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

NÁVRH A IMPLEMENTACE ŘÍDÍCÍHO PROGRAMU PRO MOBILNÍ ROBOTICKOU PLATFORMU TURTLEBOT3 BURGER

DESIGN AND IMPLEMENTATION OF CONTROL PROGRAM FOR MOBILE ROBOT PLATFORM TURTLEBOT3 BURGER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Jakub Filip
AUTHOR

VEDOUCÍ PRÁCE — Ing. Roman Parák
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Jakub Filip**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**Ing. Roman Parák**
Akademický rok:	2019/20

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:

Návrh a implementace řídícího programu pro mobilní robotickou platformu Turtlebot3 Burger

Stručná charakteristika problematiky úkolu:

Práce bude zahrnovat rešerši v oblasti mobilních robotů a seznámení se s mobilními roboty společnosti Robotis, blíž popíše zvoleného robota Turtlebot3 Burger. Teoretická část práce bude také zahrnovat rešerši frameworku ROS a simulačního prostředí Gazebo.
Předmětem práce bude sestavení robotické stavebnice, konfigurace frameworku ROS a návrh řídícího programu pro vybranou laboratorní úlohu. Závěr práce bude věnován implementaci návrhu řídícího programu a ověření funkčnosti vytvořeného řešení.
Práce předpokládá aktivní přístup studenta a nutnost práce v laboratoři.

Cíle bakalářské práce:

– Nastudujte problematiku mobilních robotů. Zpracujte přehled aktuálního stavu v dané oblasti.
– Proveďte rešerši mobilních robotů od společnosti Robotis, blíže popište zvoleného mobilního robota Turtlebo3 Burger.
– Proveďte rešerši v oblasti využití Robotického operačního systému (ROS) a simulačního prostředí Gazebo.
– Sestavte robotickou stavebnici Turtlebot3 Burger.
– Proveďte konfiguraci frameworku ROS a mobilní robotické platformy Turtlebo3 Burger.
– Navrhněte řídicí program pro vybranou laboratorní úlohu.
– Implementujte návrh řídícího programu.
– Ověřte funkčnost vytvořeného řešení pomocí simulace a na reálnem robotu.

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.

CASADO, F.: Development of control programs for the Turtlebot robot using ROS (Robot Operating System). Technická zpráva, Grupo de Robótica. Escuela de Ingenierías Industrial e Informática, 2012.

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

THRUN, S., BURGARD, W. and FOX, D. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Cílem bakalářské práce je návrh a implementace řídícího programu pro mobilní robotickou platformu TurtleBot3 Burger. V teoretické části bakalářské práce je vymezena problematika mobilní robotiky, s bližším pohledem na různé možnosti lokomoce, včetně ukázek z průmyslové praxe. Série mobilních robotů TurtleBot3 patří mezi robotické platformy, distribuované společností ROBOTIS, kde jejich hlavním znakem je kompatibilita s Robotickým Operačním Systémem (ROS). Jádro tohoto systému spadá pod BSD licenci, zaručující otevřený zdrojový kód. Integrace ROS s modelem TurtleBot3 Burger poskytuje volně přístupné robustní knihovny, tvořící základ pro pochopení ovládání diferenciálně řízeného robotu skrze ROS. V praktické části je provedena montáž a konfigurace robotické stavebnice TurtleBot3 Burger, včetně představení klíčových funkcionality této mobilní platformy, a návrh vlastního řešení. Závěr obsahuje odůvodnění zmíněného návrhu a výstup po jeho implementaci na reálném robotu.

ABSTRACT

The aim of the bachelor thesis is the design and implementation of a control program for the mobile robotic platform TurtleBot3 Burger. The theoretical part of the bachelor's thesis defines the issue of mobile robotics, with a closer look at the various possibilities of locomotion, including examples from industrial practice. The TurtleBot3 mobile robot series belongs to the robotic platforms distributed by ROBOTIS, where their main feature is compatibility with the Robotic Operating System (ROS). The core of this system is licensed under the BSD, which guarantees open source code. The integration of ROS with the TurtleBot3 Burger model provides freely accessible robust libraries, which form the basis for understanding the control of a differentially controlled robot through ROS. In the practical part, the assembly and configuration of the robotic kit TurtleBot3 Burger is performed, including the introduction of key functionalities of this mobile platform, and the design of your own solution. The conclusion contains the justification of the mentioned proposal and the output after its implementation on a real robot.

KLÍČOVÁ SLOVA

Mobilní robotika, lokomoce, ROBOTIS, Open Robotics, operační systém pro roboty, ROS, OpenCV, počítačové vidění, zpracování obrazu, detekce čáry jízdního pruhu, autonomní řízení

KEYWORDS

Mobile robotics, locomotion, ROBOTIS, Open Robotics, operating system for robots, ROS, OpenCV, computer vision, image processing, lane line detection, autonomous driving

BIBLIOGRAFICKÁ CITACE

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.

PODĚKOVÁNÍ

Poděkování směřuje k vedoucímu bakalářské práce, Ing. Romanu Parákovi, především za vzájemnou spolupráci při návrhu tématu, vytvořeného na základě společných nápadů a požadavků. Také za zajištění nepřetržitého přístupu do laboratoře, k zařízením potřebným pro zdárný návrh a implementaci řídícího programu na reálném robotu

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. Romana Paráka a s použitím zdrojů uvedených v seznamu literatury.

V Brně dne 26. 6. 2020 .....

Jakub Filip

OBSAH

1 ÚVOD

Cílem bakalářské práce je navrhnout řídící program pro mobilní robotickou platformu TurtleBot3 Burger, implementovat zvolené řešení a ověřit funkčnost na reálném robotu prostřednictvím vytvořené laboratorní úlohy.

Úvodní část je zaměřena na rešerši mobilní robotiky, především na klasifikaci této oblasti robotiky a vymezením problematiky daného oboru. V rámci dané rešerše se pojednává o různých přístupech řešení lokomoce a možnostech jejich aplikace. Součástí rešerše mobilní robotiky je představení „inteligentních“ servomotorů DYNAMIXEL a mobilních robotických platforem od společnosti ROBOTIS se zaměřením na sérii mobilních robotů TurtleBot3, především na model Burger.

Jádrem mobilních robotických platforem od společnosti ROBOTIS je Robotický Operační Systém (ROS). Úvod druhé oblasti rešerše charakterizuje, resp. softwarově zařazuje výše zmíněný systém, přičemž popisuje čtenáři klíčové vlastnosti, seznamuje ho s terminologií, základními principy, nástroji příkazového řádku, nástroji s grafickým rozhráním, kde je blíže přiblížen vizualizační nástroj RViz a simulační prostředí Gazebo, včetně detailního provedení procesem instalace a konfigurace všech nezbytných komponent, potřebných pro návrh řídícího programu.

Praktická část se skládá ze sestavení robotické stavebnice TurtleBot3 Burger, instalace potřebných softwarových závislostí pro mobilní robot a vzdálené PC, včetně konfigurace obou zařízení pro zajištění vzájemné komunikace, a tedy umožnění ovládání robotu prostřednictvím vzdáleného PC.

Návrh řídícího programu vychází z dostupných funkcionalit mobilního robotu TurtleBot3 Burger a zhodnocení všech možností s ohledem na kvalitu implementace v rámci výsledné laboratorní úlohy.

2 MOBILNÍ ROBOTIKA

2.1 Problematika a motivace

Mobilní robotika je oblast výzkumu, která se zabývá řízením autonomních a polo-autonomních vozidel. To, co odlišuje mobilní robotiku od jiných významných oblastí, jakými jsou robotické manipulátory, umělá inteligence a počítačové vidění, je důraz na problémy spojené s porozuměním rozsáhlého prostoru. Schopnost cílené navigace a inteligentního pohybu v daném prostředí je mnohonásobně snadnější v laboratorních simulovaných podmínkách než v rozsáhlém, komplexním a proměnném prostředí, kde nepatrné změny a interakce, mezi více složitými složkami, mohou vést ke vzniku mimořádného chování, které lze jen těžko předvídat. Chovat se inteligentně v rozsáhlém, komplexním a proměnném prostoru neznamená pouze postupné získávání a vyhodnocování informací, odhad polohy, rozpoznávání známých a důležitých objektů nebo míst v reálném čase, nýbrž schopnost adaptovat se na dané prostředí souběžnou kooperativní činností těchto funkcí.

Mobilní roboty nejsou jen sbírkou algoritmů pro snímání, uvažování a pohyb prostorem, ale jsou fyzickým provedením těchto algoritmů, myšlenek a teoretických konceptů, které chceme co nejvíce ztotožnit s nejasnostmi skutečného světa.

V kontextu neustálého úsilí lidstva konstruovat schopnější stroje – stroje, které odpovídají lidským schopnostem nebo je dokonce překonávají – je klíčovou překážkou vývoj systémů, které projevují mobilitu. Schopnost mobility a provádění akcí v prostoru lze nejvíce ocenit při pozorování sofistikovaných biologických organismů, které se mohou pohybovat se ve svém prostředí a plnit prostorově distribuované úkoly bez větších obtíží, neboť k tomu mají predispozice, jež je snahou uměle vytvořit a implementovat do „mozku“ robotu. Stejně jako vývoj kola je vývoj mobilních robotů důležitým odrazovým můstkem ve vývoji sofistikovaných strojů. [1]

2.2 Úvod a klasifikace mobilních robotů

Mobilní robotika je interdisciplinární oblast výzkumu, která zahrnuje strojírenství (konstrukce vozidla, mechanismy lokomoce), počítačové vědy (algoritmy snímání a plánování), elektrotechniku (integrace systému, senzory a komunikace), kognitivní psychologii, vnímání, neurovědy (vhled toho, jak biologické organismy řeší podobné problémy) a mechatroniku (kombinace strojního inženýrství s informatikou, počítačovým inženýrstvím a elektrotechnikou).

Mobilní roboty lze tedy klasifikovat technickým systémem, kde akční členy (pohony), procesory, uživatelská rozhraní, senzory a komunikační mechanismy umožňující mobilnímu robotu pracovat, musí být integrovány tak, aby celý systém fungoval jako celek.

V současné době je mnoho mobilních robotických systémů, které jsou v zásadě výzkumnými a experimentálními vozidly, nicméně, podstatná část mobilních robotických systémů je nasazena v domácích nebo průmyslových prostředích. Průmyslové aplikace, ve kterých byly současné mobilní roboty úspěšně nasazeny, se vyznačují jedním nebo více z následujících atributů: nepřítomnost lidského operátora (často kvůli nepřístupnosti), potenciálně vysoké náklady, dlouhé pracovní cykly a potřeba tolerovat podmínky prostředí, které by pro člověka nemusely být přijatelné. [1]

2.3 Mobilita a řešení lokomoce

Lokomoce je proces, kterým se autonomní robot nebo vozidlo pohybuje. Aby bylo dosaženo pohybu, musí docházet k silovému působení. Studium pohybu, ve kterém jsou tyto síly modelovány, se nazývá dynamika, zatímco kinematika je studium klasifikace a popisu pohybu, aniž by se braly v úvahu síly, které pohyb ovlivňují, tzn. kinematika se zabývá geometrickými vztahy a vytvářením kinematických mechanismů, které popisují pohyb daného robotického systému, zatímco dynamika zahrnuje energie a rychlosti spojené s těmito pohyby.

Existuje tedy mnoho různých lokomočních řešení lišících se nejen v kinetickém a dynamickém modelu, ale také ve složitosti; u kolových nebo pásových mobilních robotů jsou tato řešení snadná v porovnání s biomechanickými principy mobility [1].

2.3.1 Biomechanické principy mobility

Kolo je lidský vynález, zatímco schopnost pohybu živých organismů je vrozená vlastnost vybudovaná evolucí; příkladem může být lidský organismus, který lze vnímat jako komplexní systém spolupracujících soustav, umožňující dosáhnout složitého a různorodého pohybu.

Obecně všechny konstrukce, ať už kráčejících nebo plazících se robotů, vycházejí z biologického vzoru (princip biomechaniky pohybového ústrojí živých organismů).

Velkou skupinou této oblasti robotiky jsou kráčející roboty, které vycházejí z pohybového ústrojí savců a hmyzu. Můžeme se tedy bavit o stovkách různých konstrukčních řešeních dvou až osminohých robotů, kde efektivnost jejich použití lze specifikovat na základě následujících parametrů [2, 3, 4]:

- **Stabilita:** počet kontaktních míst, plocha kontaktního místa, úhel kontaktu mezi chodidlem a podložkou, tření, těžiště (rozložení hmotnosti), statická a dynamická stabilizace – především při plnění pracovních úloh, kdy předpokládáme přídavné zatížení (adaptace procentuálního využití nosnosti robotu).
- **Pohybové schopnosti:** možnosti pohybu a manévrovatelnosti při překonávání překážek (výška, hloubka, profil překážek, členitost terénu, materiál terénu – voda.

- **Přizpůsobitelnost prostředí:** odolnost vůči vnitřním a vnějším vlivům prostředí, tzn. mechanickým, chemickým, radiačním, klimatickým, technologickým apod.
- **Způsob pohybu v prostoru:** schopnost inteligentního chování v prostoru a způsob překonávání překážek v souvislosti s pracovními úkony a současným dodržením norem bezpečnosti a provozu.
- **Operačně a provozně přívětivé ovládání:** jednoduché a spolehlivé řízení, snadná realizovatelnost konfiguračních změn s ohledem na požadavky robotické aplikace se zohledněním norem bezpečnosti a provozu; jednoduchá údržba, obsluha atp.
- **Energetická nezávislosti:** k zajištění volného pohybu robotu prostředím, musí být konstrukce robotu navrhnuta i z pohledu energetického managementu, tzn. musí si s sebou nést zdroj energie, který je nutný k napájení akčních členů, řídícího systému, senzorů atp., zohlednit se mj. musí celková spotřeba energie, kapacita baterie, typ baterie, způsob napájení, aby nedošlo kvůli neočekávanému výpadku potřebné energie k ohrožení lidí, prostředí a technického vybavení.

Biologicky inspirované mechanismy jsou obecně složité, a proto se naskytují otázky, zda je vývoj takových systémů poháněn lidskou cílevědomostí, anebo mají skutečně praktický význam.

Existuje mnoho aplikací, zvláště pro prostředí, která jsou těžce dostupná a pro člověka nebezpečná a zároveň – jsou kvůli charakteru překážek nedosažitelná pásovými nebo kolovými mobilními roboty. Na základě získaných informací je možné shrnout obecné výhody a nevýhody spojené se skupinou kráčejících robotů [3, 4]:

Výhody:

- mohou překonávat relativně vysoké překážky, pohybovat se po schodech, překračovat příkopy a prohlubně, pohybovat se po extrémně členitém terénu, překonávat strmé svahy;
- mohou zdolávat terén, který je neřešitelný z hlediska kolových a pásových podvozků mobilních robotů;
- různorodost konstrukce umožňuje variabilitu použití a snížení možnosti poškození povrchu (podložky);

Nevýhody:

- vyšší počet koordinovaně řízených stupňů volnosti; řízení momentu setrvačnosti robotu;
- robot musí vidět podrobnou strukturu terénu;
- vyšší počet akčních členů, senzorů atd., a tedy složitější řídící systém, ať už z hlediska hardwaru, tak i softwaru; konstrukční a výrobní složitost;

Obr. 1: Robotická řešení biomechanismů od společnosti Boston Dynamics: SPOT (vlevo), ATLAS (vpravo) [5]

2.3.2 Mobilní roboty s pásovým podvozkem

Koncepce pásového podvozku vychází ze známých konstrukcí, např. vojenských, zemědělských, stavebních, sněžných vozidel atp. V mobilní robotice je využívána především v náročných robotických aplikacích, a při speciálním charakteru úloh, např. zbraňové, monitorovací, protiteroristické, manipulační systémy apod. Jejich použití nalezneme tedy převážně v nestrojírenských oblastech.

Modul pásu je z hlediska konstrukce samostatná funkční skupina, tvořící lokomoční ústrojí mobilního robotu, které lze členit do funkčních podskupin: pás (běhounová část, která je v kontaktu s povrchem), hnací kolo (spojené s pohonem robotu), hnané kolo (mechanická podpora ve smyslu tuhosti), vodicí kolo a napínací kolo, které jsou společně s hnacím a hnaným kolem prvky ze soustavy pojezdových kol, dávající tvar celého pohybového mechanismu a umožňující konstrukci pásového podvozku s proměnnou geometrií v závislosti na aktuálně zdolávaném terénu.

Celý mechanismus umožňuje zvýšenou manévrovatelnost v členitých terénech, kvůli velkému množství kontaktních bodů s povrchem, a tedy značným třením – řízení smykem/skluzem. Tento princip se liší od kolových podvozků, kde předpokládáme, že kola se nesmí smýkat po povrchu, tzn. bavíme se o alternativní formě řízení, kde k přeorientování robotu dochází v závislosti na rychlosti a směru otáčení hnaných kol.

Řízení skluzem/smykem je nevýhodou především v oblasti autonomních mobilních robotů, důvodem je obtížné určování středu otáčení, přesné polohy a orientace robotu, kdy samotné výpočetní operace probíhají s velkými nepřesnostmi, a to především díky proměnným hodnotám tření vzhledem k danému povrchu. Proto ovládání robotu probíhá manuálně (zaškoleným pracovníkem), pomocí teleoperačního zařízení [3, 4].

Obr. 2: PackBot – mobilní robot používaný především pro likvidaci bomb [6]

2.3.3 Mobilní roboty s kolovým podvozkem

Koncepce kolového podvozku u mobilních robotů patří obecně k nejpoužívanějšímu typu lokomočního ústrojí, neboť kombinace relativně jednoduché mechanické implementace a různorodost možných řešení (mnoho konfigurací – rozložení, počet a typ kol) nabízí využití v širokém spektru robotických aplikací.

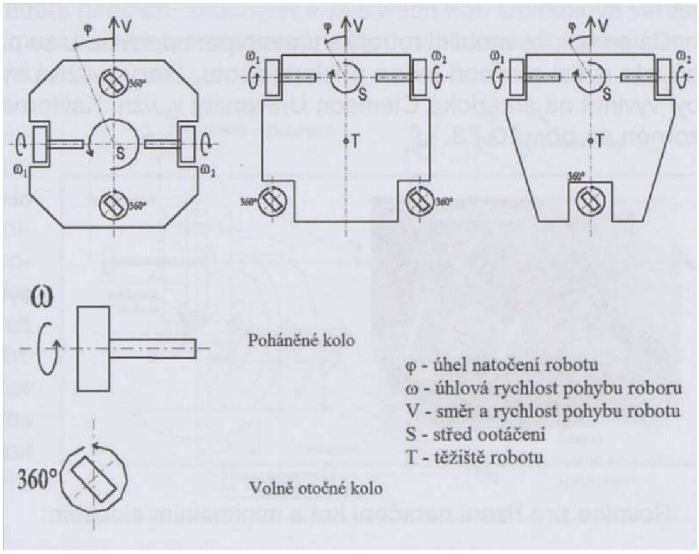
Při konstrukci kolového podvozku je vhodné vycházet z požadavků, které jsou spojené s jeho provozní činností a zejména s možnými kritickými situacemi z hlediska mechanických a fyzikálních vlastností, souvisejících s pohybem, stabilitou a interakcí podvozku s povrchem. K dosažení stability je vyžadováno, aby všechna kola byla v kontaktu s podložkou, a to i v případě členitého povrchu, díky čemuž je mj. zaručena dostatečná trakce, manévrovatelnost a ovladatelnost [3, 4].

V závislosti na dané konfiguraci kolového podvozku lze rozlišit různé koncepty řízení mobilního robotu.

Diferenčně/diferenciálně řízený mobilní robot

Velmi jednoduchý koncept řízení pro mobilní roboty, který je založený na dvou nezávisle poháněných kolech, a jednom či více nepoháněných volně otočných směrových kolech (obvykle jedno, nebo dvě). Princip řízení spočívá v rozdílných rychlostech otáčení poháněných kol, případně i ve směru otáčení.

Tento princip řešení poskytuje velmi dobrou manévrovatelnost a schopnost rotace kolem osy nacházející se v polovině rozchodu hnaných kol. Nicméně je tento typ řízení vhodný zejména pro vnitřní operační prostředí, což je způsobeno citlivostí na malé odchylky povrchu (k vyvažování robotu slouží nepoháněné volně otočné směrové kolo, popř. kola), a zároveň vykazuje velkou citlivost v souvislosti s relativní úhlovou rychlostí obou poháněných kol (výsledkem můžou být neočekáváné trajektorie – např. chceme-li vykonat jednoduchý přímočarý pohyb, tak vlivem různých relativních rychlostí obou kol dostaneme pohyb křivočarý) [1, 3].



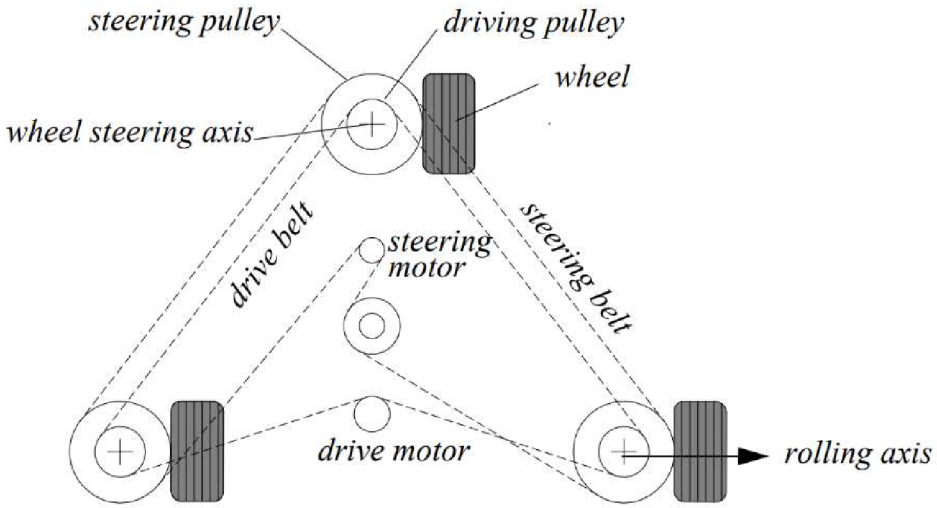
Obr. 3: Kinematické schéma diferenčně řízených podvozků [3]

Synchronně řízený mobilní robot

Typická konfigurace synchronně řízeného podvozku má tři, nebo čtyři kola, která se nacházejí ve vrcholech rovnostranného trojúhelníku, nebo čtverce. Pro příklad rozebereme trojúhelníkové rozložení kol, kde jsou použita tři poháněná, resp. tři řízená kola, která využívají pouze dvou motorů – jeden slouží k synchronnímu nastavování rychlosti a druhý k synchronní rotaci kol, tzn. kola vždy směřují vůči sobě vždy paralelně a dosahují stejných úhlových rychlostí. Toho se obvykle dosahuje komplexní sadou pásů, které fyzicky spojují kola dohromady.

Schopnost samostatně řídit rotaci a rychlost zjednodušuje celkovou kontrolu nad řízením robotu, díky čemuž lze využít pro tento princip pro idealizovaný model bodového robotu. Synchronní pohon je vhodný zejména v případech, kdy je vyžadován všesměrový pohyb, neboť dokud je každá vertikální osa řízení zarovnána s kontaktní plochou každého kola, tak se robot může vždy přeorientovat a pohybovat se po nové trajektorii, aniž by změnil svou stopu.

Translační motor obecně pohání tři kola pomocí jediného pásu. Kdykoli hnací motor uvede robot do pohybu, tak se kvůli sklonu a vůli v hnacím ústrojí nejbližší kolo začne točit před nejvzdálenějším kolem (zpoždění přenášeného výkonu na nejvzdálenějším kole), což způsobuje malou změnu orientace podvozku, a zároveň v závislosti na orientaci podvozku může být tah robotu asymetrický. Mezi další nevýhody patří nízká schopnost překovávání nerovnosti terénu. V porovnání s ostatními roboty, a jejich koncepty lokomočního řešení, nenabízí mnoho výhod. V praxi se tedy synchronně ovládané mobilní roboty příliš nevyskytují [1, 3, 4].

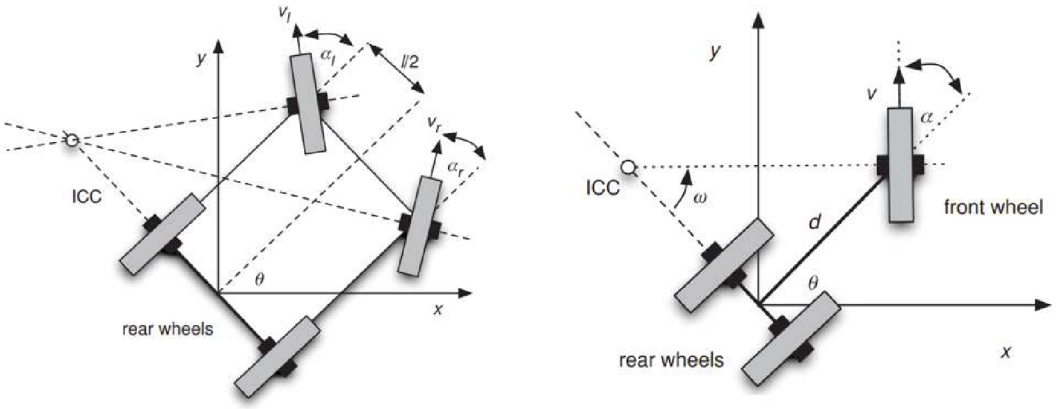


Obr. 4: Schéma synchronně řízeného podvozku mobilního robotu: směrová kladka (steering pulley), hnací kladka (driving pulley), kolo (wheel), osa řízení (wheel steering axis), motor řízení (steering motor), motor pohonu (drive motor), osa otáčení (rolling axis), řemen pohonu (drive belt), řemen řízení (steering belt) [4]

Řízení Ackerman (řízení automobilu)

Ackermanův způsob řízení nalezneme u tříkolových a čtyřkolových mobilních robotů, nicméně se jedná o typ řízení, který nalezneme u většiny automobilů, kde se přední „volantová“ kola otáčí na samostatných ramenech, tak aby hodnota pootočení osy rotace těchto kol směřovala do společného bodu, který protíná myšlenou přímku procházející osou rotace zadních kol, resp. zadní osou vozidla. Aby toto bylo vůbec možné, tak se vnitřní kolo otočí o větší úhel než vnější, a tedy vnitřní kolo vykoná kratší trajektorii než vnější.

Tříkolovou variantu Ackermanova řízení v praxi nalezneme spíše u mobilních robotů než u automobilů, neboť automobily dosahují vyšších rychlostí, při kterých se v situacích zatáčení projevuje nestabilita tříkolového řešení. Princip je založen buď na jednom poháněném a dvou řiditelných kolech, anebo dvou poháněných kolech a jednom řiditelném [1, 3, 4].



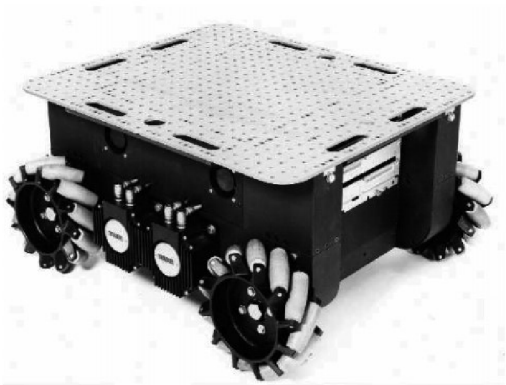
Obr. 5: Kinematické schéma čtyřkolového a tříkolového Ackermanova způsobu řízení [1]

Všesměrové způsoby řízení

Všesměrový pohyb je velice praktický, neboť umožňuje úplnou manévrovatelnost, tzn. mobilní roboty s touto vlastností nemají preferovaný směr pohybu, a mohou se tak volně pohybovat všemi směry, zatáčet a rotovat bez významné limitace úzkých prostor. Spíše než složitou konstrukcí kolových podvozků, se k dosažení těchto pohybových schopností využívají tzv. všesměrová, komplexní nebo též nazývaná složená kola, která poskytují větší kinematickou svobodu než kola jednoduchá.

Příkladem může být všesměrové lokomoční ústrojí kolového robotu se čtyřmi švédskými koly (viz obr. 6), která jsou tvořena ráfkem, na kterém jsou namontované valivé elementy soudečkovitého tvaru, které vůči náboji kola svírají úhel 45 stupňů, a zároveň je každé kolo poháněno samostatným motorem. Změnou otáčení a relativních rychlostí čtyř kol se robot může pohybovat po libovolné trajektorii, a zároveň se může otáčet kolem svislé osy.

Všesměrové roboty se ovládají podstatně snadněji než roboty založené na jednoduchých kolech, protože pohyb robotu je nezávislý na jeho póze. Mezi další přednosti patří schopnost okamžité změny směru, vysoká dynamičnost jízdy a manévrovatelnost v omezených prostorách. Sada všesměrových kol je téměř nekonečná, a proto můžeme dosáhnout různých kinematických výsledků. Nevýhodou je nutnost výrobní a montážní přesnosti, tzn. preciznost provedení těchto kol je daleko významnější z hlediska výsledného chování robotu, než je tomu u kol jednoduchých [1, 3, 4].



