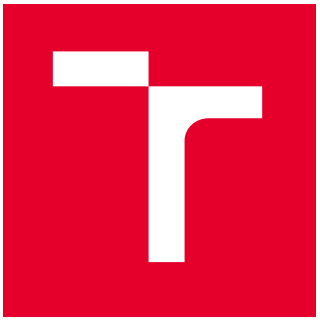


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY

INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

MOBILNÍ ROBOTICKÁ PLATFORMA TURTLEBOT3 BURGER

TURTLEBOT3 BURGER MOBILE ROBOTIC PLATFORM

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — **Petr Lisník**

AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Holoubek**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Petr Lisník**
Studijní program:	Strojírenství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Tomáš Holoubek**
Akademický rok:	2022/23

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 bakalářské práce:

Mobilní robotická platforma Turtlebot3 Burger

Stručná charakteristika problematiky úkolu:

Práce bude obsahovat rešerši mobilních robotů, seznámení a popis funkcí modelu robota TurtleBot3 Burger od společnosti Robotis.
Teoretická část se bude zabývat hlavně frameworkem ROS – Robotický Operační Systém – ve spojení se simulačním prostředím Gazebo.
Praktická část bude zahrnovat sestavení robota ze stavebnice, jeho zprovoznění v ROS a návrh řídícího programu
pro vybranou laboratorní úlohu. Závěrem bude funkčnost programového řešení otestována.
Práce předpokládá práci v laboratoři.

Cíle bakalářské práce:

– Vypracujte rešerši na téma mobilních robotů, aktuální stav a možnosti.
– Popište model robota TurtleBot3 Burger od společnosti Robotis.
– Nastudujte framework ROS ve spojení se simulačním prostředím Gazebo.
– Zprovozněte robota, implementujte vlastní návrh pro zadanou laboratorní úlohu a otestujte funkčnost.

Seznam doporučené literatury:

KOLÍBAL, Z. Roboty a robotizované výrobní technologie. Brno: Vysoké učení technické v Brně - nakladatelství VUTIUM, 2016. ISBN 978-80-214-4828-5.

THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. Probabilistic robotics: (Intelligent Robotics and Autonomous Agents series) [online]. Cambridge, Massachusetts: MIT Press, [2006] [cit. 2022-10-20]. ISBN 978-0262201629.

ROS.org. ROS.org | Powering the world's robots. [online]. 2.11.2016 [cit. 2016-11-02]. Dostupné z:

http://www.ros.org/.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2022/23

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jiří Hlinka, Ph.D. děkan fakulty

ABSTRAKT

Tato bakalářská práce se zaměřuje na návrh a implementaci řídícího programu pro mobilní robotickou platformu TurtleBot3 Burger. Úvodní část práce se věnuje rešerši mobilní robotiky, popisu mobilní robotické platformy a Robotického Operačního Systému (ROS) včetně jeho nástrojů a simulačního prostředí Gazebo. Praktická část obsahuje postupné kroky při návrhu řídícího programu včetně sestavení a zprovoznění robota, návrhu řídícího programu s využitím počítačového vidění, testování v simulačním prostředí Gazebo a následné otestování na reálném robotu v laboratoři.

ABSTRACT

This bachelor's thesis focuses on the design and implementation of a control program for the mobile robotic platformTurtleBot3 Burger. The theoretical part of the thesis is devoted to the research of mobile robotics, the description of the mobile robotic platform and the Robotic Operating System (ROS) including its tools and the Gazebo simulation environment. The practical part includes the step-by-step design of the control program, including the build and commissioning of the robot, the design of the control program using computer vision, testing in the Gazebo simulation environment, and subsequent testing on a real robot in the laboratory.

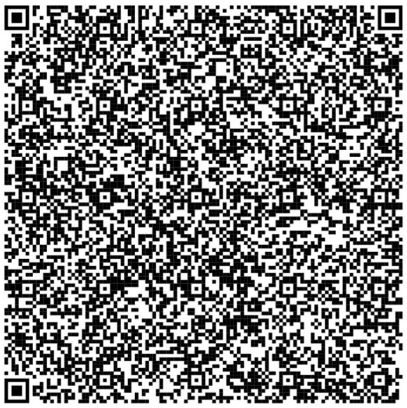
KLÍČOVÁ SLOVA

Mobilní robotika, TurtleBot3 Burger, ROS, operační systém pro roboty, Gazebo, OpenCV, počítačové vidění, zpracování obrazu, detekce čáry, LIDAR

KEYWORDS

Mobile robotics, TurtleBot3 Burger, ROS, operating system for robots, Gazebo, OpenCV, computer vision, image processing, line detection, LIDAR

ÚSTAV AUTOMATIZACE A INFORMATIKY



2023

BIBLIOGRAFICKÁ CITACE

LISNÍK, Petr. *Mobilní robotická platforma Turtlebot3 Burger*. Brno, 2023. Dostupné také z: https://www.vut.cz/studenti/zav-prace/detail/149502. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Tomáš Holoubek.

PODĚKOVÁNÍ

Chtěl bych poděkovat svému vedoucímu práce, panu Ing. Tomáši Holoubkovi, za jeho vstřícnost, ochotu a odborné vedení při psaní této bakalářské práce. Dále bych chtěl vyjádřit hlubokou vděčnost své rodině za neustálou podporu a motivaci po celou dobu mého studia. Nesmím zapomenout také poděkovat všem přátelům a kolegům, kteří se mnou sdíleli své znalosti a zkušenosti v průběhu studia.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, vypracoval jsem ji samostatně pod vedením vedoucího práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury.

Jako autor uvedené práce dále prohlašuji, že v souvislosti s vytvořením této práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následku porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestně právních důsledků.

V Brně dne 24. 5. 2023 ……

Petr Lisník

OBSAH

1 ÚVOD

Bakalářská práce se zaměřuje na návrh řídícího programu pro mobilní robotickou platformu TurtleBot3 Burger, jeho implementaci a ověření funkčnosti na zadané laboratorní úloze.

V úvodní část části práce byla provedena rešerše mobilní robotiky, kde jsou zmíněny používané technologie a různé aplikace mobilních robotických zařízení. Následuje popis mobilní robotické platformy TurtleBot3 Burger od společnosti ROBOTIS. Při řízení tohoto mobilního robota se využívá Robotický Operační Systém (ROS). Právě s tímto systémem bude čtenář dále seznámen. Budou objasněny základní pojmy a principy, jak celý systém funguje a možnosti, které nabízí. Součástí systému ROS je obrovské množství užitečných nástrojů, ze kterých bude přiblíženo simulační prostředí Gazebo.

Praktická část se skládá z několika navazujících, a zároveň prolínajících se kroků, které je třeba provést při návrhu řídícího programu pro mobilního robota TurtleBot3 Burger.

Prvním krokem je sestavení a zprovoznění robota, kontrola jednotlivých komponent a instalace potřebných softwarových prostředků. Tyto prostředky zahrnují operační systém robota včetně doplňujících knihoven použitých při programování.

Následuje samotný návrh řídícího programu, který vychází z toho, co mobilní robot poskytuje. Jednou z možností, které robot nabízí, je využití kamery při řešení zadané laboratorní úlohy. Proto je součástí návrhu řídícího programu také zpracování obrazu pomocí knihovny *OpenCV*. V průběhu tvorby řídícího programu je k testování využíváno simulační prostředí Gazebo.

Závěrem jsou výsledné schopnosti robota ze simulačního programu otestovány a porovnány s jeho chováním v reálném prostředí laboratoře.

2 MOBILNÍ ROBOTIKA

Mobilní robotika je oblast výzkumu zabývající se řízením mobilních robotů. Pod pojmem robot si můžeme představit téměř cokoliv, avšak ve většině případů se jedná o stroj, manipulátor nebo zařízení, které vykonává určitou činnost, pro kterou je sestrojeno a naprogramováno. V rámci této činnosti může, ale také nemusí, do určité míry vnímat své okolí pomocí různých senzorů a interagovat s ním. Mobilní robot má navíc schopnost přesouvat se v prostoru. Pohyb nebo obecně činnost robota může být různé povahy. Pokud je pohyb plně ovládán člověkem, například na dálkové ovládání, je označován za teleoperovaného mobilního robota. Opačným případem je úplná autonomie, kdy je robot schopen pohybu nebo činnosti bez lidského operátora a chová se podle toho, jak byl naprogramován. V případě, že pouze některé činnosti jsou autonomní, je robot označován za semiautonomního. [1, 2]

Základní problém, který mobilní robotika řeší, je spojen s lokalizací v prostoru, snímání okolního prostoru a pohybu v daném prostoru. Jaká je aktuální poloha mobilního robota nebo kam se má dostat? To lze samozřejmě zjistit několika způsoby. Jednou z možností je využít lokalizace prostřednictvím GPS, ale to nemusí být vždy vhodné a dostatečně přesné. V rámci uzavřených prostor je možné rozmístit a využívat aktivní prvky, tzv. majáky, vysílající světelné nebo ultrazvukové signály. Vzhledem k jednotlivým majákům se poté určuje poloha robota. Další častou možností je využití mapy. Jenže mapa nemusí být vždy k dispozici a v případě změny prostředí by bylo nutné mapu vždy měnit. Tento problém řeší současná lokalizace a mapování (SLAM). Robot pomocí senzorů vnímá prostředí a ukládá si data, pomocí kterých si poté vytváří vlastní mapu prostředí, kde se právě nachází. Postupně projíždí celou oblast a mapu rozšiřuje. Dále následuje samotný pohyb robota a určení trasy při pohybu. Což je další odvětví mobilní robotiky, kdy se hledá nejkratší nebo rychlejší trasa pomocí různých algoritmů. [1–4]

2.1 Rozdělení mobilních robotů

Mobilní roboty lze dělit do velkého počtu různých skupin podle vlastností a parametrů. Zaměříme se především na roboty pohybující se po zemi, neboť roboty schopné létat nebo se pohybovat pod vodou jsou další velkou samostatnou skupinou. Jedno ze běžných rozdělení je z hlediska lokomoce neboli způsobu pohybu robota, což je základní vlastnost mobilních robotů. Největší skupiny robotů jsou následující:

2.1.1 Kráčející roboty

Kráčející roboty jsou významnou skupinou v oblasti robotiky, která se inspirovala pohybovým ústrojím savců a hmyzu. Existuje mnoho různých konstrukčních variant dvou až osminohých robotů, které lze hodnotit z hlediska různých parametrů. Mezi tyto

parametry patří stabilita, pohybové schopnosti, přizpůsobitelnost prostředí, způsob pohybu v prostoru, ovladatelnost a energetická nezávislost.

Biologicky inspirované mechanismy kráčejících robotů přinášejí výhody i nevýhody. Mezi výhody patří schopnost překonávat vysoké překážky, pohyb po náročném terénu a schopnost zdolávat terény, které jsou pro kolové nebo pásové roboty nepřekonatelné. Nevýhody zahrnují vyšší počet řízených stupňů volnosti, potřebu podrobného vidění terénu, složitější řídící systém a větší konstrukční složitost.

Kráčející roboty nabízejí široké využití, zejména v prostředích, která jsou pro člověka nebezpečná nebo nedostupná pro jiné typy robotů. Přesto je stále otázkou, zda je vývoj těchto systémů motivován lidskou zvědavostí nebo mají skutečný praktický význam (viz obr. 1). [1, 5–7]

Obr. 1: Čtyřnohý robot SPOT CLASSIC od společnosti Boston Dynamics [8]

2.1.2 Pásové mobilní roboty

Koncepce pásového podvozku vychází z různých konstrukcí používaných ve vojenských, zemědělských, stavebních a sněžných vozidlech. V oblasti mobilní robotiky se pásový podvozek využívá především v náročných aplikacích, jako jsou zbraňové systémy, monitorování, protiteroristické operace a manipulační systémy. Jejich použití je častější v nestrojírenských oblastech.

Samotný pásový modul je nezávislou funkční skupinou, která tvoří pohybový mechanismus mobilního robota. Skládá se z pásu (část ve styku s povrchem), hnacího kola (pohání robot), hnaného kola (poskytuje mechanickou podporu), vodicího kola a napínacího kola. Tyto prvky společně tvoří soustavu pojezdových kol a umožňují pásovému podvozku měnit geometrii v závislosti na terénu, který je aktuálně překonáván.

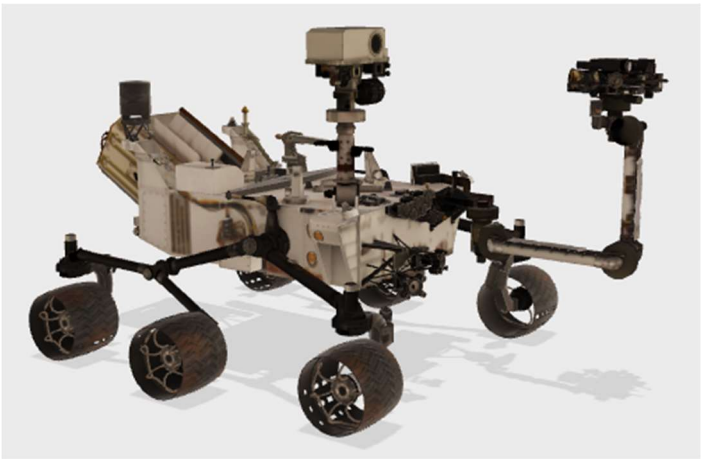
Díky velkému množství kontaktních bodů s povrchem poskytuje celý mechanismus pásového podvozku zvýšenou manévrovatelnost v členitých terénech. To je způsobeno vysokým třením mezi pásy a povrchem, které umožňuje řízení skluzem/smykem. Tento způsob řízení se liší od kolových podvozků, kde je požadováno, aby kola neklouzala po povrchu. Skluz/smyk je tedy alternativním způsobem řízení, při kterém se robot přeorientuje v závislosti na rychlosti a směru otáčení hnaných kol. [1, 5–7]

Obr. 2: Mobilní robot Centaur využíván americkou armádou pro dálkové monitorování, průzkum, odstraňování trosek, vyklízení tras a likvidaci bomb [9]

2.1.3 Kolové mobilní roboty

Kolový podvozek je nejvíce využívaným typem pohybového mechanismu u mobilních robotů, díky kombinaci jednoduché mechanické konstrukce a různorodosti možných variant (různé rozložení, počet a typ kol). Tato koncepce nabízí mnoho využití v široké škále robotických aplikací.

Při návrhu kolového podvozku je důležité zohlednit požadavky spojené s jeho provozem. Především možné kritické situace týkající se mechanických a fyzikálních vlastností, pohybu, stability a interakce podvozku s povrchem. Pro dosažení stability je nezbytné, aby všechna kola byla v kontaktu s podložkou, dokonce i na nerovném terénu. Tím je zajištěna dostatečná trakce, manévrovatelnost a ovladatelnost. [1, 5–7]



Obr. 3: 3D Model Curiosity Rover, mobilní robot určený ke zkoumání povrchu planety Mars v rámci mise Mars Science Laboratory NASA [10]

Na základě konstrukce a uspořádání kolového podvozku se kolové mobilní roboty dále dělí podle způsobu řízení.

Diferenciální pohon

Diferenciální princip řízení mobilních robotů spočívá v pohánění dvou nezávislých kol a jednoho nebo více nepoháněných otočných kol. Tímto způsobem dosahuje vysoké manévrovatelnosti a rotace kolem osy mezi hnacími koly. Tento typ řízení je ale citlivý na nerovnosti povrchu a relativní rychlost obou poháněných kol, což může vést k neočekávaným trajektoriím pohybu. [1, 5]

Synchronní pohon

Robot se synchronním pohonem umožňuje, aby každé kolo mělo vlastní pohon i řízení. Obvyklé uspořádání zahrnuje tři kola umístěná ve vrcholech rovnostranného trojúhelníku. Tato varianta využívá dva motory – jeden pro synchronní nastavení rychlosti a druhý pro synchronní rotaci kol. Všechny kola směřují vždy stejným směrem a otáčejí se stejnou rychlostí, čehož je dosaženo pomocí relativně složité sestavy řemenů spojujících kola.

