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 REALIZACE ŘÍDÍCÍCH SYSTÉMU PRO MOBILNÍ ROBOT

PROPOSAL AND IMPLEMENTATION OF MOBILE ROBOTS CONTROL SYSTEMS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Jakub Krysl
AUTHOR

VEDOUCÍ PRÁCE — Ing. Michal Růžička
SUPERVISOR

BRNO 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Jakub Krysl**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**Ing. Michal Růžička**
Akademický rok:	2015/16

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

Návrh a realizace řídících systému pro mobilní robot

Stručná charakteristika problematiky úkolu:

Práce je zaměřena na návrh a realizaci řídících systému pro mobilní robot. Řídicí jednotka mobilního robotu bude Raspberry Pi, na kterém bude běžet platforma ROS, která bude obsluhovat všechny stěžejní periferie mobilního robotu. Stěžejními periferiemi jsou rozuměny aktuátory, enkodéry, kamera, senzory pro měření vzdálenosti, komunikace pomocí WIFI technologie.Konkrétním výstupem práce by mela byt funkční platforma mobilního robotu, která je propojena se systémem ROS. Platforma mobilního robotu by mela by schopna posílat telemetrická data pomoci technologie WIFI. Další části diplomové práce je návrh a realizace plánovacího modulu.

Cíle diplomové práce:

Seznámit se s platformou ROS.
Implementovat ROS pro řídící jednotku Raspberry Pi.
Realizovat komunikaci se všemi periferiemi mobilního robotu.
Provést praktické experimenty s navrženým řídicím
systémem. Implementovat plánovací modul pro mobilní robot.
Provést praktické experimenty pro ověření funkce plánovacího modulu na dané platformě.

Seznam literatury:

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series). Intelligent robotics and autonomous agents. The MIT Press, August 2005.

Howie Choset, Kevin M. Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia E. Kavraki, and Sebastian Thrun. Principles of Robot Motion: Theory, Algorithms, and Implementations (Intelligent Robotics and Autonomous Agents). The MIT Press, June 2005.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2015/16

V Brně, dne

L. S.

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

ABSTRAKT

Tato práce se zabývá návrhem a realizací autonomního robotu s použitím platformy ROS. Jejím cílem je seznámit se s platformou ROS a s její pomocí realizovat autonomní řízení reálného mobilního robotu Leela.

ABSTRACT

This thesis deals with the design and implementation of autonomous robot with using of the platform ROS. Its goal is to get to know the ROS and use it to implement autonomous control of real robot Leela.

KLÍČOVÁ SLOVA

ROS, Raspberry Pi, řízení, navigace, autonomní mobilní robot, Linux.

KEYWORDS

ROS, Raspberry Pi, control, navigation, autonomous mobile robot, Linux.

BIBLIOGRAFICKÁ CITACE

KRYSL, J. *Návrh a realizace řídícíh systému pro mobilní robot,* Brno, Vysoké učení technické v Brně, Fakulta strojního inženýrství. 2016, 79 s., Vedoucí diplomové práce Ing. Michal Růžička.

PODĚKOVÁNÍ

Rád bych touto cestou poděkoval vedoucímu práce Ing. Michalu Růžičkovi za cenné rady a za pomoc při tvorbě této práce. Také děkuji své přítelkyni a rodině za pomoc i za trpělivost.

PROHLÁŠENÍ O ORIGINALITĚ

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

V Brně dne 26.5.2016 ……

Bc. Krysl Jakub

OBSAH

1 ÚVOD

Tato diplomová práce se zabývá návrhem a realizací řídího systému pro mobilní robot Leela s použitím open source platformy ROS. Rešerše ROSu a seznámení se s touto platformou je v kapitole 0, kde jsou popsány funkce tohoto systému, uspořádání a i historie a budoucnost ROSu. Následně je předložen návrh řešení formou seznámení se s komponenty, pro který byly tyto komponenty vybrány v souladu se zadáním diplomové práce. Tento návrh je realizován v kapitole 0, kdy nejdříve je vyřešena harwarová část a poté následuje detailní popis řešení softwaru. Následně jsou provedeny a popsány praktické experimenty k ověření funkčnosti celé práce.

Cílem práce je vytvoření řídícího systému pro robotickou platformu schopného samostatného pohybu v prostoru. Platforma musí být schopna bez vnějšího zásahu naplánovat cestu do cíle a tuto cestu i realizovat. Tato platforma slouží jako základ pro experimentální část disertační práce Ing. Michala Růžičky, která se zabývá určením globální polohy robotu. Proto určení globální polohy není v této práci řešeno a veškerá lokalizace robotu v této práci je pouze lokální.

ROS je velmi rozsáhlý projekt a tomu odpovídá i jeho dokumentace, proto byla tato práce psána formou podrobného návodu pro potřeby snadné realizace podobného problému. Tento návod zrychlí implementaci ROSu do podobného robotu a umožní další vývoj použité platformy a experimenty s ní.

2 ROS FRAMEWORK

ROS (*Robotic Operation System*) je modulární open-source platforma sestávající ze souboru balíčků, knihoven a nástrojů pro vývoj a realizaci autonomního robotu. Celý systém je navržen tak, aby si uživatel mohl zvolit, které části platformy použije a které si vytvoří sám. ROS nativně implementuje kód v jazycích Python, C++ a Lisp, experimentálně v Lua a Java. Podporované softwarové platformy jsou Ubuntu Linux a Ubuntu (armhf) Linux, experimentálně OS X, Gentoo Linux, Arch Linux a Android (NDK) či neoficiální alternativa Debian Linux při instalaci přímo ze zdroje. Tento systém je stále ve vývoji a jeho cílem je ulehčit vývoj autonomního robotu pomocí celosvětové spolupráce laboratoří i nadšenců a současně posouvat hranice ve vývoji těchto robotů. [1]

2.1 Popis

Samotný ROS se chová téměř jako operační systém, poskytuje hardwarovou abstrakci, nízkoúrovňovou komunikaci s hardwarem, implementaci běžně používané funkcionality, posílání zpráv mezi procesy a správu balíčků. Současně umožňuje získávání, kompilaci, tvorbu a spouštění programového kódu na více zařízeních. Ačkoliv v robotice je důležitá vysoká reaktivita a nízká latence, ROS není realtime platformou, avšak poskytuje možnost integrace realtime kódu. [2]

Koncept ROSu [3] lze pojmout na 3 úrovních: souborové, výpočetní a komunitní. Tyto koncepty jsou implementovány pomocí klientských knihoven (*client library*). Implementace těchto knihoven teoreticky není závislá na programovacím jazyku, ale v současnosti je ROS zaměřen na poskytnutí jejich robustní implementace v jazycích C++, Python a Lisp, zatímco podpora pro ostatní programovací jazyky (C#, Java, Go, Lua, Ruby a jiné) je zatím pouze experimentální. [4]

2.1.1 Koncept souborové úrovně

Tento koncept popisuje především soubory na disku, podle určení těchto souborů je můžeme rozdělit takto:

Packages (dále v textu pod pojmem *balíčky*) jsou základní jednotkou organizace softwaru v ROSu. Balíčky jsou v základní definici jakékoliv použitelné moduly; mohou obsahovat ROS *nodes* (viz 2.1.2 Koncept výpočetního grafu), nezávislé knihovny, datové sady, konfigurační soubory či dokonce části programů třetí strany. Každý balíček v ROS ekosystému musí mít specifikovanou licenci. Balíčky tvořící jádro ROSu jsou šířeny pod zjednodušenou licencí BSD [5], zatímco ostatní jsou licencovány podle úvahy autora balíčku. Mezi často používané licence patří Apache 2.0, GPL, MIT či dokonce i proprietární licence. [6]

Metapackages (dále v textu pod pojmem *metabalíčky*) jsou specializované softwarové balíčky, které obsahují pouze package.xml *manifest* (viz 2.1.2 Koncept

výpočetního grafu). Jsou to vlastně virtuální balíčky, jejichž cílem je odkazovat na jiné balíčky a spojovat je do jednoho celku. [7]

Package Manifests jsou XML soubory package.xml, které definují vlastnosti balíčku. Mezi tyto vlastnosti patří jméno balíčku, licence, verze balíčku, autoři, správci, závislosti na ostatních balíčcích a jiné metainformace. Tyto soubory jsou definované v REP-0127 (viz [8]). [9]

Repositories se nacházejí po celém světě, kde jednotlivá úložiště jsou spravována autory či správci balíčků, které jsou na tyto úložiště nahrávány. Cílem těchto úložišť je sdílení vytvořeného kódu napříč komunitou ROSu. Každé úložiště používá vlastní systém správy verzí. Celý indexovaný seznam úložišť lze najít na domovských stránkách ROSu. [10]

Message (msg) types, tedy typ zpráv v ROSu, slouží k popisu a definování typu zpráv posílaných procesy v ROSu. Tyto informace jsou ukládány s koncovkou .msg ve vlastním adresáři /msg v balíčku. Implementovány jsou jednoduché datové typy (integer, float a jiné) i složitější (například pole). Uživatel má možnost si nadefinovat vlastní datovou strukturu zprávy. Více o zprávách viz 2.1.2 Koncept výpočetního grafu. [11]

Service (srv) types jsou soubory obsahující popis datových struktur požadavků a odpovědí, jejich koncovka je .srv a nacházejí se ve vlastním adresáři /srv. Definují se stejný způsobem jako zprávy s tím rozdílem, že požadavky a odpovědi jsou v souboru .msg rozděleny třemi pomlčkami na novém řádku („---“). [12]

2.1.2 Koncept výpočetního grafu

Výpočetní graf je síť typu klient-klient (P2P) všech procesů ROSu, které dohromady zpracovávají data. Jednotlivé vrcholy tohoto grafu se podle ROS konceptu výpočetního grafu dělí následně:

Nodes jsou procesy provádějící jakýkoliv výpočet. Jednotlivé *nodes* mezi sebou komunikují pomocí *topics*, *services* nebo *parameter server*. Každá *node* je specializovaná na vlastní úkol, například ovládání motorů, lokalizace či výpočet odometrie. Modulární použití *nodes* přináší velkou výhodu v robustnosti systému, kdy chyby a pády jsou omezeny pouze na jednotlivé *nodes*. Mezi další výhody patří snížení komplexity kódu v porovnání s monolitickými systémy, skrytí detailů implementace a možnost alternativní implementace dokonce v jiném programovacím jazyce. Každá *node* běžící v ROSu má unikátní jméno a vlastní typ, kterým je jméno zdrojového balíčku. Spuštění jakéhokoliv balíčku v ROSu vytvoří samostatnou *node*. [13]

Master poskytuje pojmenování a registraci služeb pro všechny ostatní části výpočetního grafu. Jeho účelem je umožnění vyhledávání a spojení *nodes* mezi sebou s cílem navázání vzájemné komunikace. Po počátečním navázání přes *master* už spolu *nodes* komunikují přímo. *Master* udržuje přehled o všech *publishers* a *subscribers*. [14]

Parameter server je sdílený slovník, který *nodes* používají k ukládání a čtení statických nebinárních parametrů za chodu. Informace na tomto serveru jsou globálně přístupné, aby jednotlivé nástroje k nim mohly jednoduše přistupovat a měnit je podle potřeby. V současnosti je to jedna z částí *master*. [15]

Messages (dále v textu pod pojmem *zprávy*) jsou způsob komunikaci mezi jednotlivými *nodes*, kdy *nodes* publikují zprávy na *topic*. Každá zpráva je definovaná datová struktura (viz 2.1.1 Koncept souborové úrovně). Aby *nodes* mohly mezi sebou komunikovat, musí souhlasit nejen typ zprávy, ale i MD5 sum .msg souboru. Zprávy často obsahují údaj o čase, ve kterém byla daná zpráva vytvořena. [16]

Topics představují místa, přes která si *nodes* navzájem vyměňují zprávy za určité frekvence. *Topics* používají anonymní *publish/subscribe* sémantiku, která odděluje vytváření a používání informací. Tedy jednotlivé *nodes* nemají představu, s kým komunikují přes *topic*. Podle zájmu o informace na *topic* se *nodes* dělí na *publisher*, které data na *topic* posílají, a *subscriber*, které data z *topic* naopak získávají. Není omezený počet *subscribers* a *publishers* na jednom *topic*. Přenos zprávy přes *topic* je realizován pomocí TCP/IP nebo UDP. Způsob přenosu mohou *nodes* změnit za chodu, pokud původní není podporovaný oběma stranami. [17]

Services jsou jiný způsob výměny informací mezi *nodes*. Vzhledem k anonymní podstatě *topics* bylo nutno vytvořit alternativní způsob komunikace mezi *nodes* pro potřeby odpovědi na požadavek, což *topics* neumožňují. Nod může poskytovat službu, která se skládá ze dvou zpráv (požadavek a odpověď), které můžou mít rozdílný definovaný typ (viz 2.1.1 Koncept souborové úrovně). Komunikace zde funguje na principu klient-server, kdy klient pošle požadavek a čeká na odpověď. [18]

Bags jsou speciální formát souboru pro ukládání a přehrávání zpráv. Jejich název je odvozen od faktu, že přípona těchto souborů je .bag. V ROS ekosystému existuje mnoho nástrojů pro zpracování, analýzu a vizualizaci těchto souborů. Vzhledem k jejich povaze jsou hlavním nástrojem vytváření logů v ROSu, umožňují dodatečnou analýzu nasbíraných dat za účelem například diagnostiky. [19]

2.1.3 Koncept komunitní úrovně

V ROSu je vytvořena komunitní úroveň za účelem sdílení znalostí a vytváření zdrojů v ROS ekosystému. Tyto zdroje lze dělit následovně:

Distributions jsou sbírky balíčků o určité verzi, které si uživatel může nainstalovat. Je to velmi podobné distribucím například operačního systému Ubuntu. Jejich účelem je vytvoření stabilní základny pro vývoj vlastních balíčků. Každá distribuce má vlastní datum vydání a datum EOL, tedy ukončení její podpory. Během této doby je snaha udržet změny v distribuci pouze na opravách chyb a minimálních vylepšeních. Více o distribucích viz 2.2 Historie a vývoj. [20]

Repositories viz 2.1.1 Koncept souborové úrovně.

ROS Wiki je základním zdrojem dokumentace o ROSu. Tato wiki funguje na stejném principu jako ostatní wiki, tedy obsah je regulován uživateli. [3]

Bug Ticket System je systém správy chyb a požadování nových vlastností balíčků. Vzhledem k open source vývoji ROSu je každý ticket vytvořen přímo vůči konkrétnímu balíčku a řešen správci a uživateli onoho balíčku. [21]

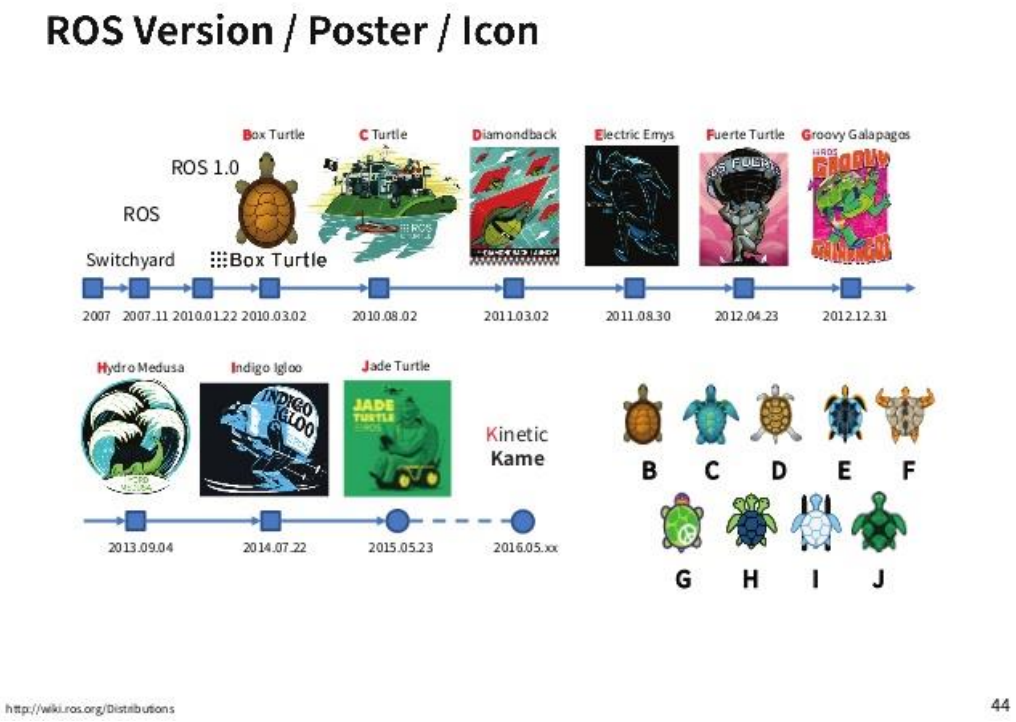
Mailing Lists je základní komunikační kanál o nových updatech ROSu a obecné diskusi o ROSu. [3]

ROS Answers slouží k pokládání dotazů ohledně jakéhokoliv aspektu ROSu. Je to též hlavní místo, kde uživatel ROSu může získat pomoc ohledně jakéhokoliv aspektu tohoto systému. [3]

Blog poskytuje pravidelné updaty týkající se ROSu včetně fotek a videí. [3]

2.2 Historie a vývoj

Počátky ROSu se datují do května 2007, kdy na Standfordské Univerzitě byl Morganen Quigley vyvíjen systém *Switchyard* pro projekty *STanford Artificial Intelligence Robot* (STAIR) a *Personal Robotics* (PR) program. Projekt PR se následně v srpnu 2007 přesunul do technologického inkubátoru Willow Garage, kde pokračoval pod názvem PR2. Zde také vznikl název *Robotic Operation System* (ROS), který se zakládal na systému *Switchyard*. Vývoj ROSu pokračoval především pro platformu PR2, první betaverze ROSu byla vydána v lednu 2009 pod názvem ROS 0.4 Mango Tango, avšak první plnohodnotná distribuce ROS 1.0 vyšla až v lednu 2010, následně v březnu pod názvem ROS 1.0 Box Turtle. Nejspíše od tohoto názvu si ROS udržuje charakteristickou ikonku želvy, která je vlastní každé distribuci. Od té doby si ROS získal velkou popularitu a vývoj rychle pokračoval. V únoru 2013 se vývoj jádra a údržba ROSu přesunula do společnosti Open Source Robotic Foundation (OSRF), kde sídlí dodnes. V současné době se připravuje v pořadí od verze 1.0 již 10. verze ROSu s názvem Kinetic Kame. Tato verze je považována za experimentální, doporučené jsou předchozí verze Jade Turtle, která je nejnovější, nebo Indigo Igloo, doporučovaná pro svou stabilitu. Přehled všech dosavadních distribucí včetně ikon je na Obr. 1. [20, 22, 23]



Obr. 1 Verze ROSu. [24]

Vzhledem k dlouhému vývoji a zpětné kompatibilitě některých balíčků až do verze ROS 0.4 je v současnosti ve vývoji verze ROS 2, která má přinést mnoho změn. Jejím hlavním cílem je rozšíření nativní podpory na jiné aplikace než roboty typu PR2. Mezi tyto aplikace patří týmy více robotů, kde je hlavním limitujícím faktorem nutnost jednoho *master*, nativní podpora malých embeded systémů, realtime systémů, nekvalitní sítě a řešení ztrát informací v síti, průmyslového výrobního prostředí a jiné. Další velkou změnou je vytvoření nového aplikačního prostředí, které je v současné době založené na stejném principu jako v ROS 0.4. Tento vývoj ROS 2 pokračuje paralelně s vývojem původního ROSu z důvodu předpokládané nekompatibility některých těchto nových vlastností s původní verzí ROSu. V budoucnu se předpokládá vytvoření způsobu komunikace mezi těmito dvěma systémy. ROS 2 je v současné době ve verzi alfa 5. Tato verze poskytuje podporu služeb v jazyce C, experimentální podporu 32 bitové a 64 bitové architektury ARM a vytvořená dema. Přehled budoucích plánovaných změn lze nalézt v plánu ROS 2. [25, 26]

2.3 Důležité balíčky, knihovny a nástroje

ROS ekosystém obsahuje obrovské množství balíčků, z nichž některé jsou důležitější než jiné. Tyto balíčky využívá téměř každý robot fungující na ROSu, protože jsou široce použitelné, stabilní, poskytují programátorský komfort či tvoří samotné jádro ROSu. Toto není kompletní výčet důležitých balíčků, pouze stručný přehled některých z nich.