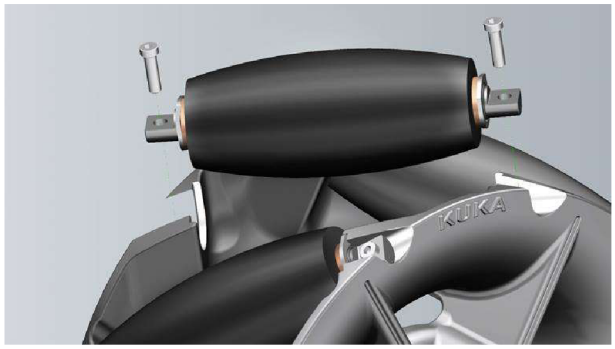
Obr. 6: Všesměrový robot CMU Uranus se čtyřmi poháněnými švédskými koly [4]

2.4 Mobilní roboty v praxi

2.4.1 Mobilní roboty KUKA

Společnost KUKA nabízí automatizační řešení v nejrůznějších průmyslových odvětvích. Ačkoliv je její působení spojováno s průmyslovými roboty, resp. manipulátory, řídícími systémy, výrobními stroji, výrobními linkami atp., tak nalezneme v jejich sortimentu i technologie spojené s mobilní robotikou.

KUKA omniMove je technologie pohonu, která vychází z kola Mecanum (obecné označení všesměrových kol, jako např. švédská kola zmíněná v podkapitole 2.3.3) a umožňuje všesměrový pohyb, tzn. neomezenou manévrovatelnost [7].



Obr. 7: Všesměrová kola KUKA omniWheel [7]

KUKA.NavigationSolution je navigační řešení pro autonomní mobilní roboty, které může být integrováno do veškerých mobilních robotických řešení KUKA. Využívá simultánní lokalizaci a mapování (SLAM), dále je systém schopen reagovat na změny v okolí a použít virtuální dráhy k pohybu po definovaných drahách. Použitím více vozidel software provede jejich synchronizaci a je schopen vzájemně přizpůsobovat jejich plánování pohybu [7].

Obr. 8: Příklad mobilního robotického systému od firmy KUKA – KMR QUANTEC [7]

2.4.2 Boston Dynamics

Společnost Boston Dynamics, se zaměřením na mobilní robotiku, řeší nejnáročnější problémy spojené s dynamickým řízením, biomechanismy, inteligentním řízení robotů v proměnném prostředí, a mnoho dalších výzev v tomto segmentu mobilních robotických systémů.

Za pomocí špičkových senzorů, elektroniky a softwaru sestrojuje vysoce výkonné roboty vybavené vnímáním, navigací a schopností algoritmizované inteligence. Na poli mobilní robotiky působí relativně dlouhou dobu. První kráčející robot schopný pohybu v náročném terénu – BIG DOG (2004), pasivně stabilní šestinohý robot s atypickým designem a pozoruhodnou pohyblivostí v drsném terénu – RHEX (2007), čtyřnohý robot navržený k následování vojáků a přenášení jejich vybavení v náročném terénu – LS3 (2012) nebo nejrychlejší čtyřnohý robot, schopný manévrování a udržování stability při rychlosti 32 km/h – WILDCAT (2013). [5]

Mezi různorodými biomechanickými řešeními nalezneme i mobilní robot s kolovým podvozkem – HANDLE, který slouží nejen k přesouvání krabic ve skladu, ale je schopný vyložit nákladní vozidlo, paletizace, a to vše automatizovaně [5].

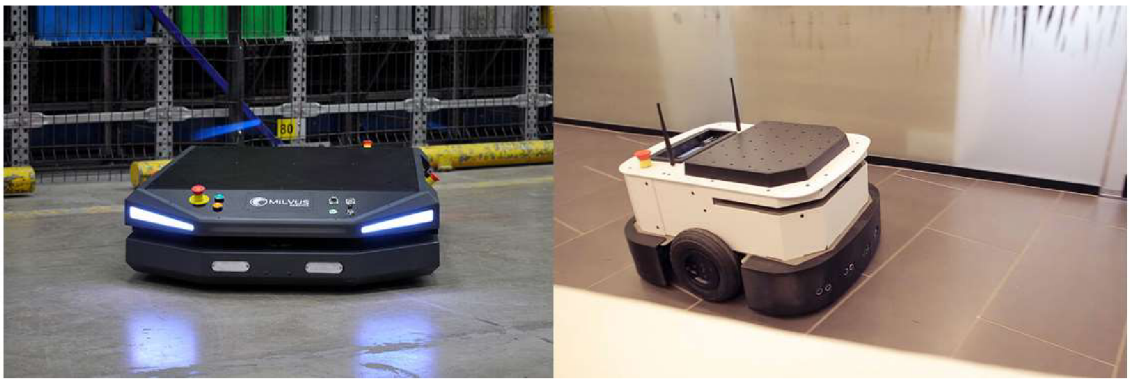
Obr. 9: Mobilní robot RHEX (vlevo) a HANDLE (vpravo) od společnosti Boston Dynamics [5]

2.4.3 Milvus Robotics

Společnost Milvus Robotics poskytuje robotické systémy ke zvýšení produktivity a kvality práce v souladu s bezpečností pracovníků. Oblasti zájmu jsou např. výrobní průmysl, marketing a výzkum, kde jsou mobilní roboty použity nezávisle na lidských pracovnících, tzn. slouží pouze k zefektivnění práce bez narušení stávajícího chodu podniku.

Autonomní mobilní robot SEIT slouží k přepravě materiálu a výrobků ve skladech nebo továrnách. Umožňuje navigaci skrze sklad nebo továrnu, aniž by potřeboval přídavné fyzické komponenty, jakými jsou např. magnety, majáky, značení na podlaze atp. Z druhé strany spektra je zde autonomní mobilní servisní robot ROBIN, který poskytuje inovativní, pohotový a efektivní způsob šíření informací, tzn. oblastí použití je marketing.

Zajímavým členem skupiny mobilních robotů od Milvus Robotics je mobilní robotická platforma MRP2 vhodná k použití ve výzkumu, neboť obsahuje funkce, jakými jsou např. navigace, mapování, rozšiřitelnost, počítačové vidění, teleoperace atp. Velkou výhodou je open-source software a CAD data použitelná v simulačním prostředí Gazebo, a to vše díky implementaci operačního systému pro roboty (ROS) [8].



Obr. 10: Mobilní robot SEIT 500 (vlevo) a MRP2 (vpravo) od společnosti Milvus Robotics [8]

2.4.4 IAM Robotics

IAM Robotics poskytuje autonomní řešení pro manipulaci s materiálem, a to v kosmetickém nebo potravinářském průmyslu a ve zdravotnictví, což je podmíněno dispozicemi autonomního mobilního robotu IAM SWIFT, jehož konstrukce je přizpůsobena manipulaci s materiálem do 7 kg, a pohybu v prostoru úzkých uliček, ve kterých se nachází regály s limitní výškou spodní police 5 cm a vrchní police 2 m.

Je vybaven manipulátorem s RAPIDVISION, což je technologie umožňující detekovat a lokalizovat v reálném čase 3D objekty v prostoru a zároveň je identifikovat pomocí informací získaných skenerem IAM FLASH. Koncovým efektorem v základu je vakuový gripper. Pro bezpečnost pohybu v rámci bezpečnostních a provozních norem je přítomna i detekce překážek [9].

Obr. 11: Mobilní robot IAM SWIFT od společnosti IAM Robotics [9]

2.5 Open Robotics

Společnost Open Robotics spolupracuje s průmyslem, akademickou sférou a vládou na vytváření a podpoře otevřeného softwaru a hardwaru pro použití v robotice, od výzkumu, vzdělávání až po vývoj produktů. Stojí za vývojem a údržbou jádra operačního systému pro roboty (ROS), a 3D multirobotického simulátoru Gazebo [10].

Nabízí služby výzkumu a vývoje v oblasti robotiky, poradenství, zakázkové inženýrství a vývoj aplikací pro průmysl a vládu. Zabývá se také řadou projektů [10]:

- soutěže v simulačním prostředí Gazebo;
- podporu nových projektů založených na ROS a Gazebo;
- vylepšení vlastních nástrojů pro konkrétní aplikace, od implementace cílené funkce po podporu nové platformy;
- vývoj nových vestavěných systémů, jako jsou inteligentní senzory a programovatelné akční členy.

Obr. 12: Logo společnosti Open Robotics [10]

2.6 ROBOTIS

ROBOTIS je globálním poskytovatelem robotických řešení a jedním z předních výrobců robotického hardwaru. Přispívá ke sdílení a šíření technologie mezi vývojáři robotů tím, že nabízí modularizované a standardizované řešení pro robotickou technologii.

ROBOTIS je výhradním výrobcem značky DYNAMIXEL, inteligentních servomotorů (akčních členů). Specializuje se na výrobu nejen robotického hardwaru, ale i robotických platforem, jejichž použití je možné ve všech oborech studia a průmyslu, mj. nabízí výukové robotické sady pro všechny věkové kategorie a úrovně dovedností [11, 12].

Obr. 13: Logo společnosti ROBOTIS [11]

2.6.1 DYNAMIXEL

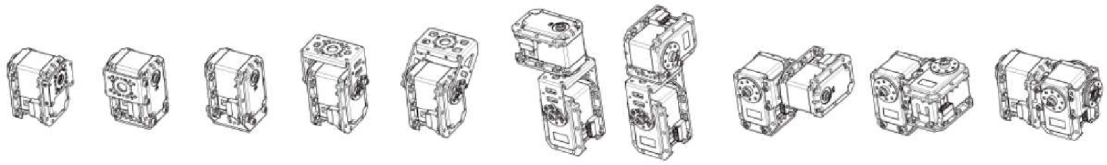
Modulární design

Modulární design typu vše v jednom (all-in-one modular design) s řadou prvků potřebných pro řízení robotu bez dalších zařízení (viz obr. 14): stejnosměrný elektromotor (DC Motor), řídicí jednotka (Controller), ovladač zařízení (Driver), senzor (Sensor), redukční zařízení (Reduction Gear), síť (Network) [12].

Obr. 14: Modulární design all-in-one DYNAMIXEL [12]

Univerzální montážní struktura

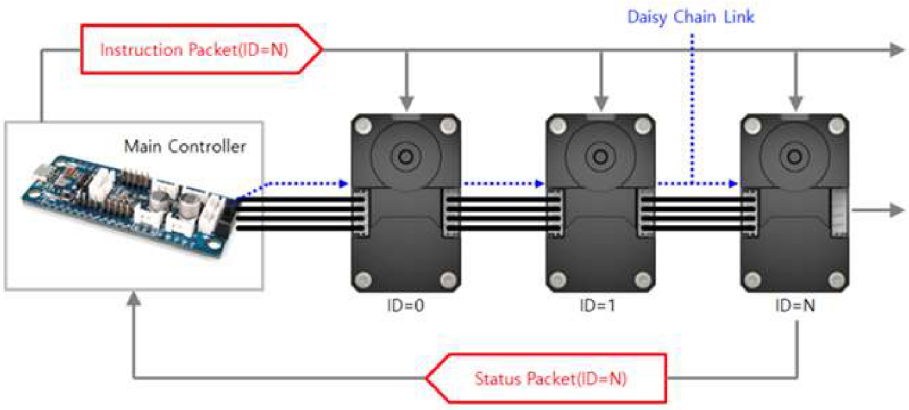
K dispozici jsou různé volitelné konstrukce rámu elektromotorů, nabízející různé možnosti montážního sestavení (obr. 15), k dosažení různorodých robotických konstrukcí pro široké spektrum robotických aplikací, mj. vývojářům poskytnutá CAD data umožňují přesné, a přitom kreativní návrhy [12].



Obr. 15: Příklady montážního uspořádání [12]

Struktura jednoduchého propojení

Každý servomotor DYNAMIXEL lze ovládat odesílaným datových paketů prostřednictvím jediné sběrnice (obr. 16). Zapojení jednotlivých prvků a možnost rozšiřování celého obvodu je snadné prostřednictvím schéma zapojení daisy (odkazuje na pletení řetězu nebo věnce ze sedmikrásek), které popisuje více zařízení spojených dohromady, a to buď v sérii, nebo do kruhu [12].



Obr. 16: Schéma zapojení daisy pro více servomotorů DYNAMIXEL [12]

Řídicí architektura

Servomotory DYNAMIXEL nabízejí různé funkce (režimy řízení) zpětné vazby (řízení momentu založené na proudu, aktuální poloha, rychlost, vnitřní teplota, vstupní napětí atd.) za současného použití sofistikovaného ovládání výkonu pomocí PID kontroléru [12].

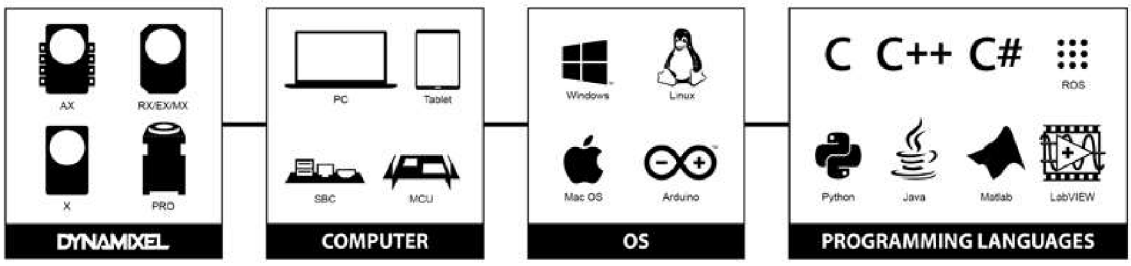
Sortiment

Široký výběr modelů s ohledem na požadavky točivého momentu, rychlosti a příkazových signálů. Nabízí od levných modelů pro začátečníky (low-end) až po vysoce výkonné modely pro profesionály (high-end). Poskytuje dokumentace, zdrojové kódy, nástroje a podporu různých vývojových prostředí [12].

Obr. 17: Přehled dostupných variant servomotorů DYNAMIXEL [12]

Podporovaná vývojová prostředí

RoboPlus někdy také označován jako R+, je software s grafickým rozhraním vyvinutý společností ROBOTIS, obsahuje sadu pro vývoj softwaru s knihovnami a zdrojovými kódy s podporou pro různé OS a programovací jazyky (obr. 18). Umožňuje ovládání prostřednictvím PC/laptopu, tabletu, jednodeskového počítače (SBC) nebo mikrokontroleru (MCU) [12].



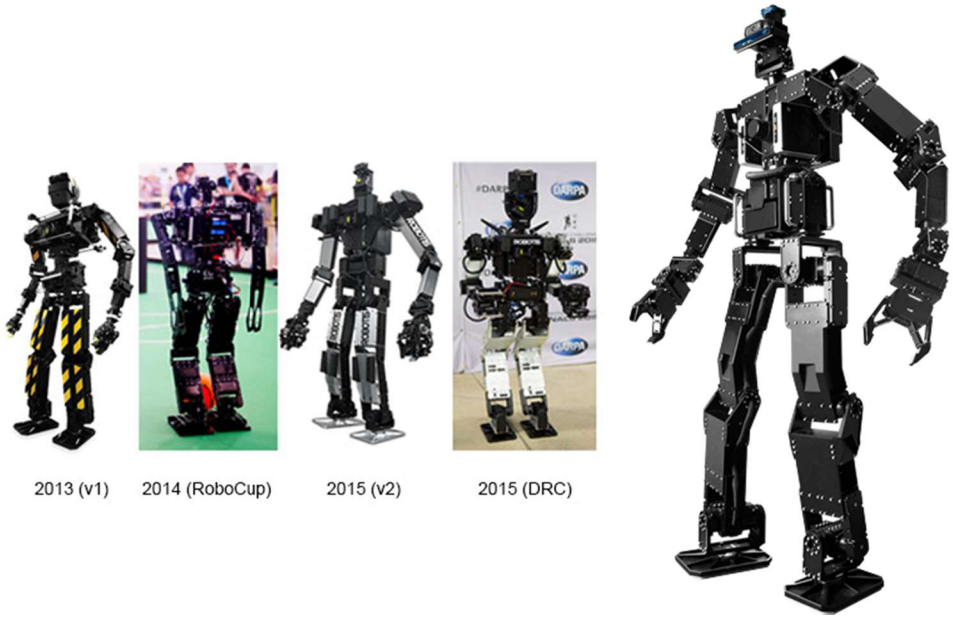
Obr. 18: Přehled podporovaných zařízení, OS a programovacích jazyků [12]

2.6.2 Robotické platformy

THORMANG3

Humanoidní open-source robotická platforma v životní velikosti s pokročilým výpočtovým výkonem, plnou podporou pro ROS, dostupnými CAD daty, open-source sadou pro vývoj, sofistikovanými senzory (např. LIDAR, RealSense kamera, FT senzor,

IMU apod.) a schopností dynamického pohybu, umožňující velké množství aplikací nejen ve výzkumu, ale i ve vzdělávání [12].



Obr. 19: Verze humanoidních robotů THORMANG (THORMANG3 první zprava) [12]

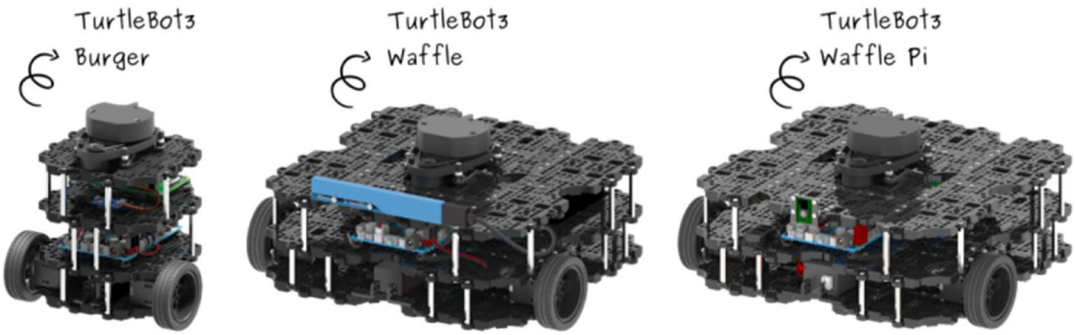
ROBOTIS OP3

ROBOTIS OP3 je nejnovější open-source miniaturní podoba humanoidní platformy, nabízející plnou podporu ROS, včetně volně dostupných CAD dat, schémat obvodů a zdrojových kódů, umožňující použití nejen v oblasti výzkumu (kinematika, dynamika, chůze) a vzdělávání, ale i soutěžích jako RoboCup nebo FIRA (Federation of International Robot-soccer Association) [12].

Obr. 20: ROBOTIS OP3 [12]

TurtleBot3

TurtleBot3 je malý, cenově dostupný programovatelný mobilní robot s plnou podporou ROS. Je vhodný pro použití ve vzdělávání, výzkumu nebo k prototypování. Cílem TurtleBot3 je dramaticky zmenšit velikost robotické platformy, snížit cenu, aniž by došlo k obětovaní funkčnosti, kvality nebo možnosti její přizpůsobitelnosti [13].

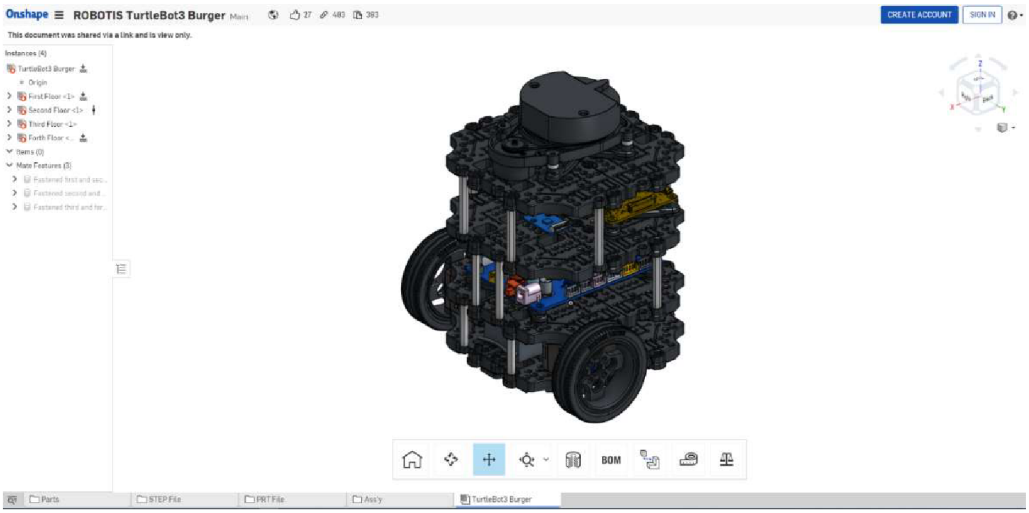


Obr. 21: Přehled série TurtleBot3 [13]

2.7 TurtleBot3 Burger

Mobilní robotická platforma TurtleBot3 byla vyvinuta ve spolupráci s Open Robotics v roce 2017. Jedná se o oficiální vzdělávací platformu pro ROS. Jak bylo zmíněno v podkapitole 2.6.2, tak výhodou je nízká cena, možnost rozšiřitelnosti a modularity, aniž by došlo ke zhoršení kvality zpracování nebo obětování funkcionalit. Aplikovaný model open-source poskytuje otevřený zdrojový kód, který je možné upravovat a sdílet, a volně dostupná CAD data, usnadňující přizpůsobitelnost a rozšiřitelnost této platformy [13].

Kapitola se bude zaměřovat na rozbor stavebnice TurtleBot3 Burger z hlediska hardwaru a softwaru. Volba tohoto modelu ze série TurteBot3 je podmíněna praktickou částí této práce, která se bude zabývat implementací řídicího programu na reálném robotu, kterým bude model TurtleBot3 Burger.



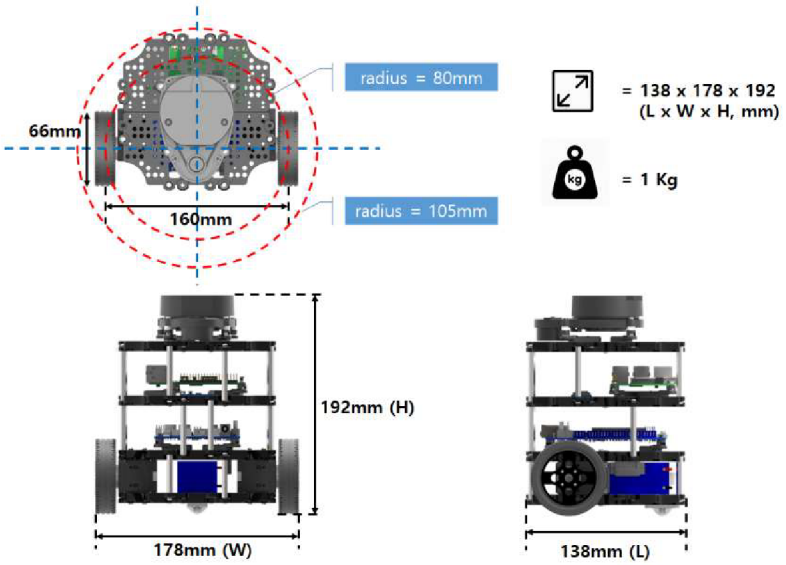
Obr. 22: Open-source 3D model TurtleBot3 Burger

2.7.1 Hardwarové specifikace

Přehled hardwarových specifikací modelu TurtleBot3 Burger je následující [13]:

Tab. 1: Přehled hardwaru a jeho charakteristik pro model Burger ze série TurtleBot3

Položka	TurtleBot3 Burger
Maximální posuvová rychlost	0,22 m/s
Maximální rychlost rotace	2,84 rad/s (162,72 deg/s)
Maximální nosnost	15kg
Rozměry (Š x V x H)	178mm x 192mm x 138mm
Váha (+ SBC + baterie + senzory)	1kg
Prahové lezení (práh výšky překážky)	10mm a nižší
Předpokládaná doba výdrže	2h 30m
Předpokládaná doba nabíjení	2h 30m
SBC (jednodeskový počítač)	Raspberry Pi 3 Model B
MCU (mikrokontroler)	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Vzdálený ovladač	–
Akční člen (hnací motor pohonu)	XL430-W250
LDS (laserový senzor vzdálenosti)	360 Laser Distance Sensor LDS-01
Kamera	–
IMU (inerciální měřící jednotka)	Gyroscope 3 Axis (3osý gyroskop)
	Accelerometer 3 Axis (3osý akcelerometr)
	Magnetometer 3 Axis (3osý magnetometr)
Napájecí konektory	3,3V / 800mA
	5V / 4A
	12V / 1A
Expanzní piny	GPIO 18 pins
	Arduino 32 pin
Komunikační obvody/periférie	UART 3x, CAN 1x, SPI 1x, I2C 1x, ADC 5x, 5pin OLLO 4x
DYNAMIXEL porty	RS485 3x, TTL 3x
Audio	Několik programovatelných pípnutí
Programovatelná LED světla	4x
Stavová LED světla	SBC LED 1x
	Arduino LED 1x
	Napájení LED 1x
Fyzická tlačítka/přepínače	Tlačítko 2x, Resetovací tlačítko 1x, DIP přepínač 1x
Baterie	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC propojení	USB
Aktualizace firmwaru	via USB / via JTAG
Napájecí adaptér	Input : 100-240V, AC 50/60Hz,
	1.5A @max
	Output : 12V DC, 5A



Obr. 23: Rozměrové charakteristiky na 3D modelu [13]

2.7.2 Základní komponenty

Jednodeskový počítač (SBC)

K dispozici jsou jednodeskové počítače Raspberry Pi 3 Model B, Raspberry Pi 3 Model B+, nebo Intel® Joule™ 570x. Pro model TurtleBot3 Burger zakoupený před rokem 2019 (případ aktuální bakalářské práce) byl v základu použit první zmíněný (viz obr. 24).

Obr. 24: Jednodeskový počítač Raspberry Pi 3 Model B [14]

Raspberry Pi 3 Model B je jeden ze série cenově dostupných počítačů, velikostně podobný kreditní kartě, který umožňuje propojení s monitorem, televizí, klávesnicí a myší. Jedná se tedy o malé výpočetní zařízení, které lze v mnoha ohledech přirovnat ke stolnímu počítači. Rozměry, absence chlazení a dalších komponent je limitující v

otázkách výpočetního výkonu, nicméně, pro mnoho robotických aplikací je poskytovaný výkon dostačující, a to samé platí i v oblasti konektivity (viz tab. 2) [14]:

Tab. 2: Hardwarové specifikace SBC Raspberry Pi 3 Model B

Položka	Specifikace
Procesor (CPU)	1.2 GHz 64-bit quad-core ARM Cortex-A53
Architektura	ARMv8-A (64/32-bit)
SoC	Broadcom BCM2837
Operační paměť RAM (SDRAM)	1 GB (sdílená s GPU)
Video (GPU)	Broadcom VideoCore IV @ 250 MHz (BCM2837: 3D part of GPU @ 300 MHz, video part of GPU @ 400 MHz)
	OpenGL ES 2.0 (BCM2835, BCM2836: 24 GFLOPS / BCM2837: 28.8 GFLOPS)
	MPEG-2 and VC-1 (with license), 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder (BCM2837: 1080p60)
USB 2.0 porty	4 (přes zabudovaný pětiportový USB hub; jeden USB port vnitřně propojen s ethernet portem)
Video vstup	15-pinový MIPI konektor kamerového rozhraní (CSI)
Video výstup	HDMI (rev 1.3 & 1.4), 14 HDMI rozlišení od 640×350 do 1920×1200 plus různé PAL a NTSC standardy, kompozitní video (PAL a NTSC) via 3,5 mm TRRS jack sdílený s výstupem zvuku, MIPI konektor rozhraní displeje (DSI)
Zvukový vstup	Přes I2C rozhraní
Zvukový výstup	Analogový (přes 3,5mm jack), digitální (přes HDMI), I2S
Interní paměť	MicroSDHC, USB Boot Mode
Integrovaná síť	10/100 Mbit/s Ethernet + WiFi 802.11n a Bluetooth 4.1
Nízkoúrovňové periférie	17× GPIO plus stejné funkce a HAT ID sběrnice
Rozměry (Š x V x H)	86,60mm x 17mm x 56,5mm

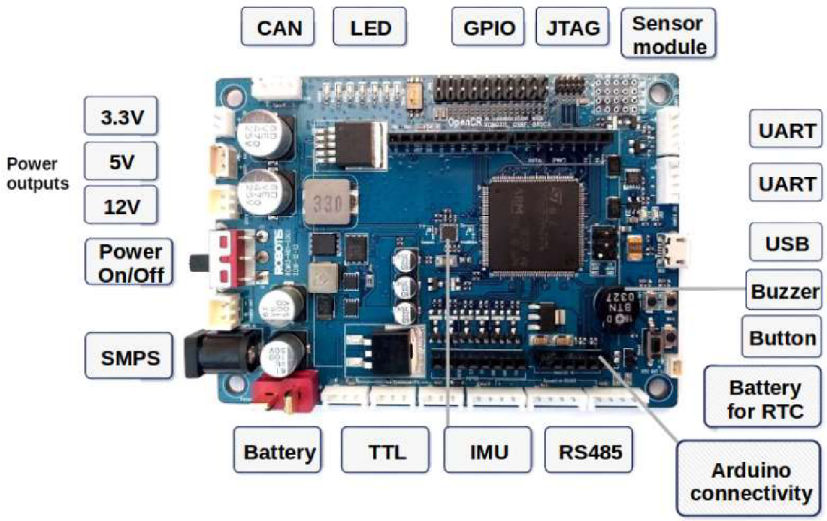
Řídicí modul (Embedded Board)

Řídící modul (embedded board) nebo také vestavěná deska označuje počítačový systém zabudovaný do vestavěného systému. Obvykle se nazývá kontrolérem, protože se používá k řízení jiných zařízení. Tato deska však sama o sobě již obsahuje vestavěný mikrokontroler pro řízení různých zařízení vestavěných v systému pomocí programů.
Vestavěný systém je definován jako počítačový systém s vyhrazenou funkčností v rámci většího mechanického nebo elektrického systému. Jedná se o kompletní zařízení, včetně hardwarových součástí, a je označován za „mozek“ systému, který vyžaduje kontrolu.

