net=host
microros/micro-ros-agent:jazzy serial --dev /dev/ttyACM0 -b
115200
```

9. Ověření funkčnosti – pomocí příkazů *ros2 topic list* a *ros2 node list* lze zkontrolovat, zda jsou správně viditelné nody a topicy publikované ze strany Pi Pico.
10. V novém terminálu spustit simulaci, kterou má Pi Pico následně řídit např. pomocí příkazu:

```
ros2 launch dp_package allinone.launch.py
```

11. V novém okně terminálu spustit skript *pico_support.py* následujícím způsobem:

```
python3 dp2025_ws/dp_package/src/scripts/pico_support.py
```

12. Start simulace v Gazebo Sim.

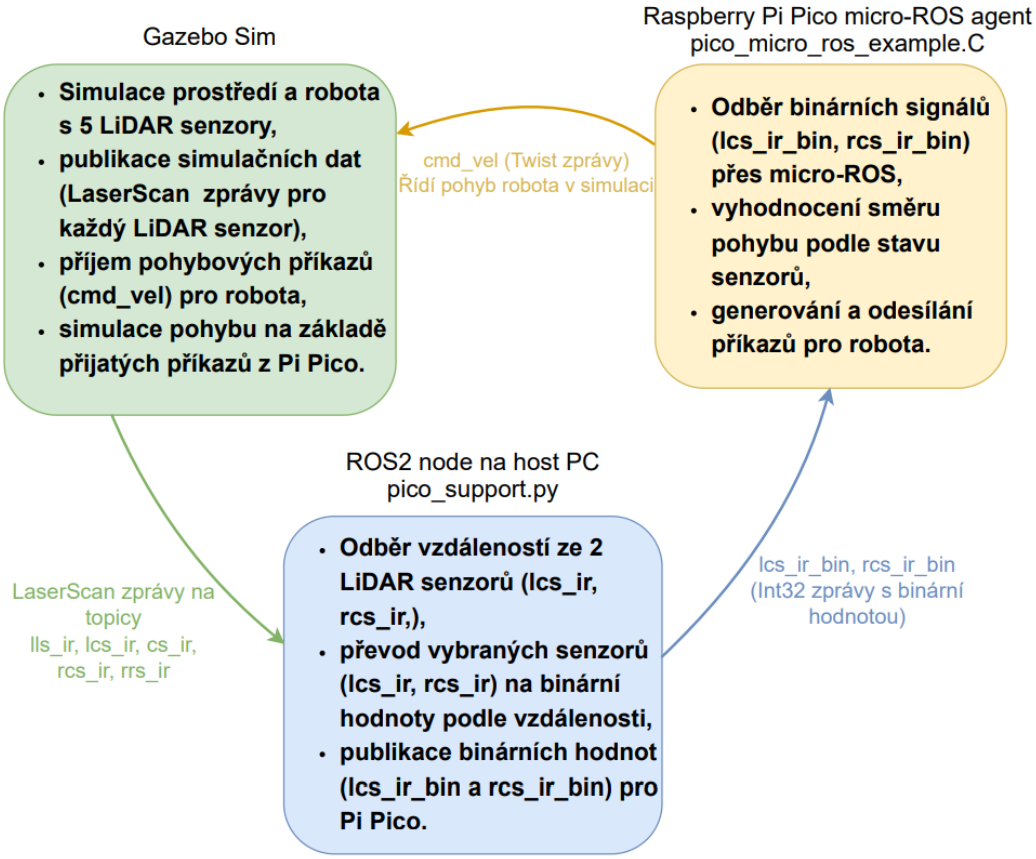
Po dokončení této konfigurace je Pi Pico plnohodnotnou součástí ROS2 systému jakožto micro-ROS agent a ovládá robot v Gazebo Sim.