2.3.1 Balíček roscore

Tento balíček je sbírka *node*, které tvoří absolutní základ pro ROS. Je nutné ho spustit, aby *nodes* mohly komunikovat mezi sebou. Toto se provede příkazem v Tabulka 1, který spustí ROS *master*, ROS *parameter server* a loggovací *node* balíčku *rosout*, která slouží především k zachytávání chyb a vnitřních zpráv ROSu. [27]

Tabulka 1 Příkaz pro spuštění jádra ROSu. [27]

```
$ roscore
```

2.3.2 Balíčky roscpp, rospy a roslisp

Balíčky roscpp, rospy a roslip jsou klientské knihovny ROSu poskytující implementaci konceptů do programovacích jazyků C++, Python a Lisp. Programátoři pracující s ROSem využívají výhod těchto jazyků jejich použitím na činnosti, na které se nejvíce hodí: roscpp je nejpoužívanější díky výpočetní rychlosti C++, rospy poskytuje sice menší výpočetní rychlost, ale má mnohem větší rychlost implementace, neboť kód psány v jazyce Python není nutné kompilovat. A poslední roslisp je používán především na vývoj plánovacích knihoven. [4]

2.3.3 Balíček rospack

Rospack je nástroj pracující v příkazovém řádku pro získávání informací o balíčcích v ekosystému ROSu. Jeho základní schopnost je získávání informací z jednotlivých souborů manifest.xml balíčků a vytváření stromu závislostí. Na základě tohoto stromu je schopen podat uživateli informace o cestě k jednotlivým balíčkům a jejich závislostech, informace o závislostech na těchto balíčcích, informace nutné k jejich kompilaci a jiné. [28]

2.3.4 Balíček rostopic

Dalším balíčkem je rostopic, který je důležitým nástrojem pro správu vytvořených *topics*. Tento nástoj pracuje v příkazovém řádku a poskytuje uživateli informace nutné k debuggování *topics* včetně výpisu *publishers*, *subscribers*, zpráv a frekvence jednotlivých *topics*. Dále obsahuje nástroje pro správu a úpravu těchto *topics* za chodu, které umožňují například ručně publikovat zprávu z konzole na daný *topic*. [29]

2.3.5 Balíček rosbash

Balíček rosbash implementuje a třídí příkazy shellu pro ROS. Obsahuje užitečné bash funkce a dovoluje uživateli používat dokončování pomocí klávesy TAB při psaní příkazů v příkazovém řádku. Jeho nejpoužívanějším nástrojem je *rosrun*, který dovoluje uživateli pouštět ROS balíčky bez znalosti absolutní cesty. Syntaxe příkazu *rosrun* a jeho příklad použití viz Tabulka 2. [30]

Tabulka 2 Příkazy balíčku rosbash. [30]

```
$ rosrun package node _parameter:=value
$ rosrun my_package my_node _my_param:=value
```

2.3.6 Balíček rqt

Rqt je metabalíček poskytující framework založený na knihovně QT. Tento framework implementuje různé nástroje grafického uživatelského prostředí (GUI) ve formě pluginů, což uživateli umožňuje jejich komfortnější správu. Pokud chce uživatel ROSu spustit balíček jako plugin rqt, při spuštění balíčku nahradí příkaz *rosrun* příkazem *rqt* (viz Tabulka 3). [31]

Tabulka 3 Příkazy pro nahrazení rosrun pomocí rqt. [31]

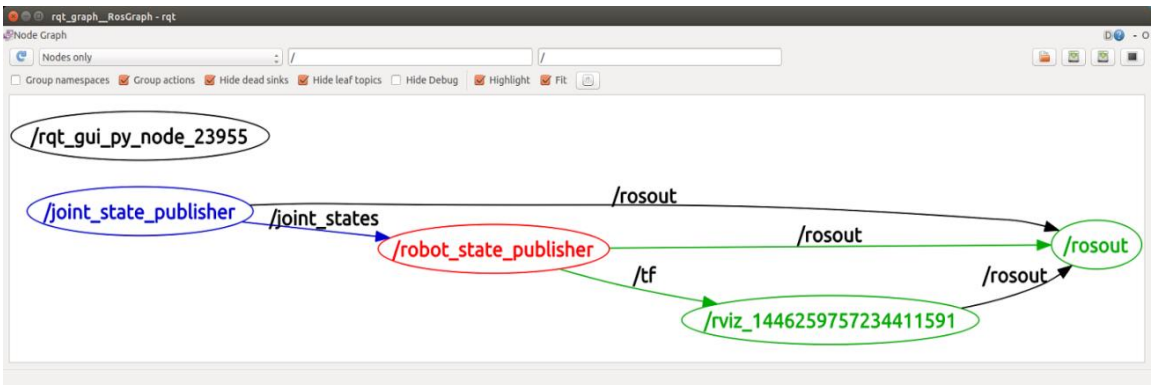
```
$ rosrun my_package my_node
$ rqt my_package my_node
```

2.3.7 Balíček rqt_graph

Tento balíček poskytuje GUI pro vizualizaci konceptu výpočetního grafu, díky kterému je uživatel schopen velice jednoduše debuggovat komunikaci mezi balíčky. Od verze ROS Indigo povoluje uživateli zobrazení statistik jednotlivých *topics* do grafu včetně například frekvence. Na Obr. 2 je příklad vizualizace pomocí balíčku rqt_graph (příkaz viz Tabulka 4), kdy hrany grafu jsou typy zpráv a kořeny jednotlivé *nodes*. [32]

Tabulka 4 Příkaz pro vizualizaci pomocí balíčku rqt_graph. [32]

```
$ rosrun rqt_graph rqt_graph
```



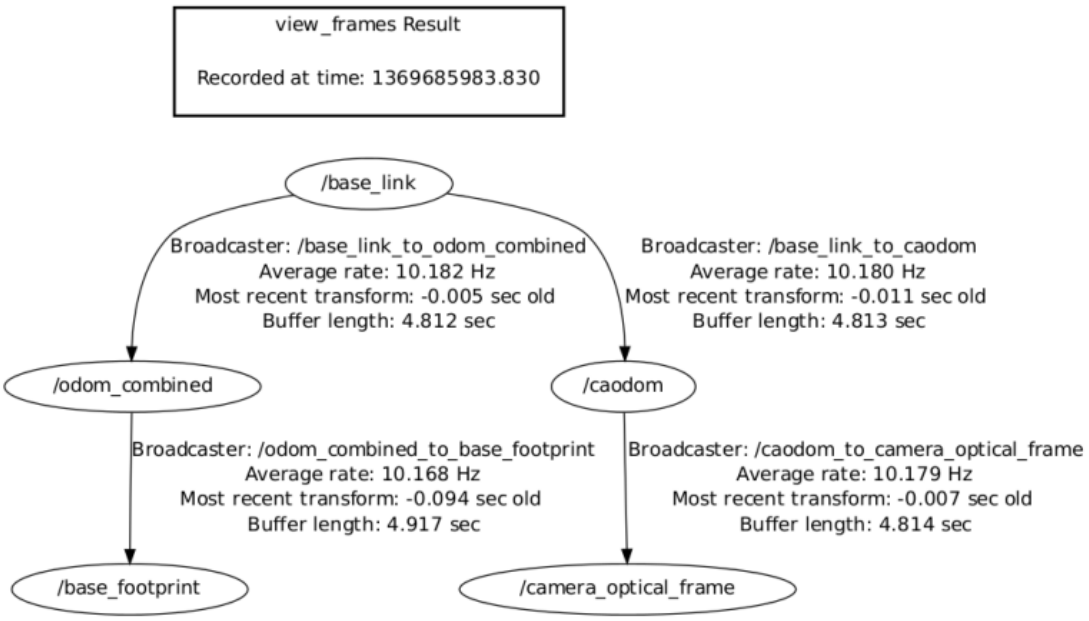
Obr. 2 Příklad vizualizace pomocí balíčku rqt_graph. [33]

2.3.8 Balíček tf

Tf je balíček pracující se souřadnicovými rámy (*coordinate frames*), na jejichž základě je schopný sledovat změnu uživatelem definovaných prostorových souřadnic v čase a vzájemnou polohu. S těmito údaji vytvoří transformační strom, pomocí kterého uživateli umožní vzájemný přepočet jakýchkoliv souřadnic definovaných v tomto stromu vůči uživatelem stanovenému času. Tento přepočet se provádí pomocí dotazů, které mohou znít například: „Jak byl natočený objekt v ruce robotu vůči jeho základně před 5 vteřinami.“. Výstupem jsou prostorové souřadnice a natočení. Vzájemný vztah jednotlivých souřadnicových rámů lze jednoduše zobrazit a debuggovat nástrojem *view_frames* (viz Obr. 3), který vytvoří graf (příkaz viz Tabulka 5) současného transformačního stromu do souboru PDF. [34]

Tabulka 5 Příkaz pro vytvoření tf grafu. [34]

```
$ rosrun tf view_frames
```



Obr. 3 Příklad transformačního stromu vizualizovaného nástrojem view_frames. [35]

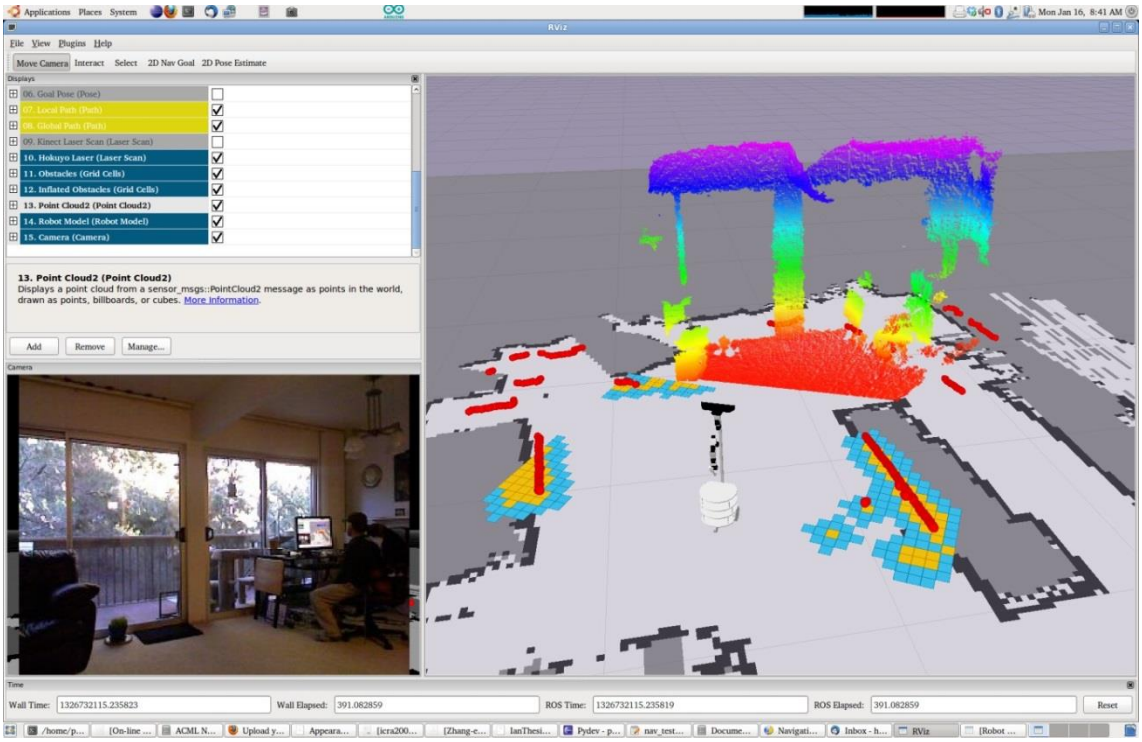
2.3.9 Balíček navigation

Velice důležitým balíčkem je metabalíček navigation, který na základě informací z odometrie, senzorů a zadaného cíle cesty je schopný posílat robotu bezpečné údaje o potřebné rychlosti v jednotlivých osách pro dosažení cíle. Pracuje pouze ve 2D prostoru, což je pro mnohé aplikace limitující. Dalšími limitujícími faktory jsou potřeba holonomického nebo diferenciálního podvozku robotu, použití planárního laseru a potřeba přibližně čtvercového půdorysu robotu. Pro jeho správné fungování musí uživatel zprovoznit na robotu ROS, transformační strom, správně publikovat data ze senzorů a být

schopný interpretovat rychlostní příkazy pro konkrétní platformu robotu. [36]

2.3.10 Balíček rviz

Posledním balíčkem je 3D vizualizační nástroj rviz (ROS Visualization), který umožňuje uživateli vizualizovat robot, jeho pohyb, okolí i data ze senzorů pomocí GUI. Příklad této vizualiza je na Obr. 4. Tento nástroj také slouží k přímému ovládání robotu včetně zadávání cílů. Uživatel pomocí GUI nadefinuje jednotlivé zobrazení, jejichž data jsou především zprávy z *topics*. [37]



Obr. 4 Vizualizace pomocí nástroje rviz. [38]

2.4 Příklady aplikací

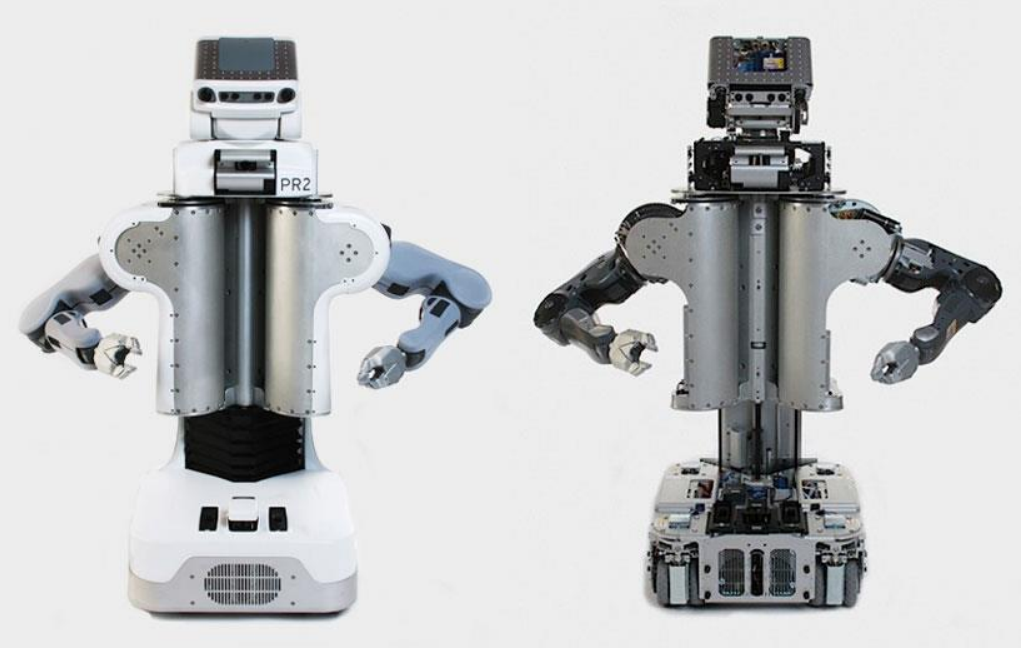
ROS se používá k nepřebernému množství aplikací. Některé z nich se velmi liší od původně zamýšlené možnosti použití tohoto systému. V následujících podkapitolách je výběr několika aplikací řízených tímto systémem. Tyto příklady jsou vybrány s ohledem na pokrytí co největší škály možných uplatnění, ač to rozhodně nejsou všechna. Kompletní seznam robotů využívajících ROS lze najít na ROS Wiki. [39]

2.4.1 PR2

Jako první je nutné se zmínit o robotu PR2 (*Personal Robot 2*) zkonstruovaným ve Willow Garage. Kvůli zániku Willow Garage je tento robot momentálně udržován společností Clearpath Robotics, na jejichž stánkách lze nalézt veškeré potřebné informace ke zprovoznění tohoto robotu. [40]

Tento robot (viz Obr. 5) je mobilní manipulační platforma, jejíž celý systém byl napsán v systému ROS. Robot sestává ze všesměrové čtvercové základny o rozměru 668 mm s maximální rychlostí 1 m/s, teleskopické páteře s rozsahem od 1330 mm do 1645 mm, hlavy s náklonem 115° a otočením 350° a 2 serverů obsahujících po 2 čtyřjádrových procesorech i7 Xeon, pamětí 24 GB RAM a 2 TB diskem. Manipulaci obstarávají 2 tříkloubová ramena zakončená uchopovacími zařízeními schopnými vyvinout sílu 80 N. Snímání prostoru tímto robotem je zajištěno laserovými scannery Hokuyo UTM-30LX v bázi a v úrovni ramen, MEMS senzorem Microstrain 3DM-GX2 IMU v úrovni ramen, Microsoft Kinect a ethernet kamerou v hlavě robotu, akcelerometry, tlakovými sensory a kalibračními led v uchopovacích zařízeních a ethernet kamerami na předloktích. Celý systém je schopný komunikovat pomocí ethernetu, WiFi a síti EtherCAT. Napájení je řešeno 1,3kWh Li-Ion bateriemi s výdrží přibližně 2 hodiny. [41]

Vzhledem k faktu, že ROS byl původně vyvíjen pro tohoto robota, je jeho celý kód velmi dobře popsán. Díky velkému množství tutoriálů pro PR2 ho lze pokládat za výbornou učící platformu pro uživatele začínající s ROSem. Jeho cena se pohybuje kolem 400 000$. [42]



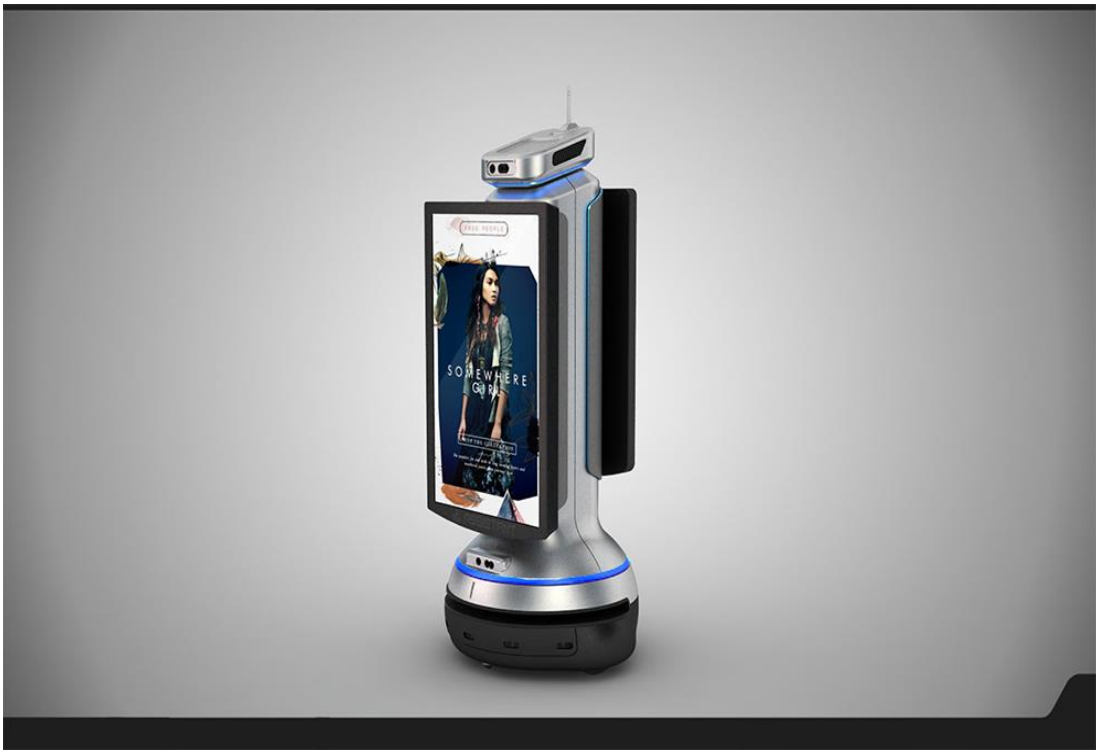
Obr. 5 PR2 robot. [43]

2.4.2 ADAS Development Vehicle Kit

Tato aplikace je kompletním hardwarovým i softwarovým řešením elektronicky ovládaného osobního automobilu. Je schopna jednolitě kontrolovat plyn, brzdy, zatáčení i řazení. Data jsou získávána z gyroskopů, akcelerometrů, GPS, enkodérů a tlakových snímačů v kolech. Společnost Dataspeed Inc. poskytuje toto kompletní řešení pro modely vozů Lincoln MKZ a Ford Fusion 2013 nebo novější. [44]

2.4.3 Robin

Robot firmy Milvus Robotics (viz Obr. 6) je sociální robot nové generace. Je schopen samostatného pohybu a mapování interiérů a audiovizuálního kontaktu s lidmi. Robin je především reklamní robot, který díky svým schopnostem funguje jako mobilní kiosk. Dokáže nejen zobrazovat a cílit reklamy, ale poskytovat rady a pomoc lidem v nouzi například zavoláním pomoci. [45]

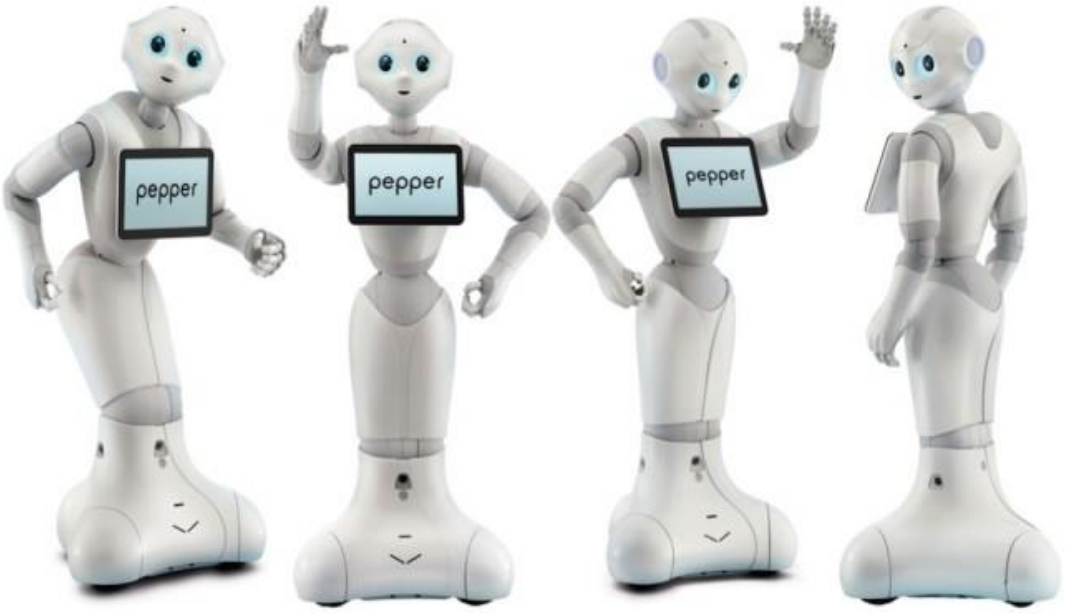


Obr. 6 Robot Robin firmy Milvus Robotics. [46]

2.4.4 Pepper

Tento humanoidní robot vyrobený firmou Aldebaran je prvním robotem schopným číst základní lidské emoce z výrazu tváře, tónu hlasu, použitým slovům a řeči těla. Na základě těchto informací Pepper upraví své chování tak, aby odpovídalo přítomnému člověku. Příklady úpravy chování jsou na Obr. 7. Vzhledem k těmto schopnostem je používán ve více než 140 SoftBank Mobile prodejnách k vítání a informování zákazníků a stal se prvním robotem adoptovaným do japonské rodiny. [47]