OpenCR (Open-source Controller) je open-source řídícím modulem pro ROS. Jedná se tedy o vestavěnou desku s plnou podporou ROS a je využívána jako hlavní řídící deskou v robotické platformě TurtleBot3. Aplikovaný model open-source poskytuje volně dostupné informace o hardwaru, jako např. schémata obvodů, PCB Gerber, BOM a zdrojový kód firmwaru pro TurtleBot3.

Budou zmíněny charakteristiky a možnosti použití této vestavěné desky (bez bodového vypsání hardwarových specifikací, neboť jsou již obsaženy v přehledu hardwaru robotu samotného) [12, 13]:

- **Vysoký výkon:** OpenCR, který používá STM32F746 od ST, používá vysoce výkonné jádro mikrokontroleru Cortex-M7 od ARM, které běží až do 216 MHz. Může být také použit pro zpracování velkého množství dat pomocí algoritmů nebo různých periferií, které vyžadují vysokorychlostní provoz.
- **Podpora Arduino:** Používáním Arduino IDE v základním vývojovém prostředí OpenCR umožňuje snadnější přístup těm, kteří nejsou s integrovaným vývojovým prostředím seznámeni. Díky podpoře rozhraní kompatibilních s hlavičkami pinů Arduino UNO je k dispozici mnoho štítových modulů spolu s mnoha knihovnami a zdrojovým kódem vytvořeným ve vývojovém prostředí Arduino. Protože deska OpenCR je přidávána a spravována prostřednictvím správce desky Arduino IDE, lze ji efektivně spravovat při aktualizaci firmwaru desky.
- **Propojení s SBC:** OpenCR poskytuje digitální a analogové vstupy / výstupy, které lze propojit s rozšiřující deskou nebo různými senzory. OpenCR má také různá komunikační rozhraní: USB pro připojení k PC, UART, SPI, I2C, CAN pro další vestavěná zařízení. Může poskytnout nejlepší řešení při použití se SBC. Podporuje 12V, 5V, 3.3V výstupy pro SBC a senzory. Podporuje také výměnné vstupy mezi baterií a SMPS, a také je kompatibilní s akčními členy DYNAMIXEL od ROBOTIS.



Obr. 25: Řídící modul OpenCR od společnosti ROBOTIS [12]

Senzorické vybavení

Série TurtleBot3 obsahuje implementační řešení pro laserový senzor vzdálenosti (360 Laser Distance Sensor LDS-01), 2D kameru (Raspberry Pi Camera Module v2.1) a 3D kameru (Intel® Realsense™ R200).

Konfigurace jednotlivých modelů nabízí v základu laserový senzor vzdálenosti + 3D kameru pro TurtleBot3 Waffle, 2D kameru pro TurtleBot3 Waffle Pi, nebo absenci počítačového vidění u TurtleBot3 Burger.

Ačkoliv ve výchozí konfiguraci TurtleBot3 Burger neobsahuje možnost počítačového vidění, tak je možné dokoupit komponenty a pomocí volně dostupného zdrojového kódu tuto skutečnost změnit.

Zvolený laserový skener vzdálenosti disponuje, s ohledem na cenu (179,9 dolarů, přepočet s daní – 5 200,-) a kompaktnost (viz obr. 26), vhodnými parametry (viz tab. 3) pro mapování a navigaci ve vnitřním prostředí [13]:

Tab. 3: Parametry laserového skeneru vzdálenosti 360 Laser Distance Sensor LDS-01

Položka	Specifikace
Rozsah snímané vzdálenosti	120 - 3500mm
Přesnost vzdálenosti (120 - 499mm)*	± 15mm
Přesnost vzdálenosti (500 - 3500mm)*	± 5,0%
Přesnost vzdálenosti (120 - 499mm)**	± 10mm
Přesnost vzdálenosti (500 - 3500mm) **	± 3,5%
Rychlost skenování	310±10rpm
Úhlový rozsah	360 deg
Úhlové rozlišení	1 deg

*přesnost = průměrná vzdálenost – referenční hodnota

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

Obr. 26: 360 Laser Distance Sensor LDS-01 [13]

Akční členy

Akční členy, resp. inteligentní servomotory, která využívá série TurtleBot3 byly představeny v podkapitole 2.6.1, tzn. od společnosti ROBOTIS servomotory DYNAMIXEL, a to řady XL.

Řada DYNAMIXEL XL přijímá nové funkce, jako bezkontaktní magnetický enkodér a strukturu sestavy dutého zadního pouzdra, umožňující režim ovládání 360 stupňů. Dále nabízí různé operační módy (kontrola rychlosti, pozice, rozšířenou regulaci polohy, PWM ovládání) a řízení profilu rychlosti pro plánování hladkého pohybu [11].

2.7.3 Přehled vestavěných funkcionalit

Základní operace

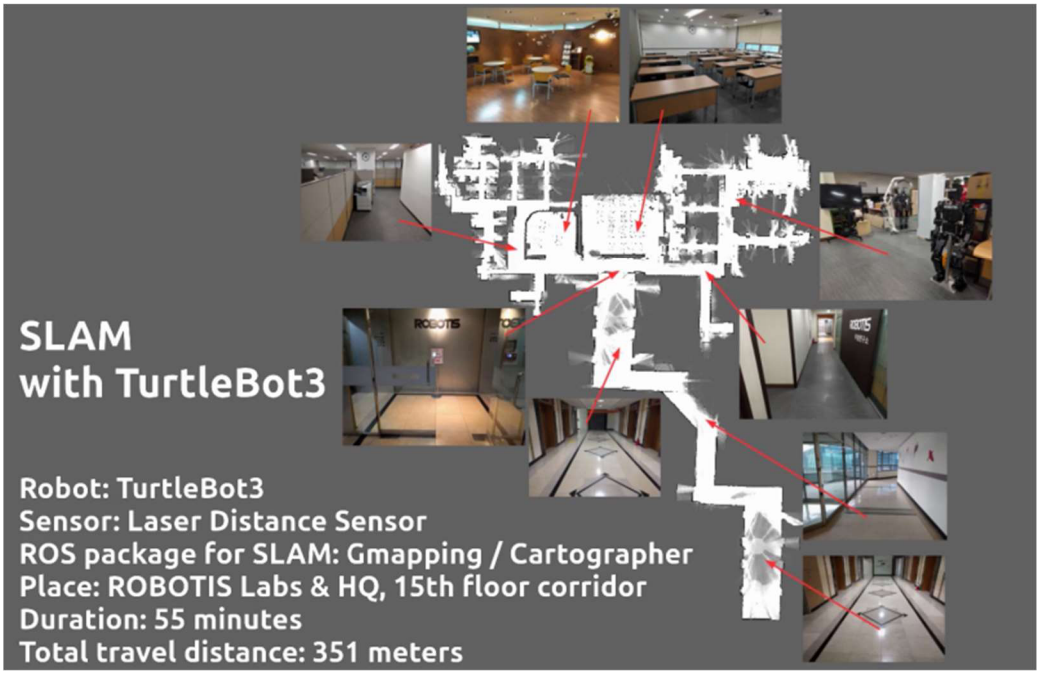
Mezi nejzákladnější operace patří teleoperace neboli manuální ovládání robotu pomocí k tomu určeného zařízení. V případě této mobilní robotické platformy existuje v této oblasti mnoho různých možností, tzn. variabilita použitelných zařízení, např. skrze osobní počítač, který je připojen k internetu, a to pomocí klávesnice, dále joystick pro PS3, XBOX 360 atp. [13]

SLAM (Simultánní lokalizace a mapování) a navigace

SLAM je souborem algoritmů, které prostřednictvím vhodných senzorů nejen konstruují mapu prostředí, ale při samotné konstrukci musí být použitý robot schopný pohybu a lokalizace.

Navigace slouží k přesunu robota z jednoho místa na určené místo v daném prostředí. K tomu je zapotřebí mapa, která obsahuje geometrické informace o nábytku, objektech a stěnách daného prostředí. Jak je popsáno v předchozím odstavci – SLAM, mapa byla vytvořena s informacemi o vzdálenosti získanými senzorem a informacemi o pozici samotného robotu.

Navigace umožňuje robotu přejít z aktuální pozice k určené cílové pozici na mapě, a to pomocí mapy, enkodéru robotu, senzoru IMU a senzoru vzdálenosti. [13]



Obr. 27: Demonstrativní ukázka použití mobilního robotu TurtleBot3 pro SLAM [13]

3 ROBOTICKÝ OPERAČNÍ SYSTÉM (ROS)

ROS (Robot Operating System) je robotický operační systém, přesněji řečeno meta-operační systém (někdy také označován za framework nebo middleware), obsahující robustní sady nástrojů a knihoven. Navržený tak, aby byl co nejvíce distribuovaný a modulární. Obrovská komunita uživatelů, a rozsáhlý ekosystém softwarové a hardwarové podpory umožňuje vývojářům vytvářet sofistikované a komplexní robotické aplikace.

ROS je open-source projekt licencovaný podle BSD licence[1], umožňující opakované použití nejen v komerčních produktech, ale i v produktech s uzavřeným zdrojovým kódem, jedná se tedy o velmi permisivní licenci. [15]

3.1 Charakterizace ROS

Přestože zkratka ROS znamená ve volném překladu operační systém pro roboty, tak se nejedná o plnohodnotný operační systém (OS), jak jej známe v případě osobních počítačů (Windows, Linux, macOS a Chrome OS) nebo chytrých telefonů (Android a iOS).

Zejména pro nové uživatele může být matoucí výše zmíněná zkratka a její překlad, a to do takové míry, že považují ROS za nový OS, avšak pro roboty. Mnohem přesnější popis je, že se jedná o meta-operační systém, který běží na již existujícím OS, a to na některé distribuci Linuxu (software pro ROS je primárně vyvíjen a testován na Ubuntu).

Ačkoliv meta-operační systém není definovaným pojmem, tak popisuje systém, který poskytuje některé funkcionality OS, a to hardwarovou abstrakci, nízkoúrovňové ovládání, multithreading a správu balíčků, mj. i nástroje a knihovny pro získávání, psaní, vytváření a spouštění kódu (framework), a to v rámci více počítačů, tzn. meziprocesová, resp. meziplatformní komunikace (middleware).

Jednotlivé zdroje se v případě stručného vymezení ROS rozcházejí. V některých publikacích je označován jako framework, middleware, nebo meta-operační systém. Nicméně, žádný z těchto termínů není chybný, neboť je obtížné stručně definovat ROS. Zjednodušeně lze ROS chápat podle rovnice (obr. 28), kde jednotlivé pojmy znamenají [15, 16, 17]:

Obr. 28: Rovnice popisující ROS [15]

- **Plumbing (instalatérství):** meziprocesová a meziplatformní komunikace – přirovnání k důmyslnému systému potrubí, např. v mrakodrapu.

[1] https://opensource.org/licenses/BSD-3-Clause

- **Tools (nástroje):** nástroje pro monitorování, zaznamenávání, vizualizaci a zpracování dat, resp. introspekci.
- **Capabilities (schopnosti, funkcionality):** funkce specifické pro roboty, např. ovládání robotu, mapování, lokalizace, navigace, knihovny geometrie robotu atp.
- **Ecosystem (ekosystém, prostředí):** aktivní komunita uživatelů, spolupracujících na vývoji ROS, angažmá společností, které rozšiřují povědomí o tomto systému.

3.2 Benefity a funkce při programování v ROS

V podkapitole 3.1 byl stručně popsán a softwarově zařazen ROS. Společně s tím byly představeny některé z jeho funkcí, které budou v rámci této kapitoly vysvětleny, a také bude rozšířen jejich výčet o další benefity a funkce, resp. zmíněn přehled vlastností, zdůvodňující, proč by naše preference v oblasti robotiky měly vést k robotickému systému ROS [15, 16, 17, 18, 19, 20, 21, 22]:

- **Meziprocesová a meziplatformní komunikace:** ROS obsahuje vestavěné rozhraní pro předávání zpráv, které poskytuje propojení v rámci různých procesů a platforem; běžně označováno jako middleware, umožňující komunikaci (předávání zpráv) mezi jednotlivými programy (uzly). Uzel může být spuštěn nezávisle na jiných uzlech, a to v libovolném pořadí. Uzly mohou být spuštěny na jednom počítači anebo distribuovány v síti počítačů. Uzel je užitečný pouze za předpokladu, že může komunikovat s jinými uzly, a nakonec se senzory a akčními členy.
- **Hardwarová abstrakce:** ROS poskytuje hardwarovou abstrakční vrstvu, pomocí které mohou vývojáři vytvářet robotické aplikace použitelné pro jakýkoliv robot; jediným limitujícím faktorem může být použitý hardware. Pokud například nový senzor pracuje na stejném principy jako stávající, tak není potřeba přepisovat fungující kód.
- **Distribuované výpočty:** Výpočetní výkon, resp. výpočetní čas je jeden z aspektů, kde tendence směřují k jeho zvyšování, resp. snižování. K dosažení požadovaných výsledků v oblasti robotiky je nutné použití senzorů, jejichž data musí být zpracována, resp. vyhodnocena. Jedná se však a velké množství dat, a tedy mnoho výpočtů, které jsme pomocí ROS schopni distribuovat do clusteru výpočetních uzlů, tzn. distribuce výpočetního výkonu a výsledné zrychlení procesu zpracování dat.
- **Souběžně prováděné výpočty:** Manipulace s hardwarem prostřednictvím dvou a více procesů je obecně problematická. Máme-li například zpracovat obraz z kamery nejen pro detekci pohybu, ale i pro detekci obličeje, tak se potýkáme s problémem komplexnosti chování programu, a tedy jeho složitosti, která se projeví ve nejen ve výpočetním čase, ale i v ladění, případně budoucí

modifikaci. ROS umožňuje přistupovat k zařízením prostřednictvím témat, kde předávanou informací je zpráva, obsahující v našem případě obraz, který můžeme na sobě nezávislými uzly zpracovat.

- **Snadné prototypování:** Hlavním cílem ROS je znovupoužitelnost již existujícího programu (kódu) a jeho sdílení napříč výzkumnou nebo vývojovou komunitou, což urychluje růst celého systému ROS, a také snižuje čas strávený při prototypování vlastního robotu.
- **Snadné testování:** ROS nabízí vestavěný testovací framework, který vyhodnotí kvalitu napsaného kódu, např. množství bugů.
- **Modularita a přizpůsobitelnost:** Jeden z problémů u většiny samostatných robotických aplikací je možnost, že některý z podprocesů hlavního kódu přestane pracovat (havaruje), a tím může dojít k zastavení celého robotu, nebo k jeho nepředvídatelnému chování. V ROS je situace jiná; právě nezávislost uzlů, ať už v rámci jednoho nebo více procesů, neovlivňuje ostatní robotické úkony. Havaruje-li jeden uzel, tak dojde k zastavení pouze daného úkonu a ostatní pracují, a tudíž nedojde k zastavení robotu. ROS obsahuje robustní metody pro obnovení provozu i za předpokladu, že některé senzory nebo motory nefungují. Daleko častější využití této vlastnosti se nachází v modifikaci robotu, resp. jeho úpravám z hlediska funkcionalit, např. výchozí konfigurace TurtleBot3 Burger má absenci počítačového vidění, tuto mezeru v použitelnosti lze vyplnit zakoupením kamery, jejího připojení, konfiguraci a následného použití bez nutnosti přepisování kódu pro již zabudované senzory a akční členy.
- **Pokročilé funkce specifické pro oblast robotiky:** ROS není pouze systémem, prostředím, ve kterém vyvíjíme software pro náš robot, ale obsahuje již přichystané balíčky k použití (ready-to-use), např. pokročilé algoritmy pro autonomní řízení mobilních robotů (např. SLAM nebo AMCL) nebo pro plánování pohybu robotických manipulátorů. „Ready-to-use“ znamená vysokou konfigurovatelnost, umožňující různorodost jejich implementací v robotických aplikacích.
- **Podpora high-end senzorů a akčních členů:** ROS je vybaven ovladači zařízení a balíčky s rozhraními různých senzorů a akčních členů. High-end senzory (např. Velodyne-LIDAR, laserové skenery, 3D kamery atp.) a high-end akční členy (např. inteligentní servomotory DYNAMIXEL) jsme schopni prostřednictvím ROS propojit bez větších obtíží.
- **Výkonná sada nástrojů pro vývoj:** Specifická forma meziprocesní a meziplatformní komunikace umožňuje spontánně zkoumat data protékající systémem, což usnadňuje ladění, vizualizaci a simulaci stavu vyvíjeného systému. Této schopnosti introspekce využívají v ROS rozsáhlé kolekce grafických nástrojů a nástrojů příkazového řádku.
- **Jazyková nezávislost:** Uzly v ROS lze naprogramovat v jakémkoliv jazyce, který má knihovny klientů pro ROS (C, C++, Python nebo Java). Tato míra

flexibility, kdy každý uzel může být naprogramovaný v jiném jazyce poskytuje možnost kolaborativní činnosti bez ohledu na jazykové specializace.

- **Integrace knihoven třetích stran:** V ROS je integrováno mnoho knihoven třetích stran, které můžou vývojáři bez obtíží používat se zárukou jejich aktuálnosti a budoucí podpory, např. OpenCV, PCL, OpenNL, OpenRave atd.
- **Aktivní komunita:** Vývoj ROS je založen na aktivní komunitě uživatelů, kteří vyvíjejí a udržují balíčky ROS aktuální. Uživatelé, ať už z oblasti výzkumu, vzdělání nebo průmyslu, a to v celosvětovém měřítku, se podílejí svojí kolaborativní činností na růstu tohoto systému.

3.3 Historie a vývoj ROS

ROS byl původně vyvinut v roce 2007 laboratoří se zaměřením na umělou inteligenci ve Stanfordu – Stanford Artificial Intelligence Laboratory (SAIL) – na podporu projektu Stanford AI Robot (STAIR). Ve stejném roce poskytl Willow Garage, vizionářský robotický inkubátor značné zdroje pro rozšíření konceptu ROS a vytvoření osvědčených implementací, které známe dnes. Toto úsilí podpořilo nespočet vědců, kteří přispěli svým časem a odbornými znalostmi, k položení softwarového základu tohoto systému. Celý projekt byl a je vyvíjen pod permisivní licencí BSD, tzn. open-source.

V roce 2009 byl vydán ROS ve verzi 4.0 a společně s tím i první robot fungující na systému ROS s označením PR2. Iniciativu převzaly i další instituce, které se začaly podílet na vývoji, a to nejen ty, které dostaly robot PR2 od Willow Garage. Za účelem snazší spolupráce byl vytvořen sdílený server, kde jednotliví přispěvatelé nahrávali své kódy. Tento model spolupráce se stal jednou ze silných stránek ROS a přispěl k vytvoření vlastního ekosystému [15, 19, 20].

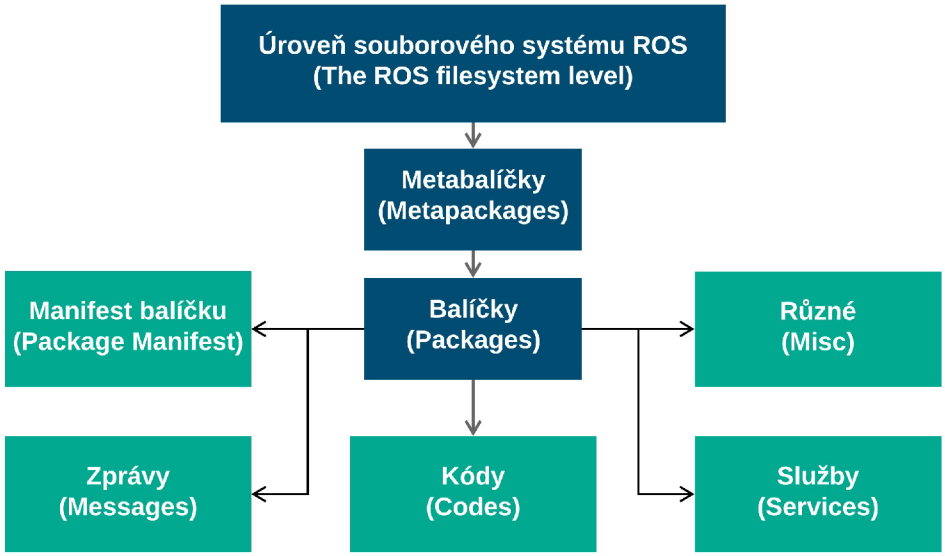
3.4 ROS koncepty a architektura

Porozumění základnímu fungování ROS a jeho terminologie je klíčové pro pochopení jak stávajících aplikací, tak k vytváření aplikací vlastních.

V ROS existují tři hlavní úrovně: úroveň souborového systému (the filesystem level), úroveň výpočtového grafu (the computation graph level) a úroveň komunity (the community level) [21].

3.4.1 Úroveň souborového systému

Soubory ROS jsou na pevném disku uspořádány specifickým způsobem, který lze vidět na následujícím diagramu (obr. 29).



Obr. 29: Úroveň souborového systému ROS

Metabalíčky (Metapackages)

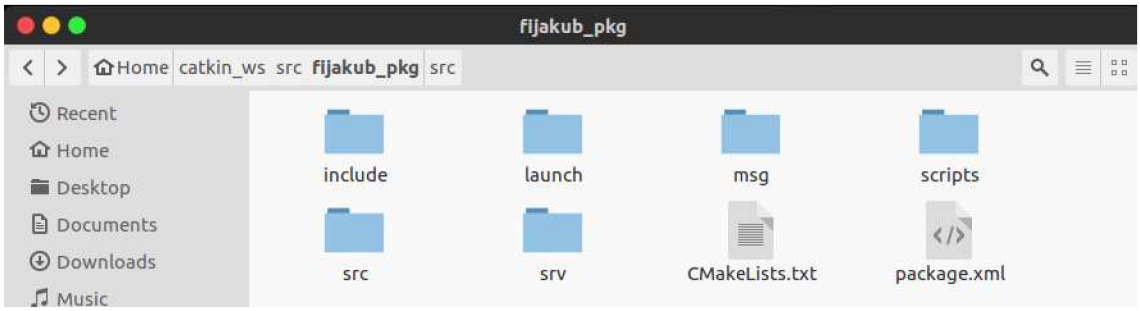
Metabalíčky jsou specializované balíčky v ROS souborovém systému; obsahují pouze jeden soubor `package.xml`. Metabalíčky jednoduše seskupí sadu více balíčků jako jeden logický balíček, tzn. uchovávají informace o souvisejících balíčcích a pomáhají nainstalovat tyto balíčky během vlastní instalace. V podstatě se jedná o normální balíček, který má v souboru `package.xml` exportní tag [16, 18, 22].

package.xml

```
<export>
     <metapackage />
</export>
```

Balíčky (Packages)

Balíčky v ROS jsou nejzákladnější jednotkou softwaru ROS. Obsahují jeden nebo více programů (uzlů), knihovny, konfigurační soubory atd. (viz obr. 30), které jsou uspořádány společně jako jedna jednotka; cílem jejich struktury je znovupoužitelnost [16, 18, 22]:



Obr. 30: Typická struktura balíčků ROS

- `include`: obsahuje hlavičkové soubory, moduly a knihovny potřebné ke spuštění programu (uzlu);
- `launch`: obsahuje spouštěcí soubory, které se používají ke spuštění jednoho nebo více uzlů jediným příkazem;
- `msg`: obsahuje vlastní definice zprávy;
- `srv`: obsahuje vlastní definice služby;
- `scripts`: obsahuje spouštěcí python programy (uzly);
- `src`: obsahuje spouštěcí C++ programy (uzly);
- `CMakeLists.txt`: soubor s informacemi pro kompilaci balíčku;
- `package.xml`: manifest balíčku.

Manifest balíčků (Package Manifest)

Uvnitř každého balíčku se nachází soubor s názvem `package.xml`. Tento soubor obsahuje informace, jako např. jméno, verze, autor, licence a závislosti, které jsou od balíčku vyžadovány. Rozdíl mezi manifestem balíčku a metabalíčku je exportní tag. Ukázkový soubor `package.xml` [16, 21, 22].

package.xml

```
<?xml version="1.0"?>
<package format="2">
     <name>fijakub_pkg</name>
     <version>0.0.0</version>
     <description>
          The fijakub_pkg package
     </description>
     <maintainer email="fijakub@todo.todo">fijakub
     </maintainer>
     <license>TODO</license>
     <author email=" fijakub@todo.todo">fijakub</author>
     <build_depend>roscpp</build_depend>
     <build_depend>rospy</build_depend>
     <build_export_depend>roscpp</build_export_depend>
     <build_export_depend>rospy</build_export_depend>
     <exec_depend>roscpp</exec_depend>
     <exec_depend>rospy</exec_depend>
</package>
```

Existuje minimální sada tagů, která musí být vnořena do sekce `<package>`, aby byl balíček kompletní, nicméně, manifest balíčků s minimální sadou tagů nespecifikuje žádné závislosti [16, 22]:

- `<name>`: jméno balíčku;
- `<version>`: verze balíčku;
- `<description>`: popis balíčku;

- `<maintainer>`: jména osob, spravujících balíček;
- `<license>`: softwarová licence, pod kterou byl balíček vydán;
- `<build_depend>`: závislosti sestavení (build dependencies) potřebné k vytvoření balíčku;
- `<build_export_depend>`: závislosti sestavení exportu (build export dependencies) potřebné k sestavení knihoven při exportu;
- `<exec_depend>`: závislosti spuštění (execution dependencies) potřebné ke spuštění kódu v tomto balíčku;
- `<test_depend>`: závislosti pro testování (test dependencies);
- `<buildtool_depend>`: závislosti sestavení nástrojů (build tool dependencies) určují systémové nástroje potřebné k sestavení balíčku;
- `<doc_depend>`: závislosti dokumentačního nástroje (documentation tool) potřebné ke generování dokumentace.

Zprávy (Messages)

Výměna dat v ROS mezi jednotlivými uzly je zprostředkována zasíláním zpráv. Uzly v systému ROS mohou zapisovat nebo číst data (zprávy), která mohou být reprezentována různými datovými typy. Pro popis hodnot různých datových typů slouží zjednodušený jazyk popisu zpráv (message description language), který usnadňuje nástrojům ROS automaticky generovat zdrojový kód pro příslušný typ zprávy v různých cílových jazycích.

Data zprávy jsou uložena v souboru s příponou `.msg` v podadresáři `msg` balíčku ROS. Definice zprávy se může skládat ze dvou typů: polí a konstant. Pole je rozděleno na typy polí a názvy polí. Typ pole je datový typ a název pole je název zprávy. Konstanty definují konstantní hodnotu v souboru zpráv [16, 19, 21, 22].

Služby (Services)

Podobně jako u zpráv používající soubor .msg, obsahující definici zprávy, tak soubory s příponou `.srv` obsahují definici služby. Nacházejí se v podadresáři `srv` balíčku ROS. Pro definování typu služeb slouží zjednodušený jazyk popisu služby (service description language). Na první pohled se může zdát, že zprávy a služby jsou duplicitním typem zapisování a čtení dat, nicméně, hlavním rozdílem je typ komunikace, pro který se používají [16, 19, 21, 22].

Repositáře (Repository)

Většina balíčků je udržována pomocí kontrolního systému verzí, Version Control System (VCS), jako je Git, Subversion (svn), Mercurial (hg) atd. Kolekce balíčků, které sdílejí VCS se nazývají repositáři [22].

3.4.2 Úroveň výpočtového grafu ROS

Výpočtový graf je peer-to-peer síť procesů ROS, které zpracovávají data společně. Základní koncepty výpočtového grafu ROS jsou uzly (Nodes), Master (ROS Master),

server parametrů (Parameter Server), zprávy (Messages), služby (Services), témata (Topics) a soubory typu bag.

Každý koncept výpočtového grafu ROS poskytuje data různými způsoby. Tyto koncepty jsou implementované v repositáři `ros_comm`, společně s balíčky souvisejícími s komunikací v ROS a knihovnami základních klientů, jako `roscpp` a `rospy` [16, 19, 21, 22].

Uzly (Nodes)

Uzel je proces, který provádí výpočet pomocí klientských knihoven ROS, jako `roscpp` a `rospy`. Jeden uzel může komunikovat s ostatními uzly pomocí témat, služeb nebo serveru parametrů. Robot může obsahovat mnoho uzlů; např. jeden uzel zpracovává obraz z kamery, druhý data z laserového skeneru, třetí řídí motory kol apod.

Použití uzlů v ROS poskytuje několik výhod; odolnost celého systému proti chybám, protože jednotlivé uzly jsou od sebe izolovány, snižují složitost kódu a zvyšují schopnost ladění celého programu, protože každý uzel zpracovává pouze jednu funkci.

Nástroj příkazového řádku `rosnode` zobrazuje informace o uzlech ROS. Aktuálně jsou podporované tyto příkazy (viz tab. 4) [16, 17, 18, 20, 21, 22].

Tab. 4: Příkazy nástroje příkazového řádku `rosnode`

Příkaz	Argumenty	Popis
`$ rosnode info`	`node_name`	Vypíše informace o uzlu.
`$ rosnode kill`	`node_name`	Ukončí aktivní uzel.
`$ rosnode list`	–	Vypíše seznam aktivních uzlů.
`$ rosnode machine`	`machine_name`	Vypíše seznam aktivních uzlů na konkrétním počítači nebo seznam počítačů.
`$ rosnode ping`	`node_name`	Otestuje připojení k uzlu.
`$ rosnode cleanup`	–	Vymaže informace o registraci nedostupných uzlů.

Zprávy (Messages)

Uzly spolu komunikují publikováním (publishing) zpráv (messages) na dané téma (topic). Zprávy jsou jednoduchou datovou strukturou obsahující typy polí. ROS zprávy podporují standardní primitivní datové typy a pole primitivních typů. Soubory zpráv s příponou `.msg` jsou jednoduché textové soubory obsahují definici datové struktury zprávy.

Uzly si také mohou vyměňovat informace pomocí servisních volání (service calls), tzv. služby (services) jsou také zprávy. Definice servisní zprávy se nachází uvnitř souboru s příponou `.srv`.