Tento typ řízení umožňuje jednoduché řízení polohy s vysokou manévrovatelností a schopnost změny směru rotace kolem osy robota mezi hnacími koly. Je vhodný především do vnitřních prostor z důvodu vysokých nároků na kvalitu povrchu bez nerovností. [1, 5–7]

Ackermannovo řízení (řízení automobilu)

Ackermannovo řízení je mechanismus používaný převážně u automobilů pro zajištění optimálního otáčení při zatáčení. Jeho princip spočívá ve specifickém úhlu natočení kol, který umožňuje diferenciaci vnějšího a vnitřního kola při zatáčení, což zajišťuje plynulé a stabilní manévrování.

Při zatáčení musí mít vnější kolo větší úhel natočení než vnitřní kolo, aby se obě kola pohybovala po oblouku stejného poloměru. Tím se minimalizuje odpor vznikající při zatáčení a snižuje se opotřebení pneumatik. V případě rovné jízdy jsou všechna kola natočena ve stejném úhlu, což umožňuje přímý pohyb. [1, 5, 7]

Všesměrové řízení

Všesměrové řízení využívá technologie všesměrových kol, které umožňují robotům dosáhnout úplné manévrovatelnosti a pohybu ve všech směrech. Oproti jednoduchým kolům poskytují větší kinematickou svobodu. Příkladem jsou všesměrová složená kola, která se skládají z ráfku a valivých soudečkovitých elementů, umístěných pod úhlem 45 stupňů vůči náboji kola. Každé kolo je poháněno samostatným motorem. Změnou otáčení a relativních rychlostí kol se robot může pohybovat libovolně a otáčet se kolem osy.

Všesměrová kola mají několik výhod. Jednou z nich je snadné ovládání, protože pohyb robotu je nezávislý na jeho poloze. Dále mají schopnost okamžité změny směru, vysokou dynamičnost jízdy a dobrou manévrovatelnost v omezených prostorech. Nevýhodou všesměrových kol je potřeba přesné výroby a montáže. Je důležité, aby tato kola byla precizně vyrobená, protože jejich provedení má významný vliv na chování robota. Celkově lze uvést, že všesměrová kola jsou inovativní technologií, která umožňuje robotům pohyb ve všech směrech s vysokou manévrovatelností. [1, 5–7]

Obr. 4: Mobilní robot Vector s technologií všesměrových kol [11]

2.2 Využití v praxi

Mobilní roboty jsou dnes hojně využívány v různých odvětvích, nejčastěji v průmyslu k transportu materiálu v rámci výrobního závodu. Tyto roboty (vozíky) jsou nejčastěji označováni zkratkami AGV nebo AMR. Automaticky naváděná vozidla (AGV – automated guided vehicle) nejsou plně autonomní a využívají fixního trasování. Jezdí pouze v předem vyznačených trasách určených pomocí vodičů nebo magnetických pásků umístěných do podlahy výrobního závodu. V případě zablokování trasy vozík čeká, dokud se trasa neuvolní. Druhou variantou jsou autonomní mobilní roboty (AMR – autonomous mobile robot), které jsou již plně autonomní. Jejich nasazení se stává čím dál běžnější s nástupem automatizace a průmyslu 4.0. Využití nacházejí mobilní roboty také v kosmickém průmyslu (mobilní robot na povrchu planety Mars, viz obr. 3), v armádě (zneškodňování nálož, viz obr. 2) a v mnoha dalších oblastech. V poslední době se velmi rozvíjí zábavní sortiment, neboť děti jsou schopné si s roboty hrát už od útlého věku. [2, 7, 12]

Obr. 5: Autonomní mobilní robot (AMR) LD-60/90 od společnosti OMRON [13]

Mobilní robot TurtleBot3 Burger není průmyslovým, kosmickým ani armádním robotem, ale lze jej zařadit do kategorie malých mobilních robotů pro vědecké a vzdělávací účely. Zmiňovat tady další zástupce postrádá smysl, protože v dnešní době jich existuje obrovské množství. Některé podobné modely jsou dokonce schopné nést na sobě malé robotické rameno. Na internetu také existují relativně jednoduché tutoriály, jak si malého robota můžete postavit sami doma. Dnes je již také na některých základních a středních školách běžné vyučovat volitelný předmět mobilní roboty. V nich se žáci mohou s podobnými roboty setkat, případně si i vlastního postavit a zúčastnit se různých soutěží. [7]

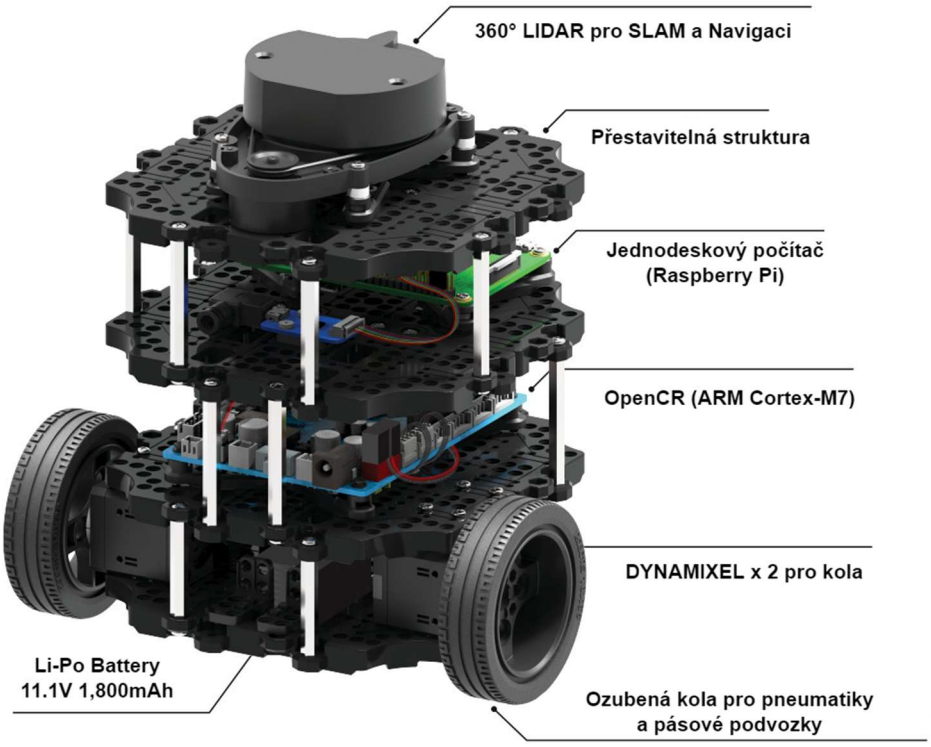
Co se týče vědeckých účelů, tak jsou tyto malé mobilní roboty používáni často v tzv. multirobotických systémech. V rámci těchto systémů se zabývají komunikací, koordinací a kooperací více robotů v jednom systému. Komunikace je nezbytnou součástí a často se využívá bezdrátového rádiového spojení prostřednictvím wifi nebo bluetooth, případně jiných technologií. Koordinace v multirobotických systémech se projevuje při vzájemné interakci robotů, aby se společně pohybovali a zároveň zachovávali formaci, jako např. hejno ptáků. Hejna jsou formou kolektivního chování velkého počtu členů interagujících společně se skupinou. Využití těchto formací zvyšuje efektivitu daného systému. Pokud by se každý robot choval jako jednotlivec, tak by v určitých situacích mohlo docházet ke vzájemnému omezování nebo dokonce ke srážkám. [2, 3, 14]

2.3 Software

Zatímco předchozí texty zmiňovaly jednotlivé druhy robotů a jejich využití, tato kapitola se věnuje jejich další důležité části, kterou je použitý software. Stejně jako robotů, existuje dnes také velké množství softwaru a knihoven, pomocí kterých lze roboty naprogramovat. V případě průmyslových robotů a vozíků bývá software součástí daného produktu zakoupeného u firmy, která roboty vyrábí. Software se liší podle toho, od jaké je firmy, protože každá firma si software většinou vytváří samostatně. To samé platí u robotických hraček, kde je programování robota uzpůsobeno dané věkové kategorii, nejčastěji pomocí funkčních bloků. Tyto typy softwaru mohou být placené i volně dostupné, ale jsou použitelné pouze u omezeného množství robotů. Většinou pouze pro roboty dané firmy. Druhou variantou je software, přesněji řečeno knihovny, které jsou většinou volně dostupné (open-source) a vývojáři si mezi sebou mohou sdílet části kódu, a tím danou knihovnu rozšiřovat. Tyto knihovny bývají naopak univerzální a lze je použít téměř u libovolného robota. Knihovny mohou být různě velké, podle toho, na co všechno se dají použít. ROS je dokonce sada těchto knihoven, proto je univerzální a nabízí obrovské množství možností pro jeho využití. Mezi zmíněné knihovny patří například *Mobile Robot Programming Toolkit* (MRPT), *Yet Another Robot Platform* (YARP), *Myrobotlab, Player*, *DART* a další. Poslední zmíněná knihovna *DART* se používá také v simulačním programu Gazebo. Mezi další programy využívané pro simulace robotů patří například *Webots* nebo *CoppeliaSim* (V-REP). [15]

3 TURTLEBOT3 BURGER

Jak je z názvu patrné, tento robot je již 3. generací robotů patřící do rodiny TurtleBot. To jsou malé robotické sady s volně dostupným softwarem. TurtleBot vytvořili poprvé Melonee Wise a Tully Foote ve Willow Garage v listopadu 2010. V dnešní době už existují celkem 4 generace, poslední je TurtleBot4. Označení Burger vzniklo hlavně díky vzhledu, stejně jako označení dalších robotů 3. generace, což jsou Waffle a Waffle Pi. Burger je z těchto robotů nejmenší, zbylí dva jsou trochu větší. Tato třetí řada vznikla ve spolupráci s firmou ROBOTIS a Open Source Robotics Foundation. [16]



Obr. 6: TurtleBot3 Burger – popis základní sestavy [17]

3.1 Hardwarové komponenty a specifikace

Celkové rozměry robota Burger jsou 138 mm × 178 mm × 192 mm. Mezi základní komponenty patří podvozek s tělem, motory, kola, OpenCR, jednodeskový počítač (SBC – Single Board Computer) Raspberry Pi, baterie (Li-Po 11.1 V 1800 mAh) a senzory. Podvozek a tělo robota je vytvořeno z tzv. vaflových desek (Waffle Plates). Na sestavení bylo těchto desek použito celkem 8 kusů. Desky jsou spojeny pomocí podpěr do 4 vrstev, což poté může připomínat burger. V prostorech na vrstvách jsou uloženy ostatní komponenty, jako jsou motory, SBC atd. Uspořádání desek lze kombinovat a vytvářet tak případně roboty různých tvarů. [18]

Raspberry Pi 3 Model B je malý jednodeskový počítač, který běží na operačním systému Linux. Pro ukládání dat a nahrání operačního systému se používá běžná mikro

SD karta. Výkonem se blíží ke slabému stolnímu PC. Je osazen procesorem Quad Core 1.2 GHz Broadcom BCM2837 64bit CPU, 1 GB RAM, 100 Base Ethernet, porty HDMI, 2x USB 2, připojení pro kameru atd. [19]

OpenCR, přesněji OpenCR1.0, je otevřená řídící jednotka robota poskytující hardware a software pro vestavěné systémy ROS. Všechno, co se týká této desky, je volně dostupné včetně zdrojového kódu softwaru pro TurtleBot3 a také volně šiřitelné jako open-source licence pro uživatele a ROS komunitu. Obsahuje mikrokontrolér STM32F7, který je založen na výkonném procesoru ARM Cortex-M7. Podporuje RS-485 a TTL pro ovládání motorů, UART, CAN a další komunikační prostředí. Výkonnost OpenCR lze zvýšit při použití s dalším SBC, což je v případě TurtleBot3 již zmíněný Raspberry Pi 3 Model B. [18, 20]

Řídící jednotka ovládá mimo jiné pohony (servomotory), které má robot dva a nesou název DYNAMIXEL od firmy ROBOTIS. TurtleBot3 Burger má pohony s označením XL430-W250. Servomotor má v sobě mimo jiné zabudovaný bezkontaktní absolutní enkodér jako senzor pozice, díky tomu může pracovat až ve čtyřech provozních režimech. Režim řízení rychlosti, řízení polohy v rozsahu 360°, rozšířené řízení polohy (více otáček) a řízení pomocí PWM (pulsně šířková modulace). [18]

Poslední důležitou součástí robota jsou senzory LIDAR a kamera.

3.2 LIDAR

TurtleBot3 Burger má v základní sadě pouze jeden senzor a tím je laserový snímač vzdálenosti s označením LDS-01. Tento 2D laserový skener se nejčastěji používá pro lokalizaci, mapování a navigaci. Parametry a specifikace (viz tab. 1) [18]:

Tab. 1: Specifikace a parametry laserového skeneru LDS-01 [18]

Položka	Specifikace
Rozsah snímané vzdálenosti	120 – 3500 mm
Přesnost vzdálenosti (120 – 499 mm)*	± 15 mm
Přesnost vzdálenosti (500 – 3500 mm)*	± 5 %
Přesnost vzdálenosti (120 – 499 mm)**	± 10 mm
Přesnost vzdálenosti (500 – 3500 mm)**	± 3,5 %
Rychlost skenování	300±10 otáček/min
Úhlový rozsah	360°
Úhlové rozlišení	1°

*přesnost = průměrná vzdálenost – referenční hodnota

**přesnost = (minimální vzdálenost - maximální vzdálenost) / 2

Obr. 7: Laserový snímač vzdálenosti LDS-01 [18]

3.3 Raspberry Pi kamera

V rámci rozšíření možností a větší flexibility byla na robota namontována také kamera. Je použitá Raspberry Pi Camera Module 2 kompatibilní se všemi modely počítačů Raspberry Pi 1 až 4, kde se připojuje pomocí páskového kabelu k portu CSI. Pro běžné použití je již vytvořena řada knihoven, například knihovna Picamera Python. Základní parametry jsou v tabulce 2. [21]

Tab. 2: Základní parametry Raspberry Pi Camera Module 2 [21]

Položka	Specifikace
Rozměry	25 mm × 24 mm × 9 mm
Rozlišení	8 megapixelů
Senzor	Sony IMX219
Horizontální zorné pole	62,2°
Vertikální zorné pole	48,8°
Snímaný obraz	640 × 480 p30

Obr. 8: Raspberry Pi Camera Module 2 [22]

Kamera se nachází v přední části robota a je umístěna ve výšce asi 150 mm. Úhel náklonu kamery je přibližně 40°, aby kamera byla schopná snímat prostor před robotem alespoň do vzdálenosti 250 mm.