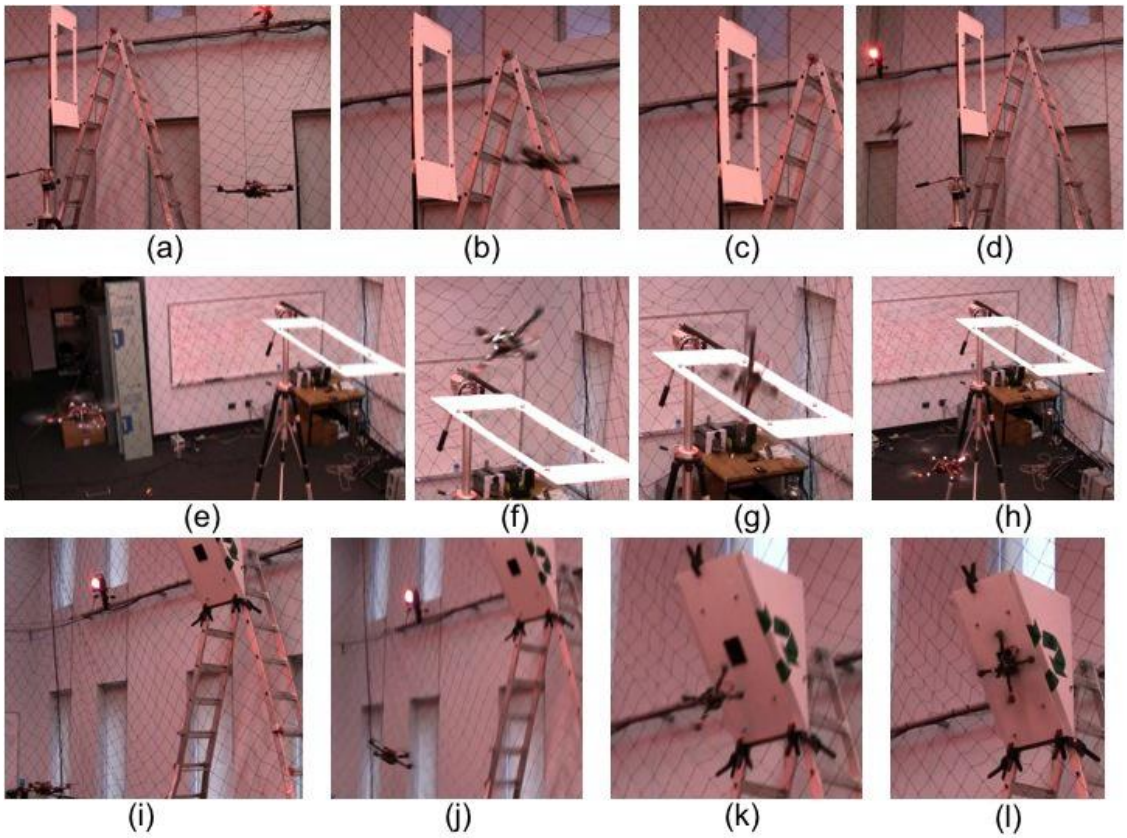
Pepper je vybavený 4 směrovými mikrofony, 1 3D a 2 HD kamerami umístěnými v hlavě, WiFi připojením k internetu a dotykovým tabletem. Pohyb po okolí je realizován pomocí 3 všesměrových kol schopných maximální rychlosti 3 km/h a pohyb částí robotu zajišťuje 20 servomotorů umístěných v hlavě, rukách a zádech k interakci s okolím. Li-Ion baterie mají dle výrobce výdrž 12 hodin. Orientaci robotu v prostoru zajišťuje součinnost 2 ultrazvukových senzorů, 6 laserových skenerů a 3 detektorů překážek, která umožňuje robotovi detekovat překážky vzdálené až 3 metry. [47]



Obr. 7 Pepper společnosti Aldebaran. [48]

2.4.5 Penn Quadrotors

Jedním z příkladů možností ROSu ovládat autonomní vzdušné roboty je projekt Pensylvánské univerzity z roku 2010 s názvem Penn Quadrorotors. Cílem projektu bylo přesné řízení kvadrokoptéry po různých agresivních trajektoriích. Celý systém byl postaven na směsi kódu pro microkontroler, který řídil výšku robotu, a ROS, který zprostředkovával komunikaci mezi ostatními programy. Modularita ROSu zde umožnila vyvíjet jednotlivé části projektu nezávisle na sobě. Hardware sestával z kvadrokoptéry, laptopu a systému snímání pohybu Vicon. Tento systém snímal polohu kvadrokoptéry s frekvencí 225Hz, kdy data byla zpracována ROSem, který vypočetl momentální rychlosti robotu. Pomocí ROS-Matlab mostu byly tyto rychlosti poslány do Matlabu, který z nich vypočetl potřebné příkazy pro samotnou kvadrokoptéru. Příkazy byly poslány zpět do ROSu, který vždy ten nejnovější poslal s frekvencí 100Hz přes ZigBee do mikrokontroleru v robotu. Tento systém řízení byl testován agresivní vzdušnou akrobacií sestávající z různých vývrtek a přetočení, průlety malým oknem pod různými úhly naklonění rámu a různými směry průletu a také přichycení k různě nakloněné ploše (viz Obr. 8). [49, 50]



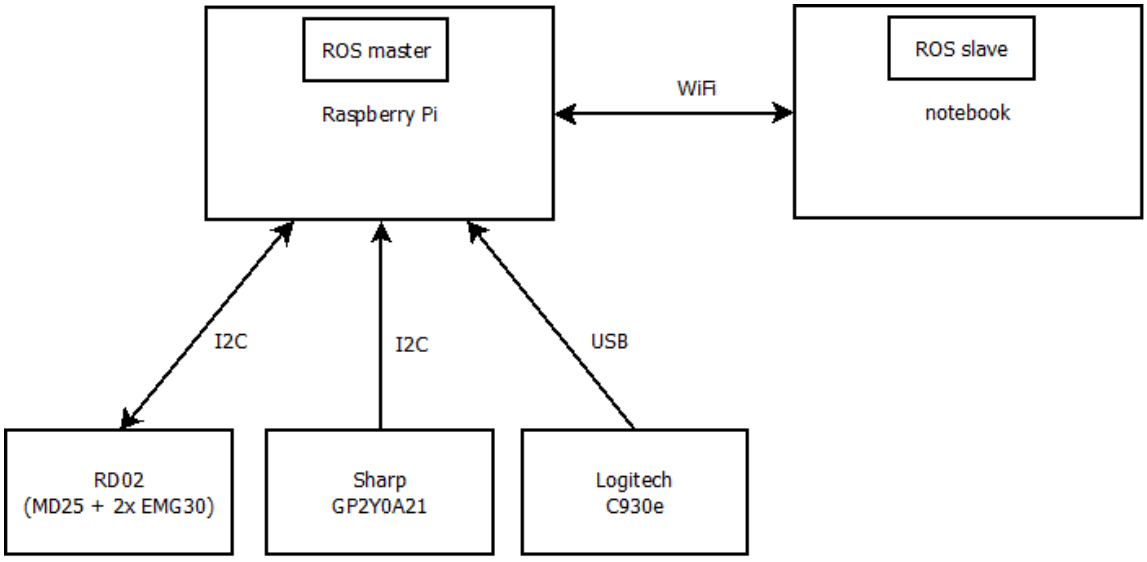
Obr. 8 Penn Quadrorotors při agresivní vzdušné akrobacii. [50]

3 NÁVRH ŘEŠENÍ

S ohledem na zadání úlohy, specifikace a možnosti ROSu bylo nutno vybrat odpovídající hardware a navrhnout software schopný řídit autonomní robot Leela (viz Obr. 9).

Obr. 9 Robot Leela.

V této kapitole jsou popsány a zdůvodněny jednotlivé komponenty nutné k realizaci řešení tohoto problému. Na Obr. 10 jsou znázorněny použité komponenty a komunikace mezi nimi.



Obr. 10 Komunikační graf komponent.

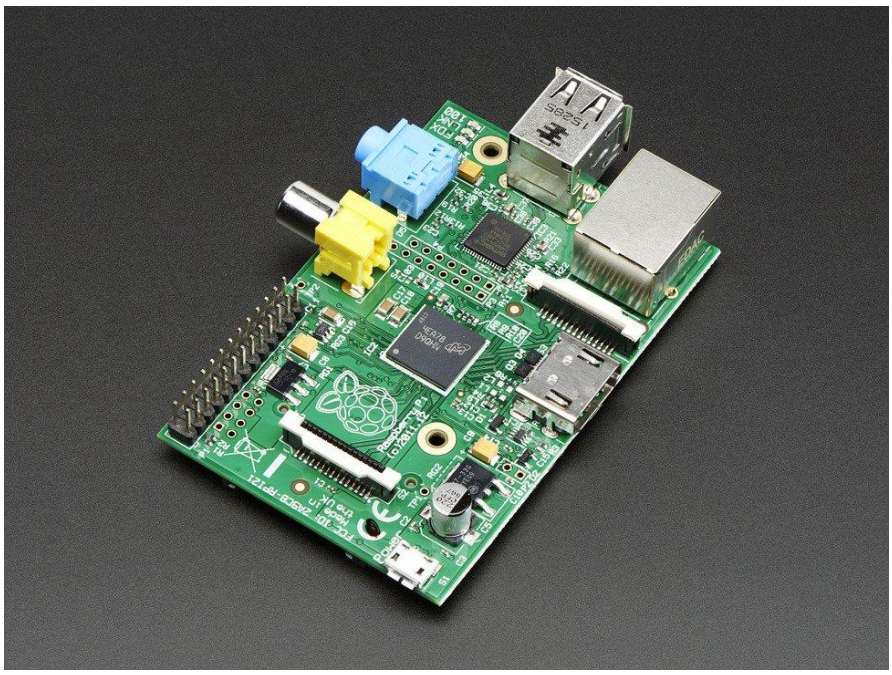
Dle zadání byl použit jednodeskový počítač Raspberry Pi pro hlavní sběr a interpretaci dat přímo na robotu. Jeho úkolem bylo získat data ze senzorů, která přes ROS posílal hlavní výpočetní jednotce přes síť WiFi. Současně přijímal přes WiFi příkazy navigace, které posílal řídící jednotce MD25 ovládající 2 motory EMG30. Ta obstarávala přes sběrnici I^2C řízení servomotorů pro pohyb a současně sběr dat z enkodérů pro lokální relativní určení polohy (odometrii). Pro určení globální polohy byla použita kamera snímající strop interiéru, vyhodnocovací software je předmětem disertační práce Ing. Michala Růžičky. Pro detekci překážek byly použity 4 infračervené senzory Sharp GP2Y0A21 směřující dopředu a do stran. Pro velkou náročnost výpočtu navigace a především globální lokalizace byla přidána další výpočetní jednotka schopna tyto složité výpočty realizovat v krátkém čase. Z důvodu snadné přenositelnosti a nezávislosti na hardware byl použit virtuální systém Ubuntu emulovaný programem VirtualBox na notebooku s hlavním operačním systémem Windows, který bylo nutno použít kvůli výpočtu globální lokalizace naprogramovanému pro systém Windows.

3.1 Raspberry Pi

Dle zadání byl použit jednodeskový počítač Raspberry Pi pro hlavní sběr a interpretaci dat přímo na robotu. Jeho úkolem bylo získat data ze senzorů, která přes ROS posílal hlavní výpočetní jednotce přes síť WiFi. Současně přijímal přes WiFi příkazy navigace, která posílal řídící jednotce motorů MD25.

Raspberry Pi byl vytvořen ve Velké Británii společností Raspberry Pi Foundation za účelem rozšíření výuky programování ve školách a vytvoření cenově dostupného

počítače pro širokou veřejnost. Původní myšlenka vznikla v roce 2006 a do roku 2008 bylo vytvořeno několik prototypů. Tou dobou na trh přišly dostupné dosti výkonné procesory pro mobilní zařízení, které umožnily vytvoření zařízení pro práci nejen s programovým kódem, ale i multimédii. Díky tomu projekt začal vypadat realizovatelně a zapojilo se do něj více lidí, čímž vznikla Raspberry Pi Foundation. V roce 2012 byly vyrobeny první kusy Raspberry Pi 1, které zanedlouho následovala sériová výroba a masivní úspěch Raspberry Pi. Od té doby bylo vytvořeno 8 různých verzí Raspberry Pi, v této práci byl použit Raspberry Pi Model B Rev. 2.0 (viz Obr. 11, dále pouze Raspberry Pi). [51]



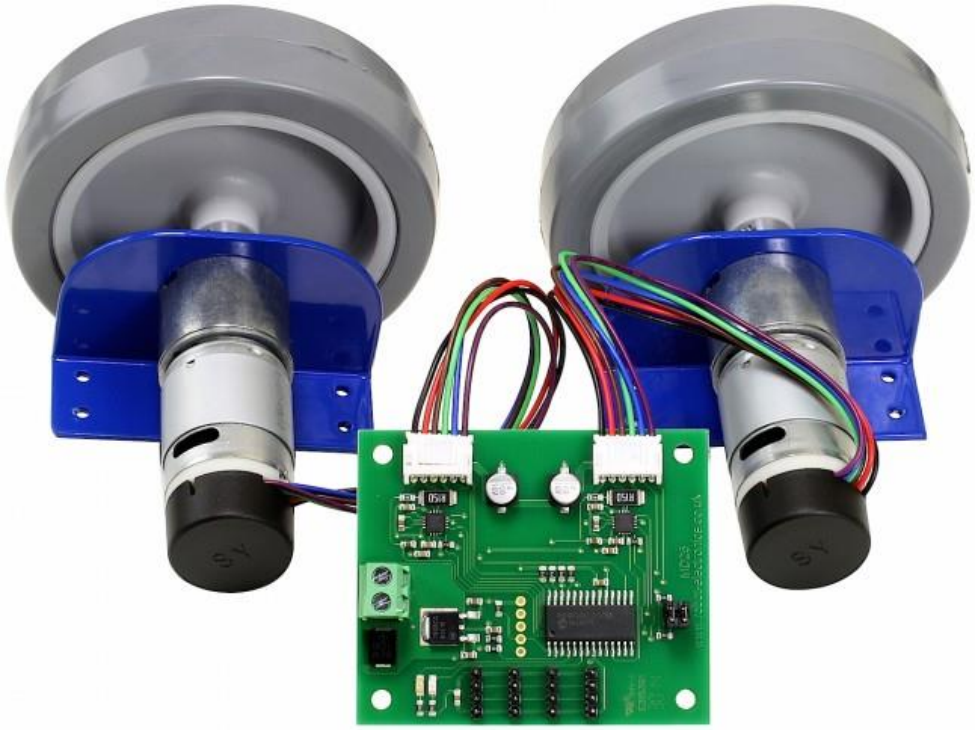
Obr. 11 Raspberry Pi Model B Rev. 2.0. [52]

Jádro tohoto jednodeskového počítače tvoří Broadcom BCM2835 SoC, který obsahuje ARM1176JZF-S 700 MHz procesor, grafickou kartu VideoCore IV a sdílených 512 MB RAM. Grafická karta pracuje na taktu 250 MHz, zvládá OpenGL ES 2.0 (24 GFLOPS), MPEG-2 a VC-1 a 1080p30 H.264/MPEG-4 AVC dekodér a kodér s vysokým profilem. Pro propojení s jinými zařízeními jsou ze 2 USB 2.0 porty a 100/10Mbs Ethernet adapter přes 3. USB port. Úložný prostor je řešen externí SD/MMC/SDIO kartou, v této práci bylo použita karta Kingston micro SDHC 8 GB class 10 s adaptérem Kingston micro SD. Rasperry Pi dále obsahuje video výstup HDMI (rev. 1.3 a 1.4) s rozlišením od 640×350 do 1920×1200 a 3.5 mm TRRS jack sdílený s výstupem zvuku pro kompozitní video, pro video vstup je implementován 15-pinový MIPI konektor kamerového rozhraní. Pro analogový audio výstup je zde 3,5 mm jack a digitání HDMI nebo I^2C a audio vstup přes GPIO I^2C. Pro nízkoúrovňové periferie je zde 8 GPIO pinů plus následující, které mohou být také využity jako GPIO: UART, I^2C sběrnice, SPI sběrnice se dvěma čipovými voliči, I2S audio +3.3 V, +5 V, GND a další 4 GPIO jsou dostupné na P5 padu po dodatečném připájení. Celá deska se napájí microUSB konektorem, kdy jmenovitý proud

je 700 mA. Raspberry Pi má rozměry 85,6 mm × 56,5 mm bez konektorů a váhu pouze 45 g. [51, 52, 53]

3.2 RD02 (MD25 a EMG30)

Pohonná souprava RD02 od společnosti Devantech (viz Obr. 12) v ceně přibližne 120 £ sestává ze 2 motorů EMG30, jejich řídící jednotky MD25, kol o průměru 100 mm a úchytů na ně. Je určená pro roboty do hmotnosti 5 kg. Celá souprava vyžaduje napájení o napětí 12 V a maximálním příkonu 6 A, kdy pro napájení logických obvodů je na desce 5 V regulátor. Pro napájení externích zařízení o napětí 5 V jsou připraveny 4 výstupy s kombinovaným maximálním skokovým proudem 1 A a stabilním prodem 300 mA. [54]

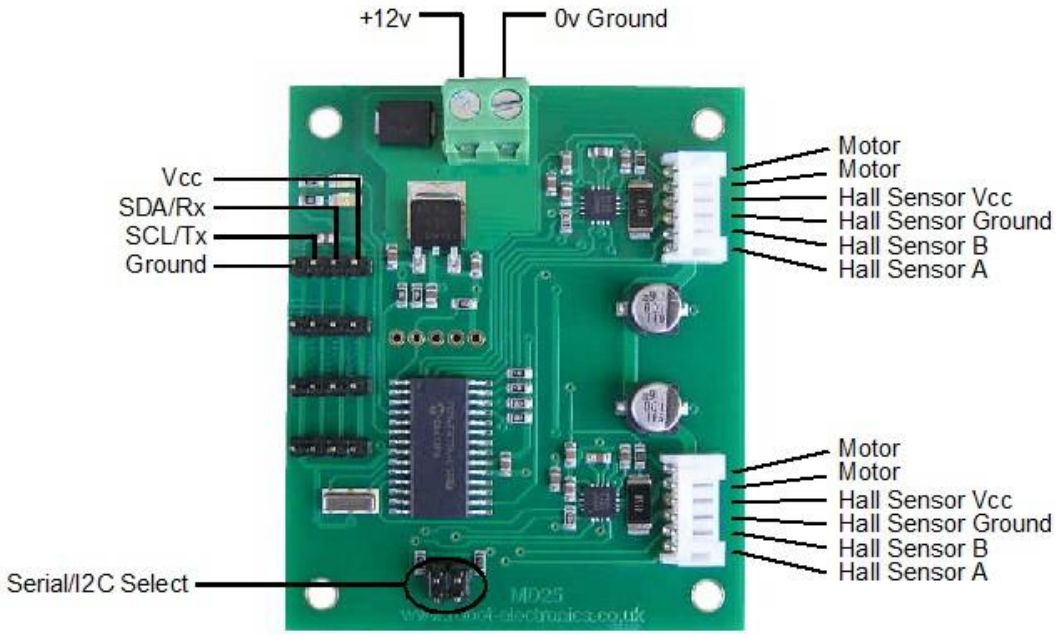


Obr. 12 Pohonná souprava RD02. [54]

Motory EMG30 (viz Obr. 13) se skládají z DC motorů, enkodérů pracujících na principu Hallova efektu a převodovky s redukčním poměrem 30:1. Tyto motory pohánějí pogumovaná kola o průměru 100 mm a šířkou 26 mm. Při napájení 12 V jsou schopny vyvinout točivý moment až 1,5 kg/cm a rychlost 170 otáček za minutu při zátěži. Bez zátěže jsou schopny maximální rychlosti 216 otáček za minutu. Spotřeba se pohybuje od 150 mA bez zátěže přes 530 mA při provozu až po 2,5 A při zaseknutí. Inkrementální ekodéry mají rozlišení 360 zubů na otočku kola. [54]

Obr. 13 Motor EMG30. [54]

Řídící jednotka motorů MD25 je schopna přes sériovou TTL nebo I^{2}C sběrnici řídit duální motory, především tedy motory EMG30, pro které je navrhnutá. Pro tyto motory deska obsahuje 10 nF kondenzátor pro potlačení šumu, kdy pro jiné motory by bylo nutno zajistit další potlačení šumu. Dle výrobce tento kondenzátor postačí pro potlačení šumu až do dvojnásobku použitého napětí. Systém je stavěný na 12 V vstupní napětí, ale zvládá výkyvy od 9 V do 14 V, kdy při nižším napětí než 9 V systém automaticky zastaví motory. Pro indikaci činnosti obsahuje deska 2 LED, červená svítí při napájení a zelená při komunikaci po sériové lince. Deska MD25 s vyznačenými konektory je na Obr. 14. [55]



Obr. 14 Řídící jednotka MD25 s popsanými konektory. [55]

Pro komunikaci s Raspberry Pi byla zvolena sběrnice I^{2}C z důvodu snadného připojení k samotnému Raspberry Pi a jednoduché programovatelnosti. MD25 komunikuje přes tuto sběrnici na adresách 0xB0 až 0xBE, kdy defaultní adresa je 0xB0, ale tuto adresu lze změnit v mezích stanoveného rozsahu pomocí odstranění adresového jumperu a poslání

sekvence 0xA0, 0xAA, 0xA5, 0xB4 (při změně na adresu 0xB4) na adresu 0xB0. Mezi každým příkazem musí být minimální pauza 5 ms. Změnu adresy potvrdí zelená LED dlouhým bliknutím následované 0 až 7 krátkými bliknutími, počet se odvíjí od čísla registru. Stejná sekvence blikání se objeví při každé inicializaci komunikace přes I^2C pro potvrzení adresy zařízení. Vlastní komunikace probíhá čtením nebo zápisem do registru na dané adrese zařízení, kdy MD25 obsahuje 17 registrů číslovaných 0 až 16. Tyto registry jsou popsány v Tabulka 6. [56]

Tabulka 6 Registry pro komunikaci přes I^2C. [56]

Register	Name	Read/Write	Description
0	Speed1	R/W	Motor1 speed (mode 0,1) or speed (mode 2,3)
1	Speed2/Turn	R/W	Motor2 speed (mode 0,1) or turn (mode 2,3)
2	Enc1a	Read only	Encoder 1 position, 1st byte (highest), capture count when read
3	Enc1b	Read only	Encoder 1 position, 2nd byte
4	Enc1c	Read only	Encoder 1 position, 3rd byte
5	Enc1d	Read only	Encoder 1 position, 4th (lowest byte)
6	Enc2a	Read only	Encoder 2 position, 1st byte (highest), capture count when read
7	Enc2b	Read only	Encoder 2 position, 2nd byte
8	Enc2c	Read only	Encoder 2 position, 3rd byte
9	Enc2d	Read only	Encoder 2 position, 4th byte (lowest byte)
10	Battery volts	Read only	The supply battery voltage
11	Motor 1 current	Read only	The current through motor 1
12	Motor 2 current	Read only	The current through motor 2
13	Software Revision	Read only	Software Revision Number
14	Acceleration rate	R/W	Optional Acceleration register
15	Mode	R/W	Mode of operation
16	Command	Write only	Used for reset of encoder counts and module address changes

Registry 0 a 1 slouží pro přímé ovládání pohybu. To se liší v závislosti na nastaveném módu (registr 15), kdy v módu 0 a 1 každý z registrů pro ovládání pohybu ovládá jedno kolo a v módu 2 a 3 slouží registr 0 pro ovládání rychlosti a registr 1 pro zatáčení. Sudé módy (0 a 2) navíc znamenají možnost zápisu hodnot 0 až 255, liché módy (1 a 3) hodnot -128 až 127. Nejnižší hodnota značí pro rychlost pohyb vzad nejvyšší rychlostí, střední (128 případně 0) zastavení a nejvyšší hodnota maximální rychlost vpřed. Mód 2 a 3 funguje na principu přičítání či odečítání hodnoty registru 1 od registru 0 a tím docílí zatočení. Program hlídá, aby daná hodnota nepřekročila limit, a pokud se to stane, automaticky upraví hodnotu druhou pro dosažení požadovaného rozdílu, tedy rychlosti zatočení. [56]