Uzly mohou komunikovat pouze pokud jsou zprávy stejného datového typu; porovnání probíhá pomocí kontrolního součtu MD5. Nástroj příkazového řádku `rosmsg` zobrazuje informace o zprávách ROS (viz tab. 5) [16, 20, 21, 22].

Tab. 5: Příkazy nástroje příkazového řádku `rosmsg`

Příkaz	Argumenty	Popis
`$ rosmsg show`	`msg_type`	Vypíše popis zprávy.
`$ rosmsg list`		Vypíše seznam zpráv.
`$ rosmsg md5`	`msg_type`	Vypíše kontrolní součet md5sum zprávy.
`$ rosmsg package`	`package_name`	Vypíše seznam zpráv v balíčku.
`$ rosmsg packages`	`package_name1 package_name2`	Vypíše balíčky, které obsahují zprávy.

Témata (Topics)

Témata ROS jsou pojmenované sběrnice, ve kterých si uzly ROS vyměňují zprávy. Témata mohou publikovat (publishing) a odebírat (subscribing) anonymně, což znamená, že odesílání a příjem dat je od sebe oddělen. Obecně uzly nevědí, s jakým uzlem komunikují. Místo toho uzly, které mají zájem o data, se přihlásí k odběru příslušného tématu, resp. hledají pouze název tématu a porovnávají, zda se shodují typy zpráv vydavatele (publisher) a odběratele (subscriber).

Uzly ROS komunikují s tématy pomocí přenosu založeného na TCP/IP známého jako TCPROS. Tato metoda je základní metodou přenosu používanou v ROS. Další typ komunikace je UDPROS, který má nízkou latenci, volný transport a je vhodný pouze pro teleoperace.

Nástroj příkazového řádku `rostopic` zobrazuje informace o tématech ROS. V současné době podporuje zobrazení seznamu aktivních témat, vydavatelů (publishers) a odběratelů (subscribers) určitého tématu, míru publikování tématu (publishing rate), šířku pásma tématu a zprávy na dané téma. Zde je přehled příslušných příkazů (viz tab. 6) [16, 17, 20, 21, 22].

Tab. 6: Příkazy nástroje příkazového řádku `rostopic`

Příkaz	Argumenty	Popis
`$ rostopic bw`	`/topic_name`	Vypíše šířku pásma používanou daným tématem.
`$ rostopic delay`	`/topic_name`	Vypíše zpoždění pro téma, které má záhlaví.
`$ rostopic echo`	`/topic_name`	Vypíše zprávu daného tématu.
`$ rostopic find`	`msg_type`	Najde téma podle typu dané zprávy.
`$ rostopic hz`	`/topic_name`	Vypíše rychlost publikování daného tématu.
`$ rostopic info`	`/topic_name`	Vypíše informace o aktivním tématu.
`$ rostopic list`		Vypíše seznam aktivních témat.
`$ rostopic pub`	`/topic_name msg_type args`	Publikace hodnoty do tématu s daným typem zprávy.
`$ rostopic type`	`/topic_name`	Vypíše se typ zprávy daného tématu.

Služby (Services)

Služby ROS se používají hlavně v distribuovaném systému, kde je často vyžadována komunikace typu požadavek/odpověď (request/response). Model komunikace publikování/odebírání (publish/subscribe) v tomto případě nelze použít, neboť témata (topics) nemohou nativně tento typ komunikace zprostředkovat; komunikace prostřednictvím témat je mnohostranně jednosměrná.

Služby ROS jsou definovány pomocí dvojice zpráv (messages); jedna obsahuje datový typ žádosti a druhá datový typ odpovědi. Soubory s příponou `.srv`, obsahující příslušné definice zpráv jsou uloženy ve složce srv uvnitř balíčku.

Jeden uzel funguje jako server ROS, na kterém může klient požádat o službu ze serveru; klient volá službu zasláním žádosti s požadavkem a čeká na odpověď ze serveru (např. uzel, který funguje jako server ROS je schopný nám poskytnout součet dvou čísel přijatých na vstupu; ostatní uzly mohou prostřednictvím této služby požadovat součet dvou čísel). Témata narozdíl od služeb slouží pro streamování nepřetržitého toku dat.

Informace o službách ROS je možné získat pomocí dvou nástrojů ROS příkazového řádku. Prvním z nich je `rossrv`, který je podobný `rosmsg`. Druhý je `rosservice`, který se používá k výpisu a dotazování na spuštěné služby ROS (viz tab. 7) [16, 17, 20, 21, 22].

Tab. 7: Příkazy nástroje příkazového řádku `rosservice`

Příkaz	Argumenty	Popis
`$ rosservice call`	`/service_name service-args`	Volání služby pomocí zadaných parametrů.
`$ rosservice find`	`service_type`	Hledání služby podle typu služby.
`$ rosservice info`	`/service_name`	Vypíše informace o dané službě.
`$ rosservice list`		Vypíše seznam aktivních služeb spuštěných na systému.
`$ rosservice type`	`/service_name`	Zobrazí typ služby dané služby.
`$ rosservice uri`	`/service_name`	Vypíše, jakou adresu služba používá.
`$ rosservice call`	`/service_name service-args`	Volání služby pomocí zadaných parametrů.

Soubory typu bag

Soubory typu bag slouží k ukládání ROS zpráv z témat (topics) a služeb (services). Přípona `.bag` reprezentuje soubory typu bag, které jsou vytvářeny pomocí příkazu `rosbag`, který předplatí (subscribe) jedno nebo více témat a ukládá data zprávy (message) do tohoto souboru; zpětně je možné nahraná témata přehrávat, nebo přemapovat na témata nová.

Hlavní aplikací nástroje `rosbag` je zaznamenávání dat; data robota mohou být zaznamenána, vizualizována a zpracována offline, např. tento princip byl využit pro dlouhodobé protokolování diagnostiky hardwaru robota PR2.

Příkazy, které se využívají pro nahrávání a zpětné přehrání souboru bag, se nachází v následující tabulce (viz tab. 8). [16, 17, 20, 21, 22]

Tab. 8: Příkazy nástroje příkazového řádku `rosbag`

Příkaz	Argumenty	Popis
`$ rosbag record`	`/topic_name1 /topic_name2 -o bag_name`	Nahraje daná témata do souboru bag.
`$ rosbag play`	`bag_name`	Přehraje existující soubor bag.

Server parametrů (Parameter Server)

Uzly používají tento server k ukládání a načítání parametrů za běhu; jedná se tedy o sdílený server, ke kterému mají všechny uzly přístup, a tedy mohou používat v něm uložené parametry, aniž by se navzájem ovlivňovaly. Typickou aplikací může být konfigurace PID regulátoru, kde hodnoty proporcionální, integrační a derivační složky uložíme jako parametry na tento server.

Nástroj příkazového řádku `rosparam` se používá k získání a nastavení parametru serveru pomocí `YAML` syntaxe; parametry jsou tedy uloženy v souboru s příponou `.yaml`. Příkazy, které k tomu slouží jsou zaznamenány v následující tabulce (viz tab. 9). [16, 17, 20, 21, 22]

Tab. 9: Příkazy nástroje příkazového řádku `rosparam`

Příkaz	Argumenty	Popis
`$ rosparam set`	`parameter_name parameter_value`	Nastaví hodnotu daného parametru.
`$ rosparam get`	`parameter_name`	Načte hodnotu daného parametru.
`$ rosparam load`	`YAML_file`	Načtení souboru `YAML` na server.
`$ rosparam dump`	`YAML_file`	Vypíše existující parametry do souboru `YAML`.
`$ rosparam delete`	`parameter_name`	Vymaže daný parametr.
`$ rosparam list`		Vypíše seznam názvů parametrů.

Master (ROS Master)

ROS Master pracuje na podobném principu jako DNS server; přiřazuje jedinečná jména a ID aktivním prvkům ROS v našem systému. ROS Master také poskytuje server parametrů.

Spuštění jakéhokoli uzlu v systému ROS je doprovázeno s hledáním ROS Master, ve kterém uzel registruje svůj název. ROS Master tedy obsahuje podrobnosti o všech uzlech aktuálně spuštěných v systému ROS. Tyto podrobnosti jsou důležité pro spojení mezi uzly (node-to-node connection). Uzel publikující téma provede registraci svého jména v ROS Master a zároveň poskytne podrobnosti o daném tématu. Následně ROS

Master zkontroluje, zda jsou některé uzly přihlášené ke stejnému tématu, a pokud ano, tak sdílí podrobnosti mezi uzlem vydavatele a předplatitele; dojde k propojení těchto uzlů pomocí protokolu TCPROS, který je založen na soketech TCP/IP. Po připojení těchto uzlů nemá ROS Master žádnou roli při jejich ovládání. Zastavíme-li jakýkoli uzel, ať už vydavatele nebo předplatitele, ROS Master provede opětovnou kontrolu. Stejná metoda se využívá i u služeb [15, 16, 17, 18, 19, 20, 21, 22].

3.4.3 Úroveň komunity ROS

Komunita ROS se skládá z vývojářů a vědců, kteří mohou nejen vytvářet a udržovat balíčky, ale i vyměňovat své poznatky, informace o existujících balíčcích. Celý tento model spolupráce, resp. kolaborativní činnosti stál za vznikem systému ROS (viz podkapitola 3.3).

Neúměrně době existence celosvětově vzrostla komunita uživatelů, a to nejen ve výzkumných oblastech, ale i v komerčním sektoru – s ROS se setkáváme v průmyslové a servisní robotice.

ROS komunita je velice aktivní – podílí se na vývoji nového softwaru, a to jak v rámci svých projektů, tak i kompatibilitě ROS distribucí napříč verzemi Linuxu. Poskytuje své znalosti ostatním vývojářům, např. prostřednictvím ROS wiki, Q&A stránek atp. O aktivitě vypovídají informace z oficiálních stránek ROS: více než 1500 účastníků e-mailové konference ros-users; více než 3300 uživatelů podílejících se na dokumentaci ROS wiki; více než 5700 uživatelů na komunitních fórech; přes 22 000 stránek v rámci ROS wiki; přes 13 000 otázek na Q&A fórech (míra odpovědí 70 %). [15, 20]

3.5 Předpoklady pro práci v ROS

3.5.1 Volba OS a distribuce ROS

Aktuálně je podporována více než jedna distribuce ROS; ROS Kinetic Kame a ROS Melodic Morenia (obr. 31). Výběr, v rámci této práce, bude podmíněn konvencemi prostředí Linuxu a doporučeními tvůrců robotické platformy TurtleBot3, mj. volba operačního systému (OS) bude vycházet nejen ze stejných předpokladů, ale bude i ovlivněna zvolením distribuce ROS.

Klíčovým faktorem, při výběru vhodné verze softwaru, je záruka dlouhodobé podpory, tzn. aktualizace softwaru, řešící zabezpečení, opravu systémových chyb atp. Obě výše zmíněná distribuce ROS tento benefit sdílejí.

Robotická platforma TurtleBot3 byla vytvořena a testována na verzi ROS Kinetic Kame. V době psaní této práce balíčky ROBOTIS ROS podporují ROS Melodic Morenia, nicméně, doporučení tvůrců odkazuje na použití verze ROS Kinetic Kame, kde je zajištěna funkčnost a podpora balíčků třetích stran.

ROS Kinetic Kame podporuje pouze Ubuntu 15.10 (Wily Werewolf), Ubuntu 16.04 (Xenial Xerus) a Debian 8 (Jessie). Mezi uživateli ROS je nejrozšířenější linuxová distribuce Ubuntu. Zvolíme tedy verzi Ubuntu 16.04.6 LTS (Xenial Xerus). [13, 16]

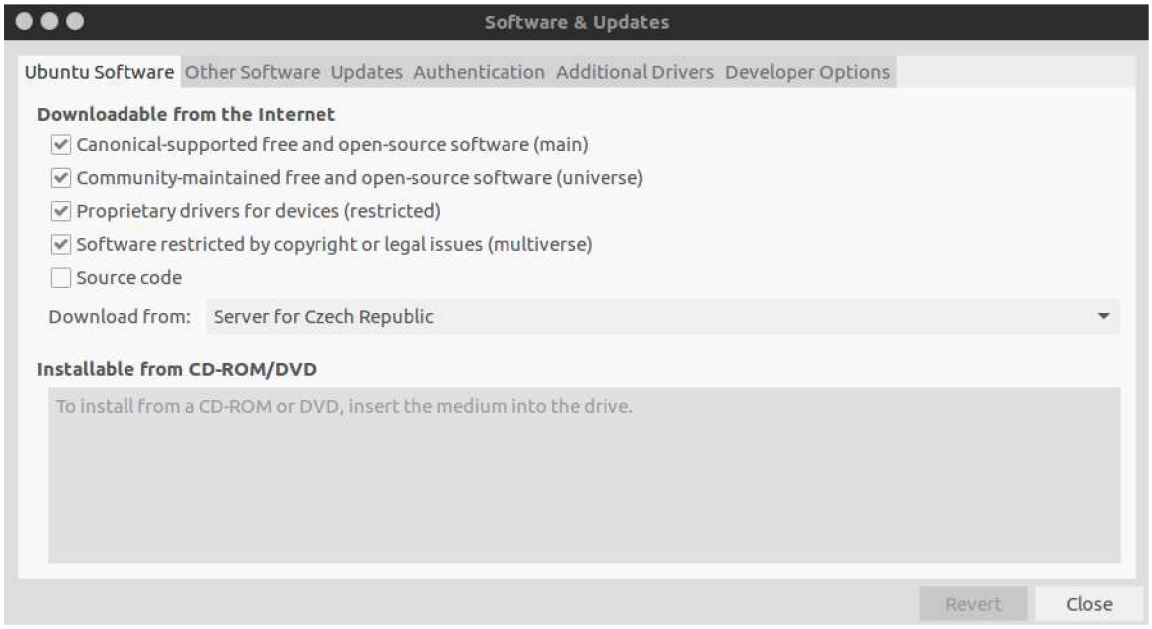
Obr. 31: Distribuce ROS s dlouhodobou podporou (LTS) [16]

3.5.2 Instalace ROS Kinetic Kame na Ubuntu 16.04.6 LTS (Xenial Xerus)

Jednotlivé kroky instalace, a jejich vysvětlení vychází z příslušných publikací a oficiálních stránek ROS [16, 18, 19, 22].

Konfigurace repositářů Ubuntu

K instalaci ROS musíme povolit přístup našeho systému k celému repositáři Ubuntu. Daný krok provedeme pomocí nástroje *Software & Updates*, a to zatržením repositářů *main, universe, restricted, multiverse* (viz obr. 32).



Obr. 32: GUI nástroj *Software & Updates*

Konfigurace Network Time Protocol (NTP)

Minimalizace časového rozdílu mezi různými počítači určením stejného serveru NTP, resp. synchronizace času je klíčovým faktorem u časově podmíněných aplikací, a aplikací citlivých na změny načasování v systému.

```
$ sudo apt-get install -y chrony ntpdate build-essential
$ sudo ntpdate ntp.ubuntu.com
```

Nastavení souboru sources.list

Soubor `sources.list` je v podstatě plánem systému, obsahující informace o zdrojích, ze kterých může stahovat software pro instalaci nebo upgrade. Přidání adresy repositáře ROS do souboru `sources.list` povolí našemu systému příjem softwaru z *packages.ros.org*.

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

Nastavení klíčů

Přidáním zabezpečeného klíče náš systém ověří původ balíčků a proces stahování, tzn. musí dojít k autentizaci balíčku na základě zabezpečeného klíče, aby byl daný balíček přidán do seznamu důvěryhodných zdrojů našeho systému.

```
$ sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80'
  --recv-key C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Aktualizace indexu balíčků Ubuntu

Po přidání repositáře ROS do souboru `sources.list` provedeme aktualizaci indexování seznamu balíčků Ubuntu a upgradujeme aktuálně nainstalované balíčky Ubuntu.

```
$ sudo apt-get update -y
$ sudo apt-get upgrade -y
```

Instalace ROS

Přestože ROS obsahuje velké množství knihoven a nástrojů, tak zvolíme jednu ze základních konfigurací dostupnou z http://wiki.ros.org/ [16], a to desktop-full, která zahrnuje ROS, nástroje příkazového řádku, nástroje s grafickým rozhraním, robotické knihovny, 2D/3D simulátory, navigaci robota s ohledem na vnímání 2D/3D prostoru.

V případě potřeby dodatečného balíčku je možná individuální instalace, např. všechny pluginy související s nástrojem rqt.

```
$ sudo apt-get install ros-kinetic-desktop-full
$ sudo apt-get install ros-kinetic-rqt*
```

Inicializace rosdep

Nástroj `rosdep` je užitečný nástroj pro instalaci závislých balíčků ROS. Rosdep zkontroluje, zda jsou dostupné závislé balíčky pro ROS balíček, který chceme kompilovat, a pokud ne, automaticky je nainstaluje. Doinstalováním závislých balíčků docílíme správné fungování daného ROS balíčku.

```
$ sudo rosdep init
$ rosdep update
```

Nastavení prostředí ROS

Pro přístup k nástrojům a knihovnám ROS musíme přidat proměnné pro prostředí ROS (ROS environment variables) do souboru `.bashrc`, který se načítá při každém spuštění nového terminálu.

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

Závislosti pro vytváření balíčků

Následujícím příkazem nainstalujeme závislosti pro vytváření ROS balíčků a užitečný nástroj `rosinstall`, usnadňující správu a instalaci velkého množství balíčků, včetně jejich závislostí.

```
$ sudo apt install python-rosinstall python-rosinstall-
  generator python-wstool build-essential
```

Vytvoření pracovního prostoru ROS (catkin workspace)

Sestavovací systém catkin zjednodušuje proces sestavování a instalace balíčků ROS, tzn. vytváření nových balíčků a instalaci stávajících balíčků ROS, a to generováním tzv. cílů (spustitelné soubory/knihovny) ze zdrojového kódu. Umožňuje jejich použití koncovým uživatelem.

```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws/
$ catkin_make
$ echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
```

Konfigurace souboru .bashrc

V krocích *Nastavení prostředí ROS a Vytvoření pracovního prostoru (catkin workspace)* byly přidány do souboru `.bashrc` cesty ke spuštění souboru `setup.bash`, pro přístup k ROS nebo catkin. Tímto jsme se vyhnuli repetitivní činnosti jejich zadávání při každém otevření nového terminálu.

Pomocí následujícího příkazu otevřeme soubor `.bashrc` v textovém editoru, který umožní jeho editaci.

```
$ gedit ~/.bashrc
```

Na posledních řádcích se nachází cesty k importování souborů setup.bash pro ROS a catkin.

~/.bashrc

```
...
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

Konfigurace sítě ROS pomocí parametrů ROS_MASTER_URI a ROS_HOSTNAME umožňuje komunikaci mezi více počítači. Pro práci na samostatném PC, bez připojení k jinému PC (robot), není potřeba specifikovat IP adresu a místo ní vložíme hodnotu *localhost*.

~/.bashrc

```
...
export ROS_HOSTNAME=localhost
export ROS_MASTER_URI=http://localhost:11311
```

Nepovinný, nicméně užitečný krok, přidání zkratek pro často používané příkazy.

~/.bashrc

```
...
alias eb='gedit ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'
```

Vložíme pro přehlednost komentáře k jednotlivým částem a provedeme uložení dokumentu klávesovou zkratkou [ctrl+s]. Všechny provedené změny nastavení budou aplikovány, tzn. budou funkční po spuštění nového okna terminálu anebo zadáním následujícího příkazu.

```
$  source ~/.bashrc
```

Upravený soubor `.bashrc`.

~/.bashrc

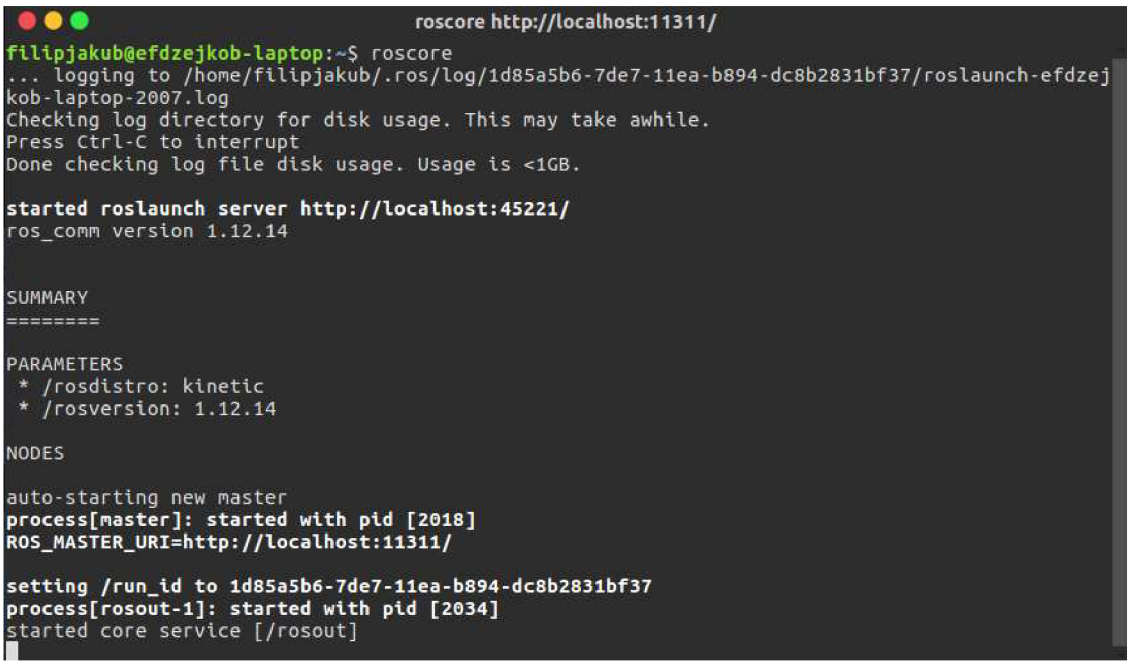
```
...
# Set ROS Kinetic Kame (ROS Development Environment)
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

```
# Set ROS Network
export ROS_HOSTNAME=localhost
export ROS_MASTER_URI=http://localhost:11311

# Set ROS alias command
alias eb='gedit ~/.bashrc'
alias sb='source ~/.bashrc'
alias gs='git status'
alias gp='git pull'
alias cw='cd ~/catkin_ws'
alias cs='cd ~/catkin_ws/src'
alias cm='cd ~/catkin_ws && catkin_make'
```

Test roscore (ověření instalace)

Správnost instalace ověříme spuštěním příkazu `roscore`, kde získáme tento výpis z terminálu (viz obr. 33).



Obr. 33: Výstup z terminálu po zadání příkazu `roscore`

3.6 ROS ukázka: vysvětlení mechanismu komunikace vydavatel/odběratel

Terminologie, resp. teoretický základ pro práci v ROS byl položen v podkapitole 3.4. Tyto znalosti budou demonstrovány na praktické úloze, která bude obsahovat dva uzly: první `talker.py` (vydavatel) a druhý `listener.py` (odběratel). Uzly jsou součástí instalace ROS.

Uzel mluvčího, `talker.py`, bude publikovat informaci, která se bude nacházet v textovém řetězci, tzn. datový typ string. Tato informace je zpráva publikována na dané

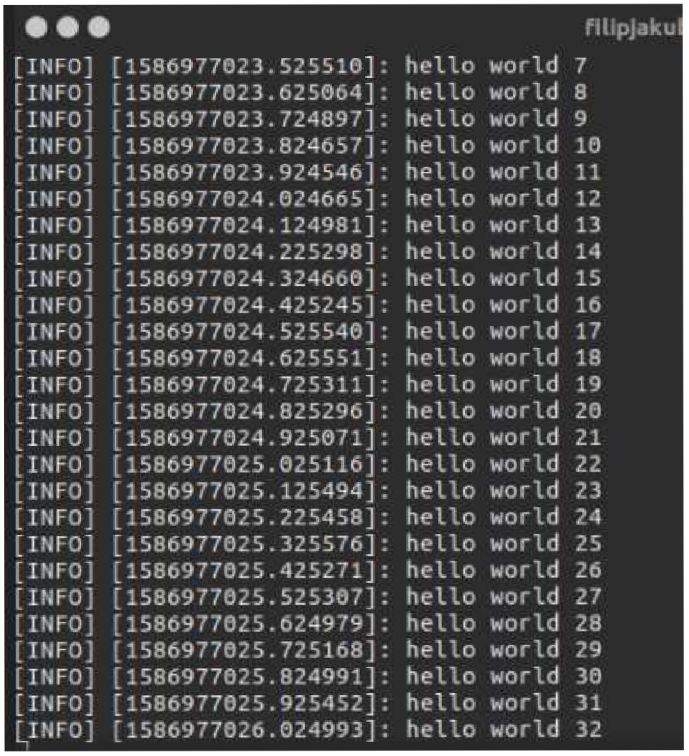
téma. Uzel posluchače, `listener.py` se přihlásí k odběru tohoto téma a zprávu obsaženou v něm vypíše do konzole.

Prvním krokem je spuštění `roscore`, aby mohlo dojít k zaregistrování uzlů, a tedy ke zprostředkování komunikace.

```
$ roscore
```

Otevřeme druhé okno terminálu a spustíme uzel `talker.py`, který začne publikovat.

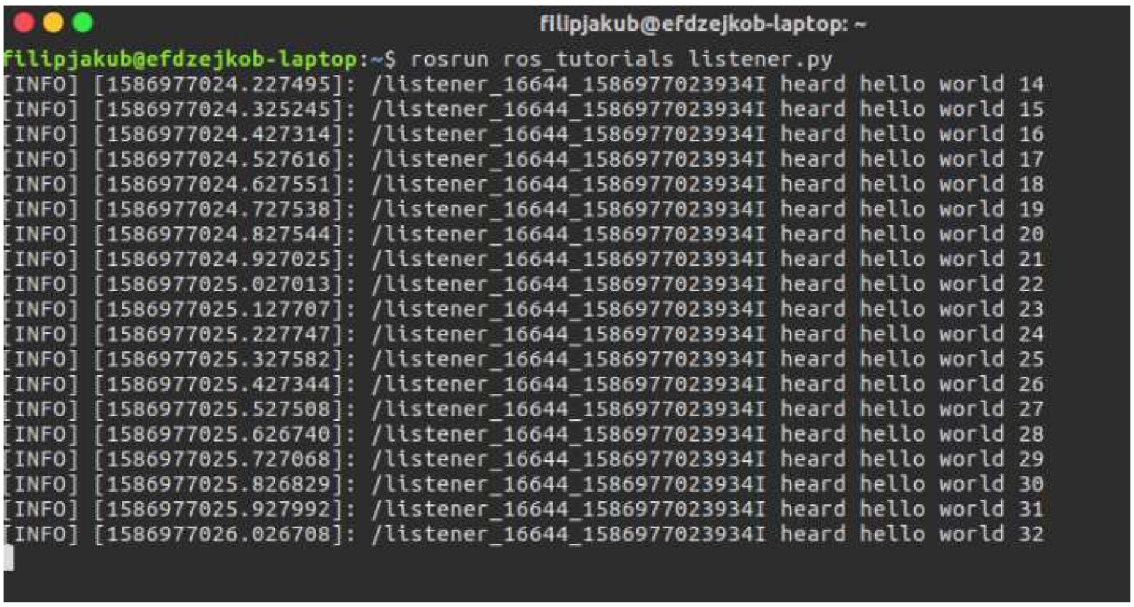
```
$ rosrun ros_tutorials talker.py
```



Obr. 34: Výstup z terminálu po spuštění uzlu `talker.py`

Ve třetím okně terminálu spustíme uzel `listener.py`, který se přihlásí k odběru daného téma

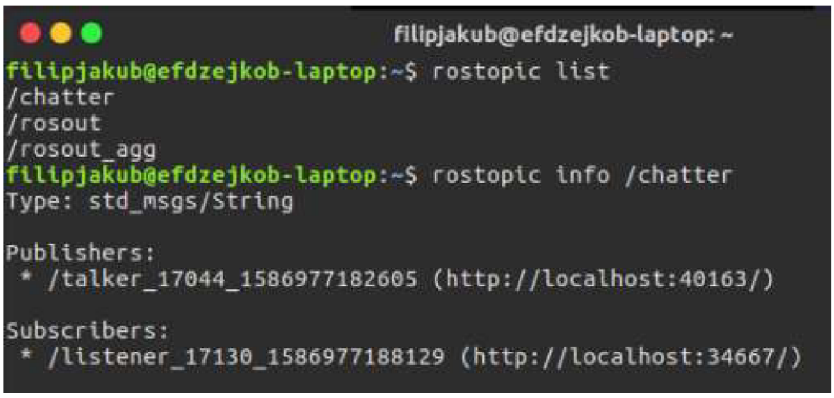
```
$ rosrun ros_tutorials listener.py
```



Obr. 35: Výstup z terminálu po spuštění uzlu `listener.py`

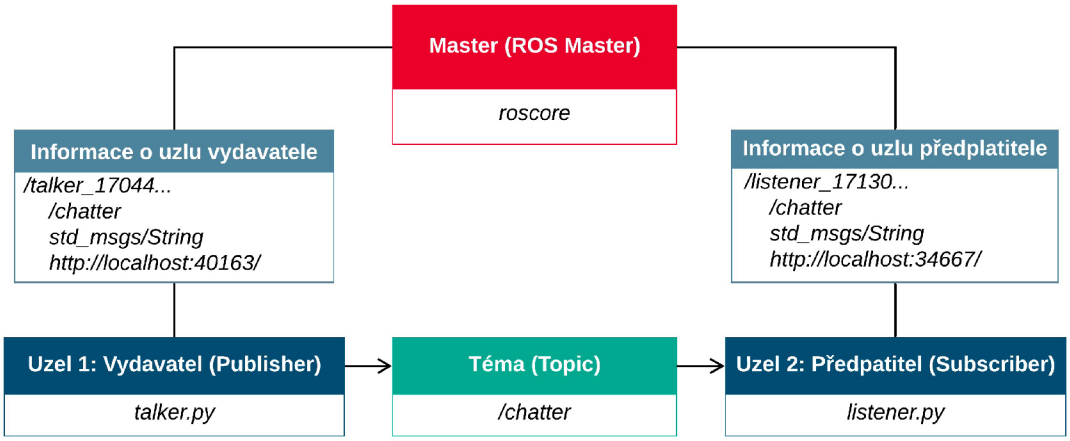
Jelikož je to přichystaný skript, tak nevíme název daného téma, a v podstatě ani o jaký datový typ se jedná. Zde využijeme již představené příkazy z podkapitoly 3.4.