Popis architektury systému

Architektura řešení je znázorněna na Obr. 34. Diagram zachycuje interakci mezi jednotlivými částmi systému:

- Gazebo simulace zajišťuje simulaci prostředí a robotu vybaveného IR (LiDAR) senzory. Simulátor publikuje data ze senzorů jako zprávy typu LaserScan, přijímá příkazy pohybu (cmd_vel topic) a simuluje chování robotu.
- ROS2 node (pico_support.py), běžící na hostitelském PC stejně jako Gazebo, odebírá data ze senzorů lcs_ir a rcs_ir a hodnoty převádí na binární data, která publikuje na topic lcs_ir_bin a rcs_ir_bin.
- Raspberry Pi Pico (micro-ROS agent) odebírá binární hodnoty z topicu lcs_ir_bin a rcs_ir_bin pomocí micro-ROS. Na jejich základě jednoduše vyhodnocuje směr pohybu a generuje odpovídající příkazy pro směr jízdy robotu, publikováním na cmd_vel topic.

C program nahraný na Raspberry Pi Pico je zjednodušenou verzí řídící logiky. Oproti původnímu python skriptu odebírá a rozhoduje směr pouze ze dvou senzorů a nemá PID regulaci. Na základě jednoduchých podmínek rozhoduje o směru pohybu robotu.



Obr. 34) Struktura projektu s Raspberry Pi Pico řízením

7.5 Tvorba Docker kontejneru pro projekt a simulaci

Pro potřeby tohoto projektu bylo celé prostředí ROS2 kontejnerizováno pomocí Dockeru. Jako základní image byl použit osrf/ros:jazzy-desktop, který obsahuje instalaci ROS2 Jazzy a potřebné nástroje pro vývoj.

Výsledný Dockerfile lze nalézt ve složce *dp2025_ws* v příloze 2. Tato složka slouží jako hlavní pracovní prostor (workspace) ROS2. Dockerfile provádí následující kroky:

- Instalace Gazebo Harmonic a ROS-GZ bridge – Během tvorby kontejneru je přidán repozitář OSRF a doinstalují se balíčky pro simulaci v Gazebo Sim ve verzi Harmonic (balíček *gz-harmonic*) a most (*ros-gz-bridge*) umožňující propojení ROS2 a Gazebo Sim. Poté jsou nainstalovány balíčky jako *nano, joint-state-publisher, joint-state-publisher-gui* a *rclpy* (ROS2 python knihovna).
- Natavení Fast DDS – Byl vytvořen soubor *fastdds.xml*, který definuje nastavení middleware pro komunikaci ROS2. Je umístěn v *dp2025_ws* složce a pro aplikaci na novém stroji je potřeba v něm přepsat lokální IP adresu daného stroje (pouze v případě, že je třeba komunikace mezi aplikací v Dockeru a mimo něj tzn. na příklad když RPI ovládá simulaci. V případě spuštění řídícího python skriptu i simulace v Dockeru není tento krok nutný). Bez tohoto nastavení by nebyla možná komunikace mezi aplikacemi běžícími na Dockeru a mimo Docker. Poté je nastavena proměnná, která přikazuje používat toto nastavení (*FASTRTPS_DEFAULT_PROFILES_SITE*).
- Překopírování zdrojových souborů – lokální složka *src* je překopírována do pracovního prostoru *docker_project/src* uvnitř Docker kontejneru.
- Sestavení ROS2 workspace – Celý pracovní prostor je sestaven pomocí nástroje colcon. Nejprve provede načtení prostředí (*source /opt/ros/jazzy/setup.bash*), aby byly dostupné všechny ROS2 funkce. V případě otevření druhého okna terminálu Docker kontejneru je příkaz nutné zavolat znovu manuálně.
- Entrypoint file – *Entrypoint.sh* (viz příloha 2 v *dp2025_ws*) byl zkopírován do root adresáře (/) kontejneru.
- Spouštění kontejneru – Kontejner je spouštěn vlastním *entrypoint.sh*, který je popsán níže.