Registry 2 až 9 slouží pro čtení hodnot enkodérů, kdy hodnota každého enkodéru sestává ze 4 bytů. Hodnoty enkodérů lze resetovat zapsáním byto 0x20 do příkazového registru (registr 16). Tyto hodnoty MD25 čte a používá při defaultně zapnutém módu automatické regulace rychlosti. Tento mód znamená, že MD25 zvyšuje proud motorů dokud zpětná vazba z enkodérů nepotvrdí dosažení požadované rychlosti nebo se motory nenacházejí v maximálním výkonu. Tento mód lze vypnout zapsáním bytu 0x30 do příkazového registru (registr 16) případně znovu zapnout bytem 0x31. [56]

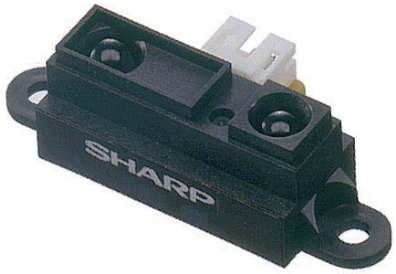
Registry 10 až 13 slouží jako informační registry poskytující informace o napětí baterie, proudu na motorech 1 a 2 a revizi software v tomto pořadí. [56]

Registr 13 ovládá rychlost akcelerace, která funguje na principu přidávání či ubírání rychlosti, kdy každý krok trvá 25ms. Počet kroků se vypočítá pomocí rozdílu současné a žádané rychlosti podělené hodnotou v tomto registru. Tato hodnota může být 1 až 10, kdy hodnota 1 znamená nejpomalejší zrychlení trvající 6,375 sekund pro změnu z maximální rychlosti vzad na maximální rychlost vpřed a hodnota 10 pouze 0,65 sekundy. Defaultní nastavená hodnota je 5, tedy maximální změna rychlosti trvá 1,275 sekundy. [56]

Posledním registrem je příkazový registr 16, který kromě již zmíněného resetování enkodérů, ovládání regulace rychlosti a nastavení adresy zařízení od verze software 2 také ovládá automatické zastavení motorů po 2 sekundách bez signálu. Toto je automaticky nastaveno, ale pro vypnutí stačí zapsat byte 0x32 případně pro znovuzapnutí byte 0x33. [56]

3.3 Senzory GP2Y0A21

Pro detetekci a měření vzdálenosti od překážek byly použity senzory GP2Y0A21 vyráběné firmou Sharp (viz Obr. 15). Tyto senzory fungují na principu emitace infračerveného světla pomocí IRED diody a detekce tohoto odraženého světla PSD detektorem. Díky použití triangulace není měření příliš ovlivněno odrazivostí objektu, teplotou okolí ani časem měření. Senzory jsou schopny měřit mezi vzdálenostmi 10 cm až 80 cm, hodnoty měření jsou předávány analogově napětím, kdy minimální napětí je 0,4 V při vzdálenosti objektu 80 cm a 2,3 V při 10 cm. Tyto hodnoty se dle výrobce mohou lišit až o 13% při maximální vzdálenosti a 37,5% při minimální. Tyto senzory potřebují napájecí napětí 4,5 V až 5,5 V a proud 0,3 mA s maximem 0,4 mA při 80 cm. [57]

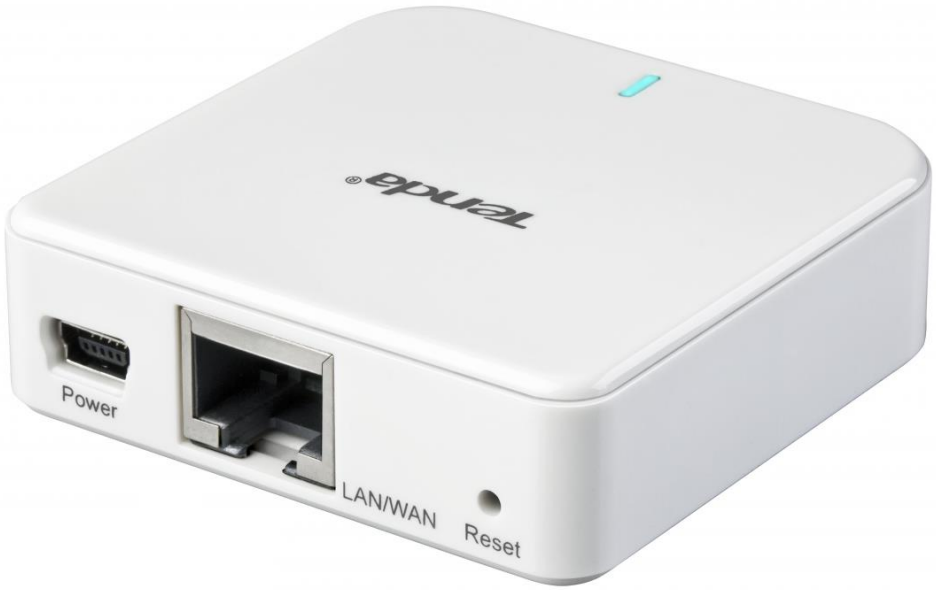


Obr. 15 Senzor Sharp GP2Y0A21. [57]

3.4 Tenda A6

Přenos dat mezi RasberryPi a notebookem byl realizován pomocí WiFi routeru Tenda A6 (viz Obr. 16 Router Tenda A6. . Tento router byl vybrán z důvodu malých rozměrů, relativně malé spotřeby a vyhovujícího napájecího napětí. Tento router je osazen chipsetem Broadcom BCM5356. Pracuje na frekvenci 2,4 Ghz se standardy 802.11b/g/n s maximální přenosovou rychlostí 150 Mb/s. Výstupní výkon je menší než 20 dBm (EIRP) a citlivost routeru je -87 dBm při 11 Mb/s, -72 dBm při 54 Mb/s a -68 dBmp při 150 Mb/s. Napájecí napětí je 5 V s 700 mA, tedý spotřeba pouze 3W. Tenda A6 má rozměry 56 x 56 x 17,3 mm. Pro připojení Raspberry Pi je zde 1 RJ45 LAN/WAN konektor. [58]

Tento router má několik režimů chodu: Acces Point, Wireless Router, WISP, Client a Universal Repeater. V režimu Acces Point jednotka vytváří vlastní WiFi síť s připojeným RJ45 konektorem. Wireless Router funguje jako klasický WiFi router, kdy jednotka zprostředkovává signál z portu RJ45 do vlastní WiFi sítě. WISP je pro připojení jednotky do stávající WiFi sítě a rozšíření tohoto signálu do stejné sítě bez možnosti připojení RJ45 kabelu. Režim Client funguje jako WiFi modul, kdy jednotka přijímá WiFi signál a distribuuje ho přes RJ45 port. Universal Repeater je režim, kdy jednotka se připojí do existující WiFi sítě a buď přímo nebo přes funkci WDS tento signál routuje do vlastní WiFi sítě a RJ45 portu. [58]



Obr. 16 Router Tenda A6. [58]

3.5 Kamera Logitech C930e

Pro získání globální polohy bylo potřeba zvolit kameru pro snímání stropu. Snímky z této kamery musely být v dostatečné kvalitě, aby algoritmus Ing. Michala Růžičky byl dostatečně přesný a eliminovaly se chyby. Z tohoto důvodu byla použita webkamera LogitechC930e (viz Obr. 17).

Tato kamera snímá video o 1080p při 30 snímcích za sekundu, rozlišení 3 MP a diagonální zorné pole 90°. Její rozměry jsou 94 x 24 x 29 mm a váha 162 g. [59]

Obr. 17 Kamera Logitech C930e. [59]

3.6 Výpočetní jednotka HP EliteBook 2540p

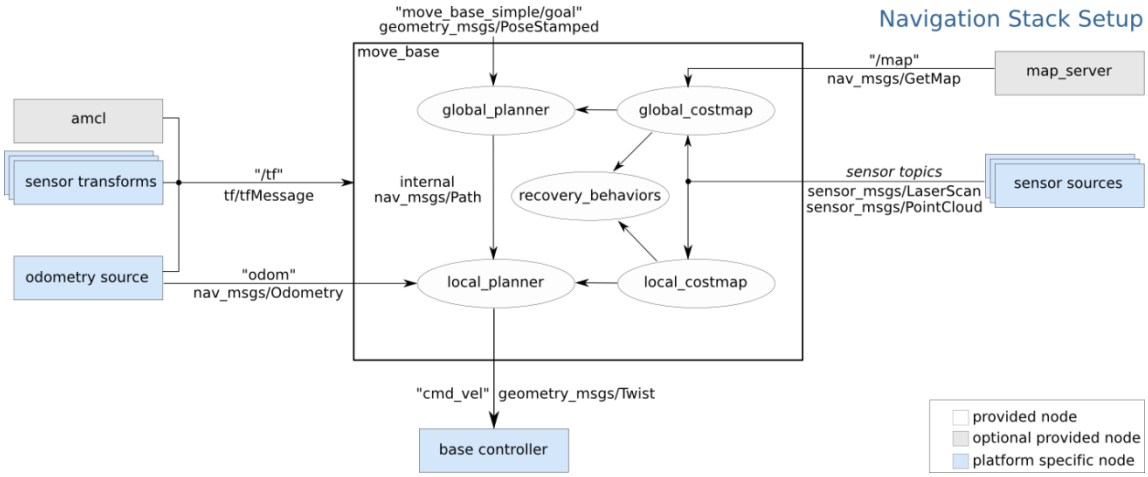
Hlavní výpočetní výkon poskytoval notebook firmy Hewlett Packard s označením EliteBook 2540p, který je dále v této práci je označován pouze jako notebook. Tento notebook je osazen procesorem Intel Core i7-620M s taktem jádra 2,66 GHz a 4 MB L3 cache, který díky technologii Intel Turbo Boost používá takt až do 3,33 GHz. V tomto notebooku je operační pamět 8192MB DDR3 PC3-10600 SDRAM s frekvencí 1333 Mhz, kdy ale paměťová sběrnice zvládá frekvenci nejvýše 1066 Mhz. Grafická karta je integrovaná Intel HD Graphics s dynamickou frekvencí, Microsoft DirectX verze 10.1 a se Shader Modelem 4.1. [60]

Na tomto notebooku je nainstalován operační systém Windows 7. Vzhledem k malé podpoře ROSu pro tento systém byl použit multiplatformí vizualizační nástroj VirtualBox, který emuloval systém Ubuntu 14.04 64 bit. Tento nástroj je vyvíjen společností Oracle jako Open Source s GNU GPL licencí verze 2. Podporuje systémy hostitele Widows,

Linux, Macintosh a Solaris a mnohem větší seznam hostů včetně Windows (verze NT 4.0 až Windows 10), DOS/Windows 3.x, Linux (verze jádra 2.4, 2.6, 3.x a 4.x), Solaris (i OpenSolaris), OS/2 a OpenBSD. Momentální nejnovější verze je 5.0, která byla použita v této práci. [61]

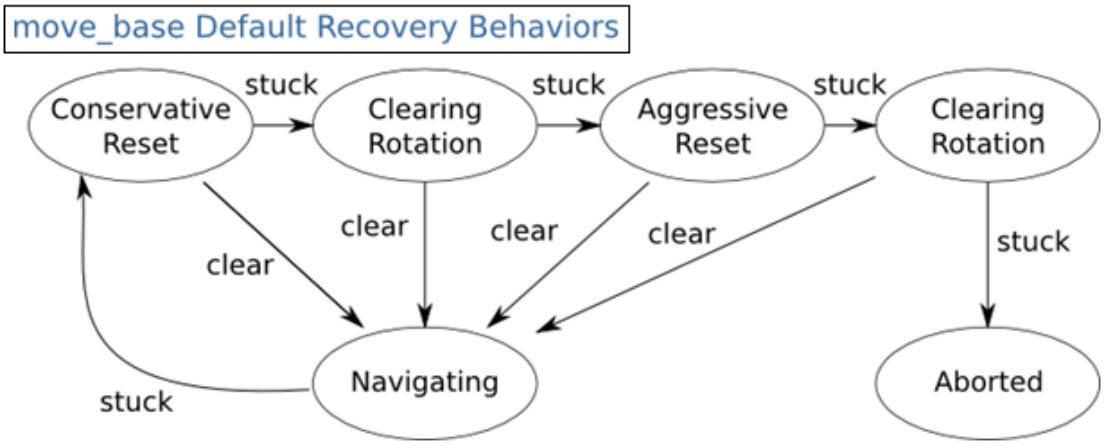
3.7 ROS Navigation Stack

Pro implementaci plánovacího modulu byl zvolen ROS Navigation Stack, který je implementací balíčku navigation s jádrem move_base z navigation. Pro jeho správné fungování je nutno robota nakonfigurovat velice specificky. Na Obr. 18 je grafické nastavení ROS Navigation Stack, aby správně fungoval. Bílé *nodes* jsou jeho součástí, modré *nodes* je nutno napsat na míru konkrétnímu robotu a šedé *nodes* jsou na uvážení uživatele, zda je použije nebo nahradí vlastními. Tyto *nodes* jsou map_server, který spravuje mapu, a amcl, který na základě *particle cloud* probabilisticky určí polohu robotu v 2D. [62]



Obr. 18 ROS Navigation Stack. [62]

Balíček move_base funguje na principu spolupráce lokálního a globálního plánovače a počítání lokální a globální *costmap*, tedy mapy překážek a míst, kam by se robot neměl dostat. Pokud robot uvízne, spustí se *recovery_behaviors*. Při tomto chování robot nejdříve odstraní překážky z mapy ve specifikovaném okolí a provede rotaci. Pokud problém přetrvává, provede to samé agresivněji, tedy odstraní všechny překážky mimo území rotace a provede znovu rotaci na místě. V případě, že toto problém nevyřeší, robot označí cíl jako nedosažitelný a přeruší navigaci. Toto chování je naznačeno na Obr. 19. [63]



Obr. 19 Základní nastavení recovery_behaviors. [63]

Výstupem move_base jsou příkazy pro pohonnou soustavu robotu na *topic* „cmd_vel“ ve formátu „geometry_msgs/Twist“, tedy 3D vektory pro lineární a úhlovou rychlost v metrech za sekundu, případně radiánech za sekundu. Vstupy jsou cíl navigace, mapa, data ze senzorů, data odometrie a transformace souřadnic. Cíl navigace je čten z *topic* „move_base_simple/goal“ ve formátu „geometry_msgs/PoseStamped“, který obsahuje časový údaj, bod v prostoru a quaternion reprezentující orientaci v prostoru. Dalším vstupem je mapa z *topic* „/map“ ve formátu „nav_msgs/GetMap“, který získá 2D mřížku, kde každá buňka reprezentuje pravděpodobnost obsazení překážkou. Move_base čte data ze senzorů z uživatelem definovaných *topics* ve formátech zprávy „sensor_msgs/LaserScan“, kde každá zpráva obsahuje jediný sken z rovinného skeneru, a „senzor_msgs/PointCloud“, která obsahuje sbírku 3D bodů a dalších informací o nich. Zpráva ve formátu „nav_msgs/Odometry“ na *topic* „odom“ obsahuje odhad pozice robotu a jeho rychlosti ve volném prostoru. Posledními vstupy jsou transformace souřadnic z *topic* „/tf“ ve formátech „tf/tfMessage“, které reprezentují tranformace souřadnic na základě transformačního stromu balíčku tf (viz 2.3.8). [63, 64, 65, 66]

4 REALIZACE

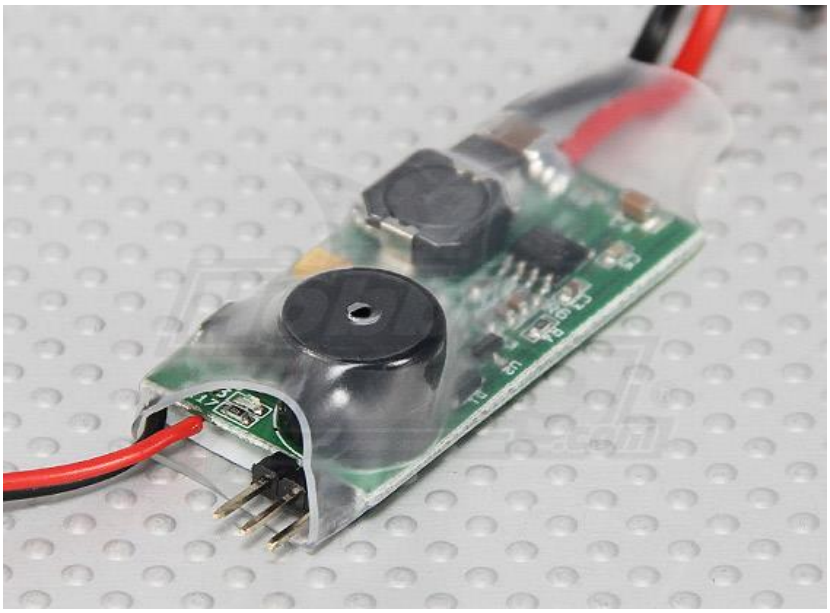
V této kapitole je podrobně popsán postup realizace řešení řídícího systému pro mobilní robot s použitím částí popsaných v kapitole 3 Návrh řešení.

4.1 Hardware

Jako první bylo nutno vyřešit hardware robotu, tedy napájení jednotlivých součástek a formu komunikace mezi nimi.

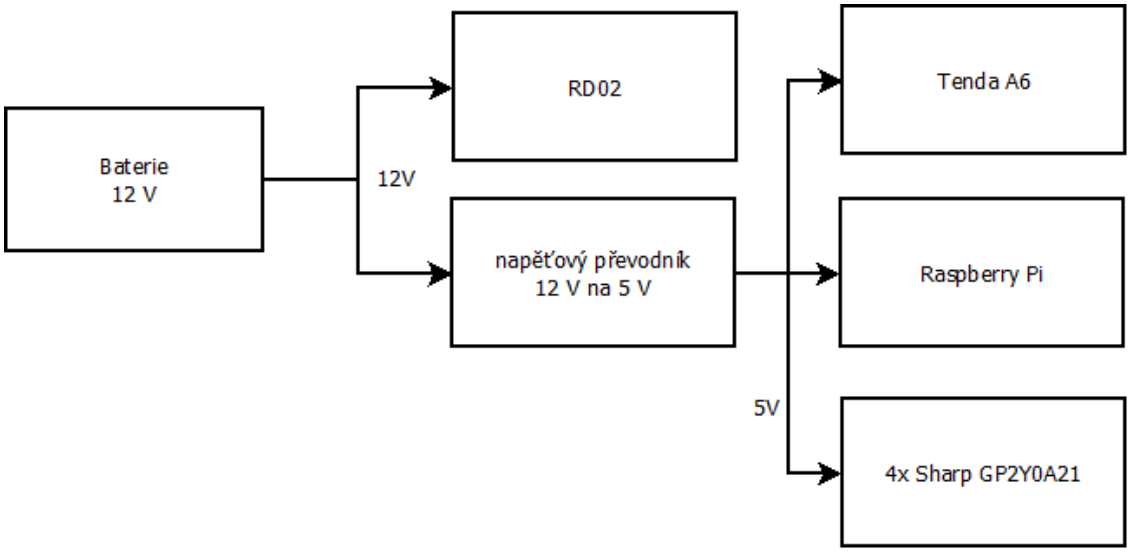
4.1.1 Napájení

Vzhledem k přepokládanému použití Li-Ion baterie o napětí 12 V bylo toto napětí použito jako základní, které bylo následně sníženo pro potřeby některých součástek na 5 V. Předpokládaný odběr těchto součástek byl 3 A po započítání rezervy, tedy výstup MD25 (s maximálním proudem 350 mA) byl nedostačující. Proto byl použit externí napěťový regulátor Turnigy 3A UBEC (viz Obr. 20) s maximálním proudem 3 A, který reguluje napětí z rozmezí 5,5 až 23 V na 5 nebo 6 V, podle polohy přepínače, s výstupním šumem menším než 50 mVp-p. Zařízení obsahuje ochranu proti přepětí i přehřátí a dosahuje účinnosti téměř 90%. Tento regulátor má navíc vestavěný bzučák pro případ poklesu vstupního napětí pod 6 V. [67]



Obr. 20 Napěťový regulátor Turnigy A3 UBEC. [68]

Dalším krokem bylo zapojení všech součástek podle schématu na Obr. 21 a otestování funkčnosti systému.

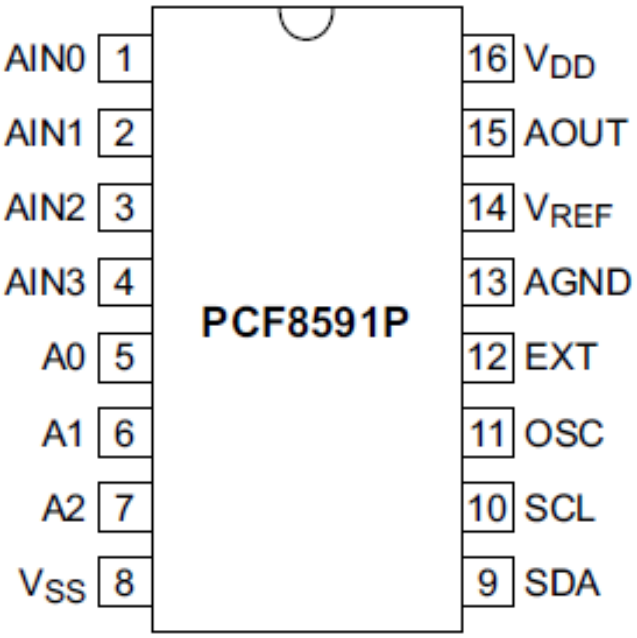


Obr. 21 Schéma elektrického zapojení součástí.