```
$ rostopic list
$ rostopic echo /nazev_tema
```



Obr. 36: Informace o daném téma pomocí nástroje příkazového řádku `rostopic`

Z výpisu je vidět, že jsou momentálně k dispozici tři témata; téma našeho zájmu je *chatter*, které přenáší standardní zprávu *std_msgs/String*, kterou je textový řetězec. Další informace poskytují přehled o tom, kolik a jaké uzly využívají toto téma ke komunikaci, a z jakého zařízení se k ní připojují.



Obr. 37: Diagram znázorňující komunikaci v ROS (odpovídající řešené úloze)

3.7 Nástroje ROS

Nástroje příkazového řádku jsou vhodné pro zaznamenávání, monitorování a zpracování dat, díky kterému jsme schopni provádět ladění robotických aplikací. Při práci na komplexnějších robotických projektech, kde je využíváno hodně balíčků, které seskupujeme do metabalíčku, je žádoucí data získaná z akčních členů, sensorů, resp. obsahy zpráv daných témat vizualizovat, a to nejen offline, ale i v reálném čase.

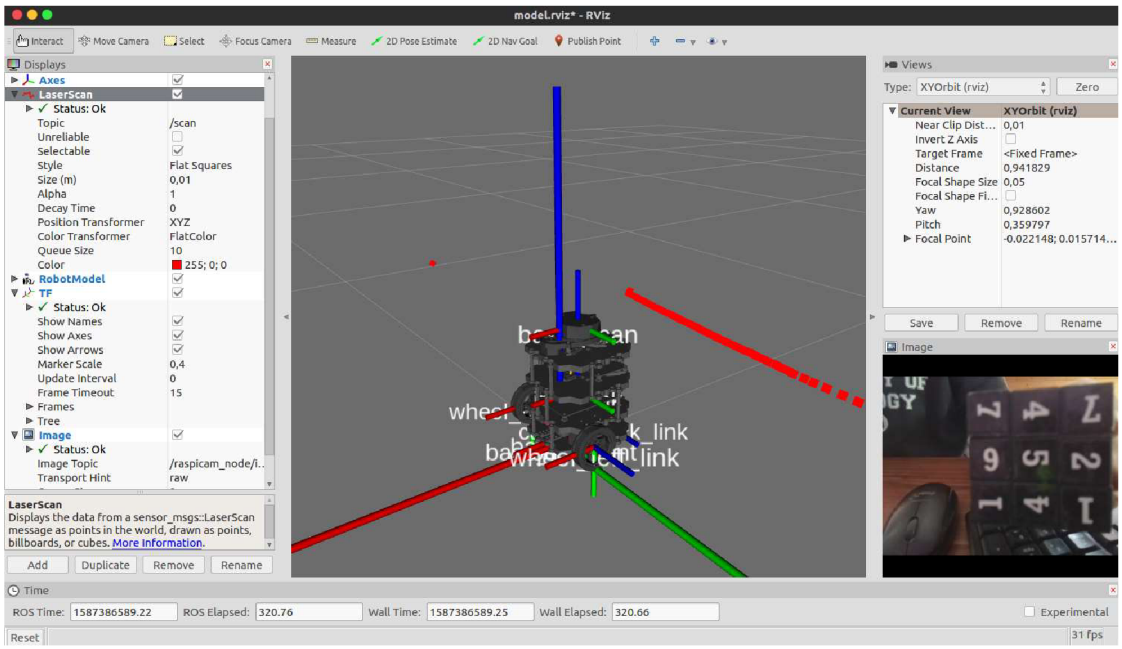
Pro tyto požadavky existují integrované vizualizační nástroje ROS, jejichž instalace byla zahrnuta v rámci podkapitoly 3.5.2.

3.7.1 RViz

RViz je nástroj s GUI (Graphical User Interface), tzn. s grafickým uživatelským rozhraním, určený pro trojrozměrnou vizualizaci různých typů dat ze senzorů, resp. různých typů zpráv, tzn. senzor musí obsahovat balíčky ROS, které mu umožní pracovat v komunikačním mechanismu vydavatel/odběratel, a také umožňuje vykreslení modelů jakýchkoli robotů popsaných souborovým formátem URDF (blíže popsáno v podkapitole 3.8).

RViz umí vizualizovat mnoho běžných typů zpráv přímo integrovaných v ROS, jako jsou data z laserového skeneru, trojrozměrných bodových mraků a obrázků z kamer. Dále je schopen využívat informace z knihoven tf , které uživateli umožňují sledovat více souřadných rámců v reálném čase, udržují vztah mezi souřadnicovými rámci ve stromové struktuře, poskytují uživateli možnost transformovat body, vektory atd. mezi dvěma libovolnými souřadnicovými rámci v libovolném časovém bodě. Robotický systém obsahuje obvykle mnoho trojrozměrných souřadnicových rámců, které se v průběhu času mění, např. světový rámec (world frame), základní rámec (base frame), rámec koncového efektoru (gripper frame) atd. Díky tomuto propojení s tf je RViz schopný zobrazit senzorová data ve společném souřadnicovém rámci podle vašeho výběru, spolu s trojrozměrným vykreslením robotu, což nám poskytuje informaci o správném vyrovnání senzorů anebo nepřesnostech modelu od reálného robotu [15, 16].

Příkladem může být vizualizace TurtleBot3 Burger, znázorněná na obr. 38, na které je možné vidět výstup z kamery (pravý dolní roh), souřadnicové rámce modelu robotu (střední oblast GUI), červenými body označené překážky v prostoru získané laserovým senzorem vzdálenosti (střední oblast GUI). V levé oblasti GUI se nacházejí informace o senzorech, modelu robotu, tf, laserovém senzoru vzdálenosti, kameře atp. Udávají informace o funkčnosti, tzn. zdali pracují správně, odebírají správná témata, včetně možnosti konfigurace některých parametrů.



Obr. 38: TurtleBot3 Burger vizualizovaný v RViz

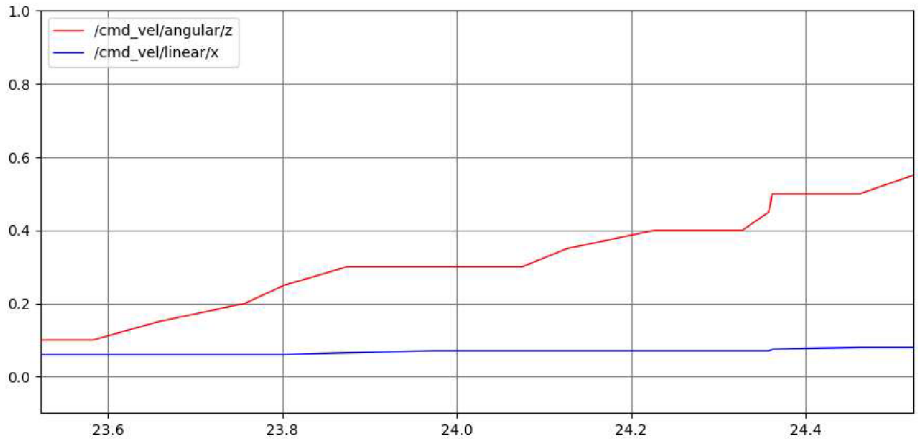
3.7.2 rqt

Framework rqt umožňuje vývoj grafických rozhraní pro váš robot, a to buď vytvořením vlastních pluginů pro rqt anebo konfigurací existujících rozsáhlých knihoven vestavěných modulů rqt.

Plugin `rqt_graph` poskytuje možnost introspekce a vizualizace aktuálně běžícího systému ROS, zobrazuje uzly a propojení mezi nimi a umožňuje snadno ladit a porozumět běžícímu systému a jeho struktuře [15].

Obr. 39: `rqt_graph` použitý pro příklad v podkapitole 3.6

Plugin `rqt_plot` slouží k vykreslování dat do grafu, tzn. může být použit při sledování chování enkodéru, průběhu napětí nebo čehokoliv, co lze reprezentovat jako číslo měnící se v čase [15].



Obr. 40: `rqt_plot` pro vykreslení závislosti rychlosti (vertikální osa) na čase (horizontální osa) u robotu TurtleBot3 Burger ovládaného klávesnicí: lineární (modrá), rotace (červená)

Plugin `rqt_topic` umožňuje sledovat a prozkoumat libovolný počet témat publikovaných v systému, a `rqt_publisher` umožňuje publikování vlastních zpráv k libovolnému tématu, což umožňuje experimentovat s právě běžícím systémem [15].

3.8 Modelování robotů v ROS

V první fázi vývoje robotů je návrh a modelování, ke kterému se využívá běžně používaných CAD nástrojů (např. Autodesk, SOLIDWORKS, Creo atd.). Důvodem modelování robotů je simulace, která dokáže zkontrolovat kritické nedostatky v konstrukci robotu, což přispívá k optimalizaci konstrukce před samotnou fází výroby.

Model virtuálního robotu musí mít všechny vlastnosti skutečného robotu, resp. jeho hardwaru, což v praxi znamená možnost absence přesných tvarů, ale nikoliv výsledných fyzikálních vlastností.

Klasické formáty reprezentující trojrozměrný model robotu nejsou v ROS podporovány. ROS obsahuje vlastní balíčky, které pomáhají při navrhování robotů. Vytvoření modelu robotu v ROS je důležité z různých důvodů, např. možnost simulace a řízení robotu, vizualizace robotu v nástroji RViz či získání informací o robotické struktuře a její kinematice pomocí nástrojů ROS [22].

URDF (Universal Robotic Description Format)

Univerzální formát robotického popisu (URDF) je formát souboru `xml` používaný v ROS k popisu všech prvků robotu. Pomocí tohoto standardizovaného formátu lze modelovat robot, tzn. kinematický, dynamický model robotu, včetně jeho vizuální reprezentace a kolizního modelu.

V oblasti moderních trendů robotiky jsou však některé jeho funkce neaktualizované, resp. zastaralé, např. nemůže specifikovat pozici robotu v prostoru, spojovací smyčky (paralelní vazby), tření, a věci, které nejsou součástí robotu [22, 23].

Xacro (XML Macros)

Problém s URDF nastává při práci se složitými modely robotů, Snižuje se jeho flexibilita použití, tj. vlastnosti, které URDF chybí jsou např. jednoduchost, opakovaná použitelnost, modularita a programovatelnost. Řešením těchto negativních vlastností je xacro, které by se dalo považovat za aktualizovanou verzi URDF nebo také vyčištěnou verzi URDF.

Na rozdíl od URDF vytváří makra uvnitř popisu robotu, která jsou znovupoužitelná v dalších částech kódu, kde mohou být rovněž zahrnuta makra z jiných souborů, což usnadňuje kód, činí ho čitelnějším a modulárnějším. Pro představu, v URDF, pokud někdo chtěl využít blok kódu 10krát, tak ho musel 10krát nakopírovat, a pokud chtěl provést změny, tak každou kopii zvlášť nakonfigurovat. Dále nebylo možné do něj zahrnout další soubory. Mezi největší výhodu patří možnost použití jednoduchých programovacích principů, tedy konstant, proměnných, matematických výrazů, podmíněných výrazů (logika) atp. [22]

3.9 Gazebo

Simulace robotů je nezbytným nástrojem v sadě nástrojů každého robotu. Umožňuje rychle a efektivně testovat algoritmy, navrhovat roboty, provádět regresivní testování a trénovat systém (strojové učení) pomocí realistických scénářů, a to nejen ve vnitřních, ale i venkovních prostředích.

Gazebo je 3D simulátor poskytující modely robotů, generování dat senzorů ve vytvořeném prostředí, modely prostředí pro 3D simulaci, která je potřebná pro vývoj robotů, a nabízí realistickou simulaci pomocí nástrojů simulujících fyzikální systémy (physics engine).

Gazebo je jedním z nejpopulárnějších simulátorů, a dokonce se stal i oficiálním simulátorem v DARPA Robotics Challenge v USA. Přestože je open-source, tak je vysoce výkonný, a nabízí mnoho funkcí, které jsou klíčové, resp. hodnotné v oblasti robotiky [19, 23]:

- **Simulace dynamiky:** Okolní prostředí, a výsledné mechanické a fyzikální vlivy vychylují chování robotů od ideálního modelu, který uvažujeme nejčastěji v kinematice, ale dynamika a schopnost její simulace umožňuje optimalizovat různé robotické aplikace. Gazebo k tomuto účelu využívá vysoce výkonné nástroje simulující dynamické vlastnosti daných robotických platforem, a to ODE (Open Dynamics Engine), Bullet Real-Time Physics Simulation, Simbody: Multibody Physics API a DART (Dynamic Animation and Robotics Toolkit).
- **Pokročilá 3D grafika:** S využitím OGRE (Open-source Graphics Rendering Engines), který se často používá ve hrách, poskytuje Gazebo realistické vykreslování prostředí včetně vysoce kvalitního osvětlení, stínů a textur.

- **Senzory a simulace šumu:** Laserový dálkoměr, 2D/3D kamera, hloubková kamera, kontaktní senzor, snímač síly, točivého momentu; generování dat senzorů, volitelně s šumem, umožňuje nastínění reálného prostředí.
- **Pluginy:** Rozhraní Gazebo API jsou poskytována uživatelům, aby mohli vytvářet vlastní pluginy pro roboty, senzory a prostředí včetně jejich kontroly.
- **Modely robotů:** K dispozici je mnoho robotů včetně PR2, Pioneer2 DX, iRobot Create a TurtleBot. Nabízí i možnost vytvoření vlastního pomocí SDF.
- **Přenos dat TCP/IP:** Simulaci je možné spustit na vzdálených serverech, a pracovat s rozhraním Gazebo prostřednictvím předávání zpráv založených na soketu pomocí Google Protobufs.
- **Simulace v cloudu:** Gazebo poskytuje cloudové simulační prostředí CloudSim pro použití v cloudových prostředích, jako jsou Amazon, Softlayer a OpenStack.
- **Nástroje příkazového řádku:** Rozsáhlé nástroje příkazového řádku usnadňují simulaci introspekce a kontroly, tzn. ověření a řízení simulace je možné prostřednictvím nejen GUI (Graphical User Interface), ale i CUI (Console User Interface).

SDFormat (Simulation Description Format)

SDFormat (formát popisu simulace) je formát souboru `xml`, který popisuje objekty a prostředí pro simulátory robotů, vizualizaci a ovládání. Původně byl vyvinutý jako součást robotického simulátoru Gazebo s ohledem na vědecké aplikace robotů.

V současné době se stal stabilním, robustním a rozšiřitelným formátem schopným popsat všechny aspekty robotů, statických a dynamických objektů, osvětlení, terénu, a dokonce i fyziky.

Záměrem vzniku tohoto nového formátu využitelného pro modelování a simulaci robotů byly odstranění nedostatků URDF (představeného v kapitole 3.8), a poskytnutí plnohodnotné dokumentace, a aktualizace s ohledem na aktuální požadavky v oblasti robotiky. [23]

Integrace ROS s Gazebo

Simulátor Gazebo je vyvíjen a distribuován společností Open Robotics, což zajišťuje plnou kompatibilitu s ROS.

Integraci ROS se samostatným simulátorem Gazebo umožňuje sada balíčků s názvem `gazebo_ros_pkgs`, která poskytuje nezbytná rozhraní pro simulaci robotu v Gazebo pomocí zpráv, služeb a dynamické rekonfigurace. A navíc poskytuje užitečné funkce, např. podporu samostatné systémové závislosti Gazebo, která nemá žádné vazby s ROS; umí zacházet s URDF a SDF (převádět z formátu URDF do SDF); vylepšuje podporu ovladačů využívajících `ros_control`; vylepšuje efektivitu řídícího modulu v reálném čase atd.

V podkapitole 3.5.2 v části *Instalace ROS* bylo Gazebo součástí zvoleného instalačního balíčku ROS, a tedy není nutné řešit jeho dodatečnou instalaci. [23]

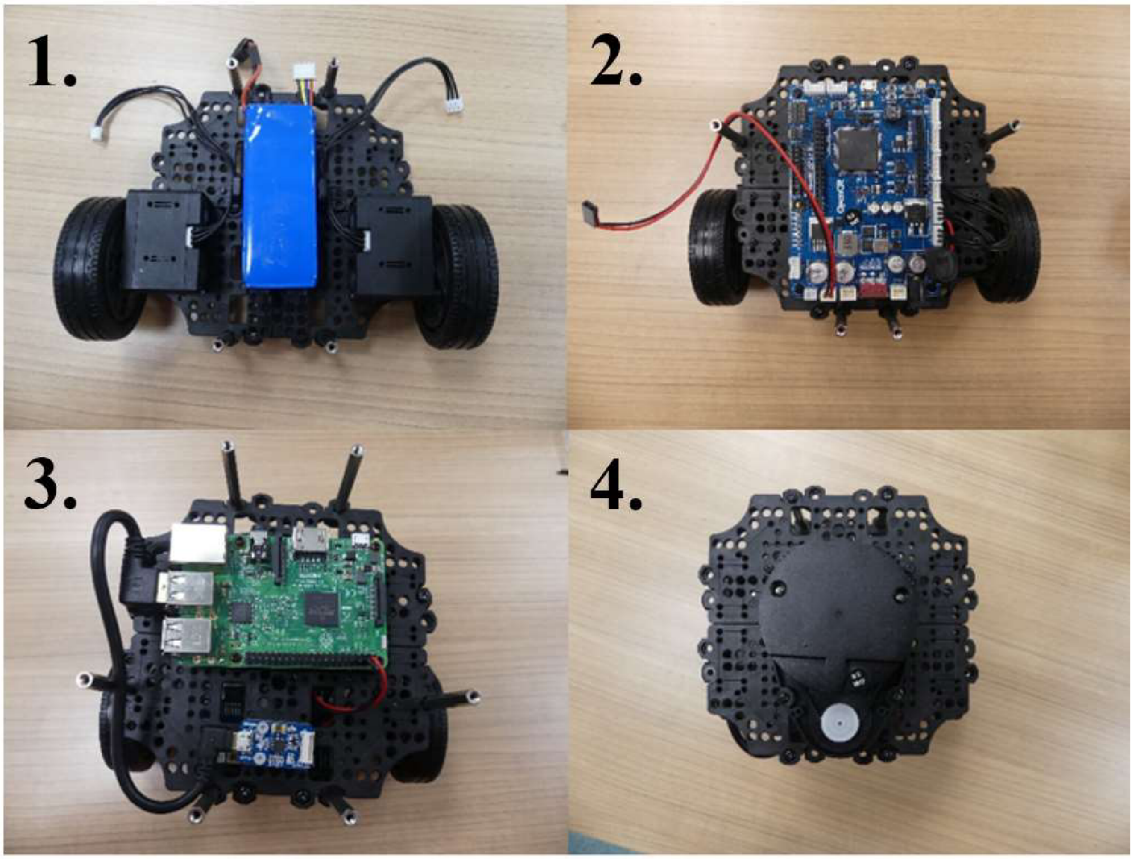
4 SESTAVENÍ A KONFIGURACE MOBILNÍHO ROBOTU TURTLEBOT3 BURGER

4.1 Sestavení robotické stavebnice

Získání komponent pro sestavení je možné, např. vytisknutím jednotlivých částí na 3D tiskárně, včetně zakoupení jednotlivých hardwarových komponent, nebo koupí stavebnice, resp. všech jejich částí v jednotně uceleném balíčku.

Obsah balení se liší s ohledem na daný model ze série TurtleBot3. V rámci téhož modelu se obsah balení neliší s ohledem na seznam jednotlivých částí, novější distribuce se mohou lišit pouze verzemi hardwarových komponent, jak bylo zmíněno v podkapitole 2.7.2, kde místo Raspberry Pi 3 Model B je Raspberry Pi 3 Model B+, který zaručuje dlouhodobou podporu s ohledem na hardware i software.

Sestavení probíhá v několika krocích, dle přiloženého manuálu, který je napsán v čínštině, japonštině, korejštině a angličtině. Jednotlivé kroky jsou sloučeny do pater, pro které jsou zaznačeny použité komponenty, zapojení výpočetního hardwaru, senzoriky a akčních členů. Celý postup je možné vidět ve zjednodušené formě na obr. 41 (dodatečné snímky, zachycující celý proces montáže, se nacházejí v příloze).



Obr. 41: Zjednodušeně znázorněná ukázka kroků sestavení

4.2 Hardwarové rozšíření o Raspberry Pi kameru V2

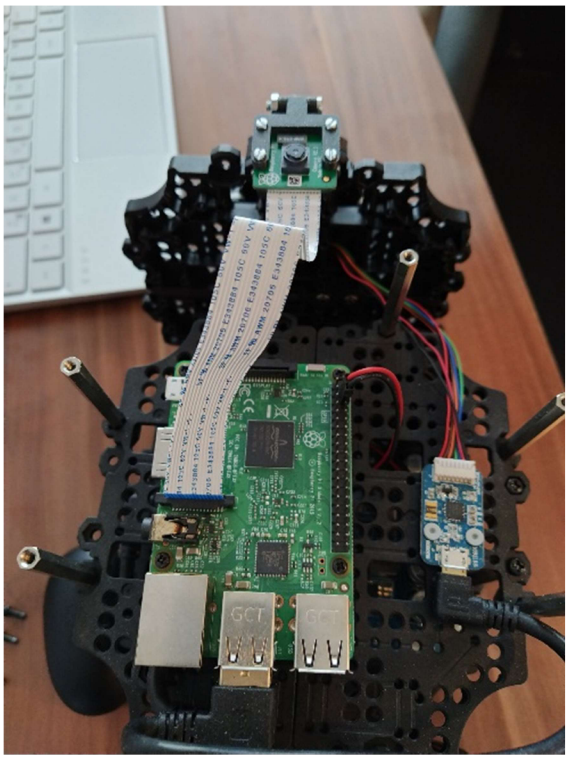
V podkapitole 2.7.2 v části *Senzorika* byla zmíněna absence možnosti počítačového vidění ve výchozí konfiguraci. Volba Raspberry Pi kamery V2 byla podmíněna existencí zdrojového kódu pro model TurtleBot3 Waffle Pi, který je použitelný i pro náš model, a dostupností jednoho kusu ve vlastnictví vedoucího bakalářské práce.

Pro upevnění kamery na již sestaveném robotu byl v programu SOLIDWORKS vymodelován držák s polohovatelným mechanismem, načež byl vytisknut na 3D tiskárně.

Raspberry Pi kamera V2

Tento typ kamery má 8megapixelový snímač Sony IMX219, což oproti minulému modelu posunuje vpřed kvalitu obrazu, věrnost barev a možnost použití i za slabého osvětlení. Podporuje režimy videa 1080p30, 720p60 a VGA90, a také umožňuje pořizovat fotografie.

Kamera pracuje se všemi modely jednodeskových počítačů (SBC) Raspberry Pi (Raspberry Pi 1, 2, 3 a 4), a je přístupná prostřednictvím rozhraní MMAL API a V4L API, včetně mnoha knihoven třetích stran. Připojení k SBC je zprostředkováno 15cm plochým kabelem k CSI portu (viz obr. 42). [14]



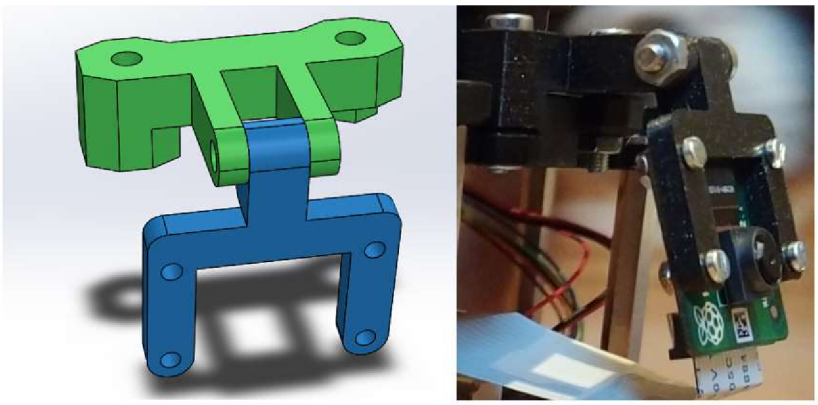
Obr. 42: Připojení Raspberry Pi kamery V2 k SBC Raspberry Pi 3 Model B

Držák pro kameru Raspberry Pi V2

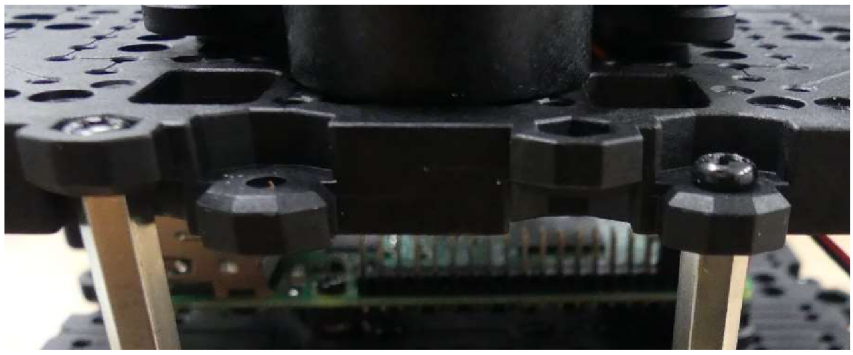
Držák se skládá ze dvou částí (obr. 43), kde fixně upevněná část (zelená) odpovídá tvarovým plochám nosných částí tzv. waffle plates (obr. 44), a pohyblivá část (modrá) je navržena s ohledem na rozměry samotné kamery.

Spojení jednotlivých částí držáku mezi sebou, robotem a kamerou bylo provedeno ocelovými šrouby s plochou hlavou a rovnou drážkou odpovídající DIN 84A (pozinkované), a ocelovými šestihrannými maticemi odpovídající DIN 934 (pozinkované), které jsou distribuovány společností MP JET s.r.o. Počty a označení: 3x M3x16 (SKU MPJ-0238), 3x matice M3 (SKU MPJ-0106 SZ), 4x M2x4 (SKU MPJ-0218) a 4x matice M2 (SKU MPJ-0102 SZ).

Cílem celé konstrukce byla nenáročnost tisku, snadná montáž, rozměry a poloha, která se nachází pod snímací hladinou laserového skeneru vzdálenosti, a tedy nenarušuje snímaná data, a zároveň je zaručena variabilita naklonění kamery pro různé požadavky natáčení okolního prostředí.



Obr. 43: Sestava modelu držáku (vlevo), výsledná podoba (vpravo)



Obr. 44: Místo montáže držáku na 4. patře nosné části

4.3 Nastavení vzdáleného PC

V podkapitole 3.5.2 byla provedena základní instalace a konfigurace ROS pro vzdálené PC, tzn. PC, kterým se budeme připojovat k robotu.

Instrukce, resp. postup konfigurace, a tedy zprovoznění vychází z oficiálních stránek pro TurtleBot3 [13].

Instalace závislostí pro ovládání Turtlebot3

Jedná se o závislé balíčky TurtleBot3, které umožňují propojení vzdáleného PC s robotem, a tedy jeho ovládání.

```
$ sudo apt-get install ros-kinetic-joy ros-kinetic-teleop-
  twist-joy ros-kinetic-teleop-twist-keyboard ros-kinetic-
  laser-proc ros-kinetic-rgbd-launch ros-kinetic-depthimage-
  to-laserscan ros-kinetic-rosserial-arduino ros-kinetic-
  rosserial-python ros-kinetic-rosserial-server ros-kinetic-
  rosserial-client ros-kinetic-rosserial-msgs ros-kinetic-amcl
  ros-kinetic-map-server ros-kinetic-move-base ros-kinetic-
  urdf ros-kinetic-xacro ros-kinetic-compressed-image-
  transport ros-kinetic-rqt-image-view ros-kinetic-gmapping
  ros-kinetic-navigation ros-kinetic-interactive-markers
$ cd ~/catkin_ws/src/
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone -b kinetic-devel https://github.com/ROBOTIS-
  GIT/turtlebot3.git
$ cd ~/catkin_ws && catkin_make
```

Nastavení ROS sítě a výchozího modelu TurtleBot3

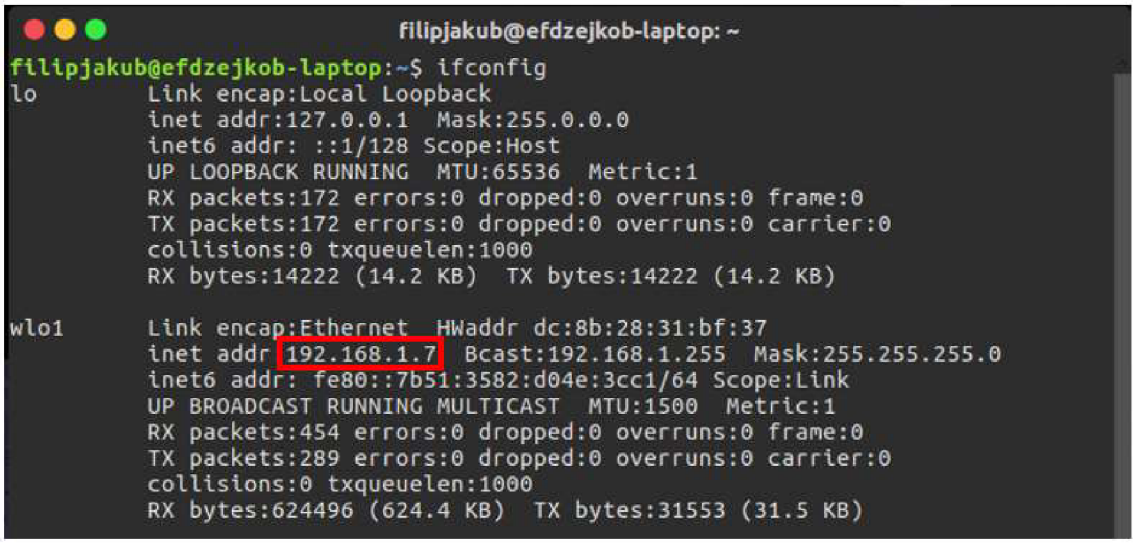
ROS vyžaduje pro komunikaci mezi vzdáleným PC a robotem IP adresy. Pro zajištění funkčnosti je nutné, aby PC i robot byly připojeny ke stejnému wifi routeru.

Změny budou prováděny v souboru `.bashrc`, kde změníme hodnoty parametrů ROS_MASTER_URI a ROS_HOSTNAME z *localhost* na IP adresu vzdáleného PC.

IP adresu vzdáleného PC zjistíme následujícím příkazem.