4 ROBOTICKÝ OPERAČNÍ SYSTÉM (ROS)

Zkratka ROS vychází z názvu Robot Operating System (Robotický Operační Systém). ROS je framework, sada softwarových knihoven a nástrojů, pro psaní robotického softwaru a vytváření robotických aplikací. Projekt ROS začal v roce 2007 a hlavní vývoj probíhal ve Willow Garage. Od té doby se komunita věnující se ROS velmi rychle rozrůstá do celého světa, a to jak na straně uživatelů, tak vývojářů. [23]

ROS obsahuje a je dodáván s připravenými funkcemi, jako jsou například Simultaneous Localization and Mapping (SLAM) nebo Adaptive Monte Carlo Localization (AMCL), což jsou balíčky použité pro autonomní navigaci mobilních robotů. Nemusí být tedy vždy nutné psát kompletní nový kód, ale stačí pouze konfigurovat a doladit již vytvořený. Dále je součástí obrovské množství nástrojů pro vizualizaci a simulaci. Zde patří *rqt_GUI*, *RViz* nebo Gazebo. Poslednímu jmenovanému se bude tato práce více věnovat. ROS je samozřejmě vybaven balíčky rozhraní a ovladači zařízení pro různé senzory a aktuátory používané v robotice. Mezi senzory můžeme zařadit laserové skenery nebo kamery a k aktuátorům například servomotory DYNAMIXEL. Co se týče programovacích jazyků, tak je ROS velmi flexibilní prostředí, kde lze využívat libovolné jazyky, které mají jeho klientské knihovny. Mezi nejznámější a nejpoužívanější patří C, C++, python nebo Java. V tomto vybraném jazyce se programují uzly neboli nody, mezi kterými poté probíhá předávání zpráv. Mezi další výhody systému můžeme zařadit již zmíněné rozdělení na uzly. V případě, že některý uzel přestane pracovat, tak zbytek systému poběží dále. U jiných robotických aplikací může nastat, že v případě pádu jednoho z vláken se zastaví celá aplikace. Využití uzlů pro psaní jednotlivých procesů je také vhodné, pokud více procesů souběžně využívá stejný hardwarový výstup. V systému ROS může libovolný počet uzlů odebírat zprávy z ovladače daného hardwaru, kterým může být například kamera nebo LIDAR. [24, 25]

V předchozím odstavci je vypsáno velké množství výhod a důvodů, proč si vybrat ROS. Na druhou stranu existují také důvody, které mohou odrazovat. Pro někoho, kdo s ROS začíná, to může být ze začátku velmi složité, vzhledem k jeho komplexnosti. Modelování robotů se provádí pomocí URDF (Unified Robot Description Format) oproti jiným systémům, kde lze robota sestavit jako 3D model. V dnešní době již existují různé programy pro převod 3D modelu na URDF, ale nemusí to být vždy úspěšné. Se simulacemi v systému ROS je situace z hlediska náročnosti podobná. Hlavním simulátorem je Gazebo, které téměř neobsahuje vestavěné funkce pro programování, a proto se kompletní simulace musí provádět pomocí kódování v ROS. ROS vychází v jednotlivých verzích, kde každá verze může mít různá omezení. O současných verzích se dozvíte více v další části. [25]

4.1 ROS verze a distribuce

V současné době existují dvě vydání robotického operačního systému, ROS 1 a ROS 2. Obě vydání vychází v takzvaných distribucích, což jsou jednotlivé verze vycházející jako soubory balíčků ROS. Distribuce ROS 1 vychází již od roku 2010 a distribuce ROS 2 vychází od roku 2017. Systém vydávání je podobný jako distribuce Linuxu (např. Ubuntu). Tento systém umožňuje vývojářům nezasahovat do základních balíčků, ale snaží se pouze odstraňovat chyby a doplňovat rozšiřující vylepšující balíčky. [26, 27]

Co se týká distribucí ROS 1, tak vycházejí podle potřeby a dostupných zdrojů. Jsou dlouhodobě podporovány (LTS) většinou po dobu 5 let. Každá distribuce je podporována přesně na jednom Ubuntu LTS, ale není to vždy podmínkou. ROS 1 běží pouze na operačním systému Linux. Poslední vydanou distribucí je ROS Noetic Ninjemys vydaný 23. května 2020 a jeho podpora končí v květnu roku 2025. Je kompatibilní se systémem Linux Ubuntu Focal Fossa (20.04). Tato distribuce by měla být také úplně poslední distribucí ROS 1. [28]

Distribuce ROS 2 ze začátku vycházely častěji, teď vycházejí jednou ročně a nově vždy 23. května na Světový den želv (World Turtle Day). Každá distribuce prochází více než rok trvajícím vývojem. Délka životnosti těchto distribucí je různá, od jednoho roku až po dobu 5 let. Novější distribuce ROS 2 běží již na více operačních systémech, jako je Windows nebo MacOS, ale doporučeným systémem je stále Linux Ubuntu. Poslední vydanou distribucí je Humble Hawksbill vydaný 23. května 2022 a jeho podpora končí v roce 2027. Jak již bylo zmíněno, kompatibilita je tady větší, např. Linux Ubuntu Jammy Jellyfish (22.04) nebo Windows 10. [26]

Ke každé distribuci ROS existuje také tzv. poster. Na následujícím obrázku jsou postery posledních distribucí obou verzí systému ROS.

Obr. 9: Postery posledních vydání verzí ROS 1 (vlevo) a ROS 2 (vpravo) [29]

V tabulce 3 lze vidět několik posledních vydaných distribucí systému ROS, u kterých je uvedeno datum vydání, datum ukončení a doporučený operační systém.

Tab. 3: Distribuce systému ROS [26, 27]:

Verze	Distribuce	Datum vydání	Datum ukončení	Doporučený systém
ROS	Noetic Ninjemys	23.5.2020	Květen 2025	Ubuntu 20.04 (Focal)
ROS	Melodic Morenia	23.5.2018	Květen 2023	Ubuntu 18.04 (Bionic)
ROS 2	Humble Hawksbill	23.5.2022	Květen 2027	Ubuntu 22.04 (Jammy), Windows 10 (VS 2019)
ROS 2	Galactic Geochelone	23.5.2021	9.12.2022	Ubuntu 20.04 (Focal), Windows 10 (VS 2019)
ROS 2	Foxy Fitzroy	5.6.2020	Květen 2023	Ubuntu 20.04 (Focal), MacOS 10.14 (Mojave), Windows 10 (VS 2019)

Porovnání ROS 1 a ROS 2

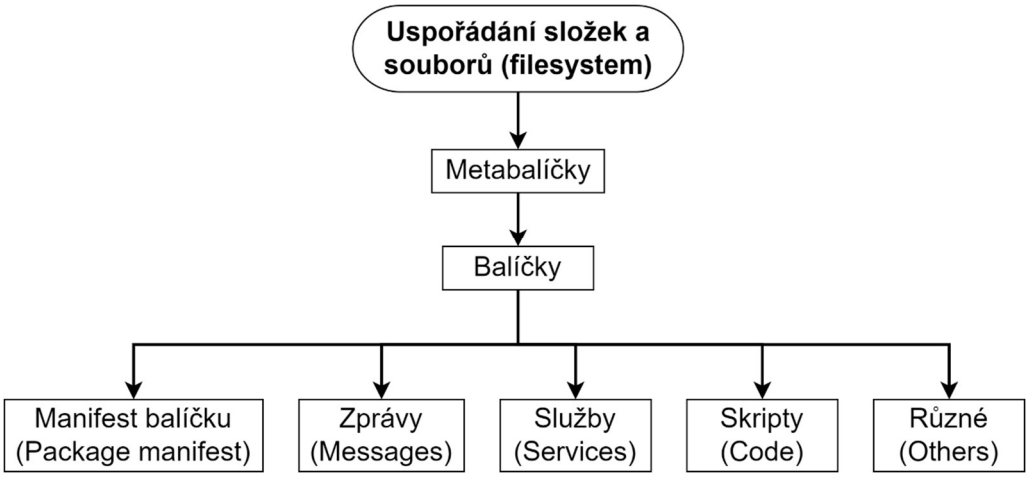
Na první pohled to nemusí být patrné, ale rozdílů mezi verzemi ROS 1 a ROS 2 je velké množství. Uživatel/programátor se s některými změnami nemusí vůbec setkat, ale s jinými se může setkávat již při vytváření projektu. Jedním z hlavních důvodu vzniku ROS 2 je kompatibilita a využití v průmyslových aplikacích. Nedostatky ROS 1 spočívaly například v bezpečnosti, nedodržení průmyslových standardů, omezené možnosti řízení v reálném čase nebo také nemožnost využití na embedded (vestavěných) systémech. Z důvodu kompatibility s průmyslovými aplikacemi, tak ROS 2 využívá middleware DDS (Data Distribution Service), což je softwarová vrstva ležící mezi operačním systémem a aplikacemi. Integruje součásti systému dohromady a umožňuje snadnější komunikaci a sdílení dat s vysokou spolehlivostí. [30–32]

4.2 Struktura ROS

Strukturu ROS je možné rozdělit na dvě části. První částí je uspořádání souborů a složek (filesystem) a druhou částí je popis komunikace mezi jednotlivými částmi systému ROS.

Uspořádání složek a souborů

Jak je patrné již z názvu, ROS není jen framework, ale má také funkce podobné operačnímu systému. Mezi tyto podobnosti můžeme zařadit právě uspořádání souborů a složek na pevném disku, viz obr. 10.



Obr. 10: Uspořádání složek a souborů v systému ROS [33]

Vysvětlení k jednotlivým blokům systému [25, 34]:

Metabalíček (Metapackage): Označuje seskupení několika balíčků (packages) dohromady. Neobsahuje žádný zdrojový kód ani další podobné soubory, pouze manifest metabalíčku (metapackages manifest). To je soubor obsahující informace o seskupení balíčků, licenci, autorovi atd. Jako příklad celého metabalíčku lze uvést metabalíček s názvem TurtleBot3, který v sobě obsahuje cca 10 balíčků. Ty obsahují informace spojené se skupinou robotů TurtleBot3.

Balíčky (Packages): Jsou základní jednotkou softwaru ROS. Obsahují jeden nebo více ROS programů (procesů), které se nazývají uzly (nodes), dále knihovny, konfigurační soubory a cokoliv dalšího, co je užitečné uspořádat dohromady. Balíčky jsou základní kameny pro sestavení a vydání softwaru ROS. Každý ROS balíček obsahuje povinný soubor manifest balíčku (package manifest), kde jsou obdobně jako u metabalíčku informace o autorovi, licenci, kompilaci atd. Obvykle soubor package.xml je tímto manifestem. Dále balíček obsahuje systém složek pro uložení jednotlivých souborů, jako jsou knihovny, skripty atd. Například ve složkách Zprávy (Messages) (zprávy) a Služby (Services) jsou uložené jejich definice. K těmto pojmům se dostaneme v další části, která popisuje komunikace v rámci systému ROS.

Cílem balíčků je poskytnout užitečné informace pohromadě, aby je bylo možné opakovaně používat i jinými uživateli. Balíček by měl obsahovat dostatek funkcí, aby byl užitečný, ale zároveň by jich nemělo být příliš mnoho. Od toho jsou tady již zmíněné metabalíčky, které spojují balíčky dohromady. Balíček si může každý vytvořit úplně kompletně celý sám, ale jednoduší a ve většině případů výhodnější variantou je použití nástrojů pro tvorbu balíčku. Nástroj pro ROS 1 se nazývá *catkin*, ROS 2 využívá jiných nástrojů, například *colcon*. Jednotlivé distribuce ROS obsahují balíčky v počtu jednotek tisíc.

Komunikace v systému

Nejdůležitější části ROS je právě komunikace mezi jednotlivými prvky systému, tj. způsob, jakým jsou data a informace předávány. Na začátek je nutno se seznámit s několika základními pojmy [25, 34]:

Uzly (nodes): Označují jednotlivé procesy, někdy označovány jako programy, které provádějí výpočty a další různé činnosti. Využívají k tomu klientské knihovny ROS, jako jsou roscpp (pro jazyk C++) nebo rospy (pro jazyk Python). Robot obsahuje mnoho uzlů tohoto typu a každý z nich je naprogramován pro specifický účel. Například pro pohon motoru, převod dat ze snímače, zpracování obrazu a další. Uzly mezi sebou komunikují pomocí témat a služeb. Použití těchto uzlů nezávislých na sobě zlepšuje odolnost systému vůči poruchám. V případě výpadku nějakého uzlu může celý robot dále fungovat pouze bez té jedné funkce, kterou daný uzel plnil. Toto platí kromě jedné výjimky, kterou je ROS Master. Dále rozlišujeme 2 varianty, jak mohou uzly pracovat: vydavatel (Publisher) a odběratel (Subscriber). V prvním případě uzel vysílá data ostatním uzlům, v druhém případě uzel data přijímá. Každý uzel může vysílat i přijímat data zároveň, být vydavatel i odběratel. Spojení mezi sebou udržují pomocí transportní vrstvy TCPROS využívající standardů TCP/IP.

ROS Master: Je hlavním uzlem v systému ROS a jeho funkce se dá přirovnat k serveru. Zajišťuje vznik propojení a komunikaci mezi uzly. Zajišťuje registraci jednotlivých uzlů, přiděluje jim jedinečné názvy, ID a má tedy přehled o všech spuštěných uzlech v systému. V případě jeho výpadku nelze spustit nové uzly ani vytvářet nová propojení. Je tedy slabým místem systému. Spojením mezi ROS Master a podřízenými uzly probíhá pomocí XML-RPC (XML Remote Procedure Call), což je protokol založený na protokolu http, který neudržuje stále spojení jako v případě TCPROS. V případě spuštění je ROS Master nakonfigurován pomocí adresy URI a čísla portu definovaných v ROS_MASTER_URI. Výchozím nastavením pro adresu URI je IP adresa počítače a číslo portu 11311. Při použití v distribuovaných sítích je vhodné si nadefinovat ROS_MASTER_URI ručně.

Server parametrů (Parameter server): Označení pro server, úložiště, který je součástí ROS Master. Uzly při svých operacích (výpočtech) využívají různé nastavené parametry, například složky PID regulátoru, maximální otáčky motoru atd. V případě, že je těchto parametrů více nebo je nutné, aby k nim měly přístup i jiné části systému, lze je uložit právě na tento server parametrů. Uzel může hodnoty parametrů ze serveru číst, zapisovat, upravovat nebo mazat. Uživatel může samozřejmě tyto parametry uložit do souboru a nahrát na server, případně stáhnout, nastavovat rozsah a přidělit práva, které uzly mají k daným parametrům přístup.

Zprávy (Messages): Komunikace mezi uzly probíhá pomocí těchto zpráv. Definice každé zprávy je uložena v balíčku, obsahuje strukturu zprávy a použité datové typy (může jich být více). V rámci systému ROS již existují sady typů zpráv specifických pro roboty, ale v případě potřeby si každý může nadefinovat vlastní typ zprávy. Pro jednosměrné doručování zpráv mezi uzly se používají témata a pro obousměrné posílání služby.

Témata (Topics): Posílání zpráv musí mít určitá pravidla, jednou z variant jsou právě témata, která jsou doslova tématy rozhovoru. Jsou to pojmenované sběrnice, ve kterých se posílají zprávy daného tématu. Jednotlivé uzly se mohou přihlásit k odběru těchto zpráv nebo také publikovat zprávy na dané téma. Pro vytvoření těchto témat musí dané téma zaregistrovat některý z uzlů (jeden z vydavatelů) u ROS Master, který poté

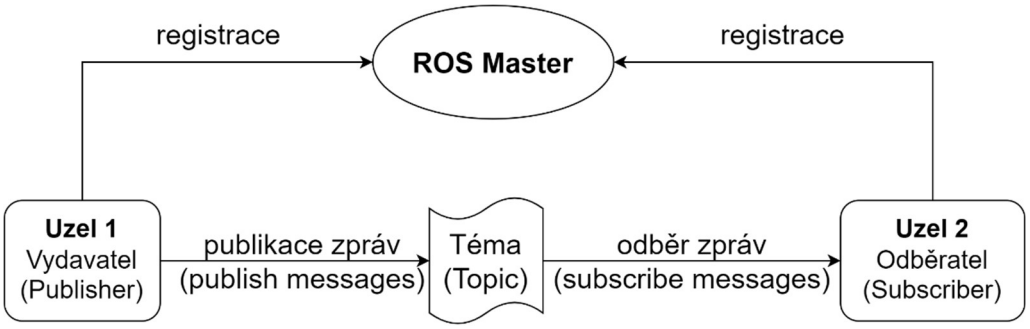
k danému tématu připojuje další uzly, které mohou být odběrateli i vydavateli. Uzly se mohou připojit k jednotlivým tématům, ale navzájem o sobě nevědí. Pouze ROS Master ví, který uzel, kde publikuje a odkud přijímá data. Proto je tato komunikace považována za jednosměrnou bez ověření. Příkladem použití může být vysílání dat ze senzorů, kdy se data posílají nepřetržitě a neověřuje se, zda data došla do všech uzlů, která je přijímají.