Skript *entrypoint.sh* zajistí správně připravené prostředí ROS2. Provádí následující kroky:

1. Načtení ROS2 Jazzy prostředí – Spustí příkaz *source /opt/ros/jazzy/setup.bash,* díky kterému je možné v daném okně terminálu používat funkce ROS2.
2. Načtení lokálního pracovního prostoru – Provede *source install/setup.bash* příkaz, který umožní pracovat se všemi balíčky v rámci projektu.
3. Spuštění interactive shell – Skript spustí *exec "$@",* což umožní uživateli pracovat s ROS2 přímo v prostředí kontejneru.

Použití Dockeru – sestavení a spuštění kontejneru

Aby bylo připravené ROS2 prostředí na možné používat, je nutné jako první sestavit Docker image a spustit kontejner následovně (na Windows nemusí GUI fungovat, testováno na Ubuntu):

1. Sestavení Docker image – Nejprve je nutné přejít v terminálu do adresáře, kde se nachází daný Dockerfile, v tomto případě *dp2025_ws* a spustit sestavení image s názvem *docker_project*:

```
cd ~/dp2025_ws
docker build -t docker_project .
```

2. Povolení X11 komunikace – Nutné pro používání GUI na Ubuntu:

```
xhost +local:docker
```

3. Spuštění Docker kontejneru s GUI – Kontejner je možné spustit pomocí následujícího příkazu:

```
docker run -it --rm --net=host -e DISPLAY=$DISPLAY -e
QT_X11_NO_MITSHM=1 -v /tmp/.X11-unix:/tmp/.X11-unix
docker_project:latest
```

   Vysvětlení parametrů:

   - *-it* – Spouští kontejner v režimu s terminálem.
   - *--rm* – Kontejner bude po ukončení automaticky uzavřen.
   - *--net=host* – Umožňuje kontejneru používat síťové rozhraní host systému, což je pro komunikaci aplikace v Dockeru a mimo něj nezbytné.
   - *-e DISPLAY=$DISPLAY* – Aby kontejner věděl, kde zobrazovat grafická okna.
   - *-e QT_X11_-unix:/tmp/.X11-unix* – Umožňuje grafický přenos dat.

Po spuštění tohoto příkazu se uživatel nachází v připraveném ROS2 prostředí, ve kterém je nyní možné spouštět všechny připravené nody, simulace atd. bez dalších úprav. Při otevření nového okna terminálu kontejneru je nutné opět sourcovat.

8 ZHODNOCENÍ A DISKUZE DOSAŽENÝCH VÝSLEDKŮ

Původní motivací projektu bylo připravit didaktickou pomůcku, tedy systém, který by byl snadno implementovatelný mezi studenty, kteří by mohli jednoduše přenášet své CAD modely do prostředí ROS2. První volbou byl Autodesk Inventor, který studenti na VUT velmi dobře znají. Inventor bohužel neumožňuje přímý export modelu do URDF formátu. Alternativou byl Fusion 360, kam je možné sestavu z Inventoru nahrát a následně exportovat odtud. Sestavu přenesenou z Inventoru do Fusionu je však nezbytné znovu zavazbit a navíc není možné přenést souřadné systémy a osy jednotlivých dílů, které jsou pro ROS2 model naprosto nezbytné a bylo nutné je definovat přímo v URDF manuálně. To se ve výsledku ukázalo jako více práce než užitku, a proto byl zvolen pro export software SolidWorks (jehož licence VUT také studentům nabízí) s pluginem SW to URDF exporter, který funguje velmi dobře.