```
$ ifconfig
```

Kde požadovaná hodnota je zvýrazněna na obr. 45 červeným obdélníkem.



Obr. 45: Výstup z terminálu po zadání příkazu `ifconfig`

Druhým krokem je nastavení výchozího modelu TurtleBot3, čímž se zbavíme repetitivní činnosti, resp. zadávání příkazu, při připojování k robotu.

Výsledná podoba upravené části v souboru `.bashrc` je následující.

~/.bashrc

```
export ROS_HOSTNAME=192.168.1.7
export ROS_MASTER_URI=http://192.168.1.7:11311
export TURTLEBOT3_MODEL=burger
```

4.4 Nastavení SBC (Raspberry Pi 3 Model B)

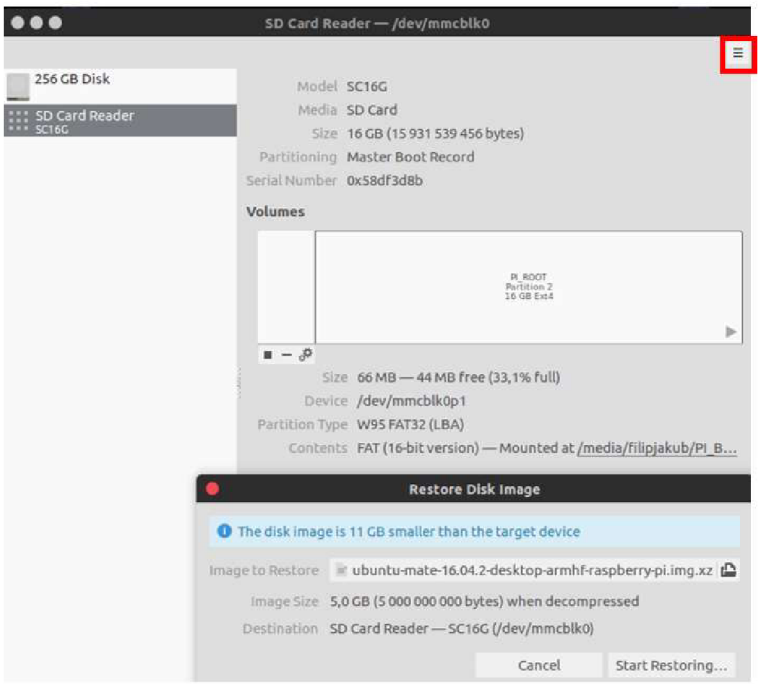
Jednodeskový počítač Raspberry Pi Model B neobsahuje úložiště, na které by bylo možné nahrát daný systém, využijeme tedy slotu pro microSD kartu. Vzhledem k modelu SBC bude námi zvolený operační systém (OS) Linux (Ubuntu Mate 16.04 LTS), a tedy minimální požadovaná kapacita microSD karty je 8 GB.

Instrukce, resp. postup konfigurace, a tedy zprovoznění vychází z oficiálních stránek pro TurtleBot3. Doporučení: při krocích prováděných na robotu nepoužívejte k napájení baterii, ale SMPS (AC adaptér). [13]

Instalace OS Linux (Ubuntu Mate 16.04 LTS)

Stažení obrazu Ubuntu Mate 16.04 LTS pro Raspberry Pi 3 Model B bude provedeno skrze PC, na kterém byly provedeny kroky v podkapitole 3.5.

K zapsání obrazu námi zvoleného OS na microSD kartu (SanDisk MicroSDHC 16 GB Ultra A1 UHS-I) provedeme pomocí programu *Disks* se zvolením možnosti *Restore Disk Image* (tlačítko v pravém horním rohu programu – červeně zvýrazněno), která podporuje XZ kompresní obrazy (viz obr. 46).



Obr. 46: Po zvolení možnosti Restore Disk Image zvolíme stažený obraz a klikneme na Start Restoring…, po dokončení operace vzniknou příslušné oddíly

Instalace ROS Kinetic Kame (pro OS na SBC TurtleBot3)

Instalace a konfigurace pro základní používaní ROS se shoduje s kroky v podkapitole 3.5.2. Pro zjednodušení instalace ROS na daném OS existují skripty, které provedou veškeré kroky za Vás. Nejprve bude provedena aktualizace nainstalovaného systému, abychom měli jistotu o aktuálnosti nainstalovaných balíčků Ubuntu, a poté bude provedena instalace ROS jedním příkazem (pozn. kroky jsou prováděny v systému robotu).

```
$ sudo apt-get update
$ sudo apt-get upgrade
$ wget https://raw.githubusercontent.com/ROBOTIS-
  GIT/robotis_tools/master/install_ros_kinetic_rp3.sh && chmod
  755 ./install_ros_kinetic_rp3.sh && bash
  ./install_ros_kinetic_rp3.sh
```

Instalace závislých balíčků pro TurtleBot3 SBC a ROS

Následuje skupina příkazů, kterými budou nainstalovány balíčky potřebné pro SBC v TurtleBot3, resp. k jeho možnému ovládání skrze vzdálené PC.

```
$ cd ~/catkin_ws/src
$ git clone https://github.com/ROBOTIS-
  GIT/hls_lfcd_lds_driver.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3_msgs.git
$ git clone https://github.com/ROBOTIS-GIT/turtlebot3.git
```

Některé balíčky jsou určené pro vzdálené PC, a tedy nejsou nutné v SBC, tzn. vymažeme je, čímž uvolníme místo v úložišti, pro případné dodatečné aktualizace.

```
$ cd ~/catkin_ws/src/turtlebot3
$ sudo rm -r turtlebot3_description/ turtlebot3_teleop/
  turtlebot3_navigation/ turtlebot3_slam/ turtlebot3_example/
```

V souvislosti s nově nainstalovanými balíčky pro TurtleBot3 je nutné nainstalovat potřebné balíčky pro ROS, aby byla zajištěna funkčnost klíčových funkcí. Po instalaci proveďte restartování SBC.

```
$ sudo apt-get install ros-kinetic-rosserial-python ros-
  kinetic-tf
```

Ověření správné instalace provedeme následujícími příkazy – během sestavování balíčků se nevyskytne žádná chybová hláška.

```
$ source /opt/ros/kinetic/setup.bash
$ cd ~/catkin_ws && catkin_make -j1
```

Nastavení práv pro USB, a ROS sítě pro komunikaci se vzdáleným PC

Následujícím příkazem se umožní používání portu USB mezi SBC a OpenCR, bez nutnosti ověření a získávání povolení pro root.

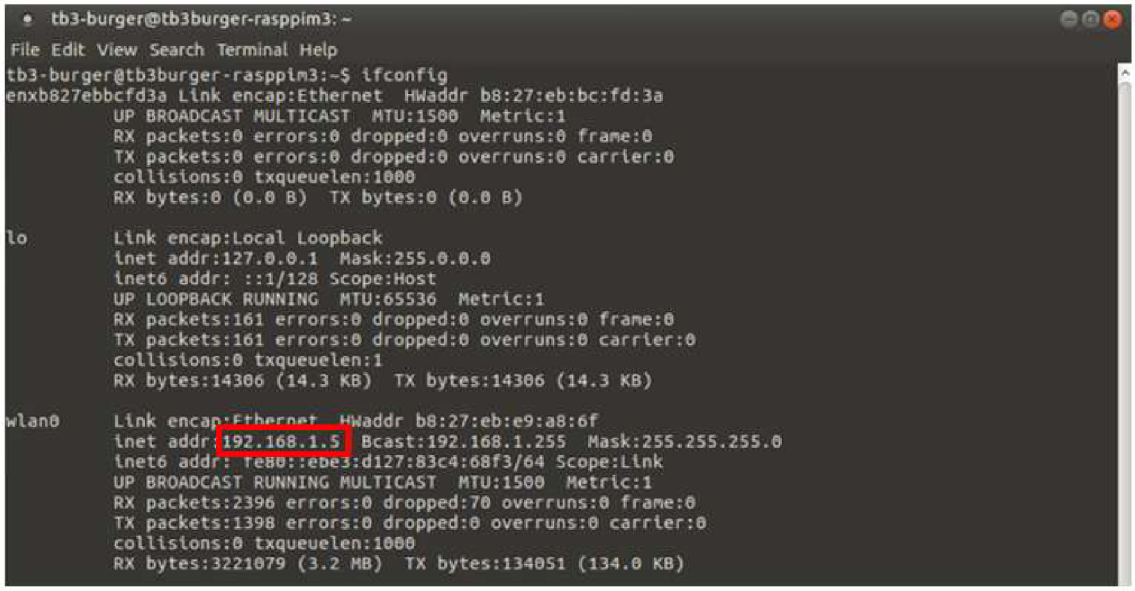
```
$ rosrun turtlebot3_bringup create_udev_rules
```

Nastavení ROS sítě pro TurtleBot3 je velmi podobné jako v případě vzdáleného PC (podkapitola 4.3), až na pár výjimek (viz obr. 47)

Obr. 47: Rozdíl mezi nastavením sítě v SBC pro TurtleBot3 a vzdáleného PC [13]

Změny budou opět prováděny v souboru `.bashrc`, kde změníme hodnoty parametrů ROS_MASTER_URI a ROS_HOSTNAME z *localhost* na IP adresu vzdáleného PC a IP adresu, která byla přidělena robotu.

IP adresu vzdáleného PC známe již z podkapitoly 4.3, stejným způsobem jako v této podkapitole získáme IP adresu pro robot (viz obr. 48), a provedeme úpravu souboru `.bashrc`



Obr. 48: Výpis z terminálu robotu po zadání příkazu `ifconfig`

Po úpravách budou hodnoty parametrů v souboru `.bashrc` následující.

~/.bashrc

```
export ROS_HOSTNAME=192.168.1.5
export ROS_MASTER_URI=http://192.168.1.7:11311
```

Doporučením na závěr je nastavení statických IP adres pro obě zařízení, po výpadku elektrického proudu nebo při údržbě na straně poskytovatele internetu, může dojít ke změně koncových hodnot IP adresy, např. z `192.168.1.5` na `192.168.1.4` atp.

4.5 Nastavení OpenCR

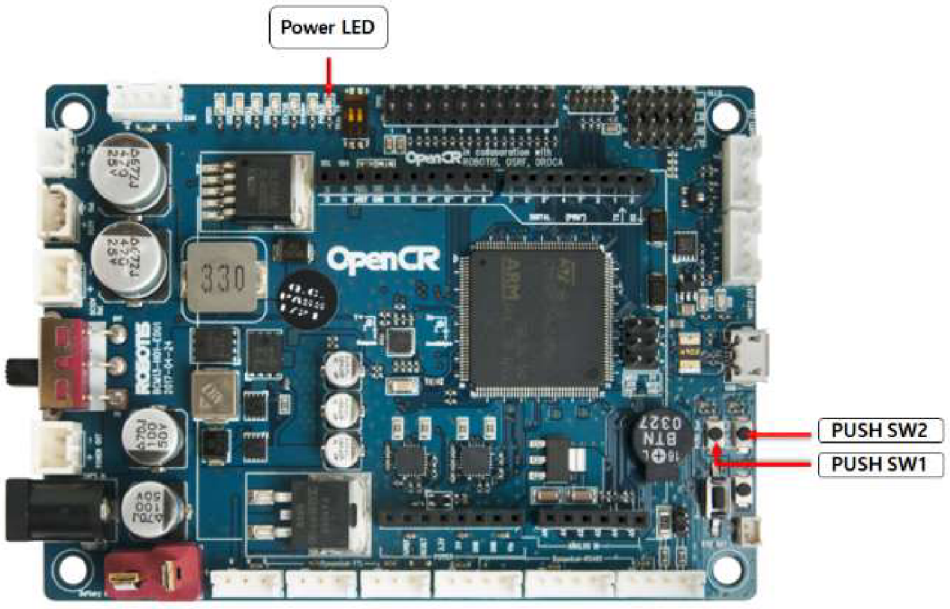
Instrukce, resp. postup konfigurace, a tedy zprovoznění vychází z oficiálních stránek pro TurtleBot3, kdy tyto instrukce byly testovány na naší kombinaci použitého hardwaru a softwaru [13].

Příkazy, resp. kroky instalace a nastavení jsou opět prováděny v systému robotu.

```
$ export OPENCR_PORT=/dev/ttyACM0
$ export OPENCR_MODEL=burger
$ rm -rf ./opencr_update.tar.bz2
$ wget https://github.com/ROBOTIS-GIT/OpenCR-
  Binaries/raw/master/turtlebot3/ROS1/latest/opencr_update.tar
  .bz2 && tar -xvf opencr_update.tar.bz2 && cd ./opencr_update
  && ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr && cd ..
```

Po úspěšném nahrání firmwaru OpenCR pro TurtleBot3 Burger bude z terminálu vytištěn textový řetězec `jump_to_fw`.

Ověření správného sestavení robotu ověříme hardwarovými tlačítky PUSH SW1 a PUSH SW2 na desce OpenCR (viz obr. 49):



Obr. 49: Znázornění tlačítek a LED diody na desce OpenCR [13]

1) Po montáži, zapojení a zapnutí robotu se rozsvítí kontrolní dioda (Power LED).
2) Položte robot na podlahu, aby nespadl při pohybu ze stolu.

3) Stiskněte a držte pár sekund tlačítko PUSH SW1 – robot se začne pohybovat vpřed (vzdálenost 30 cm).
4) Stiskněte a držte pár sekund tlačítko PUSH SW2 – robot se otočí na místě o 180 ° (po směru hodinových ručiček).

4.6 Softwarová integrace Raspberry Pi kamery pro TurtleBot3 Burger

Ve výchozí konfiguraci model Burger ze série TurtleBot3 neobsahuje integrovanou Raspberry Pi kameru. Fyzické propojení s dalšími komponenty bylo provedeno v podkapitole 4.2. Zbývá nahrání zdrojového kódu, který umožňuje přistupovat k obrazu z kamery prostřednictvím témat, tzn. skrze ROS.

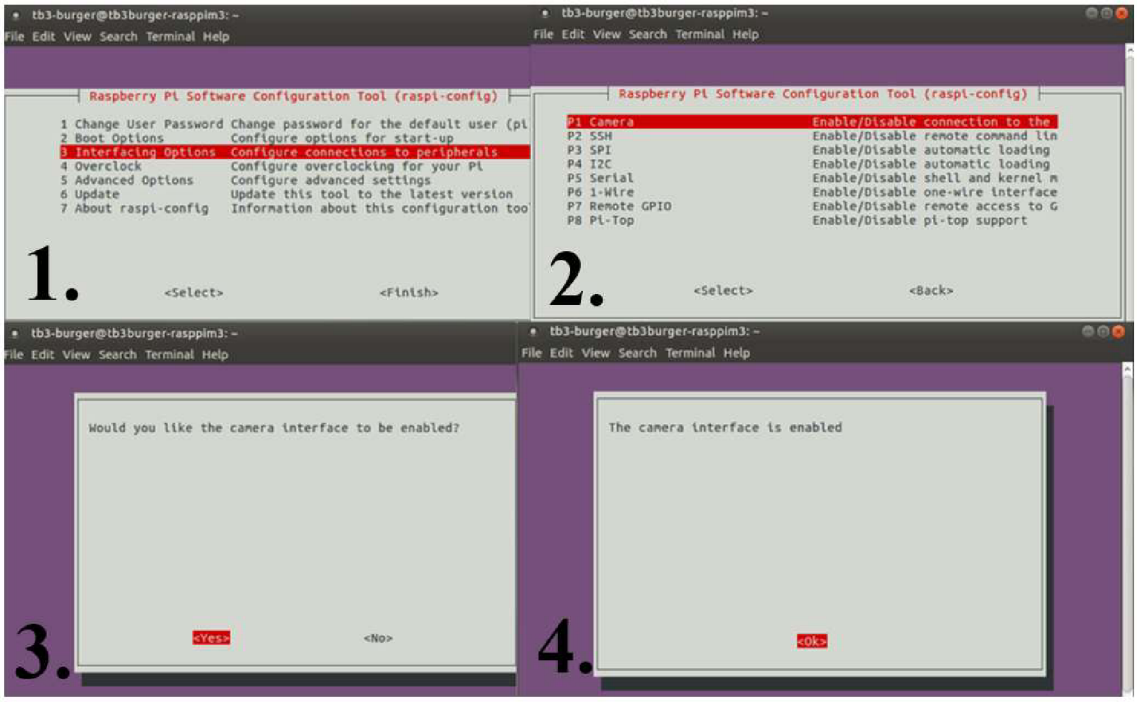
Instrukce, resp. postup konfigurace, a tedy zprovoznění vychází z oficiálních stránek pro TurtleBot3 [13]. Z důvodu neschopnosti načtení obrazu z kamery prostřednictvím nástroje RViz, a výskytu upozornění souvisejících s nemožností načtení konfiguračních souborů kamery, se bude celý postup z určité části oproti originálnímu souboru pokynů lišit.

Povolení rozhraní kamery

Přes příkazový řádek spustíme softwarový konfigurační nástroj pro Raspberry Pi (*Raspberry Pi Software Configuration Tool*).

```
$ sudo raspi-config
```

Postup pro zpřístupnění hardwaru kamery je krok za krokem znázorněn na obr. 50. Pbude nutné restartovat Raspberry Pi.



Obr. 50: Postup pro povolení rozhraní Raspberry Pi kamery V2

Instalace příslušných balíčků a závislostí ROS pro Raspberry Pi kameru
Sérií příkazů nainstalujeme balíčky a závislosti ROS pro naši kameru.

```
$ cd ~/catkin_ws/src
$ git clone
  https://github.com/UbiquityRobotics/raspicam_node.git
$ sudo touch /etc/ros/rosdep/sources.list.d/30-ubiquity.list
$ sudo nano /etc/ros/rosdep/sources.list.d/30-ubiquity.list
```

Po otevření souboru `30-ubiquity.list`, vložíme následující řádek, kterým zajistíme instalaci závislostí, které nejsou rozeznány ROS.

~/30-ubiquity.list

```
yaml
https://raw.githubusercontent.com/UbiquityRobotics/rosdep/mast
er/raspberry-pi.yaml
```

Provedeme aktualizaci seznamu v systému a instalaci závislých balíčků pomocí nástroje `rosdep`.

```
$ rosdep update
$ cd ~/catkin_ws
$ rosdep install --from-paths src --ignore-src --
  rosdistro=kinetic -y
$ sudo apt-get install ros-kinetic-compressed-image-transport
  ros-kinetic-camera-info-manager
$ catkin_make
```

Nastavení Raspberry Pi kamery, resp. její konfigurace
Pro správné načtení konfiguračních souborů zkopírujeme složku *camera_info*, která se nachází v */home/tb3-burger/catkin_ws/src/raspicam_node*, a přesuneme ji do */home/tb3-burger/.ros*.

Na vzdáleném PC spustíme příkaz `roscore` a na SBC robotu `roslaunch raspicam_node camerav2_410x308_30fps.launch`. Tím uzel `raspicam_node` začne publikovat data. Pokud byly splněny všechny dílčí kroky instalace a nastavování robotu a všech potřebných součástí, tak se po spuštění následujícího příkazu na vzdáleném PC pomocí nástroje `rqt` začnou přenášet obrazová data z kamery na obrazovku PC (viz obr. 51).

[Vzdálené PC]

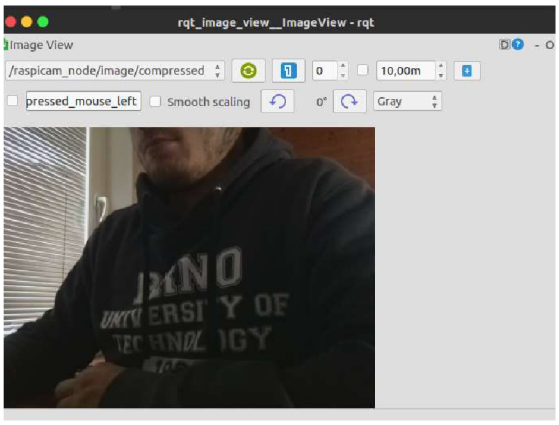
```
$ roscore
```

[TurtleBot3]

```
$ roslaunch raspicam_node camerav2_410x308_30fps
```

[Vzdálené PC]

```
$ rqt_image_view
```



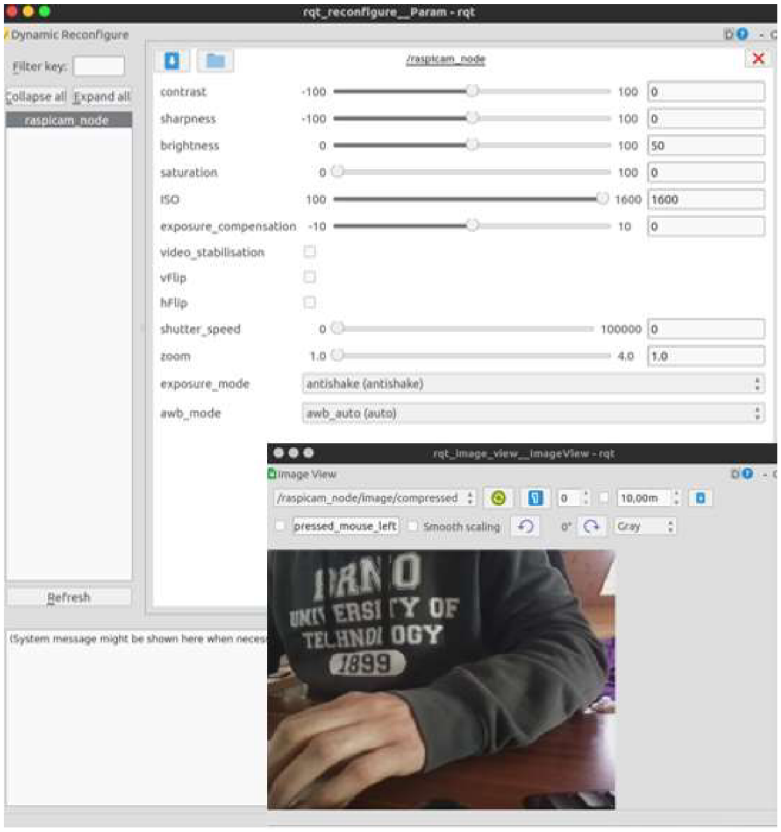
Obr. 51: Výstup po zadání příkazu `rqt_image_view`, červeným obdélníkem znázorněno odebírané téma z `raspicam_node`

Simulace a používání robotu při implementaci řídícího programu bude probíhat ve vnitřním prostředí, tzn. je nutné změnit některé parametry kamery, k čemuž slouží nástroj umožňující dynamickou konfiguraci kamery.

Cílem bude získat, co nejlepší možný obraz i za horšího osvětlení (nízké osvětlení), což znamená zvýšit hodnotu ISO (viz obr. 52).

[Vzdálené PC]

```
$ rqt_image_view
$ rosrun rqt_reconfigure rqt_reconfigure
```



Obr. 52: Zvýšení parametru ISO u kamery (závisí na míře osvětlení a množství šumu)

5 NÁVRH A IMPLEMENTACE ŘÍDICÍHO PROGRAMU

Programovatelná mobilní robotická platforma TurtleBot3 Burger, nabízející možnost rozšiřitelnosti, obsahuje v základu mnoho funkcionalit, ke kterým je volně dostupný zdrojový kód.

Zpětnou analýzou rozsáhlé knihovny, obsahující řídicí programy pro dané operace, jsme schopni nejen teoreticky, ale i prakticky porozumět chování senzoru, výstupu jeho dat a možnostem zpracování prostřednictvím ROS, tzn. přístup k datům od senzorů přes odebírání daných témat, obsahující zprávy různých datových typů, které lze zpracovat dle požadavků našeho zájmu, a vytvořit řídící podmínky pro ovládání akčních členů.

5.1 Návrh řídicího programu – řízení robotu založené na zpracování obrazu

Možnosti pro návrh programu lze rozdělit do tří kategorií. První je kontrola akčních členů zadanými instrukcemi, např. řízení pohybu robotu podle zadaného schématu, kterým může být funkce, kde výstupem bude napětí, poloha, rychlost (DYNAMIXEL servomotory umožňují různé režimy řízení).

Druhá možnost souvisí s laserovým skenerem vzdálenosti, což je 2D skener schopný snímat v rozsahu 360 stupňů. Následně shromážděná data mohou být použita pro SLAM (Simultánní lokalizace a mapování) a navigaci, neboť právě tyto funkcionality patří k základní technologii, kterou mobilní robotická platforma TurtleBot3 disponuje.

Třetí možnost je spojena s použitím počítačového vidění. Tuto variantu jsme umožnili dodatečným rozšířením našeho robotu o kameru (podkapitola 4.2 a 4.6).

Po diskuzi s vedoucím bakalářské práce bylo rozhodnuto, že návrh řídícího programu bude využívat počítačového vidění pro dosažení autonomního pohybu na základě získaných dat z kamery. Následně bude znázorněna ukázka integrovaných programů pro manuální ovládání robotu za současného využití laserového skeneru vzdálenosti, patřící k hlavní doméně série mobilní robotů TurtleBot3

Celkem budou dva výstupy (kap. 6). Prvním výstup bude tedy implementace zvoleného řešení a ověření jeho funkčnosti na laboratorní úloze. Druhý výstup bude obsahovat vytvořenou mapu místnosti.

Vzhledem k využití vestavěných řídících programů pro druhý výstup, bude kapitola 5 věnována hlavnímu výstupu bakalářské práce, kterým je aplikace počítačového vidění.

5.1.1 Problematika zvoleného řešení

která se zabývá tím, jak umožnit zařízením a strojům (obecně počítačům) nejen „vidět“, ale i zpracovat digitální vizuální data, tj. extrahovat informace z těchto dat za účelem dosažení požadovaných cílů.

Bavíme se o rozsáhlé a komplexní oblasti, kde můžeme vidět snahy o napodobení lidského vidění, za použití datového a statistického přístupu rozšířeného o strojové učení. Obecně je oblast počítačového vidění kombinací programování, modelování a matematiky.

Problém je v tom, že stále zcela nechápeme, jak náš mozek rozpoznává, organizuje a zpracovává vizuální data. Jsme pouze schopni extrahovat z obrázků prvky a ty podrobit algoritmům strojového učení za účelem získání potřebného množství dat, pomocí kterých je poté počítač schopen jejich rozeznání v reálném čase.

Množství variací jednoho vjemu nebo objektu je jedním z dalších problémů. Mějme například židli, která za různých světelných podmínek, úhlů pozorování atp. je vnímána naším mozkem jako „židle“, nicméně, pro robot se jedná o odlišný objekt. Lidská mysl rozpozná, že série různých obrazů zobrazuje ten samý objekt bez ohledu na to, jak je prezentován. Jak to tedy „vysvětlit“ strojům?

Jedním ze způsobů, jak toho dosáhnout, by bylo uložit všechny různé variace objektu, včetně velikostí, úhlů, perspektiv atd. Tento proces je však těžkopádný a časově náročný. Ve skutečnosti také není možné shromažďovat data, která mohou zahrnovat každou jednotlivou variantu.

Ke zpracování obrazu do podoby, který jsme schopni využít, tzn. izolování dat našeho zájmů, čímž zjednodušíme celkový proces učení, potřebujeme základní funkční bloky, jejichž kombinací jsme schopni formulovat složité algoritmy. Právě tyto funkce jsou obsaženy ve vysoce optimalizované knihovně OpenCV. [24, 25, 26]

5.1.2 Knihovna pro zpracování obrazu OpenCV

OpenCV je open-source knihovna počítačového vidění, obsahující solidní infrastrukturu v podobě stovek algoritmů počítačového vidění. Byla navržena pro výpočetní efektivitu se silným zaměřením na aplikace v reálném čase.

Jedním z cílů OpenCV je poskytnout snadno použitelnou infrastrukturu počítačového vidění, která pomůže lidem rychle vytvářet poměrně sofistikované aplikace.

Knihovna OpenCV obsahuje více než 500 funkcí, které pokrývají mnoho oblastí vize, včetně kontroly výrobního produktu, lékařského zobrazování, bezpečnosti, uživatelského rozhraní, kalibrace kamery, stereo vidění a robotiky. Protože počítačové vidění a strojové učení často jde ruku v ruce, OpenCV obsahuje také plně univerzální knihovnu strojového učení.

Instalace není nutná, pro naše potřeby budou stačit předdefinované binární soubory, které jsou součástí repositáře naší linuxové distribuce Ubuntu. [27]

5.1.3 Rozhraní mezi ROS a OpenCV prostřednictvím knihovny CvBridge

ROS předává obrázky ve vlastním formátu, např. `sensor_msgs/Image`, `sensor_msgs/CompressedImage` atp. Pro práci s obrazovými daty získanými prostřednictvím témat daného uzlu kamery, je nutné provést převod mezi obrazovými zprávami ROS na formát obrazu, který jsme schopni zpracovat prostřednictvím knihovny OpenCV (a naopak, zpracovaný obraz zpětně publikovat prostřednictvím jiného tématu).

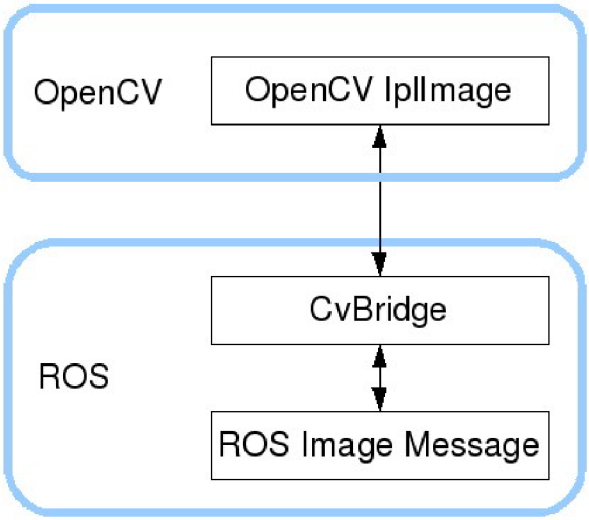
Právě k tomuto účelu slouží ROS knihovna CvBridge, která je součástí balíčku `cv_bridge`.

Logika daného převodu obrazových dat je nastíněna pomocí následujícího kódu a ilustrativního schématu (viz obr 53). [16]