Služby (Services): Těch se využívá v systému ROS, pokud potřebujeme komunikovat stylem požadavek-odpověď. Nejčastější použití je v distribuovaných systémech. Služby jsou definovány jako dvojce zpráv a jejich definice jsou opět uloženy v balíčku. Na rozdíl od témat jsou služby jednorázovou komunikací zpráv. Služba se skládá ze serveru služby (implementován v uzlu) a klienta služby. Klient vznese požadavek a od serveru očekává odpověď. Po splnění požadavku a odpovědi se spojení mezi uzly přeruší.

Bag soubory (Bags): Soubory s příponou .bag, někdy označovány přímo jako bag soubory (bagfiles). Slouží k ukládání a shromažďování dat z posílaných zpráv. Soubor bag se připojí na dané téma a zaznamenává všechny poslané zprávy. Vytvořené bag soubory je možné zpět přehrávat do daného tématu a tím provádět off-line simulaci. Využívá se například při absenci některých senzorů, nebo při opakovaných experimentech, které jsou výpočetně složité.

Tento základní koncept platí pro obě verze ROS 1 i 2. Verze ROS 2 je však rozdílná v absenci ROS Master. Jeho „práci“ zde zajišťuje již dříve zmíněná služba DDS. Tímto krokem bylo odstraněno slabé místo v případě výpadku ROS Master.

Na obrázku 11 je obecný diagram znázorňující komunikaci v ROS.

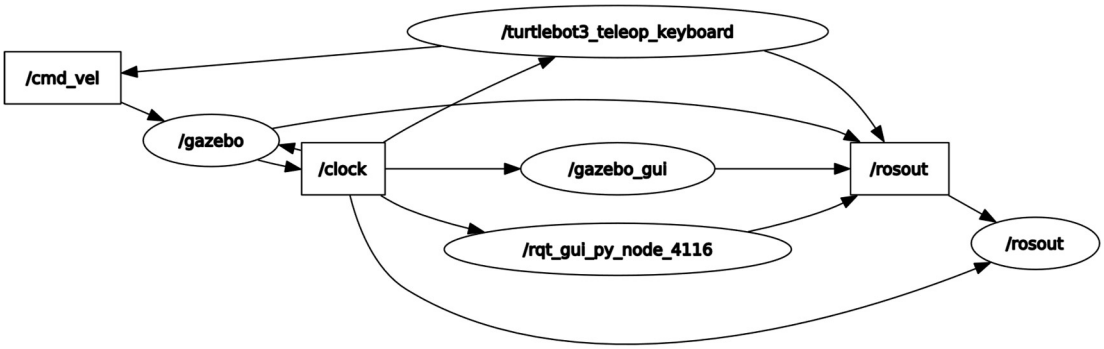


Obr. 11: Diagram komunikace v ROS

Ukázkový příklad

Na příkladu, ovládání Turtlebota pomocí šipek v simulačním prostředí Gazebo, si ukážeme, jak fungují výše uvedené pojmy. Nejprve je nutné spustit ROS Master, a to buď příkazem *roscore* nebo se spustí automaticky při spuštění simulace. Se spuštěním simulace se automaticky spustí nody */gazebo*, */qazebo_gui*, */rosout* spolu s tématy */clock* a */rosout*. Až poté je možné spustit uzel, který slouží k ovládání TurtleBota šipkami. Jeho název je */turtlebot3_teleop_keyboard* a spustí se spolu s tématem */cmd_vel*. Právě přes toto téma se posílají zprávy do simulačního prostředí Gazebo, jak se má turtlebot pohybovat. Uzel tyto zprávy generuje automaticky podle toho, jaké šipky jsou zmáčknuty.

Na obrázku 12 lze vidět všechny spuštěné uzly a témata, včetně */rqt_gui_py_node_4116*, což je uzel, jehož výstupem je právě zobrazení spuštěných uzlů a témat. Uzly jsou v elipsách a témata v obdélnících. Jednotlivým šipky v obrázku znázorňují, že některé uzly (vydavatelé) posílají zprávy do témat a některé pouze zprávy přijímají (odběratelé) nebo obojí zároveň. Při spuštění pouze ROS Master (příkaz *roscore*) se vždy spustí uzel */rosout*, který slouží pro zaznamenávání výstupů jednotlivých uzlů.



Obr. 12: Ukázka z nástroje *rqt*

4.3 Simulační prostředí Gazebo

Vývoj tohoto simulačního softwaru začal již před systémem ROS. V dnešní době je to velmi populární simulátor, a to nejen pro systém ROS, kde je jeho hlavním simulačním programem. Gazebo, obdobně jako ROS, vychází postupně v jednotlivých verzích a je volně dostupné. Pro ROS 1 je poslední verzí Gazebo 11 (classic Gazebo), pro ROS 2 došlo k několika změnám a přechodu na Ignition Gazebo. Při vydání nejnovější distribuce došlo k opětovnému přejmenování zpátky na Gazebo. Jak již bylo několikrát zmíněno, Gazebo je software navržený pro simulaci robotů, testování algoritmů, trénování umělé inteligence na realistických scénářích atd. Nabízí možnost simulace ve vnitřních i venkovních prostorech. Obsahuje robustní fyzikální engine, kvalitní grafiku a uživatelsky příjemné grafické rozhraní. [35]

Gazebo podporuje různé fyzikální engines jako jsou *ODE*, *Bullet*, *Simbody* nebo *DART*. Pro 3D grafiku používá *OGRE* (Open-source Graphics Rendering Engines), který dokáže velmi realisticky vykreslit model robota, světlo, stín nebo texturu. Podporuje obrovské množství senzorů od laserových dálkoměrů, přes 2D/3D kamery až po senzory síly a také je schopno simulovat a aplikovat šum skutečného prostředí. Nabízí velké množství dalších pluginů a kompletní modely robotů. Například PR2, Pioneer2 DX nebo TurtleBot v podobě SDF, což je XML formát používaný v rámci softwaru Gazebo. [35]

Architektura programu Gazebo je distribuovaná se samostatnými knihovnami. Knihovny jsou pro komunikaci (Communication Library), vykreslování (Rendering Library), pro fyzikální simulaci (Physics Library) a další. [35]

Při spouštění simulačního prostředí Gazebo, se ve skutečnosti spustí 2 různé programy. Jeden s názvem *gzserver* a druhý *gzclient*. *Gzserver* je jádrem systému Gazebo a je tím hlavní programem, který slouží pro simulaci fyziky, generování dat ze senzorů atd. Skrz komunikační knihovny poté poskytuje data programu *gzclient*, jehož součástí je grafické uživatelské rozhraní pro vizualizaci a interakci se simulací. Toto rozdělení může být vhodné v případě, že *gzserver* spustíme na cloudovém počítači, kde není potřeba uživatelské prostředí. [35]

4.3.1 Spojení ROS a Gazebo

Gazebo lze jednoduše stáhnout jako balíček do systému ROS. Pro vytvoření simulace poté stačí spustit soubor (většinou s příponou .launch), ve kterém jsou specifikované modely prostředí a robota, případně další parametry a programy (nody) potřebné pro simulaci. Spustí se systém ROS a prostředí Gazebo, kde simulace běží.

V rámci systému ROS se pro popis robotů používá URDF, což je formát XML pro reprezentaci modelu robota. V systému Gazebo se používá také XML formát SDF. Je to specializovaný formát pro simulace v Gazebo, lehce pozměněný a rozšířený o další možnosti oproti URDF. Běžným příkazem stačí pouze převést URDF na SDF.

Samotný model robota pro simulaci nestačí. V rámci modelu robota není specifikována funkčnost jednotlivých částí. Typickým příkladem jsou senzory nebo motory. Proto je nutné přidat pluginy (moduly), které slouží k ovládání simulovaného robota a k interakci robota s okolím. V rámci těchto pluginů je nutné specifikovat parametry těchto částí, například u kamery je to zorné pole, snímková frekvence, velikost obrazu, formát dat a další. Pluginy mají již v sobě předdefinované formáty zpráv, takže stačí doplnit název tématu, kam má Gazebo vysílat data. Případně odkud má data přijímat, pokud se jedná o ovládání motorů.

V případě použití převodů nebo celých převodovek v rámci konstrukce robota, je nutné tyto převody (transmission) specifikovat. Určují převod mezi kloubem (joint) a pohonem (actuator). V případě robota TurtleBot3 Burger to není potřeba, protože kola jsou na výstup motorů napojena přímo. [27]

5 PRAKTICKÁ REALIZACE

Praktickou částí této práce bylo sestavení robota TurtleBot3 Burger, jeho zprovoznění v ROS a návrh řídícího programu pro vybranou laboratorní úlohou. Poté tento návrh otestovat. Zadaní laboratorní úlohy je popsáno níže a nabízí určitou volnost při jeho řešení.

Zadání: Turtlebot je schopen sledovat a pohybovat se po čáře. V případě výskytu překážky, překážku objede a poté pokračuje opět ve sledování čáry. V případě zabloudění, za které se považuje ztráta čáry (například při prudké zatáčce), by měl robot být schopen čáru opět najít a pokračovat dále.

Koncepce řešení byla následující:

1) Studium frameworku ROS a simulačního prostředí Gazebo – zahrnuje volbu a instalaci systému ROS spolu se simulačním prostředím
2) Sestavení, zprovoznění robota a seznámení se s jeho možnostmi
3) Návrh řídícího programu s ohledem na možnosti, které robot nabízí
4) Testování řídícího programu v simulačním prostředí Gazebo
5) Testování řídícího programu v reálném prostředí
6) Srovnání simulace a reálného prostředí

Tento postup byl ovšem čistě orientační, neboť v průběhu testování v reálném prostředí došlo k určitým změnám, a nakonec vznikly dva návrhy řídícího programu. Druhý návrh, který vznikl při testování na reálném zařízení, byl poté ještě zpětně testován v simulačním prostředí.

5.1 Systém ROS a prostředí Gazebo

Práce byla realizována na vlastním notebooku s operačním systémem Windows 10. Dle dokumentace se tedy nabízelo využít verzi ROS 2. Na základě doporučení a zjištěných informací bylo rozhodnuto, že bude vhodnější instalace operačního systému Linux s využitím virtuálního nástroje Oracle VM VirtualBox. Až na ten se poté provede instalace ROS 2. Nabízely se hned 2 distribuce, ale protože distribuce Foxy Fitzroy měla plánované ukončení již na květen 2023, tak zůstala jediná varianta, distribuce Humble Hawksbill. K tomu doporučené verze simulačního prostředí Gazebo Garden nebo Gazebo Ignition Fortress. Posledním krokem bylo stažení a instalace balíčků pro mobilního robota TurtleBot3 Burger. Tento krok však nebyl úspěšný, proto jako náhradní řešení byla stažena a nainstalováno verze ROS Noetic Ninjemys a Gazebo 11. Tomuto kroku předcházelo přeinstalování systému Linux na kompatibilní verzi Ubuntu Desktop 20.04 LTS Focal Fossa. Tady už instalace balíčků pro TurtleBot3 Burger proběhla v pořádku. Toto je odpověď na případnou otázku, proč byl zvolen starší systém ROS a je popsán v teoretické části. Instalace systému a stahování balíčků probíhalo podle oficiálních návodů.

5.2 Sestavení a zprovoznění robota

Kompletní sestavení TurtleBot3 Burger ze stavebnice neproběhlo, neboť robot byl již sestaven. Avšak při jeho zprovoznění došlo k částečnému rozebrání a opětovnému sestavení, protože bylo nutné otestovat funkčnost některých jeho součástí. Jak již bylo zmíněno, robot je jako stavebnice, jednalo se tedy o sešroubování několika dílů k sobě a propojení elektrických součástí pomocí kabelů bez nutnosti jakéhokoli pájení. Vzhledem k tomu, že se jedná o zakoupeného robota, tak je od výrobce k dispozici kompletní návod. Při popisu robota byly zmíněny 2 senzory, LIDAR, v základní sadě, doplněný o kameru Raspberry Pi. Na základě toho, co tyto senzory nabízí, bylo rozhodnuto, že při návrhu řídícího programu budou použity oba.

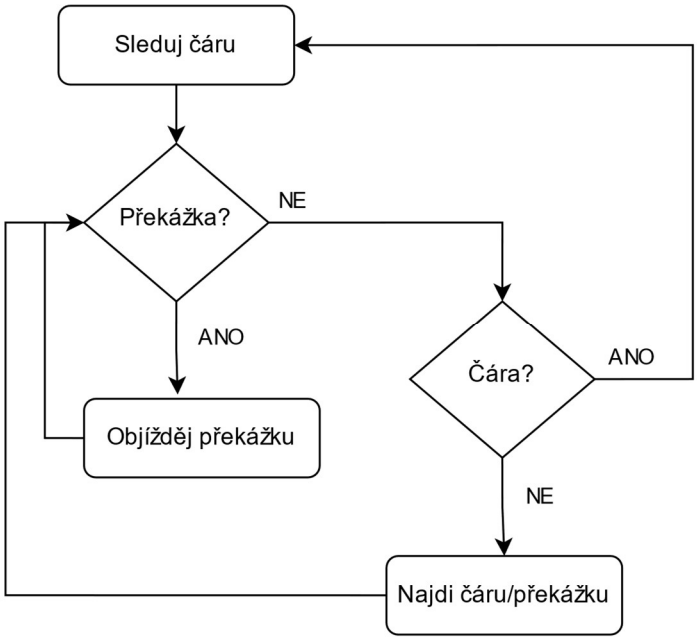
Při zprovozňování byla snaha postupovat podle oficiálního návodu, tato snaha se však nesetkala s úspěchem. Bylo proto nutné postupovat dále s pouhým přihlížením k návodu. Prvním krokem bylo stažení a instalace operačního systému na mikro SD kartu. Z doporučených verzí systému Linux pro Raspberry Pi model 3B se nabízela pouze jediná, která byla kompatibilní se zvoleným systémem ROS. Pro tento úkol ale dostačující, verze Ubuntu Server 20.04 LTS. Protože se nejedná o operační systém přímo navržený pro Raspberry Pi, bylo nutné provést několik kroků za účelem získání přístupu k obrazu připojené kamery. Instalace konfiguračního nástroje *raspi-config* pro Raspberry Pi. V tomto nástroji poté povolit rozhraní pro připojení kamery. V případě, že by toto nestačilo, tak je zde varianta, že spouštěcí/zaváděcí oddíl (boot partition) s konfiguračními soubory není připojen na správném místě na mikro SD kartě a konfigurační nástroj poté nemůže najít správné konfigurační soubory. Pomocí příkazu *mount* se musí oddíl připojit.

Po připojení kamery a získání obrazu byl další krok instalace systému ROS. Ta proběhla v pořádku, ovšem s jedním rozdílem oproti instalaci na virtuální počítač. V samotném robotu není nutné mít kompletní verzi ROS se všemi balíčky a nástroji, stačí pouze verze ROS-Base se základními balíčky. Stejně jako u počítače se musí ještě doplnit balíčky vztahující se k samotnému TurtleBot3 Burger a univerzální balíček *cv_camera,* pomocí kterého získáme přístup k obrazu z kamery. Tento balíček by měl být funkční, jak při použití kamery Raspberry Pi, tak i u jiných kamer připojených přes USB.

Další nastavení, například správné nastavení IP adres počítače a robota byly již provedeny dle návodu. Při spouštění systému a samotného programu se nabízejí 2 varianty. Při první variantě jsou všechny soubory a balíčky nahrány v robotu a program je pouze vzdáleně spuštěn přes počítač. Při druhé variantě, je systém ROS (ROS Master) spuštěn na počítači a turtlebot se k tomuto systému připojí. Aby se robot mohl připojit, je nutné správné nastavení IP adres a samozřejmě musí být připojen ke stejné síti. Program je poté spouštěn na počítači a data se posílají přes síť.