4.1.2 Senzory (A/D převodník)

Vzhledem k faktu, že Raspberry Pi nemá analogový vstup, musel být mezi senzory a Raspberry Pi připojen A/D převodník, jinak by senzory fungovaly na detekci pouze přítomnosti překážek v maximální vzdálenosti přibližně 20 cm. [57, 69]

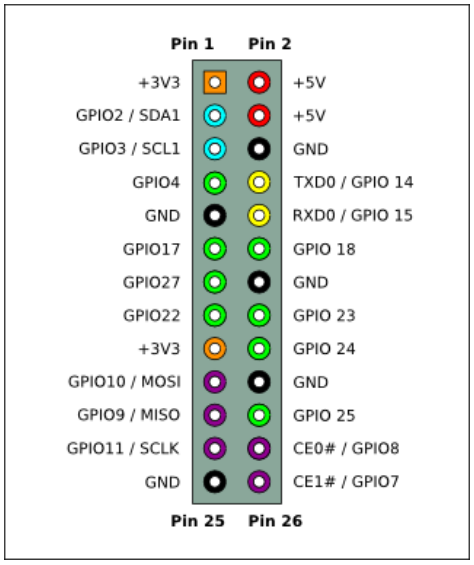
Vybrán byl NXP PCF8591T 8 bitový A/D převodník (viz Obr. 22) s napájecím napětím -0,5 až 8 V, který má 4 vstupy a podporuje až 8 I^2C zařízení. Připojení senzorů bylo proti směru hodinový ručiček, tedy senzor nejvíce vpravo (ve směru jízdy) byl připojen na pin 1 a senzor nejvíce vlevo na pin 4. [70]



Obr. 22 Piny A/D převodníku PFC8591P. [71]

4.1.3 Sběrnice I²C

Pro připojení RD02 a senzorů k Raspberry Pi byla použita sběrnice I²C. Základem této sběrnice jsou 2 kontakty (SCL a SDA), ale vždy je nutno mít i kontakt třetí (GND). SCL je hodinový signál pro synchronizaci datového toku, SDA jsou samotná data a GND je zem (0 V). RD02 (respektive MD25) má pozici pinů viz Obr. 14 v podkapitole 3.2 RD02 (MD25 a EMG30), piny pro připojení senzorů viz Obr. 22 v podkapitole 4.1.2 Senzory (A/D převodník). Sběrnici I²C lze k Raspberry Pi připojit na piny 3 (SDA), 5 (SCL) a například 6 (GND), viz Obr. 23. Raspberry Pi má vnitřně zapojené rezistory vůči zemi, tudíž I²C byla připojena přímo. [69, 72]



Obr. 23 Piny GPIO Raspberry Pi. [69]

4.2 Software

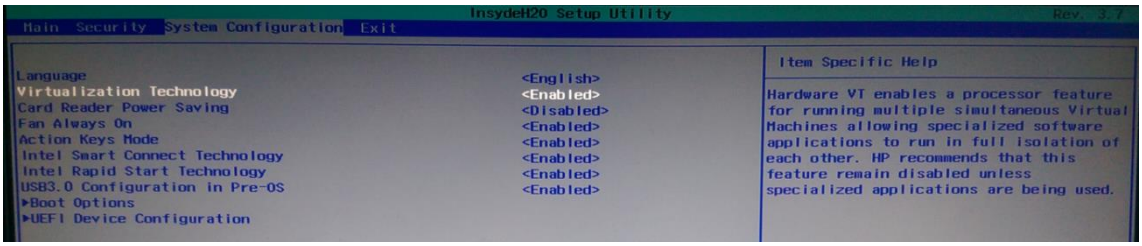
Dalším krokem bylo zprovoznění systému a vzájemné komunikace na softwarové úrovni. V této podkapitole je popis instalace operačního systému na Raspberry Pi a VirtualBox, instalaci ROSu na obou těchto systémech, nastavení sítě a nastavení ROSu pro vzájemnou spolupráci více zařízení. Následně se zde věnuji jednotlivým balíčkům nutným ke správnému fungování navigačního modulu ROSu.

4.2.1 Instalace operačních systémů

Prvním krokem bylo nainstalování virtuálního operačního systému na notebooku. Po nainstalováni programu VirtualBox, který je ke stažení na domovských stránkách tohoto programu [61], bylo potřeba stáhnout instalační obraz systému Ubuntu 14.04 (dále v práci pouze Ubuntu). Tato verze byla zvolena z důvodu stability a ověřené kompatibility s ROSem. Na oficiálních webových stránkách [73] je několik možností stáhnutí tohoto systému, pro potřeby této práce byla stažena kompletní instalace, aby nebylo nutno nastavovat síť pro VirtualBox před dokončením instalace operačního systému. Před

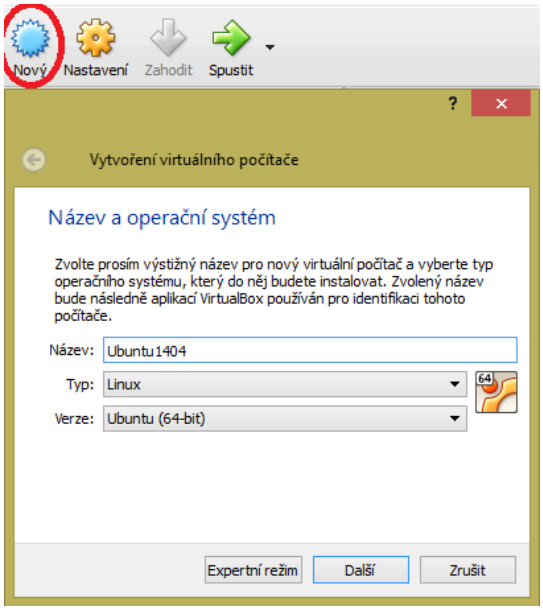
samotnou instalací bylo nutno povolit virtualizaci 64 bitového systému, jinak se v nabídce programu VirtualBox při vytvoření nového virtuálního počítače (Obr. 25) objeví pouze 32 bitové systémy. 64 bitový systém je nutný pro správné fungování 3D akcelerace pro vizualizaci pomocí balíčku rviz.

Pro povolení 64 bitové vizualizace musí procesor hostujícího počítače tuto funkci umožňovat. Po splnění této podmínky je nutno tuto funkci povolit, protože bývá v továrním nastavení zakázána. Tato změna se provádí v BIOSu hostujícího počítače, kde je nutno povolit funkci s názvem Virtualization Technology (viz Obr. 24). [74]



Obr. 24 Povoleni hardwarové virtualizace v BIOSu.

Nyní se již v menu VirtualBox při vytvoření nového virtuálního počítače (tlačítko „Nový“) objeví možnost 64 bitového systému. Toto menu se zvýrazněným tlačítkem „Nový“ a zadanými parametry (viz Obr. 25) je prvním krokem průvodce pro vytvoření nového virtuálního systému. Dalšími kroky jsou nastavení velikosti RAM, zde nastaveno 4096 MB, a vytvoření virtuálního pevného disku včetně formátu, který byl nastaven na .vdi a pevnou velikost 20 GB. VirtualBox umožňuje i dynamicky alokované disky, to s sebou přináší pouze úsporu zabraného místa na reálném disku a rychlejší vytvoření disku za cenu snížené rychlosti virtuálního počítače a možných problémů s daty.

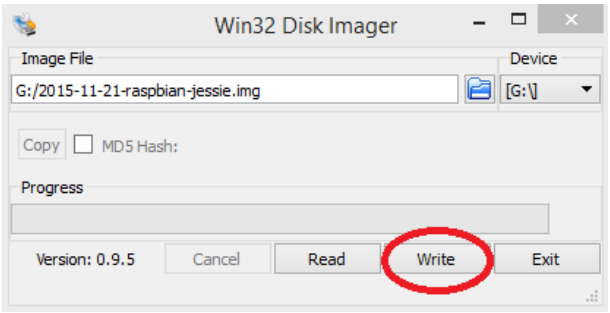


Obr. 25 Menu VirtualBox pro vytvoření nového virtuálního počítače.

Při spuštění nově vytvořeného virtuálního počítače se VirtualBox zeptá na pozici

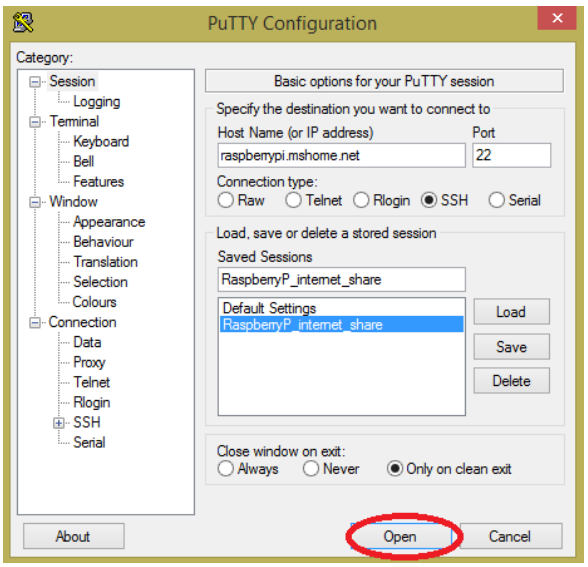
bootovacího disku. Zde je nutno najít cestu ke staženému disku Ubuntu a volbu potvrdit. Tím přistoupíme k instalaci operačního systému Ubuntu.

Po dokončení instalace Ubuntu bylo dalším krokem nainstalování operačního systému na Raspberry Pi. Pro tuto intalaci bylo nejprve nutno stáhnout obraz operačního systému Raspbian Jessie (plnou desktopovou verzi), který byl použit pro jeho vývoj přímo na jednodeskový počítač Raspberry Pi. Ke stažení je přímo na domovských stránkách Raspberry Pi [75]. Následně byl tento stažený obraz zapsán na SD kartu programem Win32DiskImager [76], který je na Obr. 26. V tomto programu je nutno najít cestu ke staženému obrazu Raspbianu, vybrat správné zařízení („Device“) a spustit zápis označeným tlačítkem „Write“.



Obr. 26 Program Win32DiskImager.

Dalším krokem je nastavení operačního systému Raspbian. SD karta s jeho obrazem byla vložena do Raspberry Pi, který byl UTP kabelem připojen k routeru poskytujícímu místní síť s internetem. Následně byl stažen program PuTTY [77], open source Telnet a SSH klient vytvořený Simonem Tathanem pro systém Windows. Tento program byl použit pro připojení k terminálu Raspberry Pi přes SSH. Na Obr. 27 je nastavení tohoto programu včetně IP adresy Raspberry Pi: *raspberrypi.mshome.net*. Spojení se vytvoří při stistku tlačítka „Open“.



Obr. 27 Nastaveni programu PuTTY.

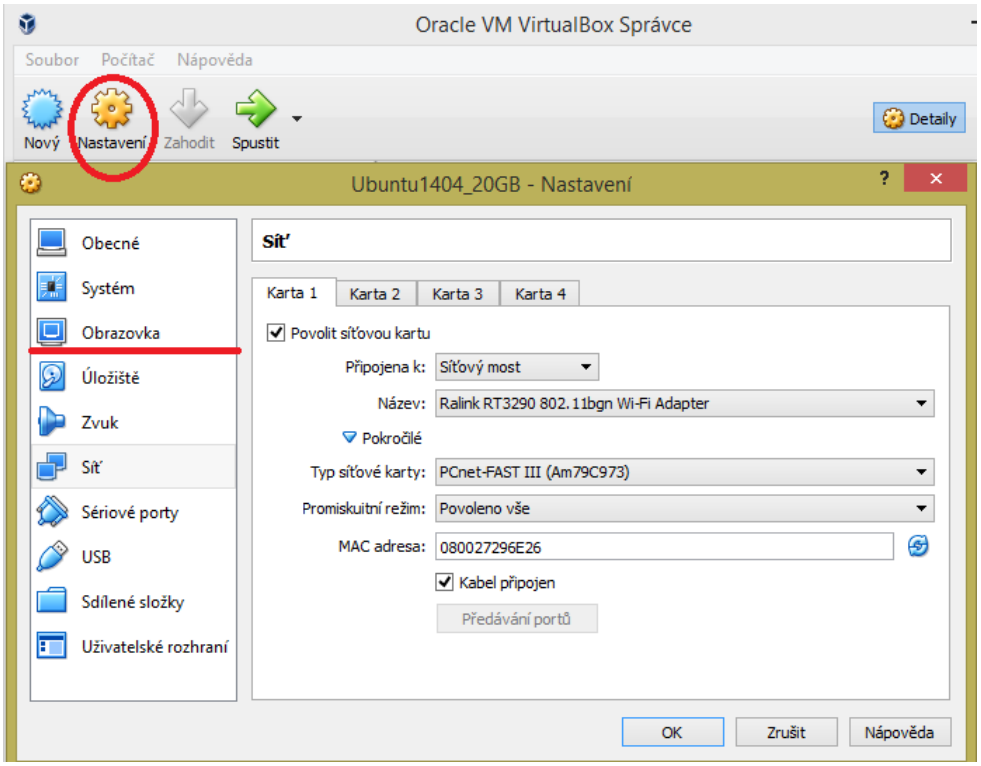
Při každém novém připojení přes SSH Raspberry Pi vyžaduje přístupové údaje (login a heslo). Tyto údaje jsou v Tabulka 7.

Tabulka 7 Defaultní přihlašovací údaje do Raspbianu. [78]

login	pi
heslo	raspberry

4.2.2 Instalace ROSu

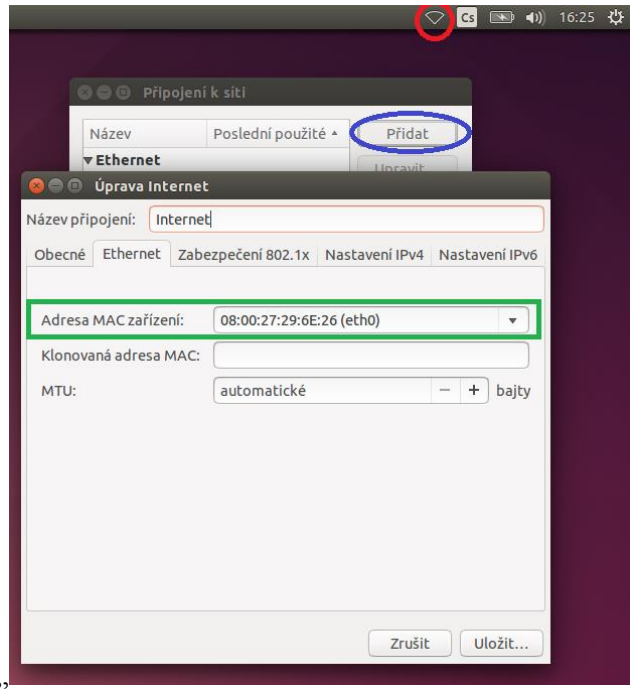
Prvním krokem při instalaci ROSu bylo nastavení připojení k internetu v programu VirtualBox v menu „Síť“ (viz Obr. 28). Zde bylo nutno nastavit režim „Síťový most“, který dovoluje Ubuntu obejít hostující systém a komunikovat přímo se síťovou kartou, což je nutné pro správné fungování takto navrženého systému řízení robotu. Další nutné nastavení je správný výběr síťového adaptéru, tedy WiFi karty. Při této příležitosti byla zvýšena video pamět virtuálního počítače z defaultních 8 MB na 128 MB pro rychlejší fungování systému. Toto nastavení bylo provedeno v menu „Obrazovka“, které je červeně podtrženo na Obr. 28.



Obr. 28 Nastavení sítě v programu VirtualBox.

Po tomto kroku zbývá nastavení připojení v samotném systému Ubuntu. Postup (viz Obr. 29) byl následující: klinutí pravým tlačítkem myši na ikonu sítě (označenou červeně), v otevřeném menu zvolit úplně poslední možnost („Upravit připojení...“), kliknout na tlačítko „Přidat“ (zakroužkované modře), zvolit možnost „Ethernet“ a potvrdit, změnit název připojení a zvolit „Adresu MAC zařízení“ na „XX:XX:XX:XX:XX:XX

(eth0)“ (v zeleně orámovaném poli). Po uložení se stačí k síti internet připojit pravým kliknutím na ikonu sítě a kliknutím na zvolený název sítě, zde „Internet“. Tímto je virtuální počítač připojen k místnímu routeru poskytující přístup k internetu.



„

Obr. 29 Nastavení sítě v systému Ubuntu.

Další postup probíhal téměr výhradně v terminálu, zkratka pro jeho otevření je CTRL + ALT + T. Pomocí příkazů v Tabulka 8 byly staženy soubory obsahující informace o nových verzích balíčků, tyto nové verze byly nainstalovány a restartován systém. Tyto příkazy nejsou nutné, ale novější verze balíčků systému Ubuntu většinou zajištují větší stabilitu.

Tabulka 8 Příkazy pro update systému.

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

Následně byl nainstalován ROS. Tato instalace proběhla pomocí postupu na ROS Wiki [79] s několika úpravami, proto je zde kompletní postup. Tato instalace je relativně rychlá, protože spočívá v instalování hotových balíčku pro tento systém, tedy to není instalace ze zdrojových souborů.

Nejprve bylo nutno přidat zdroj do sources.list, nastavit klíče a obnovit seznam balíčků příkazy v Tabulka 9.

Tabulka 9 Příkazy pro nastavení zdrojů a získání klíčů.

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list'
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
$ sudo apt-get update
```

Poté došlo na instalaci ROSu, který je předkompilovaný v několika instalačních metabalíčcích. V této práci byl nainstalován desktop-full, který obsahuje jádro ROSu, rqt, rviz, robot-generic knihovny, 2D/3D simulátory, navigaci a 2D/3D vnímání. Navíc byly nainstalovány balíčky gmapping, map-server, move-base a amcl. Příkazy jsou v Tabulka 10.

Tabulka 10 Příkazy pro nainstalování ROSu.

```
$ sudo apt-get install ros-jade-desktop-full
$ sudo apt-get install ros-jade-gmapping ros-jade-map-server ros-jade-move-base ros-jade-amcl
```

Následně bylo nutno inicializovat rosdep (viz Tabulka 11), který dovoluje jednoduchou instalaci dalších systémových závislostí a je nutný pro běh základních komponent ROSu.

Tabulka 11 Příkazy pro inicializaci rosdepu.

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

Nyní zbývalo pouze nastavit prostředí systému pro jednodušší práci s ROSem. Příkazy v Tabulka 12 slouží k zakomponování proměnných ROSu do pracovního prostředí, tedy tyto proměnné se automaticky nastaví při každém spuštění terminálu.

Tabulka 12 Příkazy pro nastavení prostředí.

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

Posledním krokem v nastavení prostředí je vytvoření ROS Workspace pro práci a tvorbu balíčků ROSu. Příkazy v Tabulka 13 vytvoří prostředí pro práci s balíčkovacím systémem ROSu catkin, toto prostředí vybudují pro ověření funkčnosti a poslední příkaz přidá systémovou proměnnou, aby ROS našel cestu ke zde vytvořeným balíčkům. V posledním příkazu bylo nutno zaměnit „USER“ za jméno uživatele systému Ubuntu. [80]

Tabulka 13 Příkazy pro vytvoření ROS Workspace.

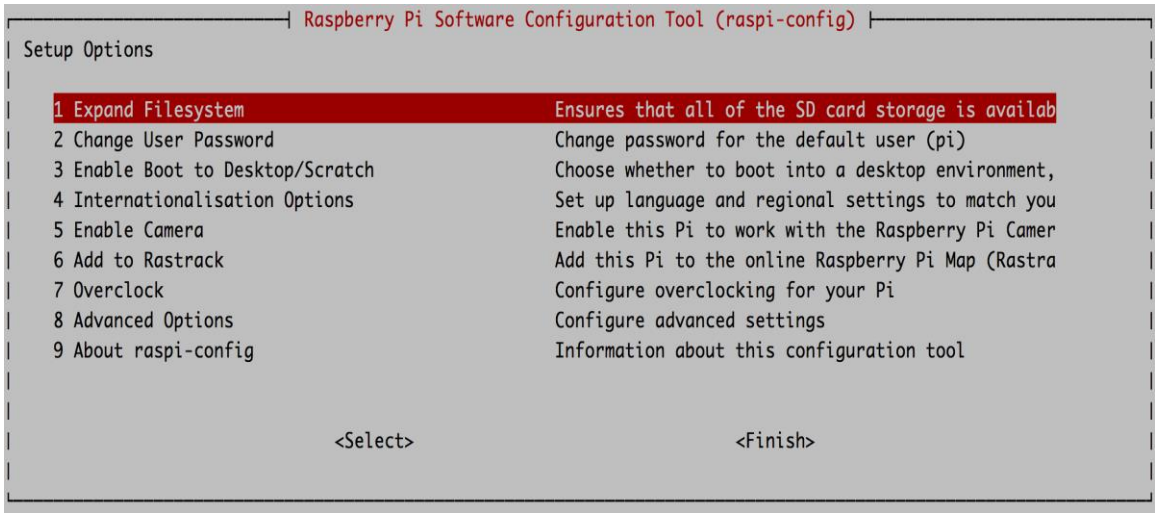
```
$ mkdir -p ~/catkin_ws/src
$ cd ~/catkin_ws/src
$ catkin_init_workspace
$ cd ~/catkin_ws/
$ catkin_make
$ source devel/setup.bash
$ echo $ROS_PACKAGE_PATH
/home/USER/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indigo/stacks
```

Tímto byl nainstalován ROS na notebooku a bylo potřeba ROS nainstalovat i na Raspberry Pi. Zde je instalace trochu komplikovanější a vzhledem k výkonu Raspberry Pi trvá podstatně déle, až 36 hodin.

Po připojení k Raspberry Pi přes SSH a přihlášení byly také spuštěny příkazy v Tabulka 8. Poté bylo nutno zvětšit obraz systému, aby vyplnil celou kartu SD a povolit I2C sběrnici. Toho lze nejjednodušeji docílit příkazem v Tabulka 14. Po jeho zadání se otevře menu konfigurace Rasperry Pi (viz Obr. 30), kde byl použit příkaz 1: „Expand Filesystem“, který zpřístupní celou velikost karty SD, a poté pod příkazem 8: „Advanced Options“ byl spuštěn příkaz A7: „I2C“ pro povolení automatického nahrávání modulu I2C do kernelu.

Tabulka 14 Příkaz pro nastavení Raspberry Pi.

```
$ sudo raspi-config
```



Obr. 30 Nastavení raspi-cofig.

Následně byl na Raspberry Pi nainstalován ROS s využitím postupu na ROS Wiki [81]. Tato instalace se liší od předchozí, protože jde o instalaci ze zdroje a tedy kompletní

kompilace přimo na Raspberry Pi. Navíc není nutno instalovat všechny balíčky jako na notebooku, protože na Raspberry nebudou používány a zbytečně by zabíraly místo.

Prvním krokem je instalace závislostí, příkazy viz Tabulka 15. Následovala inicializace rosdep (viz Tabulka 11).

Tabulka 15 Příkazy pro instalaci závislostí pro instalaci ROSu.

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

Poté přišlo na řadu samotné kompilování balíčků ROSu programem catkin. To probíhalo v ROS Workspace, tedy bylo nutno toto prostředí vytvořit příkazy z Tabulka 16.

Tabulka 16 Vytvoření ROS Workspace pro kompilaci ROSu.

```
$ mkdir ~/ros_catkin_ws
$ cd ~/ros_catkin_ws
```

Následovalo stažení základních a dodatečných balíčků nutných pro fungování navrženého systému (příkaz viz Tabulka 17).

Tabulka 17 Příkazy pro stažení balíčků ROSu ze zdroje.

```
$ rosinstall_generator desktop geometry common_msgs --
rosdistro indigo --deps --wet-only --tar > indigo-
custom_ros.rosinstall
$ wstool init -j8 src indigo-custom-ros.rosinstall
```

Před samotným vybudováním stažených balíčků je nutno vyřešit jejich závislosti. K tomu slouží příkaz v Tabulka 18, tedy k nalezení všech závislostí stažených balíčků a jejich rekurzivní instalaci. Tento krok zpravidla trvá nejdéle.

Tabulka 18 Příkaz pro řešení závislostí.

```
$ rosdep install --from-paths src --ignore-src --rosdistro
indigo -y
```

Posledním krokem instalace je vybudování připravených balíčků a vytvoření cesty k nim. Tyto příkazy jsou v Tabulka 19. Poté pro používání ROSu stačí vytvořit catkin ROS Workspace použitím příkazů v Tabulka 13.

Tabulka 19 Příkazy pro vybudování ROSu ze zdroje.