Jakmile byl model připraven, postoupilo se do fáze simulace. Původním plánem bylo použití klasických IR senzorů pro sledování čáry, avšak ty Gazebo Sim nenabízí. Bylo tedy nezbytné vyhledat alternativu. Nakonec bylo pro tento účel zvoleno 5 LiDAR senzorů, omezených na 1 paprsek (fungujících na stejném principu jako ultrazvukové senzory měřící vzdálenost) v kombinaci s 3D modelem čáry, kterou má robot sledovat. Naměřená vzdálenost se následně převáděla na binární hodnotu – pokud byla kratší než kalibrační práh (tj. čára se nachází přímo pod senzorem), interpretovala se jako logická 1. Větší vzdálenost signalizující, že pod robotem žádná čára není, odpovídala logické 0. Více je tento postup popsán v kapitole 4.2. V reálné aplikaci by bylo ovšem vhodnější tyto senzory nahradit opět IR čidly. S minimálním počtem úprav může být skript použit i v reálném nasazení s IR senzory.

Samotné prostředí ROS2 nejde popsat jako „user-friendly“ a první orientace může vyžadovat více času, zejména pokud uživatel nemá žádnou zkušenost s operačním systémem Ubuntu. Jakmile však uživatel pochopí princip nodů a topiců, modulární koncepce začne dávat smysl a umožní velmi snadno a rychle celý projekt škálovat a přidávat další funkce. Co se týče zatížení Raspberry Pi 5 při řízení simulace, zůstávalo na úrovni jednotek procent CPU a RAM a řídící skript by tedy mohl být implementován i na slabší mikrokontroler. Následně byl tedy skript ve zjednodušené formě přenesen na Raspberry Pi Pico s frameworkem micro-ROS. RPI Pico je cenově dostupné, takže by každý student mohl mít své vlastní a na něm psát svůj firmware, který by poté mohl otestovat na již připravené simulaci z této práce. Limitací ale je, že na RPI Pico, nelze použít klasický ROS2, ale pouze framework micro-ROS. Aplikace se tedy musí psát v jazyce C/C++ a implementace micro-ROS oproti ROS2 je značně komplikovanější a časově náročnější. Tvorba firmwaru na RPI Pico tedy vyžaduje pokročilejší programovací znalosti než varianta ROS2 v Pythonu na RPI 5 a jako variantu pro edukační účely ji tedy, alespoň pro méně zdatné programátory, nelze doporučit.

Na práci lze jednoduše navázat několika směry. Jedním z nich může být integrace kamerového systému, kdy lze nejen sledovat čáru pomocí počítačového vidění, ale i rozpoznávat QR kódy, překážky atd. Dále je možné rozšířit simulační prostředí a např. vytvořit scénář reálného automatizovaného skladu s více AGV, případně využít skriptů z této práce a přenést AGV do reálného světa. Navzdory uvedeným slabinám považuji dosažené výsledky za přínosné – umožňuje rychlé experimenty, ukazuje možnosti micro-ROS a výsledná Gazebo simulace je snadno rozšiřitelná a využitelná jako prostředí pro test nových ovládacích skriptů, bez nutnosti tvořit kompletní Gazebo simulaci včetně modelu od úplného začátku.

9 ZÁVĚR

Cílem této diplomové práce bylo navrhnout, implementovat a otestovat robotický systém pro sledování čáry s využitím robotického operačního systému ROS2. Výsledný systém propojuje simulační prostředí v Gazebo Sim s reálnými výpočetními jednotkami Raspberry Pi 5 a Raspberry Pi Pico. Robot je schopný sledovat čáru, reagovat na překážky a systém je díky ROS2 navržen modulárně a lze ho snadno rozšířit.

V práci byla nejprve provedena rešerše v oblasti existujících AGV, včetně porovnání jejich silných a slabých stránek. Poté byl navržen model robotu s diferenciálním pohonem, přičemž hlavním důvodem byla výborná kompatibilita Gazebo Sim a ROS2 s roboty s diferenciálním řízením. Model byl vytvořen v softwaru SolidWorks a následně byl odtud za pomoci pluginu SolidWorks to URDF exporter exportován do URDF formátu, který je pro práci v ROS2 nezbytný. Celý postup exportu je zde detailně popsán, včetně definice všech senzorů, geometrie a souřadných systémů a byl představen i způsob pro zobrazení modelu ve VS Code, což se jeví jako výborný nástroj pro úpravu detailů modelu robotu.