5.3 Návrh řídícího programu

Myšlenka celého řídícího programu není nijak složitá. Robot pomocí kamery sleduje čáru a pohybuje se po ní. V případě zaznamenání překážky pomocí LIDARu začne robot překážku objíždět. V průběhu objíždění překážky robot pravidelně zjišťuje, zda už překážku neobjel a může dále pokračovat ve sledování čáry. Program běží v nekonečné smyčce s frekvencí stejnou jako je frekvence snímání obrazu pomocí kamery. Vyšší frekvence není nutná, protože frekvence LIDARu je ještě nižší. Pomocí vývojového diagramu se program dá vyjádřit následovně:



Obr. 13: Vývojový diagram řídícího programu

Z diagramu je patrné, že podmínka výskytu překážky má přednost. Obdélníky reprezentují jednotlivé stavy, které určují, jak se bude robot pohybovat.

V první části práce již byly zmíněny různé hotové balíčky věnující se lokalizaci a mapování (SLAM) a následné navigaci robota, ale žádný ze zmíněných balíčků není vhodný pro zadanou úlohu. Ve většině případů tyto balíčky využívají pouze jeden senzor, například při mapování je to LIDAR, kdy se díky získaných informací vytvoří mapa prostředí. Z toho plyne, že primární využití má tento balíček v uzavřených prostorech, ve kterých se robot poté bez potíží dokáže pohybovat. Toto v našem případě není vůbec potřeba, protože sledování čáry může probíhat i ve venkovních prostorách, překážky je možné libovolně měnit atd. Bylo tedy nutné vytvořit vlastní balíček a řídící program, kde budou využity oba senzory zároveň, jak LIDAR, tak kamera.

Tvorba balíčku není komplikovaná, pokud je balíček vytvářen pomocí nástroje *Catkin*. Tento nástroj automaticky vytvoří strukturu složek a další nejnutnější soubory. Uživatel při vytváření zadává pouze název balíčku a jeho závislosti. Závislosti mohou být různé, ale nejčastěji to bývají klientské knihovny a definice používaných zpráv. Mezi klientské knihovny můžeme zařadit například *rospy* nebo *roscpp*, což poté umožňuje

používat při programování jazyk python nebo C++. Závislosti mohou být doplněny samozřejmě i později v průběhu tvorby programu. Příkaz pro vytvoření balíčku *robot_rasp_pkg* s několika závislostmi:

```
catkin_create_pkg robot_rasp_pkg rospy std_msgs sensor_msgs
```

Pokud je balíček vytvářen tímto způsobem, je nutné ještě předtím vytvořit pracovní prostor, tzv *catkin workspace*, a ten zařadit do prostředí ROS. Poté pomocí příkazu *catkin_make* je balíček sestaven. Po tvorbě a stavbě balíčku je již možné se pustit do tvorby algoritmu pro řízení robota. Koncovka vytvořeného souboru s řídícím programem *pohyb_rasp_1.py* již napovídá, že bude využíván programovací jazyk python. Do řídícího programu byl nutný import klientské knihovny *rospy*, definic potřebných zpráv, které budou posílány a dalších knihoven jazyka python. Definice zpráv jsou *LaserScan*, *Image* a *Twist*. Import knihoven:

```
import rospy
import numpy as np
import math
from sensor_msgs.msg import LaserScan, Image
from geometry_msgs.msg import Twist
```

Bylo využito objektově orientované programování. Uzel je vytvořený v rámci třídy, ve které jsou definovány další různé metody, které budou dále popsány. Zkrácená struktura celého programu:

```
class MyNode:
    def __init__(self):
        ...
    def laser_callback(self, msg):
        ...
    def image_callback(self, msg):
        ...
    def run(self):
        ...

def main():
    ...

if __name__ == '__main__':
    main()
```

Metoda *__init__*, kterou lze přirovnat ke konstruktoru v jazyce C, se zavolá automaticky po vytvoření objektu. Součástí této metody je samotné vytvoření uzlu a nadefinování, z jakých témat tento uzel přijímá zprávy (odběratel) a kam může zprávy

odesílat (vydavatel). Vytváří se další proměnné využívané ostatními metodami. Část metody *__init__*:

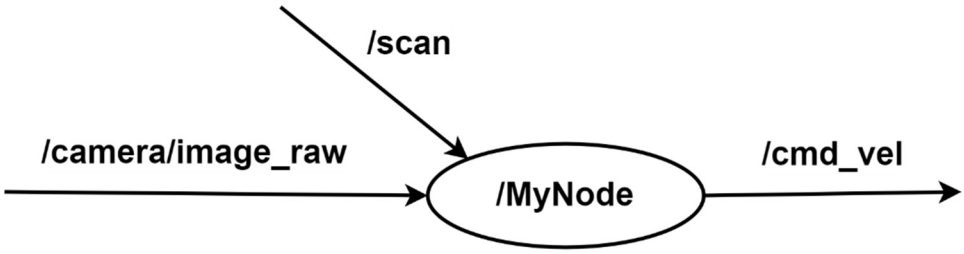
```
def __init__(self):
    rospy.init_node('MyNode')
    self.laser_sub = rospy.Subscriber("scan", LaserScan,
self.laser_callback)
    self.image_sub = rospy.Subscriber("/camera/image_raw",
Image, self.image_callback)
    self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist,
queue_size=1)
    ...
```

Při vytváření odběratele je nutné určit, co se s přijatými daty bude provádět. V ukázce kódu výše to jsou metody *self.laser_callback* a *self.image_callback*. V rámci těchto metod bude provedeno zpracování obrazu a dat ze skeneru, podle kterých bude následně určen pohyb robota. Obsah těchto metod je popsán v dalších kapitolách.

Hlavní kód *main()* pouze vytváří třídu a spouští její metodu *run()*, která přebírá hodnoty z ostatních metod a přiřazuje je do zpráv posílaných pro ovládání robota.

Robot je ovládán pomocí tématu */cmd_vel*. V tomto tématu se posílá již zmíněná zpráva typu *Twist*. Tato zpráva obsahuje dva vektory (6 hodnot), určující rychlost pohybu v jednotlivých osách a rychlost natáčení kolem jednotlivých os. V rámci programu zadáváme pouze 2 z těchto hodnot. Dle polohy souřadnicového systému robota zadáváme pro pohyb vpřed velikost rychlosti ve směru osy x a pro zatáčení velikost rychlosti otáčení kolem osy z.

V rámci systému ROS program představuje jeden uzel (viz obr. 14), který bude přijímat zprávy (data) ze senzorů a odesílat zprávy určující pohyb robota.



Obr. 14: Vlastní vytvořený uzel

5.4 Zpracování obrazu

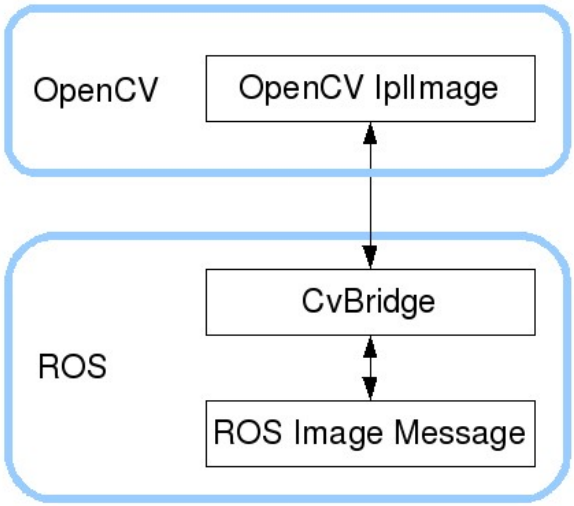
Zpracování obrazu je implementováno jako součást metody *self.image_callback*. Jak již bylo výše zmíněno, obraz z kamery byl do systému ROS získán pomocí balíčku *cv_camera*. Získaný obraz je ve formě takzvaných raw dat. Protože se jedná o barevnou kameru, tak každý pixel je definován 3 hodnotami v rozsahu 0 až 255. Hodnoty

odpovídají jednotlivým složkám RGB (Red Green Blue). Ke zpracování těchto dat byla použita knihovna *OpenCV*.

OpenCV (Open Source Computer Vision Library) je volně dostupná softwarová knihovna pro počítačové vidění a strojové učení. *OpenCV* byla vytvořena s cílem poskytnout společnou infrastrukturu pro aplikace počítačového vidění. Knihovna obsahuje velké množství optimalizovaných algoritmů, které zahrnují sady klasických i nejmodernějších algoritmů počítačového vidění a strojového učení. Tyto algoritmy lze použít k detekci a rozpoznávání obličejů, identifikaci objektů, klasifikaci lidských činností ve videích, sledování pohybu kamery, sledování pohybujících se objektů a podobně. Rozhraní je k dispozici v jazycích C++, Python, Java nebo Matlab a podporuje nejrozšířenější operační systémy Windows, Linux, Android i MacOS. [36]

V tomto úkolu použijeme pouze několik funkcí, jako jsou převod mezi barevnými modely, převod na stupně šedi, filtrace barev v obraze, ukládání videí a další. V kódu je knihovna importovaná pod názvem *cv2*.

Samotný obraz z kamery je posílán ve formě raw dat pomocí zpráv v rámci systému ROS. Do řídícího programu je ještě nutné importovat ROS knihovnu CvBridge, která je součástí balíčku s názvem *cv_bridge*. CvBridge slouží k propojení ROS a *OpenCV* pro převod obrazu z ROS zpráv na obraz vhodný pro *OpenCV* a naopak (viz obr. 15). [37]



Obr. 15: Převod obrazu mezi ROS a *OpenCV* [37]

Aby bylo možné tyto knihovny používat, musí se přidat do závislostí námi vytvořeného balíčku. Zde je ukázáno, jak vypadá import zmíněných knihoven pro zpracování obrazu a převod obrazu z ROS zpráv na obraz vhodný pro *OpenCV*:

```
import cv2
from cv_bridge import CvBridge
self.bridge = CvBridge()          # součástí metody __init__
cv_image = self.bridge.imgmsg_to_cv2(msg)
```

5.4.1 První varianta zpracování obrazu

Tato varianta bere v úvahu situaci, že čára, kterou má robot sledovat bude černé nebo velmi tmavé barvy. Získaný RGB obraz je proto převeden na obraz ve stupních šedi. Každý pixel má poté místo tří hodnot pouze jednu hodnotu 0 až 255. Nula odpovídá černé barvě a 255 je hodnota pro bílou barvu. Po převedení je nutno aplikovat takzvanou techniku prahování (thresholding). Tato technika spočívá v přiřazení hodnot jednotlivým pixelům vzhledem k zadané prahové hodnotě. Při prahování se každá hodnota pixelu porovnává s prahovou hodnotou. Pokud je hodnota pixelu menší než prahová hodnota, je nastavena na 0 (černá), v opačném případě je nastavena na maximální hodnotu 255 (bílá). Při této jednoduché technice prahování se prahová hodnota zadává manuálně. Další možností je adaptivní prahování, kdy se prahová hodnota dynamicky mění pro každý snímek a také pro jednotlivé části snímku, aby se zpracování obrazu přizpůsobilo měnícím se světelným podmínkám. [38]

Pro další zpracování je však nutné barvy prohodit. Následně aplikujeme filtr, abychom eliminovali případné nečistoty nebo tmavé body, které se vyskytovaly na podlaze v okolí čáry. Použitá funkce *cv2.erode* zmenší všechny objekty v obraze a pokud je objekt dostatečně malý tak jej úplně vymaže. Na výsledném obraze je po zpracování vidět pouze bílá čára na černém pozadí. Ukázka výše popsaného postupu:

```
gray = cv2.cvtColor(cv_image, cv2.COLOR_BGR2GRAY)
_, thresh_1 = cv2.threshold(gray, 100, 255,
cv2.THRESH_BINARY_INV)
kernel = np.ones((3, 3), np.uint8)
thresh = cv2.erode(thresh_1, kernel, iterations=2)
```

Obr. 16: Ukázka zpracování obrazu v první variantě, originál (vlevo) a po zpracování (vpravo)

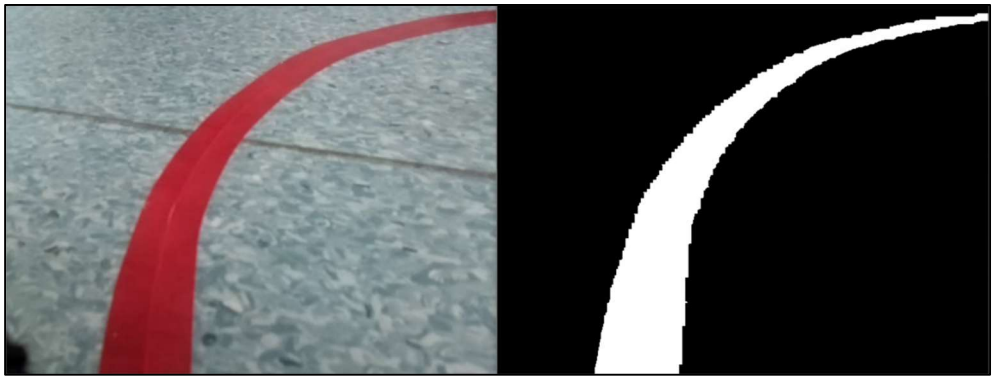
5.4.2 Druhá varianta zpracování obrazu

V průběhu testování byla přidána další varianta a je určena pro sledování čáry červené barvy. V získaném obraze jsou pomocí filtru nalezeny pixely s odstínem červené barvy a jejich hodnota je nastavena na 255 (bílá), ostatní pixely jsou nastaveny na 0 (černá). Opět aplikujeme filtr pro eliminování případných červených odlesků v okolí samotné čáry. Tímto je dosaženo stejně jako v první variantě bílé čáry na černém pozadí.

Filtrace červené barvy z obrazu jde provést mnoha způsoby. V našem případě byl použit způsob, kdy není potřeba obraz převádět do jiného barevného modelu a hodnoty pixelů jsou určeny pomocí barevného modelu RGB. Pomocí intervalů se definují pixely s odstínem červené, ze kterých se vytvoří takzvaná maska. Zbytek obrazu je převeden na černou barvu. Po této úpravě je obraz červenočerný a každý pixel je stále definován třemi hodnotami. Zbývají poslední 2 úpravy, kdy první z nich je převedení na černobílý obraz tak, aby každý pixel měl pouze jednu hodnotu 0 nebo 255. Černé části zůstanou černé a z červené barvy se stane bílá barva. Toho je dosaženo pomocí převodu na stupně šedi a prahováním obdobně jako u první varianty. Druhou úpravou je stejně jako v první variantě filtrace nežádoucích míst s odstíny červené mimo sledovanou čáru. Výsledný obraz je opět bílá čára na černém pozadí.

Výše popsaný postup:

```
lower_red = np.array([0, 0, 90], dtype = "uint8")
upper_red= np.array([60, 60, 255], dtype = "uint8")
mask_01 = cv2.inRange(cv_image, lower_red, upper_red)
lower_red = np.array([0, 0, 120], dtype = "uint8")
upper_red= np.array([140, 140, 255], dtype = "uint8")
mask_02 = cv2.inRange(cv_image, lower_red, upper_red)
mask = mask_01+mask_02
red_output = cv2.bitwise_and(cv_image, cv_image, mask = mask)
gray = cv2.cvtColor(red_output, cv2.COLOR_BGR2GRAY)
_, thresh_1 = cv2.threshold(gray, 10, 255, cv2.THRESH_BINARY)
kernel = np.ones((3, 3), np.uint8)
thresh = cv2.erode(thresh_1, kernel, iterations=2)
```



Obr. 17: Ukázka zpracování obrazu v druhé variantě, originál (vlevo) a po zpracování (vpravo)

5.5 Detekce čáry

Další postup je již stejný pro obě varianty. Ve zpracovaném obraze je potřeba čáru detekovat. K detekci je použita funkce hledání tvarů a obrysů (*contours*). Následně je z obrazu vybrán největší útvar, kterým je samotná čára. U tohoto rovinného útvaru (čáry) je nalezena poloha těžiště a spočítán jeho obsah. Souřadnice těžiště na vodorovné ose je

porovnána se středem obrazu a tím je určena odchylka. Robot je následně řízen na základě velikosti této odchylky.