```
$ ./src/catkin/bin/catkin_make_isolated --install -
DCMAKE_BUILD_TYPE=Release
$ echo "source ~/ros_catkin_ws/install_isolated/setup.bash"
```

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

4.2.3 Synchronizace času

Nyní byl ROS nainstalován a připraven k použití na obou zařízeních a bylo třeba na obouch zařízeních sesynchronizovat čas a tím umožnit vzájemnou spolupráci a předávání informací ROSem. K tomuto účelu byl použit program chrony. Tento program je implementací protokolu NTP, kdy pro účely synchronizace času mezi 2 zařízeními na LAN dosahuje přesnosti do několika setin mikrosekund. Chrony podporuje systémy Linux, FreeBSD, NetBSD, Mac Os X a Solaris. Je publikován pod licencí GNU GPL verze 2. [82]

Nejprve bylo nutné program chrony nainstalovat prvním příkazem v Tabulka 20. Poté změnit nastavení tohoto programu v jeho konfiguračním souboru druhým příkazem v Tabulka 20. Změna tohoto nastavení se liší v závislosti na faktu, zde se jedná o server nebo klient. Jako server byl zvolen notebook a klient Raspberry Pi. Pro server bylo nutno smazat všechny řádky začínající „allow“ a nahradit je jediným řádkem „allow 192.168.2“. Další změna u serveru je přidání řádku se slovem „manual“ na konec souboru. U klienta bylo potřeba také smazat všechny řádky začínající „allow“ a nahradit je jediným řádkem, který ovšem zní „allow 192.168.2.103“. Navíc zde bylo potřeba smazat řádky se „server“ a nahradit je "server 192.168.2.103". Toto nastavení zatím nebude fungovat bez správně nastavené sítě.

Tabulka 20 Příkaz pro instalaci a nastavení programu chrony.

```
$ sudo apt-get install chrony
$ nano /etc/chrony/chrony.conf
```

4.2.4 Nastavení vzájemné komunikace po síti

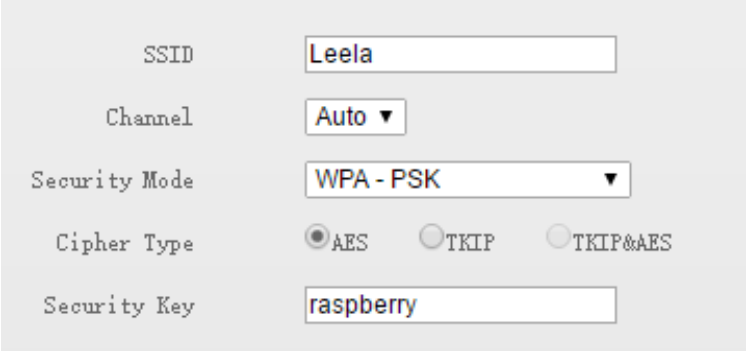
Nyní byly nainstalovány všechny potřebné programy, a proto nebylo potřeba připojení k internetu. To umožnilo spojení notebooku a Raspberry Pi přes router Tenda A6. Před samotným fyzickým připojením Raspberry Pi k routeru bylo nutno Raspberry Pi i router nakonfigurovat. Jakákoliv chyba v této konfiguraci znamená neschopnost připojit se k Raspberry Pi přes SSH a nutnost připojení externího monitoru, klávesnice a myši k Raspberry Pi.

Pro nastavení Raspberry Pi stačí zadat příkazy z Tabulka 21. První příkaz provede to, že po každém spuštění Raspberry Pi se mu nastaví statická IP adresa. Tato IP adresa je v podsíti routeru Tenda A6 a bez zpětné konfigurace se Raspberry Pi už nepřipojí k momentálně připojenému routeru. Druhý příkaz Raspberry Pi restartuje.

Tabulka 21 Příkazy pro nastavení statické IP na Raspberry Pi.

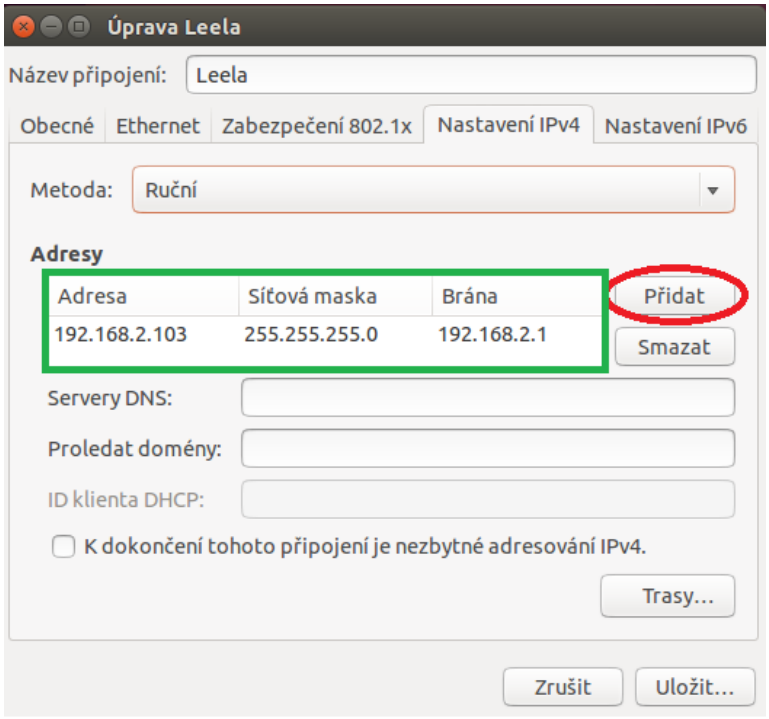
```
$ sudo echo "sudo ifconfig eth0 192.168.2.101" >>
/etc/rc.local
$ sudo reboot
```

Nyní bylo Raspberry Pi odpojeno od routeru poskytujícího internet. Router Tenda A6 byl UTP kabelem připojen k notebooku a nakonfigurován. Z možných režimů chodu byl použit Access Point, v nastavení routeru označen „Wireless AP“. Jeho nastavení je na Obr. 31.



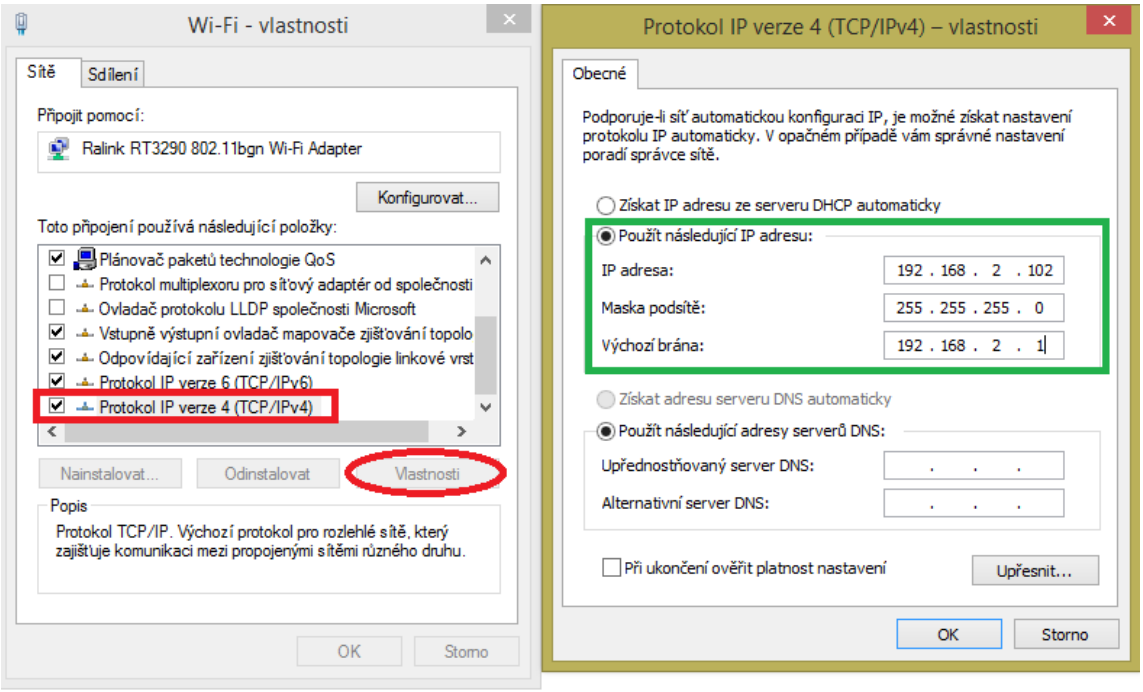
Obr. 31 Konfigurace routeru Tenda A6.

Po úspěšném nakonfigurování routeru Tenda A6 i Raspberry Pi byla tato zařízení spojena kabelem UTP a zbývalo nakonfigurovat a připojit do sítě notebook. Toho bylo docíleno vytvořením sítě jako v kapitole 4.2.1 Instalace operačních systémů s tím rozdílem, že byla nastavena statická IP adresa (viz Obr. 32). Toho bylo docíleno změnou v záložce „Nastavení IPv4“: kliknutím na tlačítko „Přidat“ (označené červeně) a vyplněním údajů specifikovaných v zeleném rámečku.



Obr. 32 Nastavení statické IP adresy v Ubuntu.

Pro správné připojení k síti bylo nutné nakonfigurovat i adaptér hostujícího systému Windows, viz Obr. 33. Nyní stačilo připojit notebook k síti Tenda A6 a ověřit konfiguraci příkazy *ping*, v Tabulka 22 je první příkaz pro notebook a druhý pro Raspberry Pi.



Obr. 33 Nastavení sítě hostujícího systému Windows.

Tabulka 22 Ověření konfigurace sítě.

```
$ ping 192.168.2.101
$ ping 192.168.2.103
```

Když byly oba počítače schopny vzájemně komunikovat po síti, bylo potřeba nastavit ROS na obou počítačích tak, aby na Raspberry Pi bežel *Master* a ROS na notebooku využíval služby tohoto *Master*. V ROS terminologii se používá označení „master“ a „slave“, tedy ROS na Raspberry Pi je „master“ a ROS na notebooku je „slave“. Toto nastavení bylo provedeno v souboru .bashrc v domovské složce. Příkazy v Tabulka 23 byly provedeny na notebooku a příkazy v Tabulka 24 na Raspberry Pi. [83]

Tabulka 23 Příkaz pro úpravu souboru .bashrc pro „slave“.

```
$ echo "export ROS_HOSTNAME=192.168.2.103" >> ~/.bashrc
$ echo "export ROS_MASTER_URI=http://192.168.2.101:11311" >>
~/.bashrc
```

Tabulka 24 Příkazy pro úpravu souboru .bashrc pro „master“

```
$ echo "export ROS_HOSTNAME=192.168.2.101" >> ~/.bashrc
$ echo "export ROS_MASTER_URI=http://192.168.2.101:11311" >>
~/.bashrc
```

4.2.5 Implementace ROS Navigation Stack

Dalším krokem bylo splnění všech podmínek pro správné spuštění plánování trasy. Tyto kroky jsou popsány v tutoriálu „Robot Setup“ viz [62]. V této práci bylo nejdříve vyřešeno čtení dat z odometrie a ovládání pohybu, tedy komunikace Raspberry PI s MD25 přes I^2C. Pro jednodušší práci s touto sběrnicí byly na Raspberry Pi nainstalovány balíky python-smbus a i2c-tools, příkazy jsou v Tabulka 25.

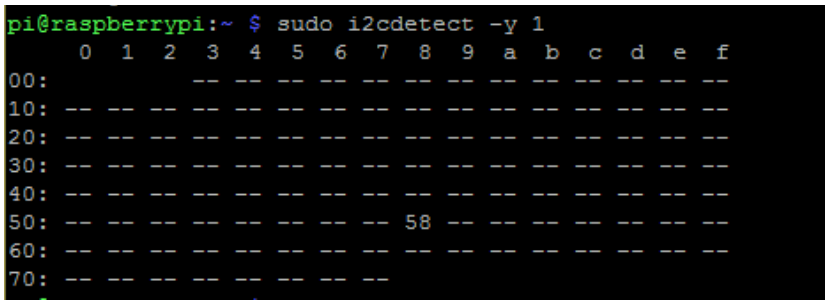
Tabulka 25 Příkazy pro instalaci balíků python-smbus a i2c-tools.

```
$ sudo apt-get install python-smbus
$ sudo apt-get install i2c-tools
```

Poté byly zjištěny hardwarové adresy zařízení připojených přes sběrnici I^2C příkazem v Tabulka 26, kdy umístění sběrnice I^2C v Raspberry Pi je /dev/i2c-1. Pro tento krok byly dočasně odpojeny senzory pro snadnější identifikaci. Adresa MD25 byla 58 (viz Obr. 34). Stejným způsobem byly zjištěny adresy senzorů po jejich opětovném připojení.

Tabulka 26 Příkaz pro detekci adres na sběrnici I²C.

```
$ sudo i2cdetect -y 1
```



Obr. 34 Adresa MD25 na sběrnici I²C.

Nyní byl vytvořen balíček s názvem „leela“ pro všechny nově napsané programy na Raspberry Pi příkazem v Tabulka 27. Tento balíček závisí na balíčcích message_generation, roscpp, rospy, std_msgs a geometry_msgs.

Tabulka 27 Vytvoření balíčku „leela."

```
$ catkin_create_pkg leela message_generation roscpp rospy
std_msgs geometry_msgs
```

V tomto balíčku byl vytvořen soubor src/enkodery_publish.py, který získával data z enkodérů přes sběrnici I²C, přepočítal je na formát „nav_msgs/Odometry“ a publikoval je na *topic* „odom“. Zpráva formátu „nav_msgs/Odometry“ se skládá z hlavičky formátu „Header“, řetězce formátu „string“ specifikující podřízený rám (zde „base_link“ vůči rodiči „odom“ specifikovaným v hlavičce), pozici ve formátu „geometry_msgs/PoseWithCovariance“ a pohyb ve formátu „geometry_msgs/TwistWithCovariance“. Současně na základě těchto dat publikoval transformaci souřadnic rámu „base_link“ vůči rámu „odom“. [84]

Následně bylo nutno publikovat data ze senzorů ve vyhovujícím formátu. Vzhledem k použití IR senzorů byl použit formát zprávy „sensor_msgs/PointCloud“. Zpráva formátu „sensor_msgs/PointCloud“ se skládá z hlavičky formátu „Header“, pole 3D bodů ve formátu „geometry_msgs/Point32[]“ a datových kanálů formátu "ChannelFloat32[]“, kde každý kanál má stejný počet prvků jako je bodů. Tyto kanály slouží například k určení barvy bodů pro zobrazení. Pro každý senzor byl vytvořen vlastní soubor, názvy byly zvoleny následovně: ir1.py, ir2.py, ir3.py a ir4.py. Tyto soubory nejen publikovaly zprávu na topic /scan ve výše zmíněném formátu, ale současně publikovaly i transformaci souřadnic daného senzoru vůči rámu „base_link“. [85]

Posledním potřebným programem byl interpret pohybových příkazů, který byl vytvořen pod názvem „listen_nod.py“. Tento program četl data z *topic* „cmd_vel“ ve formátu „geometry_msgs/Twist“ a přepočetl je na rozmezí 0 až 255, které následně poslal přes I²C DC motorům.

Pro snadnější spouštění těchto programů byl vytvořen balíček leela_2dnav (příkaz viz Tabulka 28), který závisel na balíčku leela a obsahoval soubor leela_configuration.launch, jehož obsah je v Tabulka 29. Tento spouštěcí soubor spouštěl výše zmíněné programy najednou v jediném terminálu a tím velmi zjednodušil spouštění celého systému.

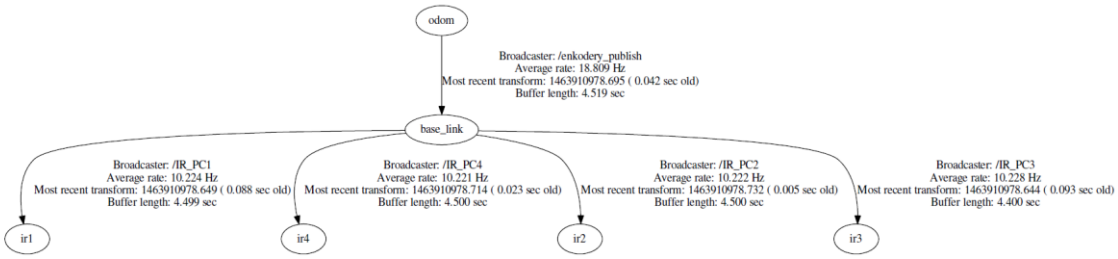
Tabulka 28 Vytvoření balíčku „leela_2dnav“.

```
$ catkin_create_pkg leela_2dnav leela
```

Tabulka 29 Obsah souboru leela_configuration.launch.

```
<launch>
      <node pkg="leela" type="ir1.py" name="IR_PC1" out-
put="screen" />
      <node pkg="leela" type="ir2.py" name="IR_PC2" out-
put="screen" />
      <node pkg="leela" type="ir3.py" name="IR_PC3" out-
put="screen" />
      <node pkg="leela" type="ir4.py" name="IR_PC4" out-
put="screen" />
      <node pkg="leela" type="listen_nod.py"
name="motor_drive" output="screen" />
      <node pkg="leela" type="enkodery_publish.py"
name="enkodery_publish" output="screen" />
</launch>
```

Implementací těchto částí byl současně nastaven transformační strom. Jeho vzhled v současném stádiu (získaný příkazem z Tabulka 5 z podkapitoly 2.3.8 Balíček tf) je na Obr. 35.



Obr. 35 Průběžný vzhled transformačního stromu.

Posledním programovatelným vstupem bylo nastavení mapového serveru pro definování prostoru pohybu robotu. Pro tento účel byl vytvořen balíček s názvem „mapovy_server“, který vygenerovanou mapu místnosti ve formátu .yaml. Tento balíček

závisí na balíčku map_server, příkaz pro jeho vytvoření je v Tabulka 30.

Tabulka 30 Příkaz pro vytvoření balíčku „mapovy_server“.

```
$ catkin_create_pkg mapovy_server map_server
```

Po vytvoření a nastavení všech nutných balíčků specifických pro robot přišla na řadu konfigurace samotné navigace. Tento balíček byl vytvořen příkazem v Tabulka 31 s názvem „leela_2dnav“. Leela_2dnav také obsahuje konfigurační soubory pro lokální i globální *costmap* a balíček base_local_planner, který je součástí balíčku navigation a stará se o výpočet rychlostí robotu.

Tabulka 31 Příkaz pro vytvoření balíčku navigace.

```
$ catkin_create_pkg leela_2dnav move_base
my_tf_configuration_dep my_odom_configuration_dep
my_sensor_configuration_dep
```

Soubor „costmap_common_params.yaml“ (viz Tabulka 32) obsahuje společné parametry pro lokální i globální *costmap*. Mezi tyto parametry patří vzdálenost překážek („obstacle_range“), maximální vzdálenost pro detekci překážek („raytrace_range“), půdorys robotu specifikovaný jako vrcholy mnohoúhelníku („footprint“) nebo poloměr kruhu („robot_radius“), minimální vzdálenost robotu od překážek („inflation_radius“) a specifikace senzorů. Jednotky jsou metry. [62]

Tabulka 32 Obsah souboru costmap_common_params.yaml.

```
obstacle_range: 0.7
raytrace_range: 0.8
footprint: [[0.1, 0.15], [0.2, 0.0], [0.1, -0.15], [-0.3, -
0.15], [-0.3, 0.15]]
#robot_radius: 0.3
inflation_radius: 0.3

observation_sources: IR_PC1, IR_PC2, IR_PC3, IR_PC4

IR_PC1: {sensor_frame: ir1, data_type: PointCloud, topic:
/scan, marking: true, clearing: true}
IR_PC2: {sensor_frame: ir2, data_type: PointCloud, topic:
/scan, marking: true, clearing: true}
IR_PC3: {sensor_frame: ir3, data_type: PointCloud, topic:
/scan, marking: true, clearing: true}
```

```
IR_PC4: {sensor_frame: ir4, data_type: PointCloud, topic:
/scan, marking: true, clearing: true}
```

Soubor „global_costmap_params.yaml“ (viz Tabulka 33) obsahuje parametry pro globální *costmap*, kterými jsou definice základního souřadnicového rámu („global_frame“), souřadnicový rám základny robotu pro referenci („robot_base_frame“), obnovovací frekvence v Hz („update_frequency“) a parametr „static_map“ znamenající, zda má být použita existující mapa (hodnota „true“) nebo vytvořena nová (hodnota „false“). [62]

Tabulka 33 Obsah souboru global_costmap_params.yaml.

```
global_costmap:
  global_frame: /map
  robot_base_frame: /base_link
  update_frequency: 5.0
  static_map: true
```

Soubor „local_costmap_params.yaml“ (viz Tabulka 34) obsahuje parametry pro lokální *costmap*. Těmito parametry jsou znovu „global_frame“, „robot_base_frame“, „update frequency“, „static_map“ (viz výše), frekvence publikování informací pro vizualizaci („publish_frequency“), zda se má střed mapy pohybovat s robotem („rolling_window“) a rozměrové parametry lokální *costmap*: šířka („width“), délka („height“) a rozlišení („resolution“) v metrech. [62]

Tabulka 34 Obsah souboru local_costmap_params.yaml.

```
local_costmap:
  global_frame: /odom
  robot_base_frame: /base_link
  update_frequency: 5.0
  publish_frequency: 2.0
  static_map: false
  rolling_window: true
  width: 3.0
  height: 3.0
  resolution: 0.05
```

Soubor „base_local_planner_params.yaml“ (viz Tabulka 35) obsahuje parametry pro balíček base_local_planner. Tyto parametry popisují kinematické chování robotu pro plánování trasy balíčkem base_local_planner. Jsou to minimální („min_vel_x“) a

maximální („max_vel_x“) dopředné rychlosti v metrech za sekundu, minimální („min_vel_theta“) a maximální („max_vel_theta“) rychlost rotace kolem osy z v radiánech za sekundu, minimální rychlost rotace kolem osy z na místě v radiánech za sekundu („min_in_place_vel_theta“), maximální rychlostí zrychlení v ose x („acc_lim_x“) a ose y („acc_lim_y“) v metrech za sekundu na druhou a maximální zrychlení rotace kolem osy z v radianech za sekundu na druhou („acc_lim_theta“), zda se jedná o holonomického robota („holonomic_robot“), tedy zda mají být generovány i rychlosti v ose y. [86]

Tabulka 35 Obsah souboru base_local_planner_params.yaml.