Práce následně detailně popisuje principy robotického operačního systému ROS2 a poté realizaci řešení s využitím vývojových desek Raspberry Pi 5 a Raspberry Pi Pico. Tyto desky slouží v rámci této práce jako řídící jednotky, které ovládají robot přímo v simulaci Gazebo Sim – komunikace se simulací probíhá po síti a na základě dat ze senzorů generují pohybové příkazy pro robot. Řídící algoritmus pro sledování čáry byl do systému implementován jako ROS2 node v jazyce Python a následně byl ve zjednodušené formě přenesen do jazyka C, pro využití na Raspberry Pi Pico s micro-ROS framework. Součástí řešení je také podrobný návod, jak ulehčit implementaci a sdílení celého ROS2 projektu pomocí vytvoření Docker kontejneru.

Celé řešení bylo řádně odsimulováno a ukázalo se, že je plně funkční, stabilní a vhodné i pro použití na slabších deskách, než je právě Raspberry Pi 5. Práce tedy naplnila všechny stanovené cíle od rešerše a návrhu až po testování a zhodnocení výsledků.

Byla tedy vytvořena plně funkční platforma, která se hodí i pro výuku moderní robotiky, kterou lze využít jako aplikaci založenou na Gazebo simulaci s připraveným robotem a scénářem. Celý systém je navržen tak, aby bylo možné jednoduše upravit řídící kód (např. ladit PID regulaci, měnit chování robotu, pokud se přiblíží k překážce, implementovat nové funkce jako je kamera atd.), bez toho, aby bylo nutné zasahovat do samotné simulace nebo měnit model robotu. Celý systém je plně připraven a nakonfigurován (robot se spawnuje na začátku trasy, senzory jsou připojeny, bridge je spuštěn atd.) a lze ho spustit pomocí jediného souboru. Je tedy možné sledovat změnu chování robotu už při jednoduché změně kódu.

10 SEZNAM POUŽITÝCH ZDROJŮ

[1] Fundamentals of Automated Guided Vehicles. *CYNGN* [online]. 5 Mar 2024 [cit. 2025-05-11]. Dostupné z: https://www.cyngn.com/blog/fundamentals-of-automated-guided-vehicles

[2] AGV vs. AMR. *AGV Network* [online]. 2023 [cit. 2025-05-11]. Dostupné z: https://www.agvnetwork.com/agv-vs-amr

[3] AGV Navigation Methods 2: Virtual Path Following. *BlueBotics* [online]. 2025 [cit. 2025-05-11]. Dostupné z: https://bluebotics.com/agv-navigation-methods-virtual-path-following/

[4] KIM, DaeEun. *Advanced Mobile Robotics: Volume 1*. MDPI - Multidisciplinary Digital Publishing Institute, 2020, 468 s. ISBN 3039219162. Dostupné z: doi:10.3390/books978-3-03921-917-9

[5] Line Follower Robot: Techniques and Technologies. *Into Robotics* [online]. 2023 [cit. 2025-01-23]. Dostupné z: https://intorobotics.com/line-follower-robot-techniques-and-technologies/

[6] *Magnetické pásky pro AGV vozíky: jak vám urychlí dopravu?* [online]. 2025, 27.3.2025 [cit. 2025-05-16]. Dostupné z: https://www.unimagnet.cz/clanek/735/magneticke-pasky-pro-agv-voziky-jak-vam-urychli-dopravu/

[7] Building a Magnetic Track Guided AGV. *RoboteQ* [online]. c2020 [cit. 2025-05-16]. Dostupné z: https://www.roboteq.com/applications/all-blogs/18-building-a-magnetic-track-guided-agv

[8] Introduction Inductive Track Guidance. *Götting KG* [online]. c1995-2025 [cit. 2025-05-16]. Dostupné z: https://www.goetting-agv.com/components/inductive/introduction