Implementace popsaného postupu detekce čáry a výpočtu odchylky zde:

```
contours, _ = cv2.findContours(thresh, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
if contours:
    c = max(contours, key=cv2.contourArea)
    area = cv2.contourArea(c)
    M = cv2.moments(c)
    cx = int(M['m10'] / M['m00'])
    err_x = cx - 320
```

5.6 Autonomní řízení – sledování čáry

V předchozí části jsou popsány způsoby zpracování obrazu, výpočtu obsahu čáry a určení odchylky těžiště čáry od středu obrazu. Právě tyto hodnoty budou použity při řízení za účelem sledování čáry.

Obsah čáry v obraze je závislý na poloze kamery na robotu a šířce čáry, kterou má robot sledovat. Poloha kamery je specifikována při popisu robota. Čára by měla být dostatečně široká, aby ji bylo možné v obraze detekovat, ale zároveň by neměla zabírat většinu snímaného obrazu. Šířka čáry je zvolena dle polohy kamery. V námi vytvořených podmínkách pro sledování čáry, široké přibližně 2,5-3 cm, bylo zjištěno, že obsah čáry v obraze se pohybuje v rozmezí mezi 20000 až 30000 pixelů, což se blíží k 10 % snímaného obrazu. V některých případech se může stát, že čára je špatně viditelná nebo dokonce přerušená a její obsah se zmenší. Aby se předcházelo případnému zabloudění, kdy robot přestane detekovat čáru, tak na základě velikosti obsahu čáry v obraze je ovlivněna rychlost robota vpřed. Pokud je obsah čáry v obraze větší nebo roven 28000 pixelů, tak rychlost zůstává na 100 % své hodnoty. V případě, že obsah čáry v obraze bude nižší než 14000 pixelů, tak se rychlost vpřed sníží až na 75 % své hodnoty. Při nižší rychlosti je nižší pravděpodobnost, že robot přestane detekovat čáru ve snímaném obraze.

Odchylka je hlavním parametrem při řízení a pohybu robota je určen podle této odchylky. Znaménko určuje směr vpravo nebo vlevo a velikost odchylky je přímo úměrná rychlost rotace robota. Závislost mezi odchylkou a rychlostí rotace robota je lineární, tzn. čím větší odchylka, tím větší rychlost rotace. Řízení lze přirovnat k jednoduchému proporcionálnímu regulátoru.

Hodnota zesílení byla získána experimentálně v průběhu testování. Velikost rotace se poté pohybuje v rozmezí 0 až 0,6 rad/s.

Tento způsob řízení umožňuje také křížení dráhy robota, ale jsou zde dvě podmínky. Čáry se musí křížit v rovném úseku a musí být na sebe kolmé. Robot tyto křižovatky projede vždy rovně.

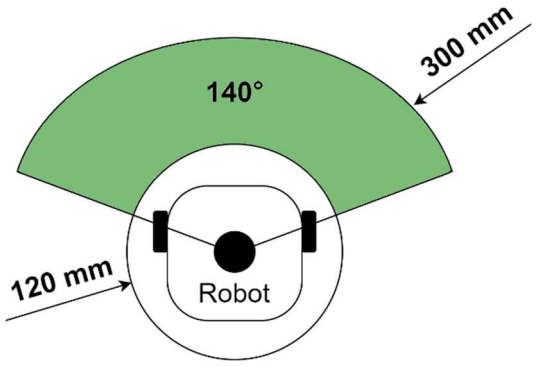
Pro případ, že robot přestane detekovat čáru v obraze a nenachází se zrovna ve fázi objíždění překážky, tak si robot pamatuje směr, vpravo nebo vlevo, svého posledního pohybu. Zastaví se a začne se v tomto směru pomalu otáčet. Otáčí se tak dlouho dokud opět ve snímaném obraze nedetekuje čáru.

5.7 Interakce s překážkami

Tato část programu je součástí metody *self.laser_callback*. Využívá laserového skeneru pro měření vzdálenosti, jehož specifikace a parametry jsou v samostatné kapitole LIDAR. Ten poskytuje data ve formě řetězce 360 hodnot udávajících vzdálenost od okolních předmětů. Na základě těchto dat je robot řízen.

Pro zajištění větší bezpečnosti a předcházení případným srážkám s překážkami na trase bylo stanoveno několik parametrů. Parametr s hodnotou 120 mm pro úplné zastavení robota, parametr s hodnotou 300 mm pro objíždění překážky a parametr s hodnotou 600 mm pro úpravu rychlosti. V případě, že robot v jakémkoliv směru detekuje těleso ve vzdálenosti nižší, než je hodnota posledního zmíněného parametru, tak sníží svoji celkovou rychlost (vpřed i rotace). Při vzdálenosti 600 mm se ještě rychlost nemění (100 %). S přibližující se překážkou se rychlost postupně snižuje až na 50 % procent své hodnoty, což odpovídá vzdálenosti 120 mm.

Vzhledem k tomu, že robot nebude nikdy couvat, ale v některých situacích se maximálně bude otáčet na místě, tak z celého rozsahu využijeme pouze 140 hodnot pro objíždění překážek. Tyto hodnoty odpovídají rozsahu -70° až +70° od osy robota. Vzdálenost, v jaké bude robot překážky objíždět, aby v průběhu objíždění nedošlo ke srážce, je určena druhým parametrem s hodnotou 300 mm. Pro zajištění bezkolizního pohybu slouží první zmíněný parametr s hodnotou 120 mm stejnou jako je hodnota minimální vzdálenosti detekce LIDARu. Pokud vzdálenost od nejbližší překážky bude nižší než hodnota parametru, tak robot úplně zastaví.



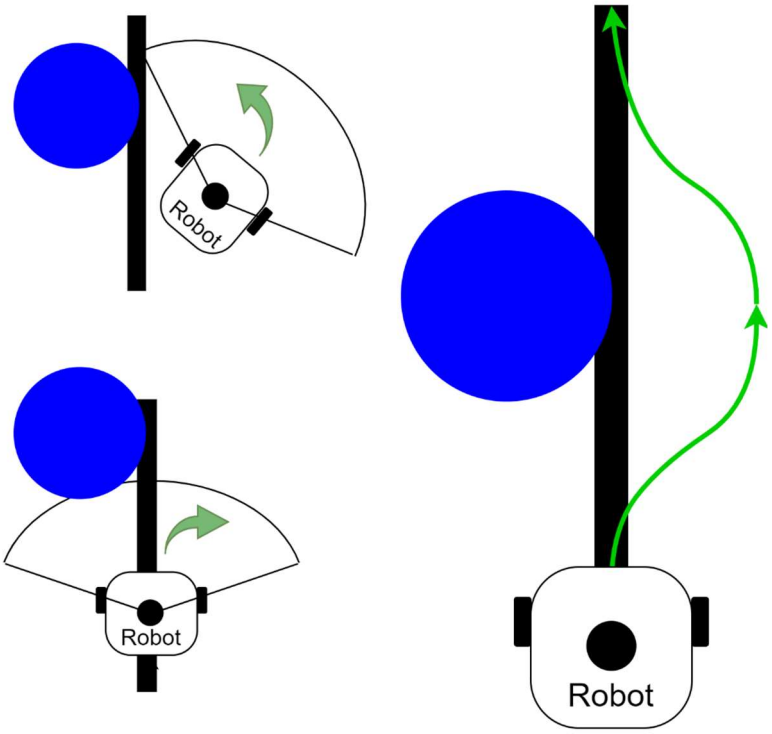
Obr. 18: Upravený pracovní prostor LIDARu

V rámci metody se tedy hledá nejnižší hodnota z již zmíněných 140 hodnot. Pokud je tato hodnota, vzdálenost od nejbližší překážky, nižší než určených 300 mm, tak se robot přepne do stavu objíždění překážek. Tento stav má přednost před sledováním čáry.

V tomto stavu je pohyb robota určen na základě polohy robota vůči překážce a vybírá se kratší objízdná trasa. Pokud je překážka detekována na pravé straně, robot začne zatáčet doleva a obráceně. Zatáčí tak dlouho, dokud překážku nepřestane detekovat v určeném pracovním prostoru LIDARu (140°, do 300 mm). Jakmile robot přestane překážku detekovat, proběhne kontrola, zda se již nenachází na čáře nebo v její blízkosti, v případě, že ne, tak se začne otáčet zpátky směrem k překážce a „hledá“ ji. Ve chvíli, kdy překážku opět detekuje, tak se proces opakuje. Jak již bylo popsáno výše, konec objíždění může nastat ve chvíli, kdy robot nedetekuje překážku a v obraze rozpozná čáru, na kterou se napojí a pokračuje po ní dále.

Ve výše popsaném procesu objíždění překážky je několikrát použito slovní spojení, že robot zatáčí/otáčí se. V průběhu celého procesu se robot stále pohybuje směrem dopředu, proto ve chvíli zatáčení se robot pohybuje po oblouku s určitým poloměrem. Velikost poloměru je závislá rychlosti rotace robota (kolem osy z). Rotace se mění dle polohy překážky vůči robotu. V případě, že je překážka přímo před robotem, rychlost rotace bude největší (poloměr nejmenší). Na druhou stranu, při „hledání“ překážky je rychlost rotace pevně daná. V obou případech je ještě poloměr pohybu robota ovlivněn jeho rychlostí vpřed. Při menší rychlosti robot zatáčí více (menší poloměr), při vyšší rychlosti zatáčí méně (větší poloměr), což je stejné jako u sledování čáry.

Robot při objíždění překážky je znázorněn na obrázku níže. Překážka se nachází po jeho levé straně, proto ji objede předpokládanou kratší cestou zprava.



Obr. 19: Obrázek pohledu shora na robota při objíždění překážky. Vlevo dole je robot v počáteční fázi detekce překážky a vlevo nahoře je v průběhu objíždění. Vpravo je znázorněná celá trasa kolem překážky

6 TESTOVÁNÍ

Testování probíhalo v simulačním prostředí Gazebo pomocí vytvořených modelů a virtuálních senzorů. Poté následovalo testování v laboratoři na reálném robotu.

6.1 Simulační prostředí Gazebo

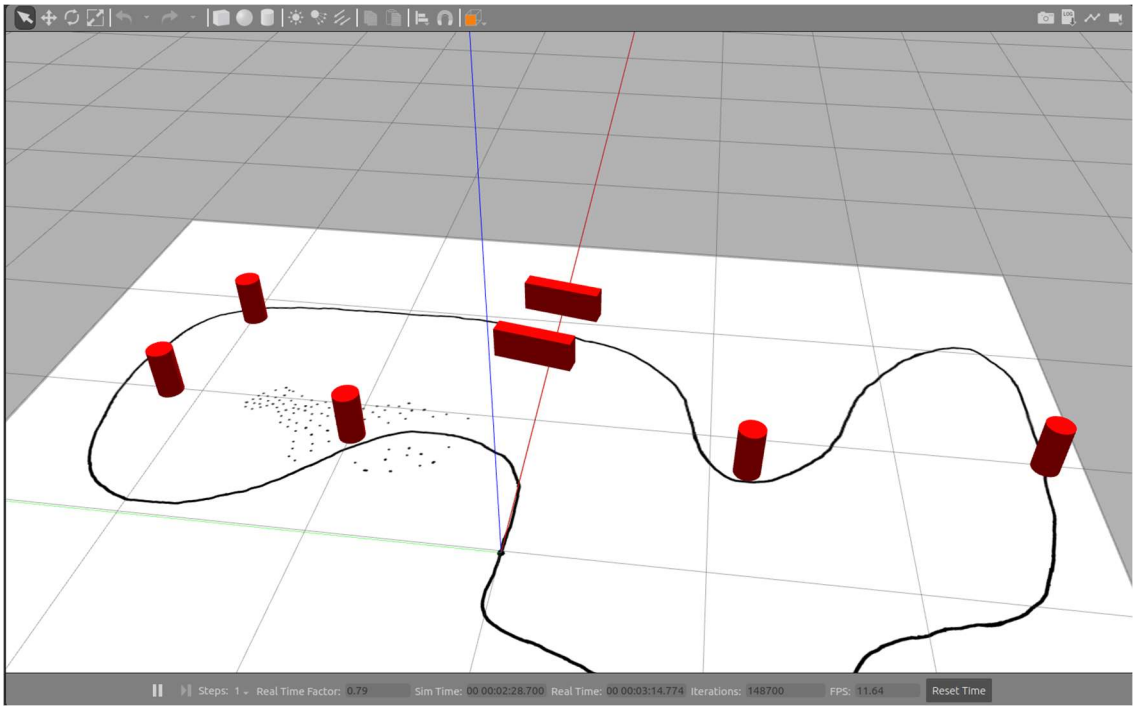
K simulaci je zapotřebí nejen samotný řídící program, ale ještě několik dalších komponent, jako je např. model samotného robota nebo model prostředí, v němž bude simulace probíhat. Model robota je součástí stažených oficiálních balíčků, nicméně není úplně kompletní. Jak již bylo zmíněno, náš robot je doplněn o kameru, která se musí do modelu implementovat, a to hned dvakrát. Poprvé přímo v URDF, kde se definuje poloha a její natočení vůči základně robota. Navíc je zde nutné provést korekci souřadnicového systému pro správnou orientaci snímaného obrazu. Kód implementující kameru do URDF:

```
<joint name="camera_joint" type="fixed">
  <origin xyz="0.04 -0.011 0.11" rpy="0 0.79 0"/>
  <parent link="base_link"/>
  <child link="camera_link"/>
</joint>
<link name="camera_link">
  <collision>
    <origin xyz="0.005 0.011 0.013" rpy="0 0 0"/>
    <geometry>
      <box size="0.015 0.030 0.027"/>
    </geometry>
  </collision>
</link>
<joint name="camera_rgb_joint" type="fixed">
  <origin xyz="0.003 0.011 0.009" rpy="0 0 0"/>
  <parent link="camera_link"/>
  <child link="camera_rgb_frame"/>
</joint>
<link name="camera_rgb_frame"/>
<joint name="camera_rgb_optical_joint" type="fixed">
  <origin xyz="0 0 0" rpy="-1.57 0 -1.57"/>
  <parent link="camera_rgb_frame"/>
  <child link="camera_rgb_optical_frame"/>
</joint>
<link name="camera_rgb_optical_frame"/>
```

Druhé přidání proběhlo v souboru definující model pro simulačním prostředí. Tady se definují vlastnosti kamery a použitý plugin s dalšími parametry. Tento soubor je nutný, aby model v simulaci odesílal data ze senzorů stejně jako reálný robot. Kód definující kameru v simulačním prostředí:

```
<gazebo reference="camera_rgb_frame">
  <sensor type="camera" name="Pi Camera">
    <always_on>true</always_on>
    <visualize>true</visualize>
    <camera>
        <horizontal_fov>1.085595</horizontal_fov>
        <image>
            <width>640</width>
            <height>480</height>
            <format>R8G8B8</format>
        </image>
        <clip>
            <near>0.03</near>
            <far>100</far>
        </clip>
    </camera>
    <plugin name="camera_controller"
    filename="libgazebo_ros_camera.so">
      <alwaysOn>true</alwaysOn>
      <updateRate>30.0</updateRate>
      <cameraName>camera</cameraName>
      <frameName>camera_rgb_optical_frame</frameName>
      <imageTopicName>rgb/image_raw</imageTopicName>
      <cameraInfoTopicName>rgb/camera_info
      </cameraInfoTopicName>
      <hackBaseline>0.07</hackBaseline>
      <distortionK1>0.0</distortionK1>
      <distortionK2>0.0</distortionK2>
      <distortionK3>0.0</distortionK3>
      <distortionT1>0.0</distortionT1>
      <distortionT2>0.0</distortionT2>
    </plugin>
  </sensor>
</gazebo>
```

Model prostředí, takzvaný svět, je k zadanému úkolu relativně jednoduchý. Na zem je umístěn obrázek s namalovanou čarou, kterou má robot sledovat a na něj je poté umístěno několik překážek. Na obrázku 20 lze vidět navržené prostředí, na kterém se testovala první varianta řídícího programu.