```
TrajectoryPlannerROS:
  max_vel_x: 0.45
  min_vel_x: 0.05
  min_vel_theta: -1.0
  max_vel_theta: 1.0
  min_in_place_vel_theta: 0.4

  acc_lim_x: 2.5
  acc_lim_y: 2.5
  acc_lim_theta: 3.2

  holonomic_robot: false
```

V tomto balíčku byl také vytvořen soubor move_base.launch, který spustí balíčky mapovy_server, amcl a move_base a tím spustí celý ROS Navigation Stack. Obsah tohoto souboru je v Tabulka 36.

Tabulka 36 Obsah souboru move_base.launch.

```
<launch>
  <master auto="start"/>
  <!-- spusteni mapoveho serveru -->
  <node name="map_server" pkg="map_server" type="map_server"
args="$(find mapovy_server)/map.yaml" />
  <!--- spusteni AMCL -->
  <node name="amcl" pkg="amcl" type="amcl" output="screen" >
    <param name="laser_max_beams" value="0" />
    <param name="odom_model_type" value="diff" />
```

```
  </node>
  <!--- spusteni move_base -->
  <node pkg="move_base" type="move_base" respawn="false"
name="move_base" output="screen">
    <rosparam file="$(find
leela_2dnav)/costmap_common_params.yaml" command="load"
ns="global_costmap" />
    <rosparam file="$(find
leela_2dnav)/costmap_common_params.yaml" command="load"
ns="local_costmap" />
    <rosparam file="$(find
leela_2dnav)/local_costmap_params.yaml" command="load" />
    <rosparam file="$(find
leela_2dnav)/global_costmap_params.yaml" command="load" />
    <rosparam file="$(find
leela_2dnav)/base_local_planner_params.yaml" command="load"
/>
    <param name="controller_frequency" value="8.0" />
  </node>
</launch>
```

Poslední chybějící částí pro fungování navigace je zadání cílového bodu. Toto lze vyřešit buď programově programovým zadáním cíle, nebo jednodušeji grafickým, který je implementován balíčkem rvit a popsán v následující podkapitole 4.2.6 GUI navigačního systému.

4.2.6 GUI navigačního systému

Před každým novým spuštením ROS Navigation Stack bylo nutno zajistit synchronizaci času. V podkapitole 4.2.3 Synchronizace času je instalace chrony a nakonfigurování jeho spojení, to ale nezajistí okamžitou sychronizaci, protože chrony se synchronizuje se serverem po malých krocích. Proto bylo nutno pokaždé zadat příkazy z Tabulka 37, které zajistí restartování serveru (a tím spojení) a okamžitou synchronizaci času. Poslední příkaz ověří, že se tak opravdu stalo (viz Obr. 36). Tyto příkazy je nutno zadat pouze na klientovi.

Tabulka 37 Příkazy pro okamžitou synchronizaci času.

```
$ sudo invoke-rc.d chrony restart
$ sudo chronyc -a makestep
```

```
$ chrony tracking
```

```
pi@raspberrypi:~ $ chronyc tracking
Reference ID    : 192.168.2.103 (192.168.2.103)
Stratum         : 11
Ref time (UTC)  : Sun May 22 09:33:29 2016
System time     : 0.000000007 seconds fast of NTP time
Last offset     : +0.002140445 seconds
RMS offset      : 0.002140445 seconds
Frequency       : 956.064 ppm slow
Residual freq   : +1007.899 ppm
Skew            : 15.158 ppm
Root delay      : 0.001230 seconds
Root dispersion : 0.021820 seconds
Update interval : 2.1 seconds
Leap status     : Normal
```

Obr. 36 Ověření správné synchronizace času.

Následně bylo spuštěno jádro ROSu na Raspberry Pi, viz Tabulka 1. Poté byly v novém terminálu Raspberry Pi spuštěny programy balíčku leela obsluhující senzory, motory a enkodéry příkazem v Tabulka 38.

Tabulka 38 Příkaz pro spuštění balíčků na Raspberry Pi.

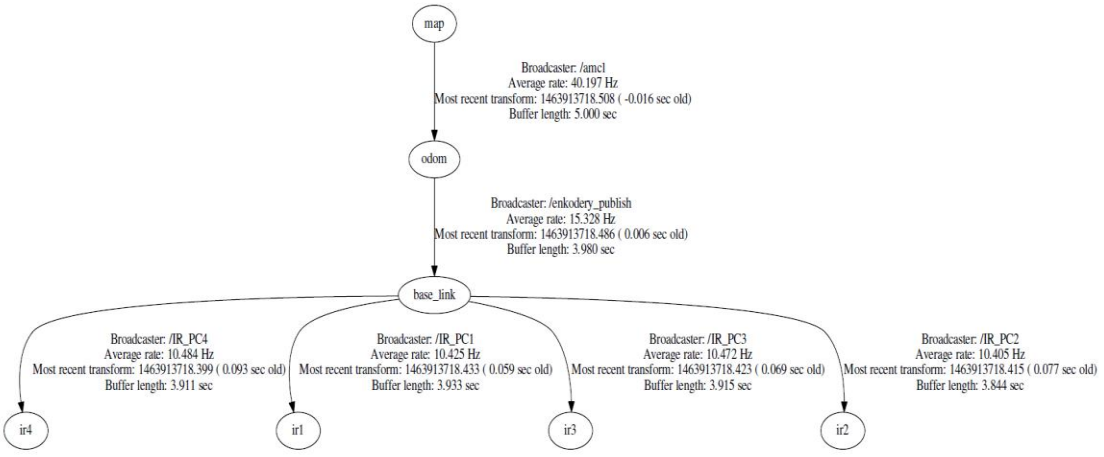
```
$ roslaunch leela_2dnav leela_configuration.launch
```

Po ověření správného spuštění těchto balíčků byl spuštěn ROS Navigation Stack příkazem v Tabulka 39. Toto spuštění trvá přibližně minutu.

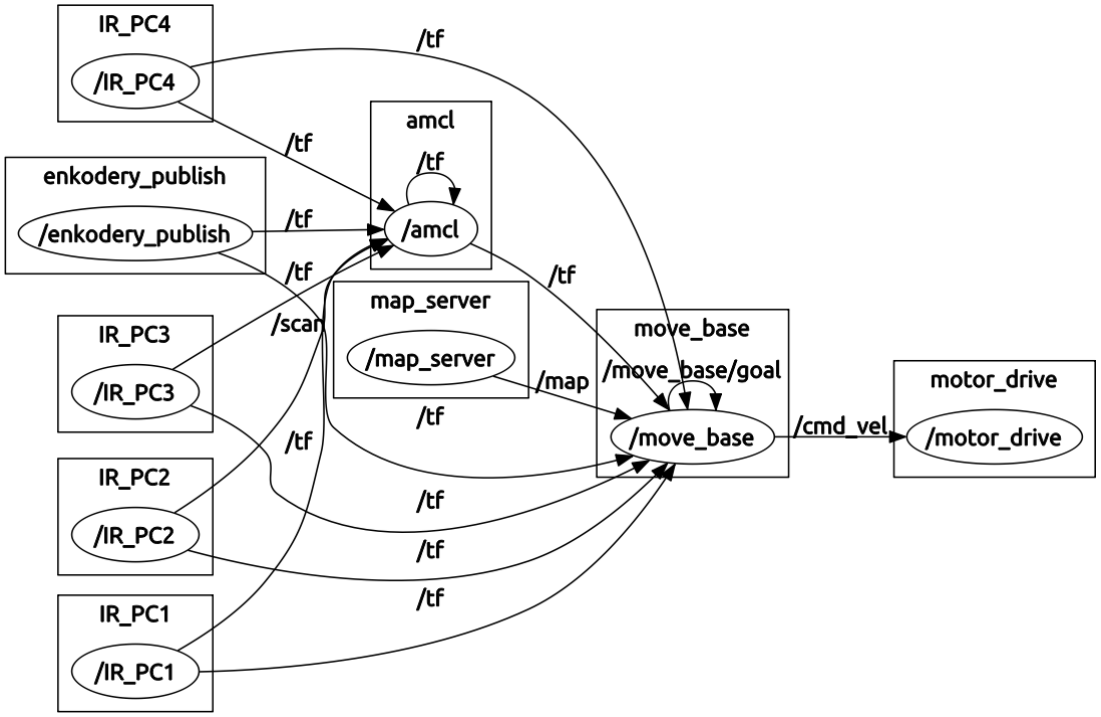
Tabulka 39 Příkaz pro spuštění ROS Navigation Stack.

```
$ roslaunch leela_2dnav move_base.launch
```

Před spuštěním vizualizace bylo ověřeno správné fungování všech částí zobrazením tranformačního stromu (viz Obr. 37) a komunikačního grafu (viz Obr. 38, příkaz viz Tabulka 4 v podkapitole 2.3.7 Balíček rqt_graph).



Obr. 37 Kompletní transformační strom.



Obr. 38 Kompletní komunikační graf.

Nyní byl spuštěn celý navigační systém a mohl být spušten rviz (viz Tabulka 40). Pokud by byl spuštěn dříve, kvůli datům z ROS Navigation Stack by bylo nemožné ho správně nastavit.

Tabulka 40 Příkaz pro spuštění balíčku rviz.

```
$ rosrun rviz rviz
```

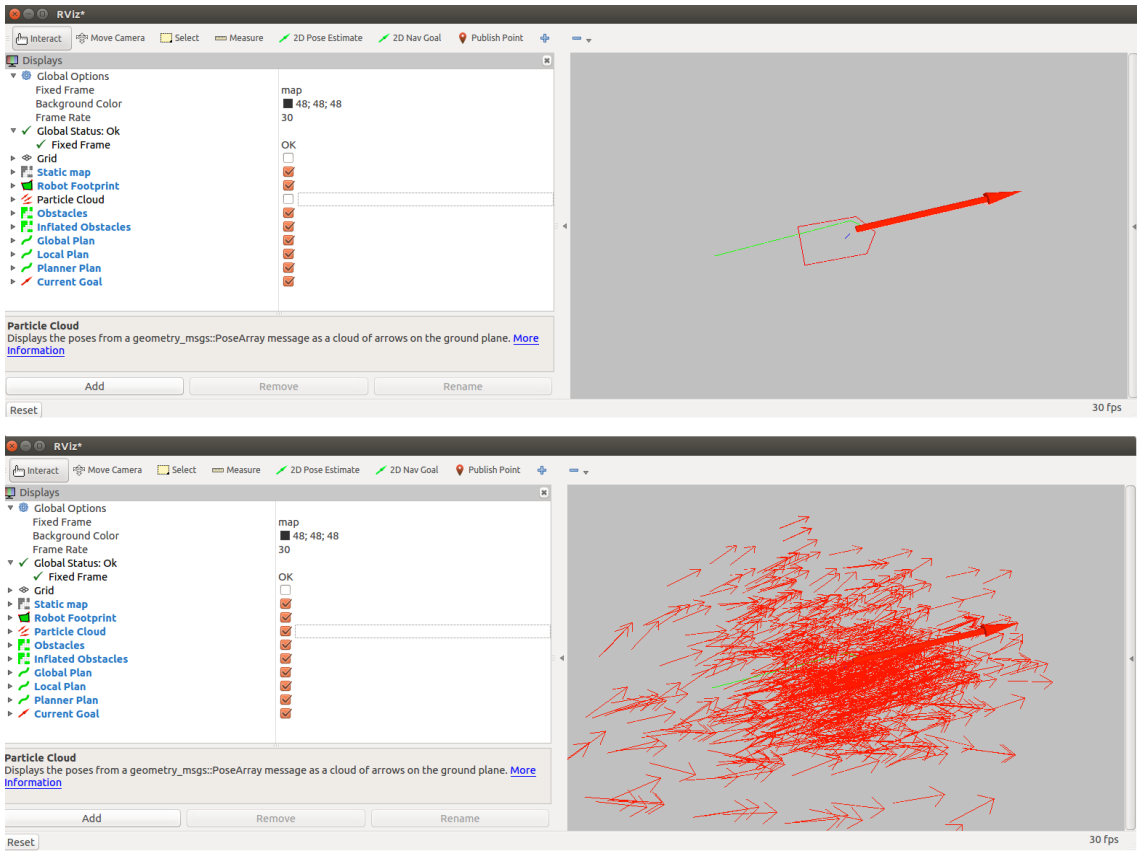
Nastavení rviz bylo nutno provést pouze při prvním spuštění, při každém dalším spuštění rviz načte předchozí uložený stav. Toto nastavení bylo provedeno dle [87] s malými změnami. Postup je popsaný v Tabulka 41.

Tabulka 41 Nastavení GUI rviz.

Načtení mapy	Add -> Map -> Display Name = Static Map
Zobrazení *particle cloud*	Add -> Pose Array -> Display Name = Particle Cloud
Zobrazení půdorysu robotu	Add -> Polygon -> Display Name = Robot Footprint Topic -> „...“ -> local_costmap -> robot_footprint (geometry_msgs/PolygonStamped) Color (255,25,0)
Zobrazení překážek	Add -> Grid Cells -> Display Name = Obstacles Topic -> „...“ -> local_costmap -> obstacles (nav_msgs/GridCells) Color (255,25,0)

Zobrazení okolí překážek	Add -> Grid Cells -> Display Name = Inflated Obstacles Topic -> „...“ -> local_costmap -> inflated_obstacles (nav_msgs/GridCells) Color (0,25,255)
Zobrazení části globálního plánu cesty, kterou robot momentálně sleduje	Add -> Path -> Display Name = Global Plan Topic -> „...“ -> TrajectoryPlannerROS -> global_plan (nav_msgs/Plan)
Zobrazení trajektorie robotu	Add -> Path -> Display Name = Local Plan Topic -> „...“ -> TrajectoryPlannerROS -> local_plan (nav_msgs/Plan) Color (0,25,255)
Zobrazení kompletního globálního plánu cesty	Add -> Path -> Display Name = Planner Plan Topic -> „...“ -> NavfnROS -> plan (nav_msgs/Plan)
Zobrazení cíle	Add -> Pose -> Display Name = Current Goal Topic -> „...“ -> current_goal (geometry_msgs/PoseStamped) 2D Nav Goal -> topic -> move_base_simple/goal Color (0,25,255)

Před samotnou navigací je potřeba robotu zadat odhad počáteční polohy. To se provádí tlačítkem „2D Pose Estimate“, kliknutím a podržením do mapy nad požádovaným bodem, pohybem myši ve směru požadovaného natočení robotu a uvolněním tlačítka myši. Zadání cíle se provádí obdobně tlačítkem „2D Nav Goal“. Na Obr. 39 je screenshot celého GUI bez a se zapnutým *particle cloud* (dole).



Obr. 39 GUI rviz.

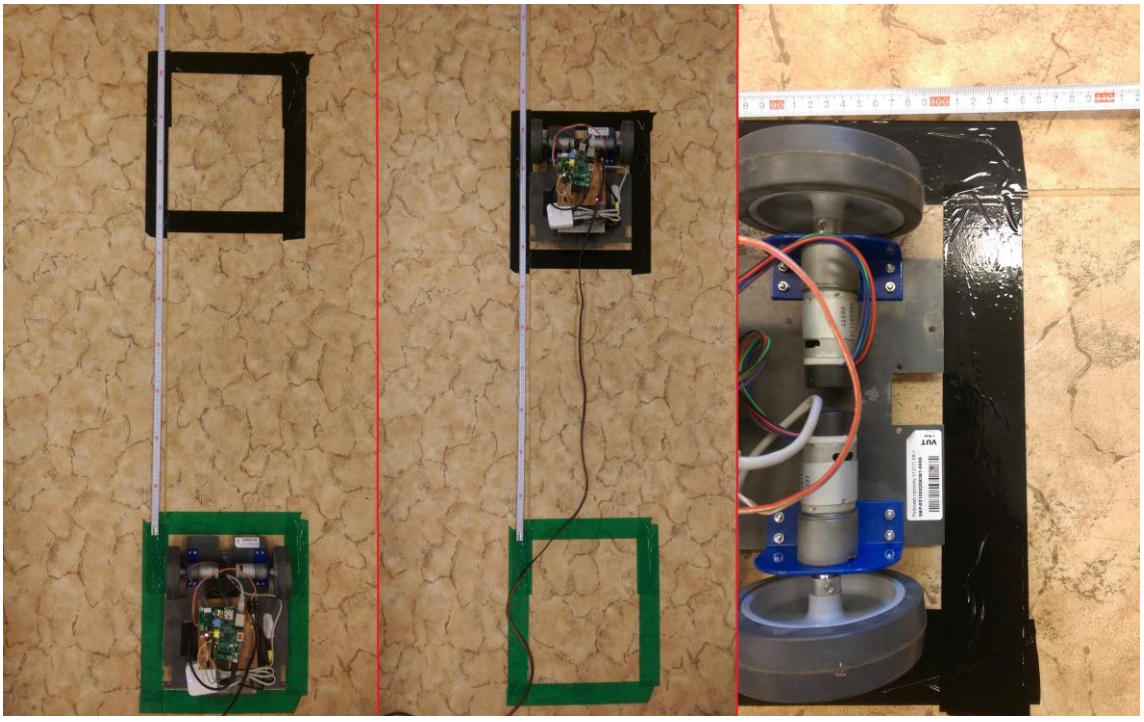
5 PRAKTICKÉ EXPERIMENTY

Cílem této kapitoly je otestovat funkčnost předloženého robota praktickými experimenty. Těmito experimenty byly experimenty pro ověření odometrie a test plánovacího modulu. Pro jednodušší manipulaci byla odstraněna šasi. V příloze A jsou přiloženy obrazové záznamy z těchto testů.

5.1 Ověření odometrie

Při ověření odometrie bylo cílem určit přesnost měření vzdálenosti robotem. Jediným zdrojem tohoto měření byly enkodéry, které ze své podstaty přinášejí do systému malé inkrementální chyby. Cílem tohoto testu bylo tedy určit přesnost implementace této metody.

První část měření probíhala přesunem dopředu o vzdálenost přesně 1 m s cílem určení přesnosti algoritmu. Robot se pohyboval z počátečního místa označeného zeleně do konečného označeného černě, viz Obr. 40, kde na fotce vlevo je počáteční stav, uprostřed konečný a vpravo detail konečného stavu. Řízení robota bylo manuální šipkami na klávesnici, ujetá vzdálenost odečtena z terminálu výstupu programu „enkodery_publish.py“, viz Obr. 41.



Obr. 40 Reálné fotky při ujetí 1m.

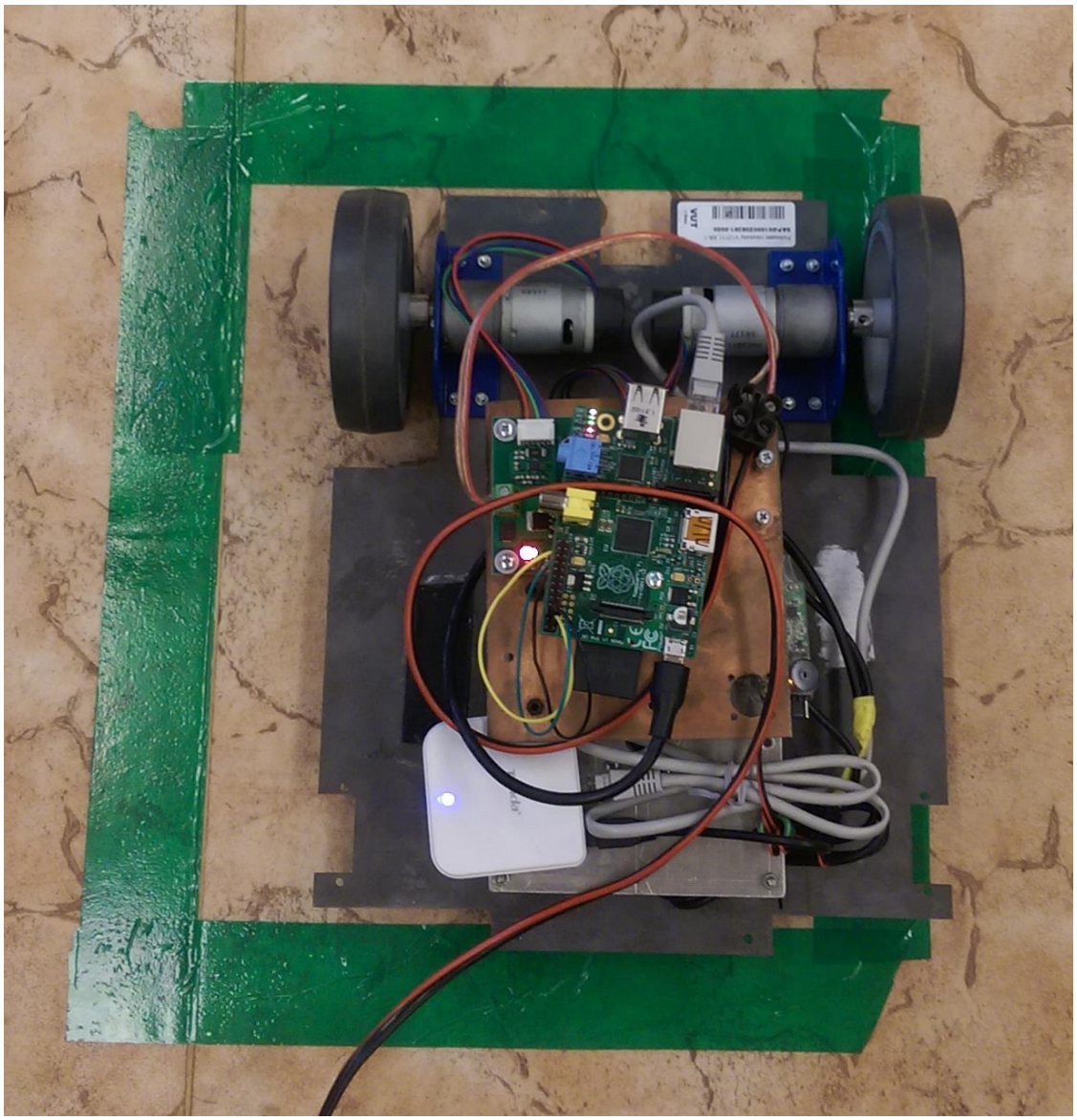
```
vzdalenost v ose X: 0.993082105373
vzdalenost v ose Y: 0.00327398995206
momentalni smer: 0.0081556537537
```

Obr. 41 Výstup z konzole po ujetí vzdálenosti 1m.

Druhou částí byl test rotace, kdy robot byl otočen o 180° a zpět, tedy výsledný „momentální směr“ se měl na konci testu blížit 0. Počáteční pozice byla stejná jako u předchozího testu. Odečtený směr z konzole po tomto testu je na Obr. 42 a reálná pozice na Obr. 43.

```
vzdalenost v ose X: -381813.880699
vzdalenost v ose Y: -3074379.16129
momentalni smer: 0.00203891981032
```

Obr. 42 Výstup z konzole po rotaci do výchozího bodu.

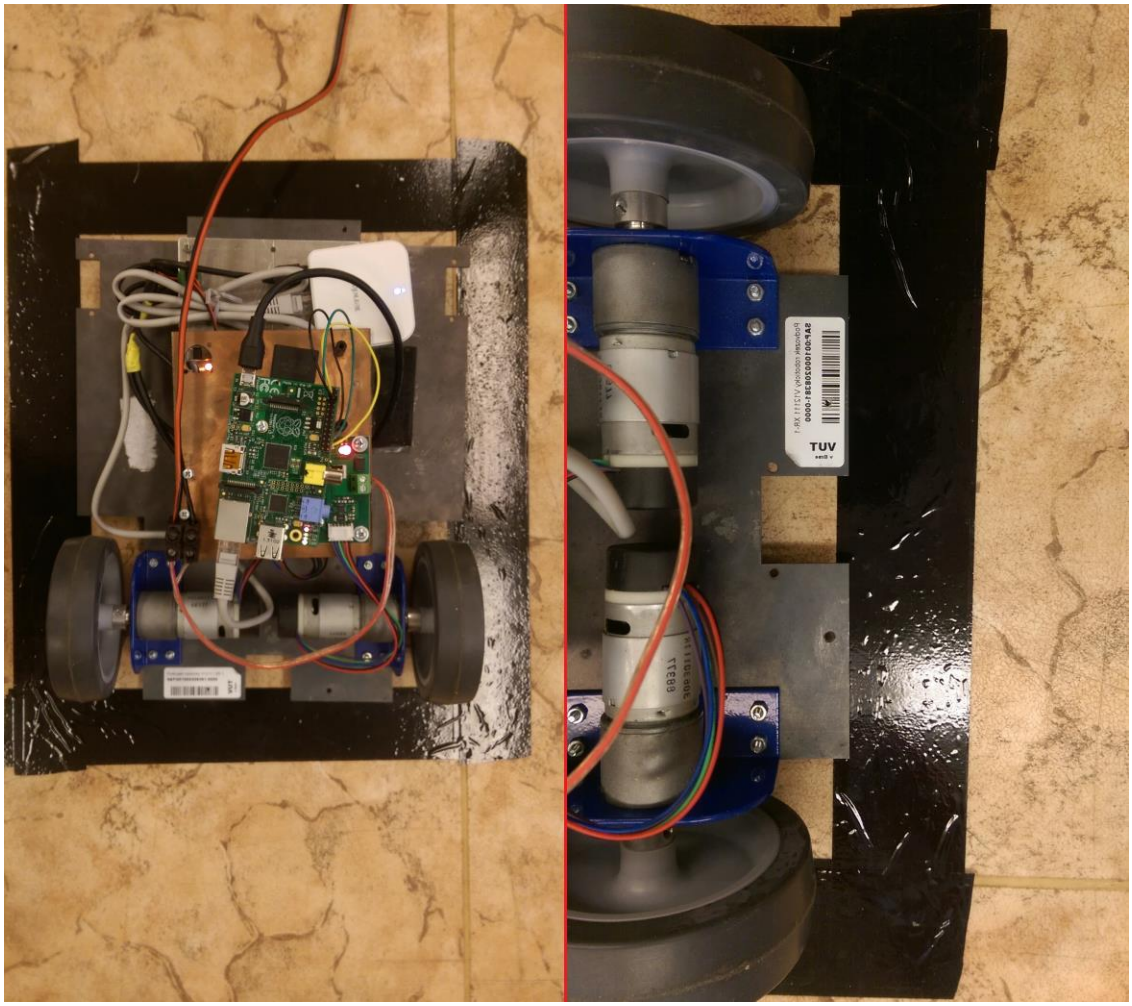


Obr. 43 Výsledná pozice robotu po rotaci.

5.2 Ověření plánovacího modulu

Pro testování plánovacího modulu byl použit pouze test ujetí 1m, protože plánovací modul i při pouhé translaci používá rotaci na korigování chyb. Tedy není nutno provádět

test rotace. Zadání cíle bylo realizováno programem dle ROS Tutoriálu, viz [88]. Výchozí pozice byla stejná jako v předchozím testu, cílová pozice je na Obr. 44.



Obr. 44 Cílová pozice testu plánovacího modulu.

6 ZÁVĚR

Zadáním této diplomové práce bylo vytvořit návrh řešení řídícího systému reálného mobilního robotu Leela v ROSu a tento návrh následně realizovat.

Vzledem k rozsáhlosti projektu ROS bylo nutno se s tímto systémem nejdříve seznámit, čemuž je věnována celá kapitola 2 ROS Framework. Následně bylo nutno se seznámit s jednotlivými poskytnutými komponenty a vybrat jakékoliv další potřebné komponenty pro řešení tohoto řídícího systému. Po zajištění napájení a hardwarové komunikace mezi těmito komponenty byly nainstalovány operační systémy a ROSy. Zajištěním synchronizace času a spojením systémů přes síť bylo přistoupeno k dalšímu kroku, implementaci plánovácího modulu.

Plánování trasy bylo realizováno pomocí ROS Navigation Stack, kdy programy pro obsluhu senzorů, motorů a enkodérů byly vytvořeny přímo pro potřeby této práce. Mapový server a lokalizační software amcl byly implementovány ROSem. Po ověření funkčnosti všech částí byl spuštěn plánovací modul a bylo nakonfigurováno grafické uživatelské rozhraní rviz pro vizualizaci pohybu, zadávání předpokládané polohy a zadávání cíle robotu.

V poslední části jsou posány experimenty pro ověření funkčnosti realizovaného řešení. Těmito experimenty jsou ověření funkčnosti odometrie a plánovácího modulu. Při testu měření vzdálenosti byl naměřen rozdíl 7 mm vůči předpokládané hodnotě, který lze pokládat za zanedbatelně malý. Na konci testu odometrie při rotaci se robot nacházel mimo požadovanou pozici o několik centimetrů v ose y, což bylo způsobeno rozdílnou adhezí povrchu pod koly při dané rotaci. Vzhledem k výslednému rozdílu 0,002 radiánu vůči předpokládanému výsledku byl tento test označen za úspěšný. Při testování plánovacího modulu byla nastavena maximální odchylka od cíle 0,03 radiánu a 0,03 metru. Z poskytnuté dokumentace je patrné, že výsledná pozice robotu je v těchto mezích a tedy plánovací modul byl úspěšně implementován.

Celá práce byla psána formou podrobného návodu pro umožnění jednoduchého zopakování a navázání na ni dalším vývojem. ROS v současné době podstupuje intenzivní vývoj a dá se předpokládat, že tento vývoj bude pokračovat i v budoucnosti. Tato práce poskytuje do budoucnosti vytvořenou platformu pro další implementaci a testování nejrůznějších součástek a řešení různorodých problémů s pomocí ROSu.

SEZNAM POUŽITÝCH ZDROJŮ

[1] Open Source Robotics Foundation. *ROS* [Online]. [Citováno 18. dubna 2016]. Dostupné z: <http://www.ros.org/>.

[2] THOMAS, Dirk. *ROS/ Introduction* [Online]. 22.5.2014 [Citováno 16. dubna 2016]. Dostupné z: <http://wiki.ros.org/ROS/Introduction>.

[3] AaronMR. *ROS/ Concepts* [Online]. 21.6.2014 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/ROS/Concepts>.

[4] XU, Hao. *ROS/ Client Libraries* [Online]. 12.12.2015 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/Client%20Libraries>.

[5] Open Source Initiative. *The BSD 3-Clause License* [Online]. [Citováno 18. dubna 2016]. Dostupné z: <https://opensource.org/licenses/BSD-3-Clause>.

[6] HANSEN, Karl. *ROS/ Packages* [Online]. 19.10.2015 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/Packages>.

[7] WEAVER, Josh. *ROS/ Metapackages* [Online]. 28.3.2014 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/Metapackages>.

[8] THOMAS, Dirk. *Specification of package manifest format* [Online]. 12.9.2012, 22.9.2012 [Citováno 18. dubna 2016]. Dostupné z: <http://www.ros.org/reps/rep-0127.html>.

[9] VENATOR, Ed. *catkin/ package.xml* [Online]. 30.7.2015 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/catkin/package.xml>.

[10] TullyFoote. *Repositories* [Online]. 3.4.2014 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Repositories>.

[11] Eric. *msg* [Online]. 15.4.2016 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/msg>.

[12] THOMAS, Dirk. *srv* [Online]. 19.11.2013 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/srv>.

[13] CONLEY, Ken. *Nodes* [Online]. 3.2.2012 [Citováno 18. května 2016]. Dostupné z: <http://wiki.ros.org/Nodes>.

[14] CONLEY, Ken. *Master* [Online]. 3.2.2012 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Master>.

[15] THOMAS, Dirk. *Parameter Server* [Online]. 7.8.2013 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Parameter%20Server>.

[16] GvdHoorn. *Messages* [Online]. 22.5.2015 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Messages>.

[17] FOROUHER, Dariush. *Topics* [Online]. 1.6.2014 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Topics>.

[18] CONLEY, Ken. *Services* [Online]. 3.2.2012 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Services>.

[19] SAITO, Isaac. *Bags* [Online]. 2.5.2015 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Bags>.

[20] jkay. *Distributions* [Online]. 19.3.2016 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Distributions>.

[21] SCHULTZ, Jarvis. *Tickets* [Online]. 3.12.2013 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/Tickets>.

[22] Open Source Robotic Foundation. *ROS History* [Online] . [Citováno 18. dubna 2016]. Dostupné z: <http://www.ros.org/history/>.

[23] Willow Garage. *ROS* [Online]. [Citováno 18. dubna 2016]. Dostupné z: <https://www.willowgarage.com/pages/software/ros-platform>.

[24] YOONSEOK, Pyo. *ROS Version / Poster / Icon* [Online]. 9.7.2015 [Citováno 18. dubna 2016]. Dostupné z: <http://image.slidesharecdn.com/20150708rosseminarinbusankorea-150709053607-lva1-app6891/95/20150708-ros-seminarinbusankorea-44-638.jpg?cb=1436420777>.

[25] GERKEY, Brian. *Why ROS 2.0?* [Online]. [Citováno 18. dubna 2016]. Dostupné z: <http://design.ros2.org/articles/why_ros2.html>.

[26] GERKEY, Brian. *ROS 2.0 Roadmap* [Online]. 6.4.2015 [Citováno 18. dubna 2016]. Dostupné z: <https://github.com/ros2/ros2/wiki/Roadmap>.

[27] THOMAS, Dirk. *roscore* [Online]. 31.21.2013 [Citováno 18. dubna 2016]. Dostupné z: <http://wiki.ros.org/roscore>.

[28] Willow Garage. *rospack* [Online]. 2011 [Citováno 19. dubna 2016]. Dostupné z: <http://docs.ros.org/independent/api/rospkg/html/rospack.html>.

[29] MOORE, Tom. *rostopic* [Online]. 23.4.2015 [Citováno 19. dubna 2016]. Dostupné z: <http://wiki.ros.org/rostopic>.

[30] HENDRIX, Austin. *rosbash* [Online]. 13.6.2015 [Citováno 19. dubna 2016]. Dostupné z: <http://wiki.ros.org/rosbash>.

[31] LIVINGSTON, Scott. *rqt* [Online]. 11.2.2015 [Citováno 19. dubna 2016]. Dostupné z: <http://wiki.ros.org/rqt>.

[32] FOROUHER, Darius. *rqt_graph* [Online]. 1.6.2014 [Citováno 19. dubna 2016]. Dostupné z: <http://wiki.ros.org/rqt_graph>.

[33] KEN, Mori. *Robot State Publisher.* [Online]. 9.1.2016 [Citováno 19. dubna 2016]. Dostupné z: <http://4.bp.blogspot.com/-K03xtoSUmAk/VpEgsH1ubBI/AAAAAAAAAiI/QtLD0ClDXNo/s1600/rqt_joint_state_publisher_rviz.png>.

[34] SAITO, Isaac. *tf* [Online]. 19.7.2015 [Citováno 19. dubna 2016]. Dostupné z: <http://wiki.ros.org/tf>.

[35] antonella. *ROS tf tree* [Online]. 27.3.2013 [Citováno 19. dubna 2016]. Dostupné z: <http://answers.ros.org/upfiles/13696863696590874.png>.

[36] Sunshine. *navigation* [Online]. 1.5.2016 [Citováno 19. května 2016]. Dostupné z: <http://wiki.ros.org/navigation>.

[37] Rethink Robotics. *Rviz* [Online]. 2015 [Citováno 19. dubna 2016]. Dostupné z: <http://sdk.rethinkrobotics.com/wiki/Rviz>.

[38] GOEBEL, Patrick. *Pi Robot Passes First Navigation Endurance Test* [Online]. 16.1.2012 [Citováno 19. dubna 2016]. Dostupné z: <http://www.pirobot.org/blog/0025/nav_test_rviz_1.png>.

[39] LUCETTI, Walter. *robots* [Online]. 13.5.2016 [Citováno 19. května 2016]. Dostupné z: <http://wiki.ros.org/Robots>.

[40] Clearpath Robotics. *PR2 Support* [Online]. [Citováno 22. dubna 2016]. Dostupné z: <https://support.clearpathrobotics.com/hc/en-us/categories/200217239-PR2>.

[41] Willow Garage. *PR2 Hardware Specs.* [Online]. [Citováno 22. dubna 2016]. Dostupné z: <https://www.willowgarage.com/pages/pr2/specs>.

[42] ASH, Devon. *Robots/ PR2* [Online]. 2.3.2016 [Citováno 22. dubna 2016]. Dostupné z: <http://wiki.ros.org/Robots/PR2>.

[43] Willow Garage. *PR2* [Online]. 9.8.2010 [Citováno 22. dubna 2016]. Dostupné z: <http://www.smartroboticsys.eu/wp-content/uploads/2014/12/pr22.jpg>.

[44] Dataspeed. *Dataspeed products* [Online]. 2014 [Citováno 22. dubna 2016]. Dostupné z: <http://dataspeedinc.com/products>.

[45] Milvus Teknoloji Mekatronik Müh. *Robin* [Online]. 2015 [Citováno 22. dubna 2016]. Dostupné z: <http://milvusrobotics.com/en/robin-smart-service-and-advertisement-robot>.

[46] Milvus Teknoloji Mekatronik Müh. *Robin* [Online]. 2015 [Citováno 22. dubna 2016]. Dostupné z: <http://milvusrobotics.com/themes/milvus/assets/images/robin/gallery-01.jpg>.

[47] SoftBank Robotics. *Find out more about Pepper* [Online]. [Citováno 22. dubna 2016]. Dostupné z: <https://www.aldebaran.com/en/cool-robots/pepper/find-out-more-about-pepper>.

[48] LADYANN. *Pepper* [Online]. 20.6.2015 [Citováno 22. dubna 2016]. Dostupné z: <http://nootrix.com/wp-content/uploads/2014/06/pepper-680x365_c.jpg>.

[49] kwc. *Robots using ROS: Penn Quadrorotors* [Online]. 28.5.2010 [Citováno 23. dubna 2016]. Dostupné z: <http://www.ros.org/news/2010/05/robots-using-ros-penn-quadrotors.html>.

[50] D. Mellinger, N. Michael a V. Kumar, Trajectory generation and control for precise aggressive maneuvers with quadrotors, *The International Journal of Robotics Research*, 2012, vol. 31 (5), pp. 664-674.

[51] Raspberry Pi Foundation. *Raspberry Pi* [Online]. [Citováno 13. května 2016]. Dostupné z: <https://www.raspberrypi.org>.

[52] Adafruit. *Raspberry Pi Model B 512MB RAM* [Online]. [Citováno 13. května 2015]. Dostupné z: <https://www.adafruit.com/products/998>.

[53] PRENNER, Michal. *Raspberry Pi Model B 512MB RAM Rev 2.0* [Online]. [Citováno 13. května 2016]. Dostupné z: <http://rpishop.cz/raspberry-pi-pocitace/8-raspberry-pi-0766897151323.html>.

[54] Devantech. *RD02 – 12v robot drive* [Online]. [Citováno 23. dubna 2016]. Dostupné z: <http://www.robot-electronics.co.uk/products/drive-systems/drive-kits/rd02-12v-robot-drive.html>.

[55] Devantech. *MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive* [Online]. [Citováno: 23. dubna 2016]. Dostupné z: <http://www.robot-electronics.co.uk/htm/md25tech.htm>.

[56] Devantech. *MD25 - Dual 12Volt 2.8Amp H Bridge Motor Drive (I2C mode documentation)* [Online]. [Citováno: 22. dubna 2016]. Dostupné z: <http://www.robot-electronics.co.uk/htm/md25i2c.htm>.

[57] Sharp. *GP2Y0A21 Documentation* [Online]. [Citováno 24. dubna 2016]. Dostupné z: <http://www.sharpsma.com/webfm_send/1489>.

[58] tenda_cz. *Tenda A6 Wireless-N Travel AP/ Router/ Client* [Online]. 24.3.2014 [Citováno 19. května 2016]. Dostupné z: <http://www.tenda.cz/article/tenda-a6-wireless-n-travel-ap-router-client>.

[59] Logitech. *Logitech Webcam C930e Technical Specifications* [Online]. [Citováno 19. května 2016]. Dostupné z: <http://support.logitech.com/en_us/article/39606?product=a0qi00000069v0MAAQ>.

[60] HP Development Company. *HP EliteBook 2540p Notebook PC - Overview* [Online]. 2016 [Citováno 19. května 2016]. Dostupné z: <http://h20564.www2.hp.com/hpsc/doc/public/display?docId=emr_na-c02206692>.

[61] Oracle. *VirtualBox* [Online]. 28.4.2015 [Citováno 19. května 2016]. Dostupné z: <https://www.virtualbox.org/>.

[62] ECORCHARD, Gael. *navigation/ Tutorials/ RobotSetup* [Online]. 6.8.2015 [Citováno 20. května 2016]. Dostupné z: <http://wiki.ros.org/navigation/Tutorials/RobotSetup>.

[63] LU, David. *move_base* [Online]. 3.3.2016 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/move_base>.

[64] FOOTE, Tully. *geometry_msgs* [Online]. 25.2.2016 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/geometry_msgs>.

[65] FOOTE, Tully. *nav_msgs* [Online]. 25.2.2016 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/nav_msgs>.

[66] FOOTE, Tully. *sensor_msgs* [Online]. 25.2.2016 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/sensor_msgs>.

[67] TUM Technische Universität München. *User Manual of 3A UBEC* [Online]. 26.5.2009 [Citováno 20. května 2016]. Dostupné z: <http://www.daedalus.ei.tum.de/attachments/article/186/UBEC-3A.pdf>.

[68] HobbyKing. *Turnigy 3A UBEC with Low Voltage Buzzer* [Online]. [Citováno 19. května 2016]. Dostupné z: <http://www.hobbyking.com/hobbyking/store/__22494__Turnigy_3A_UBEC_with_Low_Voltage_Buzzer.html>.

[69] eLinux.org. *RPi Low-level peripherals* [Online]. 3.7.2015 [Citováno 22. května 2016]. Dostupné z: <http://elinux.org/RPi_Low-level_peripherals>.

[70] Premier Farnell. *NXP PCF8591T/2,512 ADC, DAC, 8BIT, I2C, 16SOIC* [Online]. 2016 [Citováno 22. května 2016]. Dostupné z: <http://cz.farnell.com/nxp/pcf8591t-2-512/adc-dac-8bit-i2c-16soic/dp/2296062>.

[71] NXP Semiconductors. *PCF8591 Data Sheet* [Online]. 27.6.2013 [Citováno 22. května 2016]. Dostupné z: <http://www.farnell.com/datasheets/1759207.pdf>

[72] Devantech. *Using the I2C Bus* [Online]. [Citováno 22. května 2016]. Dostupné z: <http://www.robot-electronics.co.uk/i2c-tutorial>.

[73] Canonical. *Alternative downloads* [Online]. 2016 [Citováno 20. května 2016]. Dostupné z: <http://www.ubuntu.com/download/alternative-downloads>.

[74] Oracle. *Configuring virtual machines – 64-bit guests* [Online]. [Citováno 20. května 2016]. Dostupné z: <https://www.virtualbox.org/manual/ch03.html#intro-64bitguests>.

[75] Raspberry Pi Foundation. *Raspbian* [Online]. 10.5.2016 [Citováno 20. května 2016]. Dostupné z: <https://www.raspberrypi.org/downloads/raspbian/>.

[76] Adam. *Win32DiskImager* [Online]. [Citováno 20. května 2016]. Dostupné z: <http://www.raspberry-projects.com/pi/pi-operating-systems/win32diskimager>.

[77] TATHAM, Simon. *Putty* [Online]. [Citováno 20. května 2016]. Dostupné z: <http://www.putty.org/>.

[78] Linux New Media USA. *Passwords* [Online]. 2016 [Citováno 20. května 2016]. Dostupné z: <http://www.raspberry-pi-geek.com/howto/Passwords>.

[79] WOODALL, William. *Ubuntu install of ROS Jade* [Online]. 18.5.2016 [Citováno 20. května 2016]. Dostupné z: <http://wiki.ros.org/jade/Installation/Ubuntu>.

[80] LAMPRIANIDIS, Nick. *Installing and Configuring Your ROS Enviroment* [Online]. 11.1.2015 [Citováno 20. května 2016]. Dostupné z: <http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment>.

[81] Dhood. *Installing from Source* [Online]. 17.2.2016 [Citováno 20. května 2016]. Dostupné z: <http://wiki.ros.org/indigo/Installation/Source>.

[82] LICHVAR, Miroslav. *Chrony* [Online]. 4.4.2016 [Citováno 20. května 2016]. Dostupné z: <https://chrony.tuxfamily.org/>.

[83] YOUNG, Matt. *ROS/ Networksetup* [Online]. 13.4.2016 [Citováno 20. května 2016]. Dostupné z: <http://wiki.ros.org/ROS/NetworkSetup>.

[84] autogenerated. *nav_masgs/Odometry Message* [Online]. 25.2.2016 [Citováno 22. května 2016]. Dostupné z: <http://docs.ros.org/api/nav_msgs/html/msg/Odometry.html>.

[85] FOOTE, Tully. *rviz/ DisplayType/ PointCloud* [Online]. 8.1.2014 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/rviz/DisplayTypes/PointCloud>.

[86] WALTER, Lucas. *base_local_planner* [Online]. 5.10.2014 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/base_local_planner>.

[87] FOOTE, Tully. *Using rviz with the Navigation Stack.* [Online]. 8.1.2014 [Citováno 21. května 2016]. Dostupné z: <http://wiki.ros.org/navigation/Tutorials/Using%20rviz%20with%20the%20Navigation%20Stack>.

[88] LEIGH, Angus. *Sending goals to the Navigation Stack* [Online]. 21.3.2014 [Citováno 22. května 2016]. Dostupné z: <http://wiki.ros.org/navigation/Tutorials/SendingSimpleGoals>.

A. OBSAH PŘILOŽENÉHO DVD

Název	**Popis**
DP_Krysl.pdf	Textová část diplomové práce.
Odometrie_trans.mov	Obrazový záznam z ověření odometrie – přesunu dopředu.
Odometrie_rot.mov	Obrazový záznam z ověření odometrie – rotace.
Planovací_modul.mov	Obrazový záznamk z ověření plánovacího modulu.