```
from cv_bridge import CvBridge
bridge = CvBridge()
cv_image = bridge.compressed_imgmsg_to_cv2(
   cmprs_img_msg,
   desired_encoding='passthrough')
```

Import modulu `cv_bridge.CvBridge`, nadefinování objektu CvBridge skrze proměnnou `bridge` a získaní obrazu ve zvoleném obrazovém kódování pomocí `compressed_imgmsg_to_cv2`, kde parametr `cmprs_img_msg` je zprávou ROS, a `desired_encoding='passthrough'` specifikuje požadované kódování obrazu.

Všechny možnosti kódování a další ukázky možného použití naleznete v online dokumentaci[2].



Obr. 53: Schéma rozhraní mezi ROS a OpenCV [16]

5.2 Detekce čáry jízdního pruhu pro autonomní řízení robotu TurtleBot3 Burger

Vybraná laboratorní úloha, která spojuje počítačové vidění a autonomní řízení robotu, vyžaduje splnění tří úkolů – percepci, rozhodnutí, akci.

Percepci neboli vnímání, lze chápat v příjmu vizuálních dat prostřednictvím Raspberry Pi kamery V2, a následném provedení jejich zpracování za účelem dalšího použití. Rozhodování je součástí nejen percepce, kdy podle kritérií stanovujeme

[2] http://docs.ros.org/.

relevantnost získaných dat, ale také na jejich základě stanovujeme míru ovlivnění pohybu robotu. Provedení akce v našem případě bude obsahovat pouze ovládání akčních členů pro dosažení sledování čáry jízdního pruhu, tvořícího výslednou trasu.

Ke splnění požadovaného cíle je nutné provést extrahování prvků obrazu, které jsou pro nás relevantní, tzn. vnímat pouze charakteristiky pruhu, které získáme využitím knihovny OpenCV. Získaná data je nutné podrobit analýze, pomocí vytvořeného algoritmu, který jejich rozborem provede vyhodnocení a interpretaci do podoby, díky které jsme schopni vytvořit řídicí podmínky.

Snímky pořízené prostřednictvím kamery, která je upevněna na robotu, jsou v rozlišení 720p, a budou na nich demonstrovány jednotlivé kroky algoritmu.
Pro reálnou aplikaci, v reálném čase, je s ohledem na limitující faktor, kterým je rychlost bezdrátového připojení, nutné použití nižšího rozlišení, které je v našem případě 308p (souvisí s dostupnými spouštěcími soubory z `raspicam_node`, viz podkapitola 4.6 v části *Nastavení Raspberry Pi kamery, resp. její konfigurace*). V případě omezujícího faktoru, který by souvisel s výkonem vzdáleného PC, které je používano pro připojení s robotem, a tedy sloužící k hlavním výpočtovým operacím, je možné pomocí knihovny OpenCV provést dodatečné změnšení.

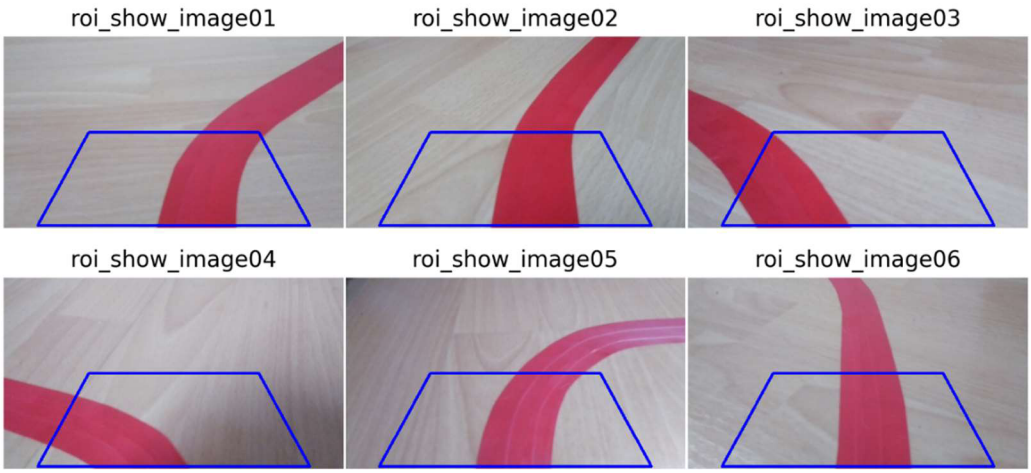
Obr. 54: Testovací snímky z pohledu robota

5.2.1 Transformace obrazu a vymezení oblasti zájmu

Geometrická transformace obrazu pomocí metody inverzního perspektivního mapování, která je součástí knihovny OpenCV, odstraňuje efekt perspektivy, který způsobuje zkreslení jízdního pruhu, např. pruhy se v dálce sbíhají do jednoho, nemají konstantní tloušťku atp.

Jednoduše řečeno provedeme převod obrazu z pohledu robotu (pozorovací bod umístěn na konstrukci robotu v určité výšce, a pod konkrétním úhlem) na pohled z ptačí perspektivy, čímž dosáhneme jednotné tloušťky pruhu, a prvky trajektorie budou odpovídat skutečnosti, tzn. rovnoběžnost a zakřivení jsou snadněji popsatelné, čímž dojde ke zjednodušení procesu segmentace pruhu.

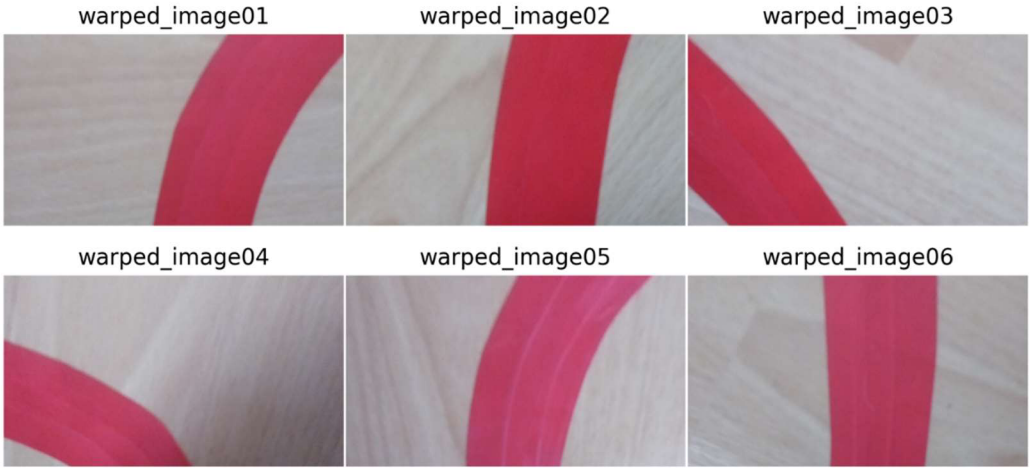
K definování perspektivní transformace je nutné vybrat oblast, kterou chceme převést (viz obr 55). Ta odpovídá oblasti našeho zájmu. Volíme ji s ohledem na nutnou část, potřebnou pro zpracování, čímž eliminujeme vlivy, které by způsobily nepřesnost v naší robotické aplikaci, a zároveň zmenšíme množství dat posílaných do dalších algoritmů zpracování obrazu.



Obr. 55: Vyznačená oblast zájmu, která bude zároveň podrobena transformaci

```
import cv2
M = cv2.getPerspectiveTransform(src, dst)
warped_image = cv2.warpPerspective(img, M, (width, height))
```

Import modulu `cv2`, pro použití funkcí OpenCV. Proměnná `M` představuje matici, ve které jsou uloženy vypočtené hodnoty pro perspektivní transformaci. Parametr `src` specifikuje souřadnice vrcholů ve zdrojového obrázku (vrcholy čtyřúhelníku, viz obr 55), a `dst` souřadnice cílového obrázku. Proměnná `warped_img` je transformovaný obrázek (viz obr. 56), kde parametry `width` a `height` jsou rozměry cílového obrázku (šířka a výška).



Obr. 56: Testovací snímky s oblastí zájmu z pohledu ptačí perspektivy

5.2.2 Segmentace obrazu metodou prahování

Předzpracovaný obraz vyřešil problém nesourodosti zobrazení podle úhlu, pod kterým sledujeme čáru jízdního pruhu. Nyní je nutné provést izolaci dat pruhu. Toho docílíme segmentováním obrazu, resp. rozdělením obrazu na oblasti se společnými vlastnosti.

K tomu využijeme metodu prahování, která využívá hodnocení úrovně intenzity jednotlivých pixelů. Určením prahových hodnot získáme po výsledném zpracování binární reprezentaci obrazu (pixel může nabývat pouze dvou hodnot, a to 0 a 1).

Demonstrativní snímky, zobrazující výstupy po aplikování metody prahování, jsou v netransformovaném stavu, za účelem předejití případným nespojitostem v získaných datech, které by se poté projevily v oblasti zájmu, při pohybu.

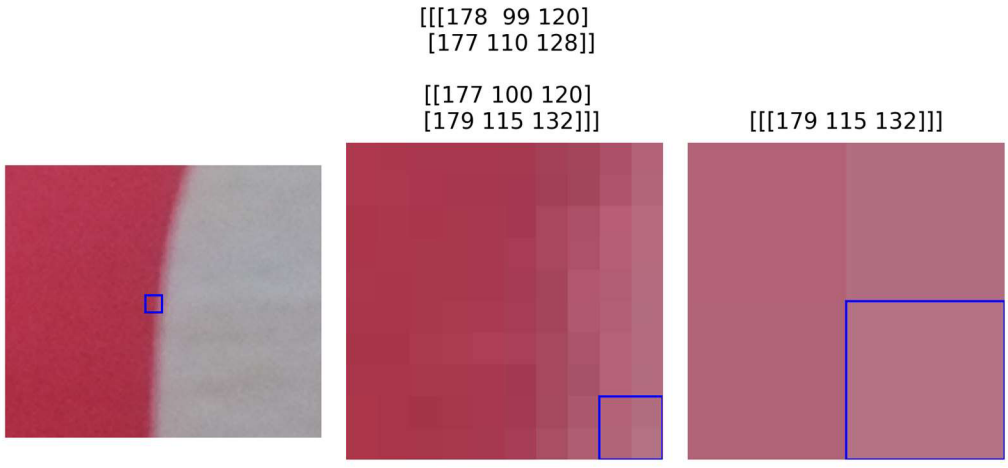
Detekce čáry jízdního pruhu, v rámci zvolené metody, nabízí dva možné přístupy: selekci barev a detekci hran. Vzhledem k charakteristikám pruhu, což jsou šířka (relativně úzký) a počet (pouze jeden), bude použita selekce barev. Pro případ dvou pruhů, mezi kterými by robot vykonával pohyb, je možné zvolení kombinace obou dvou přístupů. V analogii pohybu po silnici lze tyto dvě situace rozdělit na detekci středního, nebo vnějšího pruhu.

Naleznutí čáry jízdního pruhu za použití selekce barev

Obecně jsou obrázky uchovávány v RGB nebo BGR barevném formátu, kde každý pixel je charakterizován mohutností tří komponent: červená (R), zelená (G) a modrá (B), kde pro 8bitové obrázky nabývá každá komponenta v rozmezí od 0 do 255, tzn. že pro většinu obrázků platí, že jejich barevná hloubka je 8 bitů na bod, kdy počet barev se poté vypočítá jako 2 na počet bitů, tedy 2 na 8, cože je 256 (indexujeme od 0, proto 0 až 255).

Pro práci s obrazovými daty je nutné pochopit, že obraz z pohledu počítače je reprezentován číselně, tzn. matice, skládající se z řádků a sloupců, kde její jednotlivé prvky jsou opět maticemi, uchovávající informaci o daném pixelu (viz obr 57).

S touto znalostí je možné pomocí převodní vztahů převést obrázky mezi různými barevnými prostory. Bez nutnosti znalosti teorie, stojící za těmito vztahy, je převod umožněn funkcemi knihovny OpenCV.

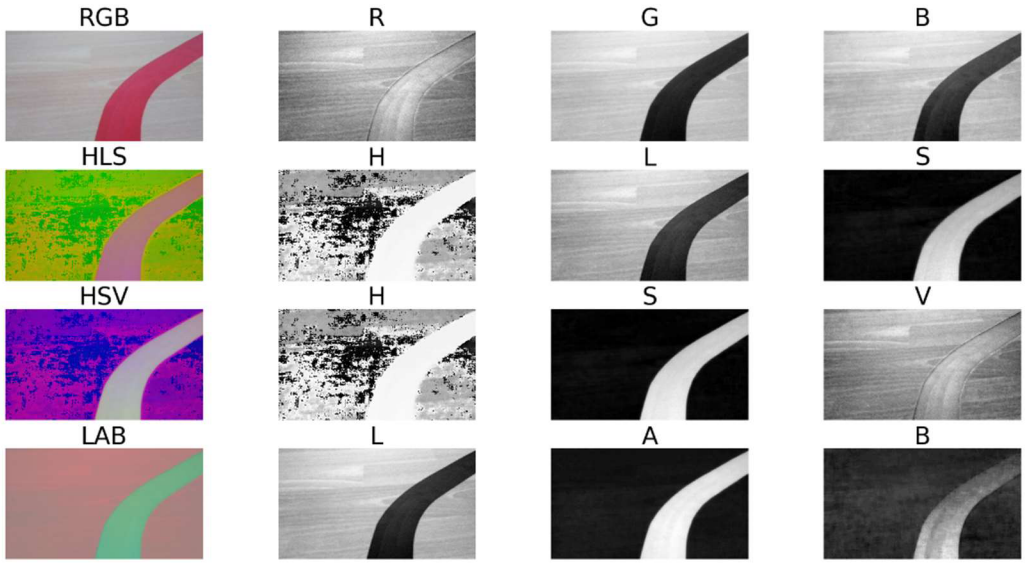


Obr. 57: Číselná reprezentace 8bitového digitálního obrazu

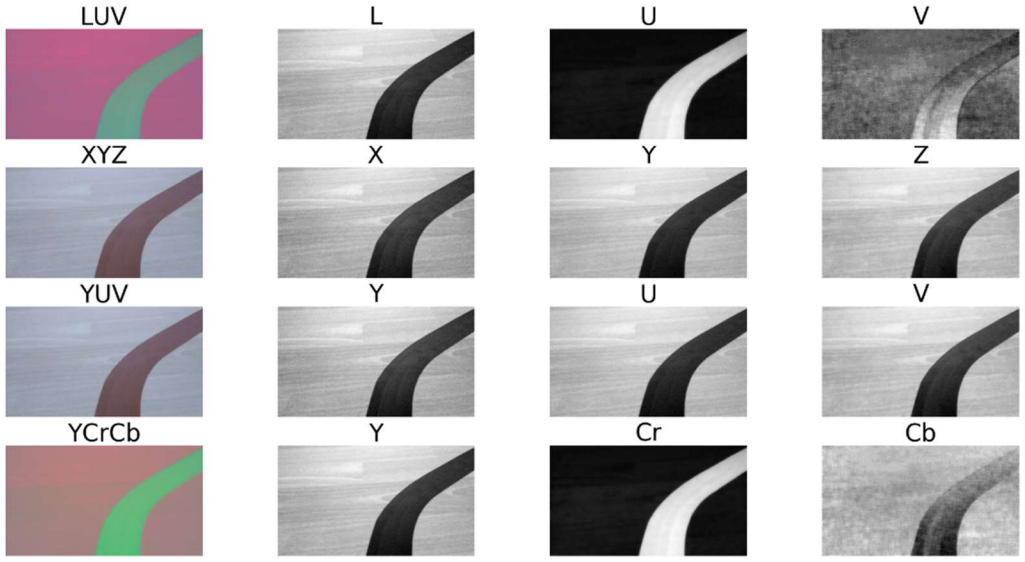
Selekce barev u testovacích snímků a jejich následné převedení do binární podoby se zpravidla provádí vybráním jedné komponenty barevných prostorů, kdy po získání obrazu ve stupních šedi se volí prahová hodnota v rozmezí od 0 do 255.

Volbu zvolené komponenty, resp. její vhodnost, lze vyhodnotit vizuálně, kdy hlavním kritériem je oddělení čáry jízdního pruhu od podkladu. Ke stanovení prahových hodnot budeme přistupovat experimentálně, tedy bude stanovena s ohledem na měnící se světelné podmínky, kterými je charakterizováno prostředí, ve kterém probíhá praktická zkouška daného řešení.

Po převodu testovacích snímků do jednotlivých barevných prostorů a izolováním jejich komponent je zřejmě viditelné, že některé se hodí více pro daný převod do binární podoby než jiné (viz obr. 58 a 59).



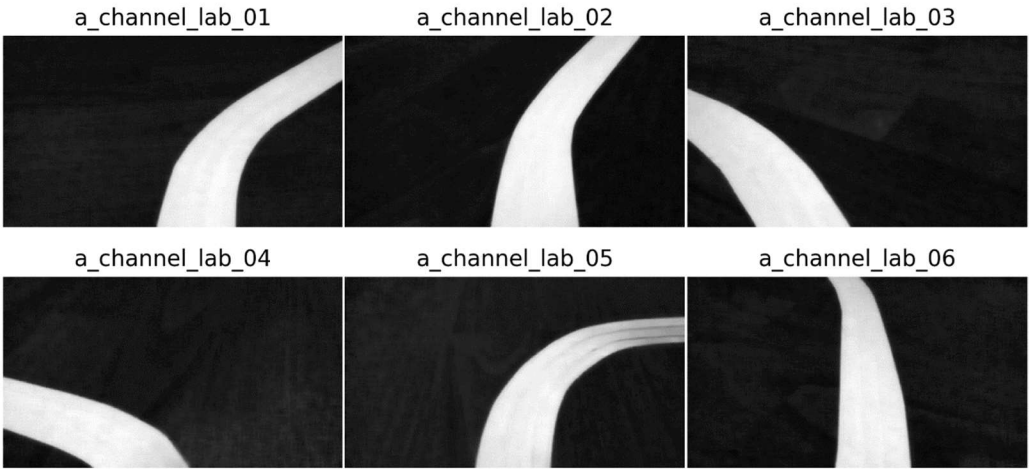
Obr. 58: Přehled možných variant pro selekci barev (část 1)



Obr. 59: Přehled možných variant pro selekci barev (část 2)

Výběr vhodných variant je následující (barevný prostor-zvolená komponenta): HLS-S, HSV-S, LAB-A, LUV-U, YCrCb-Cr. Všechny tyto varianty jsou rovnocenné jak po vizuální stránce, tak jsou i shodné po následném převedení do binární podoby, kdy produkují po kvalitativní stránce rovnocenné výstupy pro další zpracování.

Demonstrativní ukázka (viz obr. 60 a 61) bude nastíněna na variantě LAB-A, včetně kódu pro provedení binarizace.



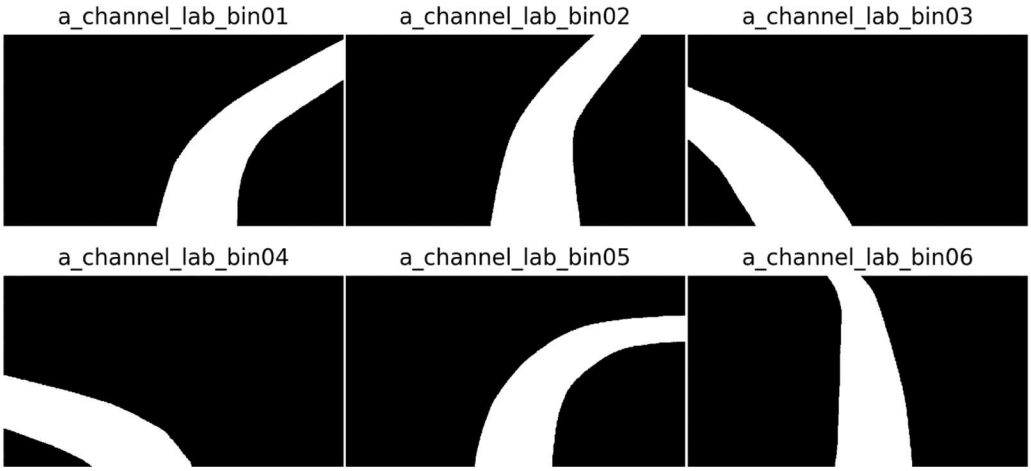
Obr. 60: Ukázka komponenty A z LAB pro všechny testovací snímky

```
import cv2
import numpy as np
lab = cv2.cvtColor(image, cv2.COLOR_RGB2LAB)
a_channel_lab = lab[:, :, 1]
a_thresh = (140, 255)
a_binary = np.zeros_like(a_channel_lab)
a_binary[(a_channel_lab >= a_thresh[0]) & (a_channel_lab <=
a_thresh[1])] = 1
a_binary = np.dstack((a_binary, a_binary, a_binary)) * 255
```

Import modulu `cv2` pro práci s OpenCV, a `numpy` pro práci s maticemi. Funkce `cv2.cvtColor` slouží k převodu mezi barevnými prostory, kde vstupními parametry jsou obraz, tj. `image` a poté varianta převodu, tedy z RGB do LAB pomocí `cv2.COLOR_RGB2LAB`. Proměnná `a_channel_lab` uchovává komponentu A, která byla izolována z proměnné `lab`, kdy tvar `lab[:, :, 1]` specifikuje, že chceme všechny řádky a sloupce z dané matice, tvořící digitální obraz, a pro každý element , resp. pixel, pouze hodnotu, kterou nabývá v komponentě A.

Vytvoříme pomocný obraz, resp. pole (matici), kdy funkce `np.zeros_like(a_channel_lab)` vrátí pole nul o stejné velikosti, jako je vstupní pole. Z vizuálního hlediska dostaneme černý obrázek o rozměrech vstupního. Prahové hodnoty specifikuje proměnná `a_thresh` (byly stanoveny experimentálně). Kde prostřednictvím logických operátorů zaznačíme oblasti, spadající do prahových hodnot,

do pomocného obrazu, kde je k jejich vykreslení nutné použít funkci `np.dstack()` vynásobenou hodnotou bíle barvy pro jednokomponentový obraz, tedy hodnotou 255.



Obr. 61: Binární podoba testovacích snímků po aplikování metody prahování na komponentu A z LAB

5.3 Proložení získaných dat čáry pruhu polynomem druhého stupně za použití metody posuvných oken

Metoda posuvných oken patří mezi robustní metody k získávání souřadnic bílých pixelů v binárním obrazu, a jejich následnému proložení polynomem. Tato metoda je spojena s méně robustní metodou, která využívá předešlého proložení, získaného metodou posuvných oken. Celá myšlenka této kombinace spočívá ve snížení výpočetní náročnosti celého algoritmu, kdy mezi hlavní požadavky patří využití druhé metody tak často, jak je to jen možné.

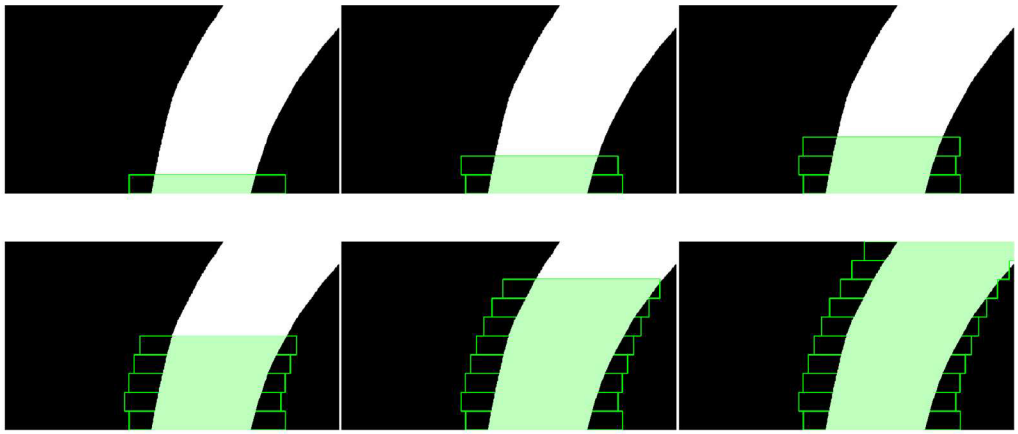
Prvním krokem je uložení souřadnic nenulových pixelů v binárním transformovaném obrazu, s kterým budeme pracovat. Právě pozice těchto souřadnic se porovnávají s hranicemi, definovanými základním prvkem této metody, kterým je okno, definovaných rozměrů, tzn. spadá-li jejich pozice do množiny okna, tak se tyto souřadnice uloží a jsou použity k následnému proložení polynomem.

Přestože se metodami segmentace obrazu izolovala potřebná data, tak by vlivem nepředvídatelného rušení mohlo dojít k chybovému proložení, které by neodpovídalo charakteristice průběhu čáry pruhu. Proto je nutné stanovit startovací pozici, ve které bude začínat algoritmus metody posuvných oken, resp. pozici startovacího okna, od které se bude vyvíjet poloha dalších oken. Určí se sumou bílých pixelů vzhledem k ose x, kde poloha nejvyšší sumy bude startovací pozicí. Dojde k zajištění, že souřadnice bílých pixelů, které budou použity k proložení, odpovídají oblasti pruhu, tzn. oblasti nejvyšší koncentrace bílých pixelů. Princip hledání polohy startovací pozice je vyjádřen následujícím kódem (import modulů odpovídá předchozím ukázkám, a tedy nebudou znovu zaznamenány).

```
histogram =
np.sum(binary_warped_image[binary_warped_image.shape[0]//2:,
:], axis=0)
x_base = np.mean(np.argmax(histogram, axis=0), dtype=np.int32)
x_current = x_base
```

Kdy každé další okno změní svoji pozici s ohledem na střední hodnotu bílých pixelů okna předcházejícího, včetně podmínky minimálního počtu pixelů k přepozicování (viz obr. 62).

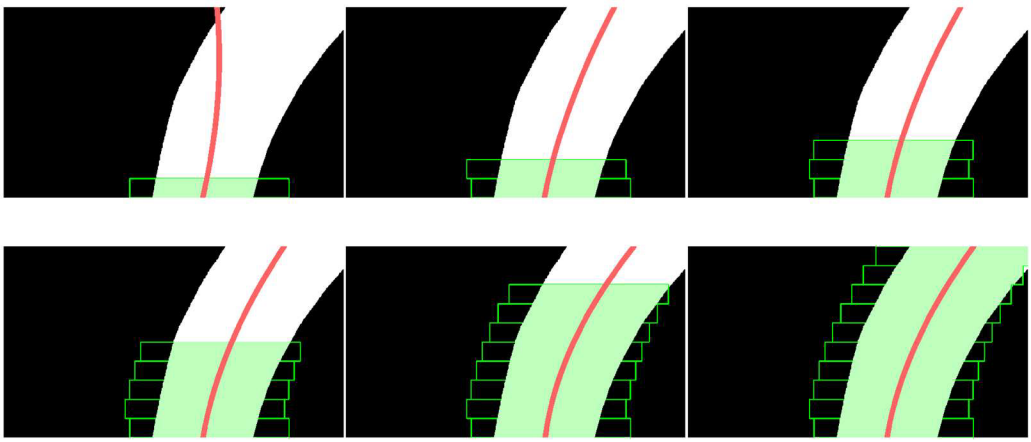
```
if len(found_coor) > minpix:
            x_current = np.int(np.mean(nonzerox[found_coor]))
```



Obr. 62: Startovací pozice a princip přepozicování dalších oken

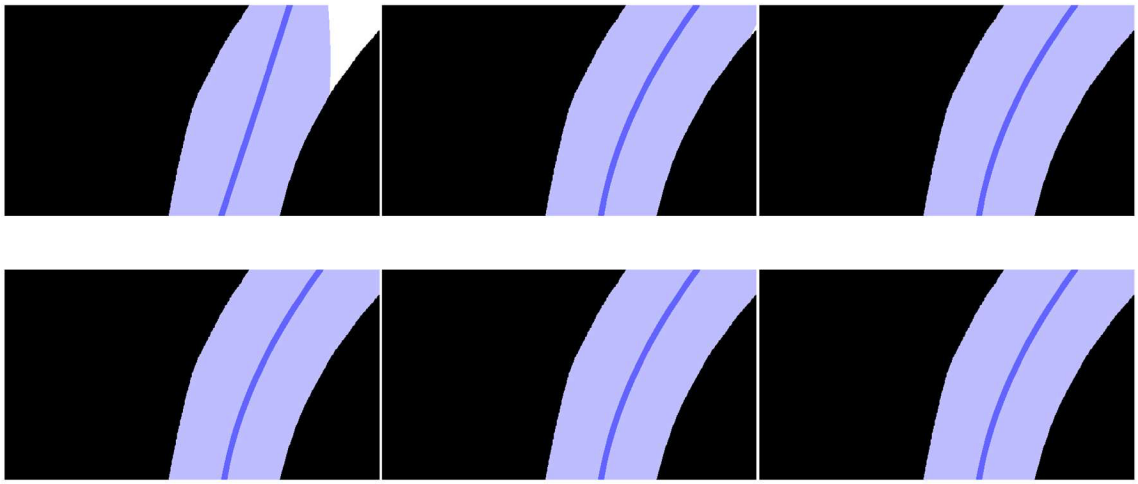
Pro demonstrativní účely bylo zvoleno celkem deset posuvných oken, počet je nicméně libovolný a závisí pouze na potřebě jemnosti výsledné sítě bodů , kdy příliš jemná síť zvyšuje výpočetní nároky, obzvláště pro naši aplikaci, kdy jsou segmenty pruhu snadno rozpoznatelné, a neobsahují složitou geometrii.
Pomocí následujícího kódu bude provedeno proložení těchto bodů polynomem druhého stupně (viz obr. 63).

```
lane_coor = np.concatenate(lane_coor)
x = nonzerox[lane_coor]
y = nonzeroy[lane_coor]
lane_fit = np.polyfit(y, x, 2)
```



Obr. 63: Proložení získaných dat polynomem druhého stupně

Výstupní data metody posuvných oken budou použita jako vstupní pro druhou metodu, která získaná data využívá jako referenční hodnoty, kolem kterých vytvoří hranici, která pracuje na podobném principu jako metoda posuvných oken, kdy jakýkoliv bílý pixel nacházející se uvnitř bude zapsán do sítě bodů, které budou opět proloženy. Výsledná podoba, včetně znázornění této hranice světle modrou barvou, je zobrazena na obr. 64.



Obr. 64: Použití druhé metody po získání dat z metody posuvných oken

5.4 Autonomní řízení robotu na základě získaných dat z obrazu

Funkce `np.polyfit()` vrací polynom druhého stupně jako matici o jednom řádku a třech sloupcích, ve kterých jsou koeficienty A, B a C, tedy `[A, B, C]`, kde za použití algebraického vztahu, dle rovnice (1) se dostane pozice x dané křivky, charakterizující čáru jízdního pruhu, a to pro jakýkoli bod po výšce oblasti zájmu.

$$x = A \cdot y^2 + B \cdot y + C \tag{1}$$

Úkolem je získání hodnoty, resp. odchylky této pozice od referenční polohy, kterou je střed zmíněné oblasti zájmu, kde na základě její velikosti je nutné stanovit míru ovlivnění pohybu robotu, tzn. rychlost rotace.

5.4.1 Regulace úhlové rychlosti proporcionální, integrační a derivační složkou

Princip zpracování žádané hodnoty je založen na vytvoření nového uzlu, který přijme zprávy datového typu `Float32` z tématu `/detect_lane/lane_error`, kde zmíněná hodnota vyjadřuje odchylku od referenční hodnoty, tzn. chybu/vychýlení v reálném čase. K zajištění tendence snižování této chyby bude využito znalosti regulace pomocí tří složek: proporcionální (P), integrační (I) a derivační (D).

Pro systém podléhající regulaci úhlové rychlosti není známá jeho charakteristika, a tedy je obecně problém stanovení jednotlivých složek regulátoru, nicméně, existuje metoda pro stanovení těchto složek za provozního zapojení – heuristická metoda seřizování parametrů známá jako Ziegler-Nicholsova metoda , využívající empirických vztahů pro stanovení parametrů regulace. Postup pro aplikování této metody je následující:

1) Vyřazení vlivu integrační a derivační složky ($K_i \rightarrow 0, K_d \rightarrow 0$).
2) Přivedení systému na hranici stability zvyšováním hodnoty proporcionální složky ($K_p \rightarrow zvyšování\ od\ nuly\ až\ do\ konečné\ hodnoty$), kdy konečná hodnota proporcionální složky způsobí na výstupu regulační smyčky stabilní a konzistentní kmitání.
3) Získané hodnoty – proporcionální složka neboli tzv. kritické zesílení K_u, a doba mezi dvěma kmity, tzv. kritická doba kmitu T_u.
4) Na základě těchto složek a tabulky (tab. 10) s odvozenými empirickými vztahy se stanovují hodnoty složek regulace.

Tab. 10: Empirické vztahy pro stanovení složek regulace s ohledem na typ regulátoru

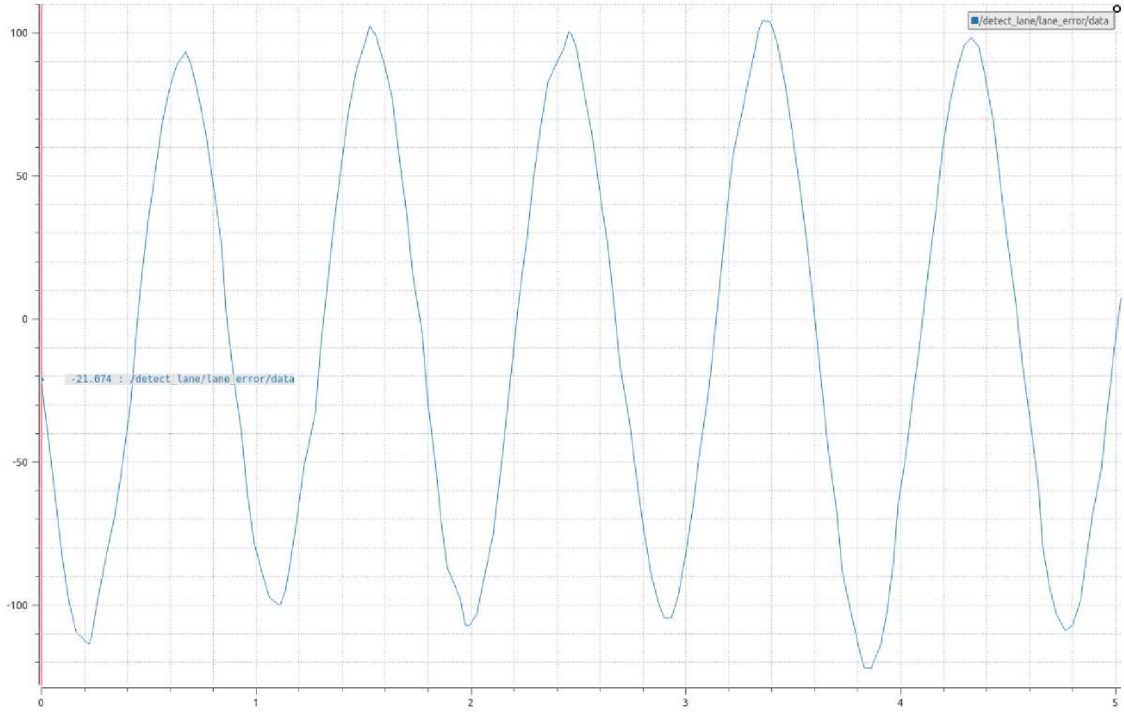
Typ regulace	K_p	K_i	K_d
P	$0{,}5 \cdot K_u$	–	–
PI	$0{,}45 \cdot K_u$	$0{,}54 \cdot K_u/T_u$	–
PD	$0{,}8 \cdot K_u$	–	$K_u \cdot T_u/10$
PID	$0{,}6 \cdot K_u$	$1{,}2 \cdot K_u/T_u$	$3 \cdot K_u \cdot T_u/40$

Při stanovování kritického zesílení se bude vycházet z maximální možné odchylky, která je rovna polovině šířky oblasti zájmu, a maximální rychlosti rotace mobilního robotu.

Z podkapitoly 2.7.1 je maximální rychlost rotace rovna 2,84 rad/s, a polovina šířky oblasti zájmu vychází z rozlišení, které bylo zmíněno v podkapitole 5.2, kde šířka rozlišení 308p v poměru 4:3 odpovídá 410px, a tedy maximální odchylka činí 205px. Výpočet výchozí hodnoty (2) a experimentálně zjištěné hodnoty (3) kritického zesílení. Pro jednoduchost jsou hodnoty vnímané jako bezrozměrné veličiny.

$$K_{u\_approx} = \frac{2{,}84}{205} = 0{,}01385 \tag{2}$$

$$K_{u\_experimental} = 1{,}1 \cdot K_{u_{approx}} = 1{,}1 \cdot 0{,}01385 = 0{,}0152 \tag{3}$$



Obr. 65: Závislost výchylky [px] (vertikální osa) na čase [t] (horizontální osa) pro hodnotu z rovnice (3)

Data byla získána prostřednictvím nástroje příkazového řádku `rosbag`, a vykreslena nástrojem `PlotJuggler`. Z grafu (obr. 65) odpovídá hodnota kritické době kmitu $T_u = 0{,}926\ s$.

Použitím empirických vztahů (tab. 10) a následném otestování parametrů na dráze (obr. 66) bylo stanoveno optimální nastavení v podobě PD regulátoru s hodnotami $K_p = 0{,}0122$ a $K_d = 0{,}00141$.

Obr. 66: Testovací dráha pro řídící program mobilního robotu

6 REALIZACE ÚLOHY A TESTOVÁNÍ

6.1 Sledování čáry s využitím počítačového vidění

Zdrojové soubory k testovací laboratorní úloze se nacházejí v příloze. Pro jejich spuštění je nutné provedení veškerých kroků v podkapitole 3.5 a kapitole 4. Následné přesunutí balíčku `bt_uai_jf` do vlastního pracovního prostoru catkin `/catkin_ws/src/`. Následné provedení jeho sestavení pomocí následujících příkazů.

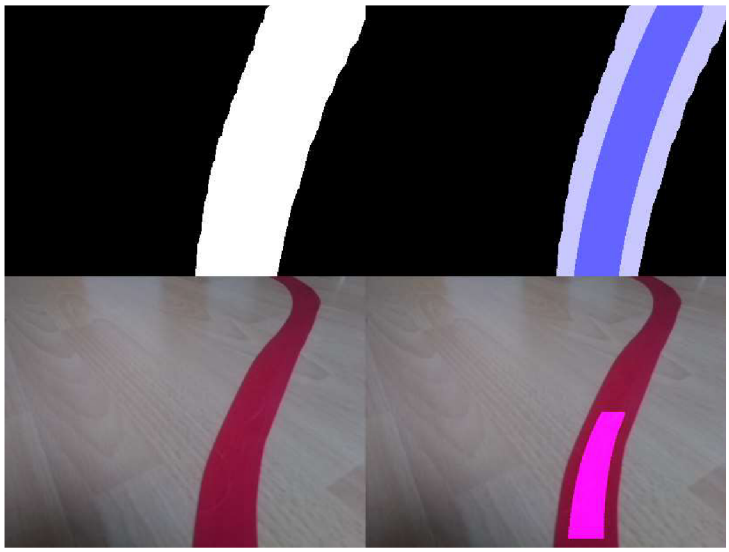
```
cd ~/catkin_ws/
catkin_make
```

Jednotlivé kroky pro spuštění se zadávají pokaždé do nového okna terminálu. Zahájení ROS Master pro umožnění komunikace mezi uzly. Spuštění základních balíčků TurtleBot3, a `raspicam_node` pro příjem témat obsahující obraz z kamery. Skript `detect_lane.py` využívá algoritmu z podkapitoly 5.2 a 5.3 a skript `control_lane.py` simuluje chování PD regulátoru s parametry z podkapitoly 5.4.

```
[Vzdálené PC]
$ roscore
```

```
[TurtleBot3]
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
$ roslaunch raspicam_node camerav2_410x308_30fps.launch
```

```
[Vzdálené PC]
$ rqt_image_view
$ rosrun bt_uai_jf detect_lane.py
$ rosrun bt_uai_jf control_lane.py
```



Obr. 67: Ukázka výstupu z programu při reálné aplikaci

6.2 Mapování místnosti s využitím laserového skeneru vzdálenosti a manuálního ovládání skrze vzdálené PC

Veškeré predispozice pro výstup programu vycházejí z kapitoly č. 4. Jedná se o demonstraci klíčové funkcionality mobilního robotu TurtleBot3, tzn. veškeré řídící programy jsou součástí samotné instalace mobilní robotu.

Zahájení ROS Master pro umožnění komunikace mezi uzly. Spuštění základních balíčků TurtleBot3 a uzlu spouštějícího vizualizační nástroj RViz společně s programem odebírajícího data z laserového skeneru vzdálenosti a následné vizualizace v RVizu, a to v reálném čase. S ohledem na znalost parametrů použitého skeneru (viz podkapitola 2.7.2 v části *Senzorické vybavení*) je nutné uvést robot do pohybu, aby bylo možné na základě dat ze skeneru vytvořit mapu místnosti. K tomuto účelu bude využito teleoperačního uzlu pro ovládání robotu skrze klávesnici (viz obr. 68).

```
[Vzdálené PC]
$ roscore

[TurtleBot3]
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch

[Vzdálené PC]
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
  slam_methods:=gmapping

[Vzdálené PC]
$ roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

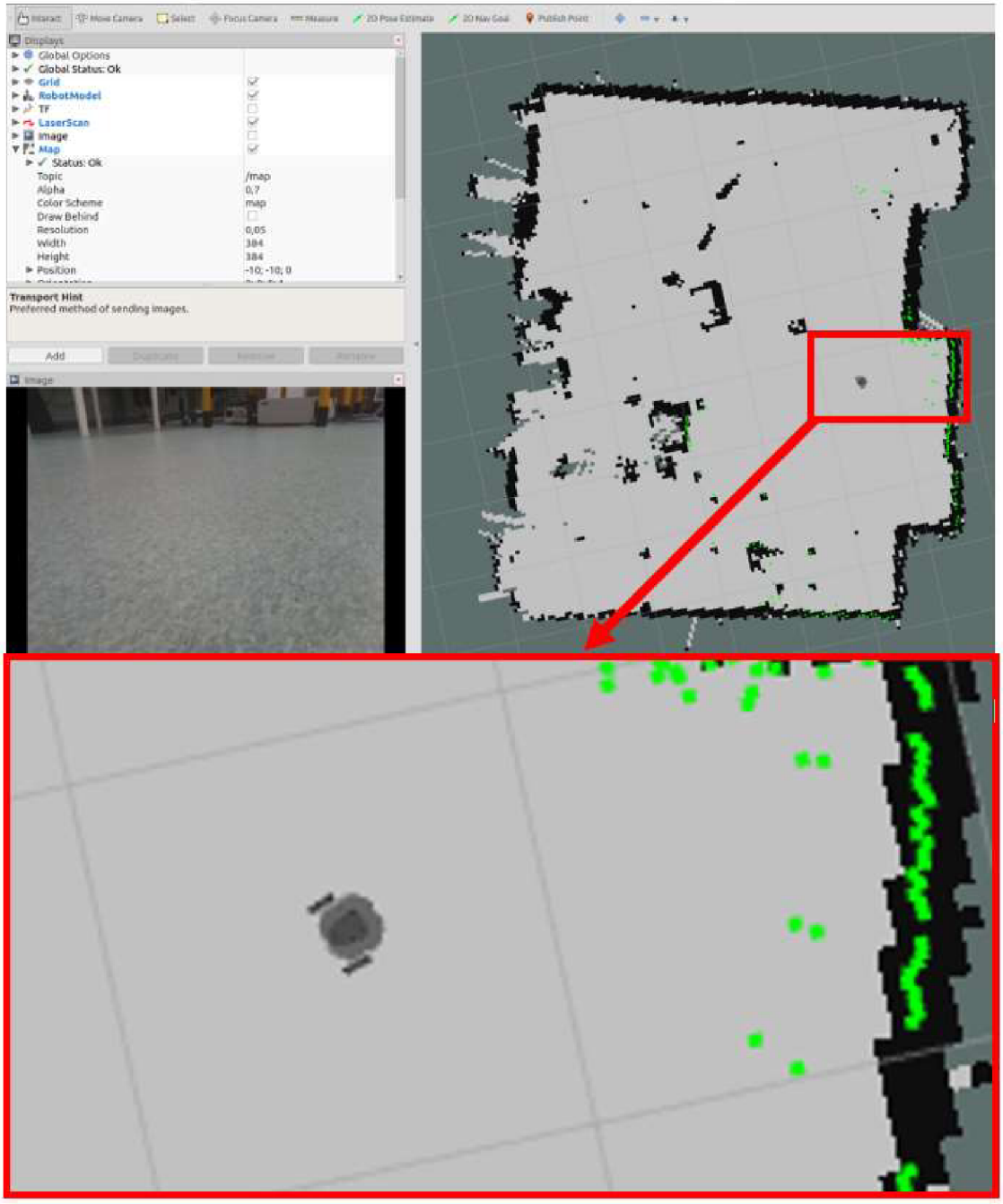
```
Control Your TurtleBot3!
---------------------------
Moving around:
        w
   a    s    d
        x

w/x : increase/decrease linear velocity (Burger : ~ 0.22, Waffle and Waffle Pi : ~ 0.26)
a/d : increase/decrease angular velocity (Burger : ~ 2.84, Waffle and Waffle Pi : ~ 1.82)

space key, s : force stop

CTRL-C to quit
```

Obr. 68: Spuštění uzlu ovládajícího robot skrze klávesnici vzdáleného PC



Obr. 69: Vizualizace dat ze skeneru v podobě mapy pomocí nástroje RViz (pozice TurtleBota přiblížena dodatečným softwarem), zelené body značí pozici překážky v reálném čase

7 ZÁVĚR

Cílem bakalářské práce bylo seznámení s problematikou mobilní robotiky, a aktuálním vývojovým stavem dané oblasti. Provedení rešerše zahrnující mobilní roboty od společnosti ROBOTIS se zaměřením na mobilní robot TurtleBot3 Burger. Nastudování základních principů fungování Robotického Operačního Systému ROS, včetně simulačního prostředí Gazebo. V rámci praktické části provést sestavení robotické stavebnice, včetně instalace a konfigurace závislostí, nutných pro první spuštění robotu. Následné navržení řídícího programu s ohledem na znalost hardwaru a softwaru tvořící jádro mobilní robotické platformy TurtleBot3 Burger. Implementace zvoleného řešení a ověření jeho funkčnosti pomocí simulace na reálném robotu.

První rešeršní část se zabývá problematikou mobilní robotiky, převážně z hlediska možností mobility v proměnném prostředí. Klasifikace interdisciplinární oblasti mobilní robotiky. Popis aktuálně existujících řešení lokomoce, s bližším pohledem na možnosti jejich využití s ohledem na operativní kvalitu v daném prostředí. Objektem zájmu jsou nejen principy založené na kolovém nebo pásovém podvozku, ale i na mechanismech, které se inspirují živými organismy. Výstupem jsou ukázky různých robotických aplikací ať už v průmyslové či výzkumné sféře, kde jsou vidět tendence propojování schopnosti mobility a robotických manipulátorů.

Společnost ROBOTIS stojí nejen za distribucí série mobilních robotů TurtleBot3, ale i dalších robotických platforem a řešení, jakými jsou humanoidní robotické platformy, řídící moduly, „inteligentní“ servomotory DYNAMIXEL a mnoho dalšího. Akční členy DYNAMIXEL nabízejí velice výkonná řešení pro pohon různých robotických aplikací, kde právě jejich variabilita, architektura a možnosti řízení stojí za označením „inteligentní“ servomotory. V rámci spektra robotických platforem od společnosti ROBOTIS je blíže popsán mobilní robot TurtleBot3 Burger. Popis hardwarové výbavy, kdy i přes kompaktnost této mobilní robotické platformy nedochází k obětování počtu nebo kvalitě vestavěných funkcionalit.

Za vznikem série mobilních robotů TurtleBot3 nestojí pouze společnost ROBOTIS, ale i společnost Open Robotics, která spolupracuje s průmyslem, akademickou sférou a vládou na vytváření otevřeného softwaru a hardwaru pro použití v robotice. Stojí za vývojem a údržbou Robotického Operačního Systému , na jehož softwarovém základě je tato robotická platforma postavena.

Druhá rešeršní oblast se zabývá tímto systémem. Charakterizuje ho, softwarově zařazuje, představuje základní benefity, které přináší do oblasti vývoje softwaru pro různorodé robotické aplikace. Stěžejním tématem je nejen pochopení základních konceptů, terminologie a principu komunikace mezi zařízeními, ale také instalace a znalost nástrojů , které umožňují introspekci získaných dat pro laděni a vizualizaci.

Sestavení robotické stavebnice TurtleBot3 Burger. Instalace a konfigurace závislostí potřebných pro spuštění a propojení mobilního robotu se vzdáleným PC. Umožnění vzájemné komunikace, a tedy ovládání robotu pomocí vzdáleného PC skrze

bezdrátovou síť. Základní postup zprovoznění je rozšířen o tipy a triky, usnadňující samotný proces a možnost rozšíření zvoleného robotu a schopnost počítačového vidění skrze Raspberry Pi kameru V2. K dosažení této dodatečné integrace bylo nutné vytvořit model držáku pro kameru, odpovídající rozměrům kamery a robotu. Následné zhotovení držáku na 3D tiskárně. A umožnění příjmu dat z kamery skrze ROS , tedy přenos obrazu mezi kamerou, upevněnou na robotu, a vzdáleným PC.

Návrh řídicího programu pro vybranou laboratorní úlohu bude souviset s využitím počítačového vidění. Varianta s využitím laserového skeneru vzdálenosti byla vyřazena, protože by bylo využíváno primárně vestavěných funkcionalit, což by znehodnotilo přínos výsledné implementace na reálném robotu. Nicméně bude alespoň představena funkce SLAM (simultánní lokalizace a mapování).

Zvolená laboratorní úloha, s využitím počítačové vidění, bude zaměřena na autonomní pohyb robotu na základě sledování čáry jízdního pruhu. K dosažení požadovaného pohybu po trajektorii jsou využívána obrazová data, zpracovaná metodou prahování pro oddělení charakteristiky čáry od podkladu. Výsledkem zpracování obrazu je jeho binární podoba, umožňující snadnější získání dat pro předpis, kterým bude definována odchylka dat pruhu od referenční polohy robotu. Předpis charakteru čáry bude vycházet z proložení dat binárního obrazu, a to optimalizovanou metodou posuvných oken. Data budou proložena polynomem druhého stupně, který lépe popisuje zakřivení trajektorie, v jejích kritických místech. Ovlivnění pohybu robotu s ohledem na velikost odchylky a typ lokomočního ústrojí – diferenciálně řízený mobilní robot, bylo navrženo skrze znalost chování regulátorů, tj. znalost proporcionální, integrační a derivační složky pro regulaci. Znalostí nástrojů ROS byla získaná data použita pro seřízení výše zmíněných parametrů Ziegler-Nicholsovou metodou, protože není známa charakteristika regulovaného systému. Reálná aplikace ukázala vhodnost použití kombinace proporcionální a derivační složky. Na základě výše zmíněných postupů byla ověřena funkčnost daného řešení na zvolené laboratorní úloze.

V rámci demonstrace vestavěných funkcionalit, které nabízí mobilní robotická platforma TurtleBot3, byla prostřednictvím laserového skeneru vzdálenosti a algoritmu pro SLAM vytvořena mapa místnosti.

8 SEZNAM POUŽITÉ LITERATURY

[1] DUDEK, Gregory a Michael JENKIN. *Computational principles of mobile robotics*. 2nd ed. New York: Cambridge University Press, 2010. ISBN 05-216-9212-1.

[2] TZAFESTAS, S. G. *Introduction to mobile robot control*. Amsterdam: Elsevier, [2014]. Elsevier insights. ISBN 978-012-4170-490.

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

[4] SIEGWART, Roland, Illah Reza NOURBAKHSH a Davide SCARAMUZZA. *Introduction to autonomous mobile robots*. 2nd ed. Cambridge, Mass.: MIT Press, c2011. ISBN 02-620-1535-8.

[5] *Home | Boston Dynamics* [online]. Waltham, MA: Boston Dynamics, c2020 [cit. 2020-04-13]. Dostupné z: https://www.bostondynamics.com/

[6] PackBot – ROBOTS: Your Guide to the World of Robotics. In: *ROBOTS: Your Guide to the World of Robotics* [online]. Piscataway, NJ: IEEE, c2020 [cit. 2020-04-13]. Dostupné z: https://robots.ieee.org/robots/packbot/?gallery=photo1

[7] *Industrial intelligence 4.0_beyond automation | KUKA AG* [online]. Augsburg, Bavaria: KUKA, c2020 [cit. 2020-04-13]. Dostupné z: https://www.kuka.com/

[8] *Autonomous Mobile Robots | Milvus Robotics* [online]. Ankara, Ankara: Milvus Robotics, c2020 [cit. 2020-04-13]. Dostupné z: https://milvusrobotics.com/

[9] *IAM Robotics | The Leader in Autonomous Mobile Manipulation Robots* [online]. Sewickley, Pennsylvania: IAM Robotics, c2020 [cit. 2020-04-13]. Dostupné z: https://www.iamrobotics.com/

[10] *Open Robotics* [online]. Mountain View, CA: Open Robotics, c2020 [cit. 2020-04-13]. Dostupné z: https://www.openrobotics.org/

[11] *ROBOTIS STORE | Robot is...* [online]. Lake Forest, CA: ROBOTIS, c2020 [cit. 2020-04-13]. Dostupné z: https://www.robotis.us/

[12] *ROBOTIS* [online]. Lake Forest, CA: ROBOTIS, c2020 [cit. 2020-04-13]. Dostupné z: http://en.robotis.com/

[13] *TurtleBot3* [online]. Lake Forest, CA: ROBOTIS, c2020 [cit. 2020-04-13]. Dostupné z: http://emanual.robotis.com/docs/en/platform/turtlebot3/overview/

[14] *Teach, Learn, and Make with Raspberry Pi – Raspberry Pi* [online]. Cambridge, Cambridgeshire: Raspberry Pi Foundation, c2020 [cit. 2020-04-13]. Dostupné z: https://www.raspberrypi.org/

[15] *ROS.org | Powering the world's robots* [online]. Mountain View, CA: Open Robotics, [2020] [cit. 2020-04-13]. Dostupné z: https://www.ros.org/

[16] *Documentation – ROS Wiki* [online]. Mountain View, CA: Open Robotics, [2020] [cit. 2020-04-13]. Dostupné z: http://wiki.ros.org/

[17] NEWMAN, Wyatt S. *A Systematic Approach to Learning Robot Programming with ROS*. New York: Chapman and Hall/CRC, 2017. ISBN 978-149-8777-827.

[18] LENTIN, Joseph, a . *Robot Operating System (ROS) for Absolute Beginners: Robotics Programming Made Easy*. New York: Apress, 2018. ISBN 978-1-4842-3404-4.

[19] YOONSEOK, Pyo, Cho HANCHEOL, Jung RYUWOON a Lim TAEHOON. *ROS Robot Programming*. Seoul: ROBOTIS Co., 2017. ISBN 979-11-962307-1-5.

[20] LENTIN, Joseph. *Learning Robotics Using Python*. 2nd ed. Birmingham, UK: Packt Publishing, 2018. ISBN 978-1-78862-331-5.

[21] LENTIN, Joseph a Gandhinathan RAMKUMAR. *ROS Robotics Projects: Build and control robots powered by the Robot Operating System, machine learning, and virtual reality*. 2nd ed. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1-83864-932-6.

[22] LENTIN, Joseph a Jonathan CACACE. *Mastering ROS for Robotics Programming: Design, build, and simulate complex robots using the Robot Operating System*. 2nd ed. Birmingham, UK: Packt Publishing, 2018. ISBN 978-1-78847-895-3.

[23] *Gazebo* [online]. Mountain View, CA: Open Robotics, c2014 [cit. 2020-04-13]. Dostupné z: http://gazebosim.org/

[24] GOLLAPUDI, Sunila. *Learn Computer Vision Using OpenCV: With Deep Learning CNNs and RNNs*. 978-1-4842-4261-2. New York: Apress, 2019. ISBN 978-1484242605.

[25] SOLEM, Jan Erik. *Programming computer vision with Python*. Sebastopol: O'Reilly, 2012. ISBN 978-1-449-31654-9.

[26] JOSHI, Prateek, David Millán ESCRIVÁ a Vinícius GODOY. *OpenCV By Example: Enhance your understanding of Computer Vision and image processing by developing real-world projects in OpenCV 3*. Birmingham, UK: Packt Publishing, c2016. ISBN 978-1-78528-094-8.

[27] KAEHLER, Adrian a Gary R. BRADSKI. *Learning OpenCV 3: computer vision in C with the OpenCV library*. Sebastopol: O'Reilly, 2016. ISBN 978-1-4919-3799-

9 SEZNAM PŘÍLOH

PŘÍLOHA A. CD-ROM

Tab. 11: Obsah CD – hlavní adresář („\Bakalářská_práce“)

Adresář	Soubor
\BP_pdf\	`2020_BP_FILIP_Jakub_200532_VUT_FSI_UAI.pdf`

\ROS_workspace\	`control_graphs`
	`include`
	`output_images`
	`scripts`
	`src test_images`
	`CMakeLists.txt`
	`package.xml`
	`setup.py`

\držák_rpi_kameraV2\	`držák_rpi_kameraV2.zip`
\Readme\	`readme.txt`