Obr. 20: Ukázka simulačního prostředí s mapou a překážkami

Ukázka popisu překážky:

```
<collision name='obstacle_1'>
 <pose>0.75 1.0 0.15 0 0 0</pose>
 <geometry>
  <cylinder>
    <radius>0.075</radius>
    <length>0.3</length>
  </cylinder>
 </geometry>
 <max_contacts>10</max_contacts>
</collision>
<visual name='obstacle_1'>
 <pose>0.75 1.0 0.15 0 0 0</pose>
 <geometry>
  <cylinder>
   <radius>0.075</radius>
   <length>0.3</length>
  </cylinder>
 </geometry>
 <material>
  <script>
   <name>Gazebo/Red</name>
  </script>
 </material>
</visual>
```

Simulace se spouští pomocí tzv. launch souboru. V rámci tohoto souboru je definován model robota a jeho parametry. Prostředí, které chceme spustit,

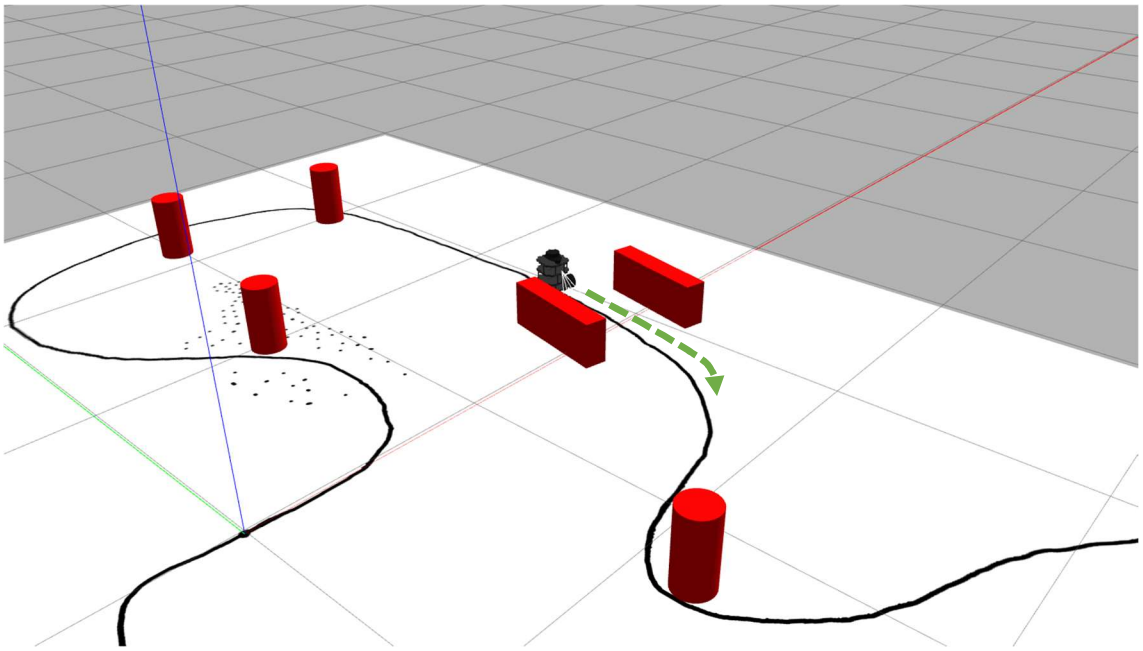
což je v našem případě vytvořený svět vhodný pro otestování zadané úlohy. Dále už zbývají jen samostatné programy (nody). Launch soubor:

```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)"
doc="model type [burger, waffle, waffle_pi]"/>
  <arg name="x_pos" default="0.0"/>
  <arg name="y_pos" default="0.0"/>
  <arg name="z_pos" default="0.0"/>
  <include file="$(find
gazebo_ros)/launch/empty_world.launch">
    <arg name="world_name" value="$(find
turtlebot3_gazebo)/worlds/muj_svet.world"/>
    <arg name="paused" value="false"/>
    <arg name="use_sim_time" value="true"/>
    <arg name="gui" value="true"/>
    <arg name="headless" value="false"/>
    <arg name="debug" value="false"/>
  </include>
  <param name="robot_description" command="$(find
xacro)/xacro --inorder $(find
turtlebot3_description)/urdf/turtlebot3_$(arg
model).urdf.xacro" />
  <node pkg="gazebo_ros" type="spawn_model"
name="spawn_urdf" args="-urdf -model turtlebot3_$(arg
model) -x $(arg x_pos) -y $(arg y_pos) -z $(arg z_pos) -
param robot_description" />
  <node name="rviz" pkg="rviz" type="rviz" args="-d
$(find
turtlebot3_gazebo)/rviz/turtlebot3_gazebo_model.rviz"/>
  <node name="joint_state_publisher"
pkg="joint_state_publisher"
type="joint_state_publisher"> <param name="use_gui"
value="TRUE" /> </node>
  <node name="robot_state_publisher"
pkg="robot_state_publisher" type="robot_state_publisher"
/>
  <node name="MyNode" pkg="robot_rasp_pkg"
type="pohyb_rasp_1.py" />
</launch>
```

Ne všechny spuštěné nody jsou k simulaci nutné. Lze si všimnout nody s názvem *rviz*, která spouští program *RViz*. Jedná se o 3D vizualizační nástroj, ve kterém lze zobrazit model robota, jeho kinematiku, data ze senzorů a mnoho dalšího. Při řešení byl využit pro kontrolu dat v průběhu simulace.

6.1.1 Simulace první varianty řídícího programu

Simulace první varianty, kdy je sledovaná čára černé barvy, proběhla bez potíží a na obrázku 21 lze vidět záběr ze simulace, kdy byly testovány různé polohy překážek a robot zrovna projíždí mezi dvěma z nich. Zelená šipka znázorňuje pohyb robota.



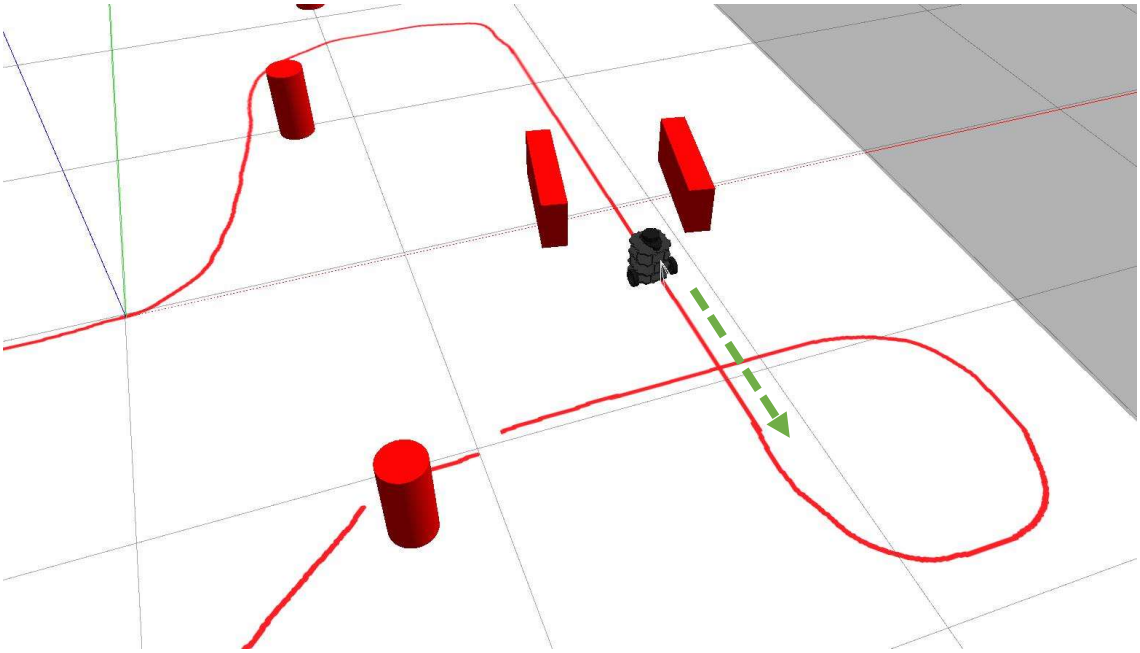
Obr. 21: Záběr ze simulace první varianty

Na obrázku 22 je ukázka záběru z kamery v průběhu simulace. Vlevo je originální obraz před zpracováním a vpravo je obraz po zpracování.

Obr. 22: Obraz z kamery v průběhu simulace první varianty, před zpracováním (vlevo) a po zpracování (vpravo)

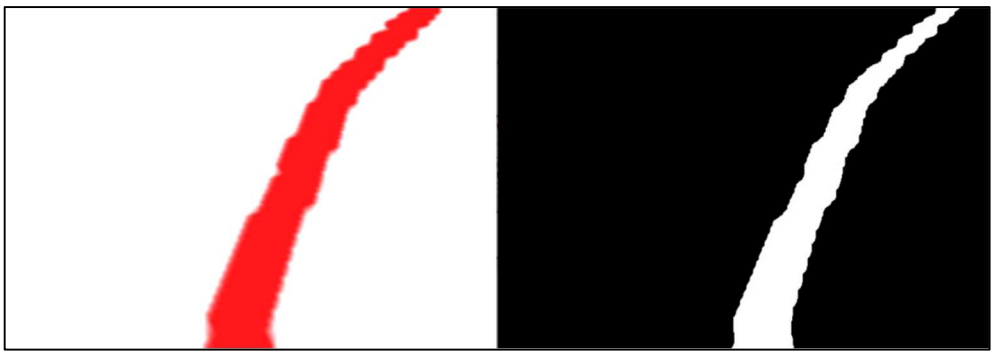
6.1.2 Simulace druhé varianty řídícího programu

Tato simulace proběhla zpětně po vytvoření druhé varianty řídícího programu při testování v reálném prostředí. Simulace sledování červené čáry:



Obr. 23: Záběr ze simulace druhé varianty

Na obrázku 24 je ukázka záběru z kamery v průběhu simulace. Vlevo je originální obraz před zpracováním a vpravo je obraz po zpracování.



Obr. 24: Obraz z kamery v průběhu simulace druhé varianty, před zpracováním (vlevo) a po zpracování (vpravo)

6.2 Reálné prostředí

Zajištění spojení a komunikace mezi počítačem s řídícím programem testovaným v simulaci a robotem probíhalo pomocí *SSH* (Secure Shell). Návod, jak toto propojení zajistit, je součástí oficiálního návodu k robotu. [18]

V kapitole 5.2 jsou zmíněny dva způsoby spouštění systému ROS a samotného programu. Oba způsoby byly úspěšně vyzkoušeny. V případě, že je systém ROS spuštěn na počítači, tak obrovskou výhodou je možnost použití různých nástrojů. Například *Rviz* nebo *rqt*, pro vizualizaci dat posílaných z robota.

6.2.1 Úprava dat z laserového skeneru

Při testování byly zjištěny určité nedostatky ve funkčnosti LIDARu. V posílaných datech se místy náhodně objevovala hodnota 0. To by znamenalo, že se nějaký předmět nachází v těsné blízkosti. Dle technické dokumentace, je schopen detekovat předměty až od vzdálenosti 120 mm. Tyto chybné hodnoty bylo nutné odstranit. Po obdržení jsou data z LIDARu zkontrolována a případné chybné hodnoty jsou nahrazeny hodnotami dostatečně velkými, aby neovlivňovaly řízení robota.

Eliminace chybných hodnot pomocí cyklu FOR:

```
for i in range(len(ranges)):
    if ranges[i] < 0.01:
        ranges[i] = 3.0
```

6.2.2 Ukládání kamerového záznamu

V průběhu testování byla snaha ukládat kamerový záznam, aby na originálním a zpracovaném obraze bylo možné zpětně vyhodnotit úspěšnost zpracování obrazu. Ukládání probíhalo pomocí funkcí z knihovny *OpenCV*. Aplikace těchto funkcí v programu:

```
self.out = cv2.VideoWriter('/home/ubuntu/output.avi',
cv2.VideoWriter_fourcc('M','J','P','G'), 10, (640,480))
self.out.write(cv_image)
```

Ukládání záznamu bylo však časově náročné, zpomalovalo chod programu a při řízení se projevovalo zpožděním. Pohyb robota pak nebyl plynulý, měl tendenci přetáčet zatáčky apod. Při vypnutí ukládání záznamu, kdy se zpoždění neprojevovalo, byl pohyb robota mnohem plynulejší a přesnější. Aby bylo možné dosáhnout podobné plynulosti a přesnosti pohybu robota, bylo nutné celkové snížení rychlosti (vpřed i rotace) až na polovinu původních hodnot.

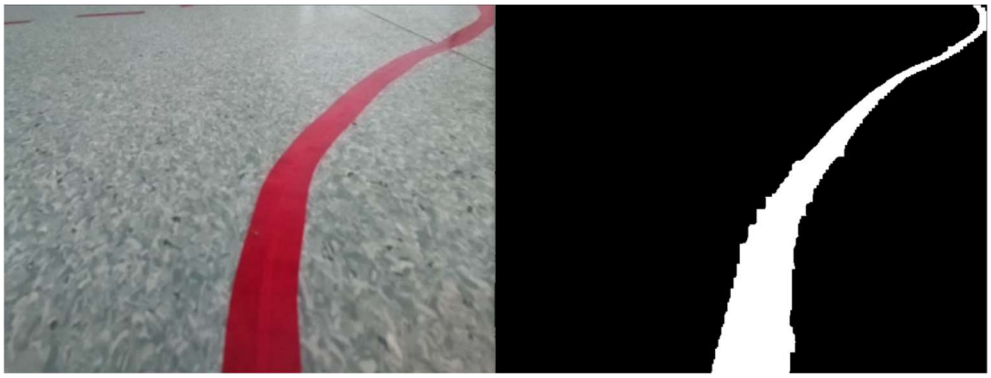
6.2.3 Porovnání variant řídícího programu

Již několikrát bylo zmíněno, že druhá varianta vznikla v průběhu testování na reálném robotu. První varianta zpracování obrazu fungovala, ale bez dalších hardwarových úprav nebyla vhodná pro vytvořené podmínky. Černá čára pro první variantu vznikla nalepením černé izolační pásky na podlahu. Při nerovnoměrném osvětlení a měnících se světelných podmínkách se páska místy natolik leskla, že při převodu do odstínů šedi, se jevila dokonce světlejší než její okolí. Byla vyzkoušena také zmíněná adaptivní metoda prahování. Většinu času byl její výstup lepší než u jednoduché metody prahování, ale v určitých situacích byl výsledek přesně opačný. Adaptivní metoda vůbec nesplnila svůj účel a výstup byl dokonce horší než u jednoduché metody prahování. Na obrázku 25 lze vidět obraz z kamery před a po zpracování při testování první varianty. V situacích, kdy se čára leskla, bylo možné po zpracování obrazu vidět pouze její část. Tato část se nacházela přímo před robotem, z čehož robot usoudil, že čára vede rovně, přestože se nacházel před zatáčkou a měl by už pomalu začít zatáčet.