[9] Camera-Based Track Guidance (PGV). *Pepperl+Fuchs* [online]. c2025 [cit. 2025-05-16]. Dostupné z: https://www.pepperl-fuchs.com/en/products/industrial-sensors/positioning-systems/camera-based-track-guidance-pgv-gp32265

[10] PUPPIM DE OLIVEIRA, Diogo, Wallace PEREIRA NEVES DOS REIS a Orides MORANDIN JUNIOR. A Qualitative Analysis of a USB Camera for AGV Control. *Sensors (Basel, Switzerland)* [online]. Basel: MDPI, 2019, **19**(19), 4111 [cit. 2025-05-16]. ISSN 1424-8220. Dostupné z: doi:10.3390/s19194111

[11] ROY, Abhishek a Mathew Mithra NOEL. Design of a high-speed line following robot that smoothly follows tight curves. *Computers & electrical engineering* [online]. OXFORD: Elsevier, 2016, **56**, 732-747 [cit. 2025-01-23]. ISSN 0045-7906. Dostupné z: doi:10.1016/j.compeleceng.2015.06.014

[12] PRAKASH, M. Surya, K. Ajay VIGNESH, J. SHYAMSUNTHAR, K. RAMAN, J. Senthil RAJU a N. RAJU. Computer Vision Assisted Line Following Robot. *Procedia engineering* [online]. Elsevier, 2012, **38**, 1764-1772 [cit. 2025-05-11]. ISSN 1877-7058. Dostupné z: doi:10.1016/j.proeng.2012.06.215

[13] What is ROS? *The Robotics Back-End* [online]. 2025 [cit. 2025-05-11]. Dostupné z: https://roboticsbackend.com/what-is-ros/

[14] *ROS2 Documentation* [online]. [cit. 2024-10-23]. Dostupné z: wiki.ros.org

[15] SUBRAMANIAN, Rajesh. *Build autonomous mobile robot from Scratch using ROS: simulation and hardware*. New York: Apress, 2023, xxii, 563 stran : ilustrace ; 24 cm. ISBN 978-1-4842-9644-8.

[16] Explore ROS Topics: Publisher and Subscriber Guide. In: *Mathworks* [online]. 2025 [cit. 2025-05-11]. Dostupné z: https://www.mathworks.com/help/ros/gs/ros-topics.html

[17] Actions. In: *ROS 2 Design* [online]. May 2020 [cit. 2025-05-11]. Dostupné z: https://design.ros2.org/articles/actions.html

[18] TOLA, Daniella a Peter CORKE. Understanding URDF: A Dataset and Analysis. *IEEE robotics and automation letters* [online]. Piscataway: IEEE, 2024, **9**(5), 4479-4486 [cit. 2025-05-04]. ISSN 2377-3766. Dostupné z: doi:10.1109/LRA.2024.3381482

[19] Describing robots with URDF. *Articulated Robotics* [online]. 2021 [cit. 2025-01-23]. Dostupné z: https://articulatedrobotics.xyz/tutorials/ready-for-ros/urdf/

[20] KOUBAA, Anis. *Robot Operating System (ROS): The Complete Reference*. Volume 7. Netherlands: Springer Nature, 2017. ISBN 3319549278. Dostupné z: doi:10.1007/978-3-319-91590-6

[21] Create and Visualize a Robotic Arm with URDF – ROS 2 Jazzy. In: *Automatic Addison* [online]. 8 Nov 2024 [cit. 2025-05-11]. Dostupné z: https://automaticaddison.com/create-and-visualize-a-robotic-arm-with-urdf-ros-2-jazzy/

[22] RIVERA, Zandra B., Marco C. DE SIMONE a Domenico GUIDA. Unmanned ground vehicle modelling in Gazebo/ROS-based environments. *Machines (Basel)* [online]. BASEL: Mdpi, 2019, **7**(2), 42 [cit. 2025-01-20]. ISSN 2075-1702. Dostupné z: doi:10.3390/machines7020042

[23] *Gazebo Sim* [online]. 2025 [cit. 2025-01-20]. Dostupné z: https://gazebosim.org/home

[24] MARIAN, Marius, Florin STÎNGĂ, Marian-Terxinius GEORGESCU, Horațiu ROIBU, Dorin POPESCU a Florin MANTA. *A ROS-based Control Application for a Robotic Platform Using the Gazebo 3D Simulator*. Slovakia: 2020 21th International Carpathian Control Conference (ICCC), 2020. ISBN 978-1-7281-1951-9.

[25] PADALKAR, Pratik, Pawan KADAM, Shantanu MIRAJGAVE, et al. Industrial Warehouse Robot Simulation Using ROS. In: *Artificial Intelligence*. 1695. Switzerland: Springer, 2023, s. 87-95. ISBN 9783031224843. ISSN 1865-0929. Dostupné z: doi:10.1007/978-3-031-22485-0_9

[26] ANDERSON, Charles. Docker [Software engineering]. *IEEE software* [online]. LOS ALAMITOS: IEEE, 2015, **32**(3), 102-c3 [cit. 2025-04-27]. ISSN 0740-7459. Dostupné z: doi:10.1109/MS.2015.62

[27] BOETTIGER, Carl. An introduction to Docker for reproducible research. *Operating systems review* [online]. 2015, **49**(1), 71-79 [cit. 2025-05-11]. ISSN 0163-5980. Dostupné z: doi:10.1145/2723872.2723882

[28] Docker Workflow. *Medium* [online]. 28 Jun 2023 [cit. 2025-05-11]. Dostupné z: https://medium.com/@augustineozor/docker-workflow-b9fe71d32184

[29] RABIEE, Sadegh a Joydeep BISWAS. A Friction-Based Kinematic Model for Skid-Steer Wheeled Mobile Robots. In: *2019 International Conference on Robotics and Automation (ICRA)* [online]. Montreal, QC, Canada: IEEE, 2019, s. 8563-8569 [cit. 2025-04-20]. ISBN 1538660261. ISSN 1050-4729. Dostupné z: doi:10.1109/ICRA.2019.8794216

[30] ROS Architecture and Concepts. *Packt* [online]. 2016 [cit. 2025-01-20]. Dostupné z: https://www.packtpub.com/en-us/learning/how-to-tutorials/ros-architecture-and-concepts?srsltid=AfmBOorDJs-fh0jVb6v-71EDkgTQXM36FeX9ZeI6yUMGta_bqnEM9Mao

[31] Micro-ROS module for Raspberry Pi Pico SDK. *GitHub* [online]. 2021 [cit. 2025-05-20]. Dostupné z: https://github.com/micro-ROS/micro_ros_raspberrypi_pico_sdk

[32] MODEX 2024 – Come and see MasterMover at Booth C2893. In: *MasterMover* [online]. 2024 [cit. 2025-05-11]. Dostupné z: https://www.mastermover.com/en-us/blog/modex-2024-come-and-see-mastermover-at-booth-c2893

[33] AMRs vs. AGVs: The Differences Explained. In: *Conger* [online]. 2025 [cit. 2025-05-11]. Dostupné z: https://www.conger.com/amr-vs-agv/

[34] IR Sensor Working and Applications. In: *ROBU* [online]. 19 May 2020 [cit. 2025-05-11]. Dostupné z: https://robu.in/ir-sensor-working/

[35] How to use LDR Sensor Module with Arduino. In: *How To Electronics* [online]. 2 Feb 2025 [cit. 2025-05-11]. Dostupné z: https://how2electronics.com/how-to-use-ldr-sensor-module-with-arduino/

[36] 5 Wheeled robot steering mechanisms. In: *Kshitij Tiwari* [online]. 2025 [cit. 2025-05-11]. Dostupné z: https://kshitijtiwari.com/all-resources/mobile-robots/robot-steering/#differential-steering

11 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

11.1 Seznam zkratek

AGV	Automatic Guided Vehicle
AMR	Automatic Mobile Robot
IR	Infra Red
LED	Light Emitting Diode
LDR	Light Dependent Resistor
PWM	Pulse Width Modulation
ROS	Robot Operating System
URDF	Unified Robot Description Format
SDF	Simulation Description Format
API	Application Programming Interface
RGB	Red, Green, Blue
LiDAR	Light Detection and Ranging
XML	eXtensible Markup Language
XACRO	XML Macro Language
RFID	Radio Frequency Identification

11.2 Seznam obrázků

Obr. 1) MasterMover AGV [32] 19
Obr. 2) Rozdíly mezi AGV a AMR [33] 20
Obr. 3) Magnetický senzor Roboteq MGS1600 [7] 22
Obr. 4) Indukční senzor HG G-19370 [8] 22
Obr. 5) Kamery a pásky pro PGV od Pepperl+Fuchs [9] 23
Obr. 6) IR senzor [34] 23
Obr. 7) LDR senzor [35] 24
Obr. 8) Schéma komunikace mezi nody přes topic /example [16] 26
Obr. 9) Schéma akce [17] 26
Obr. 10) Podpora formátů různými softwary pro robotiku [18] 27
Obr. 11) Linky a jointy na robotickém ramenu [21] 28
Obr. 12) Prostředí RViz [25] 29
Obr. 13) Schéma Docker workflow [28] 30
Obr. 14) Differential steering princip [36] 31
Obr. 15) Skid-steering princip [36] 32
Obr. 16) Princip náhrady IR senzorů v Gazebo Sim 34
Obr. 17) Schéma ROS2 balíčku [14] 36

Obr. 18) Sestava robotu 38
Obr. 19) Sestava s definovanými souřadnými systémy a osami 39
Obr. 20) Proces nastavení URDF exportu 40
Obr. 21) Změna kolize pojezdového kola zobrazená v RViz 41
Obr. 22) Struktura URDF souboru 42
Obr. 23) RViz a joint_state_publisher_gui slider 45
Obr. 24 Zobrazení modelu ve VS Code 46
Obr. 25) Gazebo Sim a výběr modelu z Fuel 50
Obr. 26) Test LiDAR senzorů v Gazebo Sim 52
Obr. 27) RViz vizualizace outputu z LiDAR senzorů 52
Obr. 28) Vývojový diagram logiky sledování čáry 54
Obr. 29) Robot sledující čáru 55
Obr. 30) Robot stojí a LED svítí červeně z důvodu bezpečnosti 55
Obr. 31) LIDAR detekující kvádr v blízkosti robotu a žlutá LED 56
Obr. 32) Výstup z rqt_graph 57
Obr. 33) Schéma fungování Raspberry Pi s ROS2 a Gazebo 58
Obr. 34) Struktura projektu s Raspberry Pi Pico řízením 60

11.3 Seznam tabulek

Tab. 1) Rozdíly mezi AGV a AMR [2] 20
Tab. 2) Vlastnosti navigačních metod [2; 3; 4] 21
Tab. 3) Vlastnosti verze s diferenciálním pohonem a skid-steer pohonem [29] 31

12 SEZNAM PŘÍLOH

Příloha 1: Vývojový diagram skriptu safety_pid.py – vyvojovy_diagram_safety.pdf

Příloha 2: ROS2 workspace – složka dp2025_ws

Příloha 3: Soubory pro implementaci RPI Pico s micro-ROS – složka micro_ros_files

Příloha 4: Video ze simulace - line_follow_video.mp4