Obr. 25: Obraz z kamery při testování první varianty v situaci, kdy se čára leskla, originál (vlevo) a po zpracování (vpravo)

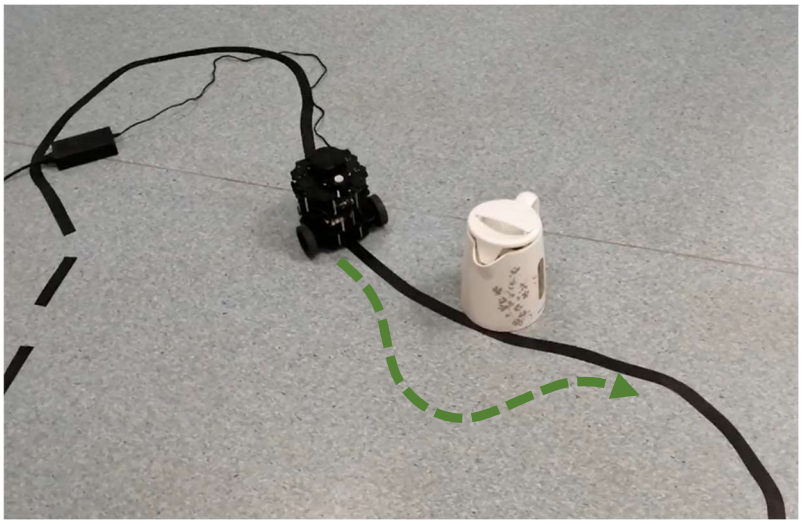
Dále si lze povšimnout černé skvrny v levém dolní rohu. To je část napájecího kabelu, který musel robot při jízdě tahat za sebou, protože nebyla k dispozici funkční baterie. Při změně úhlu kamery, aby kabel nebyl součástí obrazu, by detekovaná část čáry byla ještě menší než na obrázku 25.

Řešení tohoto problému mohou být různá, například umístění přídavných světel na robota, aby se světelné podmínky, pokud možno měnily co nejméně nebo doplnění dalších senzorů. Tyto řešení by zahrnovaly přidání dalšího hardwaru a investice. Proto vznikla navíc druhá varianta, kdy černá páska byla nahrazena červenou. Červená páska se místy také leskla, ale ne tolik jako černá páska. Princip zpracování obrazu je také trochu odlišný, tudíž zde problém nenastal. Dále na obr. 26 je opět ukázka z kamery v průběhu testování, kdy se červená páska místy jeví světlejší a zpracování obrazu je přesto úspěšné.

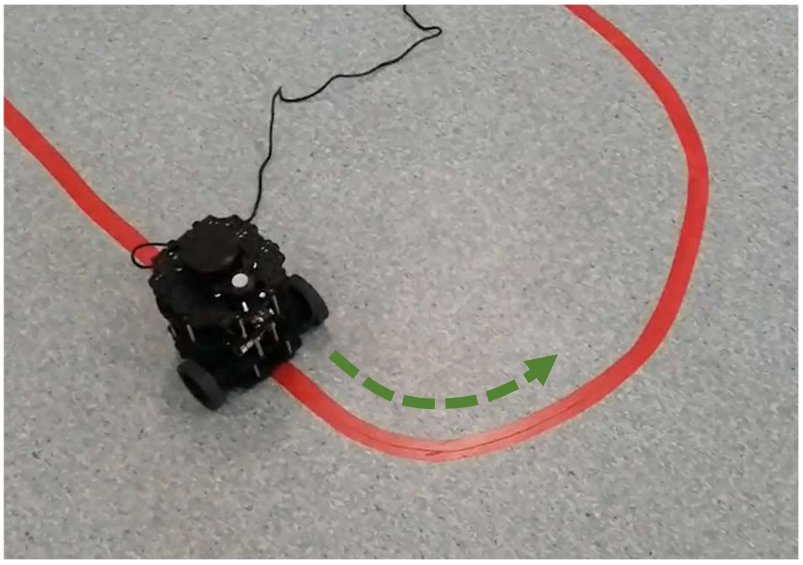


Obr. 26: Obraz z kamery při testování druhé varianty, originál (vpravo) a po zpracování (vlevo)

Na obr. 27 a 28 jsou fotografie pořízené v průběhu testování doplněné o zelenou šipku znázorňující očekávaný pohyb robota:



Obr. 27: Fotografie v průběhu testování s černou čarou



Obr. 28: Fotografie v průběhu testování s červenou čarou

6.3 Porovnání simulace a reálného prostředí

Pohyb robota v simulaci byl samozřejmě plynulejší než pohyb robota v reálném prostředí, což potvrdilo předpoklady. Rozdíly, které odlišovaly simulační prostředí od reality jsou zmíněny už v předchozí kapitole. LIDAR v simulaci neposkytoval žádná chybná data, proto nebylo potřeba provádět žádnou úpravu dat. Ukládání záznamu z kamery v průběhu simulace probíhalo bez zpoždění a nebylo nutné provádět žádnou korekci rychlosti robota. Největší rozdíl mezi simulací a testováním v reálném prostředí byl spojen s vlastnostmi objektů a světelnými podmínkami. Světlo v simulaci nevychází z jednoho bodu, ale intenzita světla je všude stejná. Proto žádný z předmětů poté nevytváří stín, který by měnil světelné podmínky při sledování čáry. Díky stejné intenzitě světla a nulové odrazivosti nakreslené čáry byl její odstín při zpracování obrazu vždy stejný. Nikdy pak nenastala situace, že by robot detekoval pouze určitou část čáry v obraze. Simulace byla proto úspěšná pro obě varianty, s černou i červenou čarou.

Zaměříme-li se na funkčnost řídícího programu a pohyb robota, tak simulace splnila svůj účel. Byla dostatečně věrnou kopií reálného prostředí a urychlila návrh řídícího programu díky možnosti rychlého testování jednotlivých částí programu.

7 ZÁVĚR

Úvodní rešeršní část se zabývá tématem mobilní robotiky se zaměřením na mobilní roboty pohybující se po zemi. Seznamuje čtenáře se základním rozdělením dle způsobu pohybu, který je dán jejich konstrukcí a způsobem řízení. Zmíněno je také využití těchto robotických aplikací v praxi včetně ukázek nejenom z průmyslové oblasti.

Následuje popis a přiblížení konkrétní mobilní robotické platformy s názvem TurtleBot3 Burger společnosti ROBOTIS. Tento mobilní robot byl navíc vůči základní sadě rozšířen o kameru Raspberry Pi.

Další část práce je zaměřena na Robotický Operační Systém a simulační prostředí Gazebo. Popsány jsou jejich klíčové vlastnosti, funkce a výhody pro vývoj a testování robotických systémů. ROS poskytuje robustní infrastrukturu pro komunikaci mezi robotickými komponentami a umožňuje efektivní vývoj softwaru pro různé robotické aplikace. Simulační prostředí Gazebo pak nabízí prostor pro simulaci a testování chování robotů v realistickém virtuálním prostředí. Jejich kombinace přináší významnou podporu pro výzkum a vývoj v oblasti mobilní robotiky.

Praktická část obsahovala zprovoznění mobilní robotické platformy TurtleBot3 Burger a návrh řídícího programu pro zadanou úlohu, kterou je sledování čáry a vyhýbání se překážkám na trase. Realizace zahrnovala několik kroků od instalace Robotického Operačního Systému včetně simulačního prostředí Gazebo až po otestování zadané úlohy v laboratoři. Při zprovozňování robota a doplnění kamery nebylo možné postupovat přesně podle oficiálního návodu, proto je postup doplněn o různé postřehy a tipy.

Následovala tvorba řídícího programu v systému ROS s využitím počítačového vidění, ale také s využitím laserového skeneru. Řídící algoritmus je kombinací těchto technologií. Robot spolehlivě sleduje vyznačenou trasu díky počítačového vidění s podporou LIDARu při objíždění překážek, které se na trase mohou vyskytovat.

Ke zpracování obrazu je využita volně dostupná knihovna OpenCV. Obraz z kamery je převeden do binární podoby a následně je v něm detekována čára. Podle polohy detekované čáry v obraze je určena odchylka, která je dále využita při řízení. Robot je řízen diferenciálně pomocí dvou samostatných pohonů, proto je možné odchylku využít při řízení jako vstup do proporcionálního regulátoru.

Překážky jsou objížděny na základě polohy vůči robotu. Robot dokáže zvolit efektivní směr objížďky překážek a udržet si při tom bezpečný odstup. V průběhu objíždění kontroluje, zda se již nenachází opět na trase, po které má pokračovat dále. V případě neúspěšného nalezení trasy si robot pamatuje poslední směr svého pohybu a je schopen se vrátit.

Funkčnost daného řešení byla úspěšně otestována v simulačním prostředí a následně také v reálném prostředí laboratoře.

SEZNAM POUŽITÉ LITERATURY

[1] DUDEK, Gregory a Michael JENKIN. *Computational principles of mobile robotics*. 2nd ed. New York: Cambridge University Press, 2010. ISBN 978-0-521-87157-0.

[2] KOŠNAR, Karel. *Mobilní robotika* [online]. B.m.: Praha: České Vysoké Učení Technické v Praze, Fakulta elektrotechnická, Katedra kybernetiky. 2007. Dostupné z: http://www.roznovskastredni. cz/dwnl/pel2007/06/Kosnar

[3] GAUTAM, Avinash a Sudeept MOHAN. A review of research in multi-robot systems. In: *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)*: *2012 IEEE 7th International Conference on Industrial and Information Systems (ICIIS)* [online]. Chennai, India: IEEE, 2012, s. 1–5 [vid. 2023-05-06]. ISBN 978-1-4673-2605-6. Dostupné z: doi:10.1109/ICIInfS.2012.6304778

[4] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. *Probabilistic robotics*. Cambridge, Mass: MIT Press, 2005. Intelligent robotics and autonomous agents. ISBN 978-0-262-20162-9.

[5] KOLÍBAL, Zdeněk. *Roboty a robotizované výrobní technologie*. První vydání. Brno: Vysoké učení technické v Brně - nakladatelství VUTIUM, 2016. ISBN 978-80-214-4828-5.

[6] FILIP, Jakub. *Návrh a implementace řídícího programu pro mobilní robotickou platformu Turtlebot3 Burger*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí práce Ing. Roman Parák.

[7] SIEGWART, Roland a Illah Reza NOURBAKHSH. *Introduction to autonomous mobile robots*. Cambridge, Mass: MIT Press, 2004. Intelligent robots and autonomous agents. ISBN 978-0-262-19502-7.

[8] Legacy Robots. *Boston Dynamics* [online]. [vid. 2023-05-22]. Dostupné z: https://www.bostondynamics.com/legacy

[9] *Centaur®* | *Teledyne FLIR* [online]. [vid. 2023-05-22]. Dostupné z: https://www.flir.eu/products/centaur?vertical=ugs&segment=uis

[10] MARS.NASA.GOV. Curiosity Rover, 3D Model. *NASA Mars Exploration* [online]. [vid. 2023-05-22]. Dostupné z: https://mars.nasa.gov/resources/24584/curiosity-rover-3d-model

[11] AGO, Dion66in #steemhunt • 5 Years. Vector - Autonomous Omni-Directional, Mobile Robot. *Steemit* [online]. [vid. 2023-05-22]. Dostupné z: https://steemit.com/steemhunt/@dion66/vector-autonomous-omni-directional-mobile-robot

[12] ROBOTNIK. What is the difference between AGVs vs. AMR? *Robotnik* [online]. 28. únor 2022 [vid. 2023-05-08]. Dostupné z: https://robotnik.eu/what-is-the-difference-between-agvs-vs-amr/

[13] *LD-60/90* [online]. [vid. 2023-05-22]. Dostupné z: https://industrial.omron.cz/cs/products/ld-60-90

[14] STAN, Alexandru-Calin. A decentralised control method for unknown environment exploration using Turtlebot 3 multi-robot system. In: *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)*: *2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI)* [online]. Ploiesti, Romania: IEEE, 2022, s. 1–6 [vid. 2023-05-06]. ISBN 978-1-66549-535-6. Dostupné z: doi:10.1109/ECAI54874.2022.9847497

[15] *ROS Alternatives - C++ Robotics | LibHunt* [online]. 12. únor 2023 [vid. 2023-05-08]. Dostupné z: https://cpp.libhunt.com/ros-alternatives

[16] *TurtleBot* [online]. [vid. 2023-05-06]. Dostupné z: https://www.turtlebot.com/

[17] PRESTASHOP 1.5. *RoboSklep* [online]. [vid. 2023-05-06]. Dostupné z: https://robosklep.com

[18] ROBOTIS e-Manual. *ROBOTIS e-Manual* [online]. [vid. 2023-05-06]. Dostupné z: https://emanual.robotis.com/

[19] LTD, Raspberry Pi. Buy a Raspberry Pi 3 Model B. *Raspberry Pi* [online]. [vid. 2023-05-06]. Dostupné z: https://www.raspberrypi.com/products/raspberry-pi-3-model-b/

[20] *ROBOTIS STORE | Robot is...* [online]. [vid. 2023-05-06]. Dostupné z: https://www.robotis.us/

[21] LTD, Raspberry Pi. Buy a Raspberry Pi Camera Module 2. *Raspberry Pi* [online]. [vid. 2023-05-06]. Dostupné z: https://www.raspberrypi.com/products/camera-module-v2/

[22] *Raspberry Pi Camera Module 2 - 8MP* [online]. [vid. 2023-05-21]. Dostupné z: https://www.kiwi-electronics.com/en/raspberry-pi-camera-module-2-8mp-2359

[23] *ROS: Home* [online]. [vid. 2023-05-06]. Dostupné z: https://www.ros.org/

[24] BARTOŠ, Pavel. *Turtlebot v rámci frameworku ROS*. Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky.

[25] JOSEPH, Lentin a Jonathan CACACE. *Mastering ROS for robotics programming: design, build, and simulate complex robots using the Robot Operating System*. Second edition. Birmingham Mumbai: Packt, 2018. ISBN 978-1-78847-895-3.

[26] *ROS 2 Documentation — ROS 2 Documentation: Humble documentation* [online]. [vid. 2023-05-06]. Dostupné z: https://docs.ros.org/en/humble/

[27] *Documentation - ROS Wiki* [online]. [vid. 2023-05-06]. Dostupné z: http://wiki.ros.org/

[28] *noetic - ROS Wiki* [online]. [vid. 2023-05-06]. Dostupné z: https://wiki.ros.org/noetic

[29] *ROS Documentation* [online]. [vid. 2023-05-08]. Dostupné z: https://docs.ros.org/

[30] *Changes between ROS 1 and ROS 2* [online]. [vid. 2023-05-06]. Dostupné z: http://design.ros2.org/articles/changes.html

[31] ELMOFTY, Omar. ROS2 — how is it better than ROS1. *Medium* [online]. 4. srpen 2022 [vid. 2023-05-06]. Dostupné z: https://medium.com/@oelmofty/ros2-how-is-it-better-than-ros1-881632e1979a

[32] *What is DDS?* [online]. [vid. 2023-05-06]. Dostupné z: https://www.dds-foundation.org/what-is-dds-3/

[33] Introduction to ROS. *VNAV* [online]. [vid. 2023-05-06]. Dostupné z: https://vnav.mit.edu/labs/lab2/ros101.html

[34] PYO, YoonSeok, HanCheol CHO, RyuWoon JUNG a TaeHoon LIM. *ROS Robot Programming* [online]. B.m.: ROBOTIS, 2017. ISBN 979-11-962307-1-5. Dostupné z: http://community.robotsource.org/t/download-the-ros-robot-programming-book-for-free/51

[35] *Gazebo* [online]. [vid. 2023-05-06]. Dostupné z: https://classic.gazebosim.org/

[36] OpenCV. *OpenCV* [online]. [vid. 2023-05-07]. Dostupné z: https://opencv.org/

[37] *cv_bridge - ROS Wiki* [online]. [vid. 2023-05-07]. Dostupné z: http://wiki.ros.org/cv_bridge

[38] Python | Thresholding techniques using OpenCV | Set-1 (Simple Thresholding). *GeeksforGeeks* [online]. 6. květen 2019 [vid. 2023-05-10]. Dostupné z: https://www.geeksforgeeks.org/python-thresholding-techniques-using-opencv-set-1-simple-thresholding/

SEZNAM OBRÁZKŮ A TABULEK

Seznam obrázků:

Seznam tabulek: