VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

# NÁVRH A REALIZACE SENZORICKÉHO SYSTÉMU PRO MOBILNÍ ROBOT S VYUŽITÍM FRAMEWORKU ROS

SENSOR SYSTEM DESIGN FOR MOBILE ROBOT BASED ON ROS FRAMEWORK

AUTOR PRÁCE
AUTHOR

ING. STANISLAV VĚCHET, PH.D.

VEDOUCÍ PRÁCE
SUPERVISOR

BC. PETR TOMÁŠ

BRNO 2014

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2013/2014

# ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Petr Tomáš

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

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 senzorického systému pro mobilní robot s využitím frameworku ROS**

v anglickém jazyce:

**Sensor system design for mobile robot based on ROS framework**

Stručná charakteristika problematiky úkolu:

Hlavním cílem práce je návrh a realizace senzorického systému vhodného pro použití při navigaci mobilních robotů. Navržený senzorický systém musí využívat framework ROS (Robot Operating System), který je určený pro tvorbu navigačního software mobilních robotů. Jako hlavní senzor bude sloužit ultrazvukový dálkoměr SRF komunikující pomocí sběrnice I2C s jednodeskovým počítačem. Naměřená data budou přenášena do PC pomocí protokolu TCP/IP.

Cíle diplomové práce:

Podrobně se seznamte s možnostmi frameworku ROS.

Navrhněte způsob komunikace senzorů SRF s nadřazeným počítačem.

Proveďte oživení modulu ROS vhodného pro ultrazvukové dálkoměry.

Navržený systém prakticky otestujte a ověřte jeho funkčnost na mobilním robotu.

Seznam odborné literatury:

[1] 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.

[2] 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.

[3] Robot Operating System: online dostupné z www.ros.org

Vedoucí diplomové práce: Ing. Stanislav Věchet, Ph.D.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2013/2014.

V Brně, dne 9.12.2013

L.S.

| | |
|---|---|
| Ing. Jan Roupec, Ph.D. | doc. Ing. Jaroslav Katolický, Ph.D. |
| Ředitel ústavu | Děkan fakulty |

## Anotace

Podstatou této diplomové práce je návrh a implementace senzorického systému využívající robotický framework, který nese název ROS (Robot Operating System). Hlavním úkolem je provést podrobnou analýzu a otestování možností robotického frameworku se závěrečnou implementací na konkrétní testovací robotické aplikaci (senzorický systém) s následným ohodnocením použitelnosti tohoto systému v mobilní robotice. Dále je paralelně cílem vytvořit detailní obecný a praktický průvodce pro začátečníky s ROS, kteří jsou také málo zkušení v linuxových operačních systémech.

## Annotation

The essence of this master thesis is design and implementation of sensor system based on robotic framework which is called ROS (Robot Operating System). The main task is to perform detailed analysis and test of capabilities of the framework with final implementation on specific robot application (sensor system) with following evaluation of applicability of the system in mobile robotics. As parallel aim is to create detailed general and practical guide for beginners with ROS which they are also beginners in Linux based operating systems.

## Klíčová slova

ROS, ultrazvukový dálkoměr SRF, mobilní robotika, Linux, Lego NXT, Raspberry Pi

## Keywords

ROS, ultrasonic rangefinder SRF, mobile robotics, Linux, Lego NXT, Raspberry Pi

## Prohlášení o originalitě

*Já Petr Tomáš, prohlašuji, že jsem diplomovou práci vypracoval samostatně a že jsem uvedl všechny použité prameny a literaturu.*

*V Brně dne 30. 5. 2014* .....

Bibliografická citace VŠKP
TOMÁŠ, P. *Návrh a realizace senzorického systému pro mobilní robot s využitím frameworku ROS.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2014. 85 s. Vedoucí diplomové práce Ing. Stanislav Věchet, Ph.D..

## Poděkování

Na tomto místě bych rád především poděkoval vedoucímu diplomové práce Ing. Stanislavu Věchetovi, Ph.D. za pomoc, cenné rady a věnovaný čas při tvorbě závěrečné práce.

V neposlední řadě bych také rád poděkoval celé mé rodině a všem blízkým, kteří mi svou tolerancí a trpělivostí vytvořili podmínky pro vytvoření této práce.

# Obsah

# 1. ÚVOD

Mobilní robotika je v dnešní době ve fázi vývoje a výzkumu, čímž se stále řadí do oblasti experimentální robotiky. Tento obor je v praxi rozvíjen z velké části z řad universit, výzkumných institucích nebo z řad dobrovolných nadšenců a expertů. To má za výhodu, že většina nově vzniklých robotických aplikací nebo software knihoven není tvořena s myšlenkou co největšího profitu, ale s přístupem obohatit celou mobilní robotiku jako takovou. Nesnaží se například podléhat licencím, které by bránily volnému šíření pro uživatele z celého světa. Vzniklé aplikace, které posouvají mobilní robotiku kupředu, vznikají v hojném počtu jako universitní práce či to jsou jen výsledky z mnoha robotických soutěží.

Jeden takový universitní projekt začal vznikat v roce 2007 na prestižní americké universitě Stanford, který následně v roce 2008 převzal robotický výzkumný inkubátor Willow Garage, který je mimo jiné tvůrcem světově známého nástroje OpenCV (Open Source Computer Vision). Projekt dostal název ROS - Robot Operating System, kde z přímého překladu je patrné, že by se mělo jednat o operační systém, ale skutečnost je taková, že ROS není plnohodnotným operačním systémem a tak je spíše nazýván jen jako framework. Od prvopočátku byl ROS distribuován jako *open source*, tudíž zdarma, a i díky tomu se každým rokem rozrůstal a stal se rapidně populárním. Dostal se do stavu, kde jeho možnosti dnes využívají nejpřednější světové university a instituce zabývající se robotikou. Vzniklá komunita zaujímá co do počtu přes 100 subjektů z celého světa (viz Obr. 3), kteří mimo jiné vzájemně spolupracují a podílí se na neustálém vývoji. Zajímavým faktem je také to, že tento framework sekundárně začala používat i sféra průmyslové robotiky, kde mluvíme o projektu ROS Industrial (viz kapitola 2.2). Členy tohoto projektu jsou známé firmy jako ABB, BMW, Boing, Ford či Jonh Deere [1].


*Obr. 1 – Logo projektu ROS. [1]*

Z důvodu velké popularity v mobilní robotice a neznalosti zmíněného systému ROS, který nebyl doposud v žádné české literatuře analyzován, byl tento systém hlavním předmětem zkoumání této práce. A jelikož veškerá dokumentace jsou rozsáhlé webové stránky v anglickém jazyce, záměrně byl proto zvolen jazyk práce český.

Hlavním cílem této diplomové práce je navrhnout a realizovat senzorický systém (testovací robotická aplikace), který bude využívat možnosti uvedeného frameworku. V soustavě senzorů bude disponovat ultrazvukový dálkoměr SRF (Sonic Range Finder) komunikující pomocí sběrnice $I^2C$ (Inter-Integrated Circuit) s jednodeskovým počítačem. Naměřená data budou přenášena pomocí TCP/IP

(Transmission Control Protocol/ Internet Protocol) protokolu do nadřazeného počítače. Výsledné řešení bude následně zhodnoceno z hlediska použitelnosti v mobilní robotice.

Sekundárním cílem práce je vytvořit rychlý a srozumitelný průvodce pro osoby, které nemají žádné zkušenosti se zmíněným robotickým systémem. Proto v práci je detailně popsáno krok po kroku jak systém správně nainstalovat, nastavit a ovládat pro bezproblémový chod.

Smyslem práce je předat čtenáři potřebné informace, tak aby byl schopen co nejrychleji používat framework ROS, a tím co nejvíce urychlil vývoj vlastní robotické aplikace. Požadavkem na čtenáře je alespoň minimální znalost unixových operačních systémů a dobrá obecná znalost informačních technologií.

Práce je rozdělena hned na několik částí. V první fázi práce je provedena obecná analýza se zaměřením na motivaci k používání robotického frameworku. Dále je text zaměřen na specifikaci použitého softwaru a hardwaru, který byl použit ve výsledné testovací robotické aplikaci. Ve specifikaci softwaru se práce nejvíce detailně zaměřuje na framework ROS. V další části textu je popsán přístup a metody, které vedly k dosáhnutí konečného řešení. V předposlední kapitole práce jsou navíc uvedené další praktické zkušenosti s uvedeným robotickým systémem. Závěrem je hodnoceno výsledné řešení a obecná použitelnost robotické software platformy ROS při vývoji robotických aplikacích.

# 2. Jak začít vytvářet robotickou aplikaci?

V té chvíli, kdy vznikne nápad, je připravený hardware (roboti) a promyšlený robotický projekt, nastává úkol začít „oživovat“ vlastní či zakoupené roboty. Zmíněné oživování lze přeformulovat do tvaru začít programovat, vymýšlet algoritmy. Rozsah programování je v každém robotický projektu individuální, záleží jaký software je potřeba vytvořit. Zda je nutné vše naprogramovat od „nuly“ (vlastní ovladače pro senzory, motory atd.) nebo již stačí začít implementovat algoritmy např. pro lokalizaci robota.

Dnešní trend vede programátory robotů k tomu, aby více a více začali používat software, který byl již vytvořený a otestovaný, čímž není potřeba ztrácet čas vytvářením vlastního softwaru. Mluvíme o používání nejrůznějších robotických nástrojů, knihoven nebo komplexních robotický frameworků.

Zmíněné robotické frameworky představují systémy, které kombinují nejrůznější robotický software, jako jsou např. ovladače pro senzory, motory nebo celé roboty, nástroje pro 2D/3D simulaci, zpracování obrazu, plánování pohybu, mapování, navigaci, lokalizaci atd. Jednoduše řečeno tyto frameworky se snaží co nejvíce usnadnit a urychlit realizaci robotického projektu. Zástupci těchto systémů jsou například: Player, YARP, Orocos, CARMEN, Orca, MOOS, Microsoft Robotics Studio a ROS [2]. Poslední zmíněný systém ROS je hlavní předmětem této práce, který byl předem vybrán a který je v následujících kapitolách podrobně popsán.


*Obr. 2 - Loga různých robotických frameworků [1][3][4][5][6][7][8][9].*

## 2.1 ROS – Robot Operating System

Nejprve budou zodpovězeny otázky co je a co nabízí framework ROS a proč ho začít používat? Následná praktická specifikace systému je provedena v kapitole 4.1.

### 2.1.1 Základní popis a design

Nepřehlédnutelnou skutečností je, že zmíněný systém se především v oblasti mobilní robotiky stal velice populárním a používaným. Tomu nasvědčuje veliká komunita aktivních uživatelů. Dle posledních statistik (září 2013) je oficiálních uživatelů tzv. *ros-users* přes 1500, více než 3300 osob podílející se na tvorbě online dokumentace,

která je umístěna na www.wiki.ros.org, a kolem 5700 lidí, kteří poskytují online podporu při problémech na www.answers.ros.org. [10]


*Obr. 3 – Mapa oficiální uživatelů ROS (university, robotické instituty). [10]*

Komunita se rozléhá do celosvětového měřítka, viz Obr. 3, výhodou projektu je, že je koncipován tak, aby se na vývoji mohl podílet takřka každý. Smyslem je navázat spojení a společně vytvořit systém s těmi nejlepšími lidmi v oboru. Tím se myslí navázaní spolupráce např. s universitou v Americe, která má experty na mapování ve venkovním prostředí, spolu s universitou v Evropě, která má specialisty na používání map pro navigaci robotů. Velkou výhodou celého projektu je, že podléhá tzv. *BSD* (Berkeley Software Design) licenci, což znamená, že celý systém je distribuován jako otevřený, volně šiřitelný software, tudíž zdarma. [10]

Dalším aspektem říkajícím o rozsáhlosti projektu je to, že jsou každoročně pořádány velké mezinárodní konference zvané ROSCon. Pro rok 2014 se konference bude konat ve městě Hong Kong nazývaná ROS Kong 2014. [11]


*Obr. 4 - Logo mezinárodní konference ROSCon pro rok 2014. [11]*

Primárním cílem celého projektu je podporovat znovupoužitelnost zdrojových kódů ve výzkumu a vývoji robotů.

ROS si klade za cíl stát se flexibilní universální robustní robotický software, který bude používán k implementacím nejrůznějších robotických problémů. ROS je navržen tak, aby byl co nejvíce flexibilní, modulární a distribuovaný.

Modularita dovoluje, aby si uživatel sám zvolil, kterou jeho část použije a kterou část si spíše vytvoří sám, což také poskytuje možnost použít ROS např. i do tzv. *embeeded* systémů s malou pamětí (viz kapitola 7.5).

Distribuovaný koncept umožnuje rozložit systém do více autonomních počítačů, které vzájemně komunikují, a které se celkově jeví jako jeden integrovaný systém. Jednotlivé počítače zpracovávají procesy neboli tzv. *uzly* (viz kapitola 4.1.7), které jsou vykonávány na zvolené stanici. Každý uzel je individuálně navržen a časově libovolně spouštěn za běhu celého systému. Aktuálně ROS pracuje jako tzv. *Master/Slave* systém, který se skládá z jednoho nadřazeného (Master) a z mnoha podřízených (Slave) pracovních stanic. Určitá skupina vývojářů pracuje i na Multi/Master provedení. [10]

Jak už bylo řečeno, ROS není operační systém (OS), tudíž musí být provozován pod nějakým plnohodnotným operačním systémem. Největší optimalizace a plná podpora z řad vývojářů je nastavena pro linuxový OS Ubuntu. Nicméně existují další tzv. experimentální verze ROS, které byly upraveny tak, aby mohly fungovat na jiných známých OS jako např. OS X, Windows, Raspbian, Fedora atd. (viz Obr. 5).

Díky velkému počtu podporovaných OS vzniká jedna z největších výhod, která nám dovoluje ROS framework používat na různých počítačových platformách. Systém je tedy označován jako multiplatformní. [2]


*Obr. 5 – ROS podporuje mnoho operačních systémů. [12]*

Další významnou výhodou ROS je podpora nejrůznějších sensorů, motorů, a také celých robotů. Dostupné jsou například ovladače pro různé 2D/3D laser skenery, kamery, gyroskopy, akcelerometry atd. Z hlediska robotů je aktuálně podporováno cca 32 komerčních robotů. [13]


*Obr. 6 – ROS podporuje mnoho senzorů a robotických platforem. [13]*

Díky multiplatformní možnosti a distribuovanému designu vzniká velká oblast použití. Pokud jednotlivé počítačové platformy jsou schopné z hlediska výkonnosti framework provozovat a mohou komunikovat protokolem TCP/IP, lze jednoduše tyto platformy spojit a sjednotit a vytvořit tím jeden globální řídicí systém. V praxi to dovoluje např. jednoduše vytvořit síť vzájemně komunikujících různorodých robotů.

Při vývoji robotické aplikace ROS dovoluje určitou nezávislost při výběru programovacího jazyka. Primární jazyky jsou C++ a Python, dále lze vyvíjet programový kód v jazyku LISP, Java a Lua. Samotné prostředí ROS umožnuje sestavení zdrojových kódů, interpretaci skriptů a běh programů. Dále je potřebná znalost značkovacích jazyků XML a YALM (YAML Ain't Markup Language).

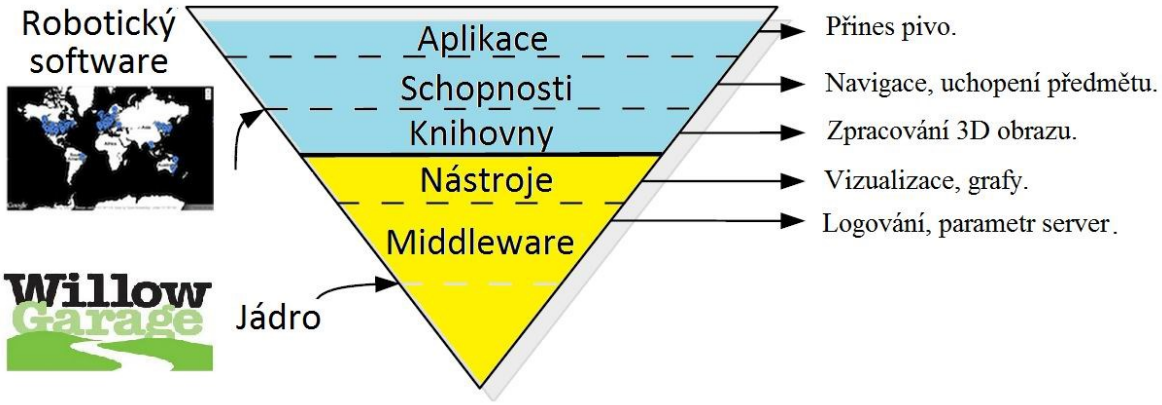
Další poznámkou je, že ROS není tzv. *real-time* systém, i když je ho možné integrovat s real-time kódem. Willow garage institut provedl také úspěšnou integraci s real-time frameworkem Orocos. [2]

Hlavním prostředkem, kterým se ROS prezentuje je webový prostor *www.ros.org*, který je moderně a přehledně zpracován. Slouží jako hlavní zdroj informací s kapacitou cca 22 000 webových stránek. Jsou zde umístěny velmi užitečné tutoriály jak pro začátečníky, tak i pro pokročilé uživatele. Dále je zde možné nalézt dokumentaci robotického softwaru, jako jsou knihovny, ovladače a jiné užitečné balíčky, které vznikly díky komunitě ROS.

ROS framework je každým dnem vyvíjen a vylepšován velkou skupinou lidí. Jejich aktivita je daná například tím, že v květnu 2014 bude vydána opět nová verze (viz kapitola 4.1.1). [10]

### 2.1.2 Struktura a funkce systému

Díky rozmanité struktuře je ROS nazýván termínem *ekosystém* (viz Obr. 7), který je rozdělen na dvě základní vrstvy: jádro a robotický software. Jádro je spravováno společností Willow Garage a software jako knihovny, schopnosti a aplikace zaměřené na robotiku jsou dílem ROS komunity. [14]




*Obr. 7 - Struktura ekosystému ROS.*

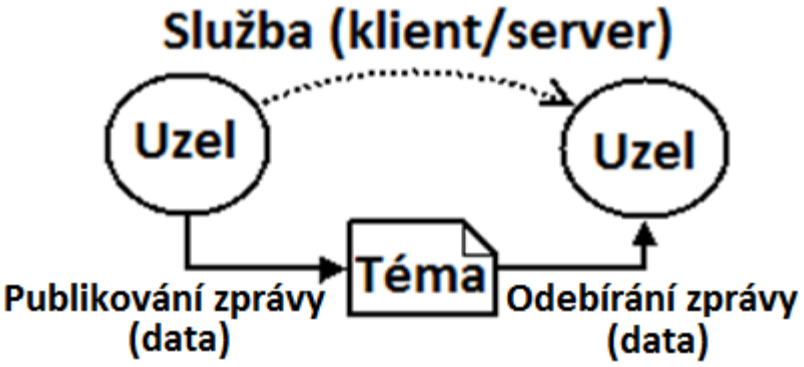
#### *2.1.2.1 Jádro*

Jak bylo již v úvodu specifikováno, ROS není operační systém, ale z hlediska jádra poskytuje služby podobně jako OS. Například obsahuje hardware abstrakci HAL (Hardware Abstraction Layer), která umožnuje snadno přenést jádro systému na různou platformu, bez jakéhokoliv zásahu do vnitřní implementace.

Dále obsahuje management balíčků, který je obdobně implementován jako u unixových systémů. ROS je v podstatě tvořen různějšími balíčky. Díky tomu může být systém zmenšen či zvětšen do libovolné velikosti. Balíčky obsahují uzly, které jsou základními stavebními prvky celého systému ROS. Správa a distribuce balíčků je prováděna pomocí tzv. *repositářů*, které představují převážně internetové servery, kde jsou balíčky uloženy.

Framework ROS patří do kategorie softwaru nazývaný termínem *middleware*. Middleware představuje softwarové lepidlo, které umožnuje propojení různého softwaru na různých platformách. A jako middleware poskytuje následující služby: zasálání zpráv mezi procesy, logování, parametr server, vzdálené volání procedur, nástroje, knihovny.

První služba, která je v robotice velmi důležitá, je komunikační systém (viz Obr. 8). Systém zasílání zpráv je ve frameworku postaven na mechanismu anonymního publikování/odebírání dat mezi procesy neboli uzly (viz kapitola 4.1.7). Výměna informací mezi uzly probíhá přes tzv. *téma* (viz kapitola 4.1.8), které dovoluje přijímat/odesílat zprávy v libovolné datové struktuře (viz kapitola 4.1.9). Aktuálně je transport dat podporován protokolem TCP/IP a UDP. [14]




*Obr. 8 – Komunikační model systému ROS.*

Další velmi užitečnou službou při vývoji nejrůznějších aplikací je možnost logování dat. ROS poskytuje jednoduché ovládání a spolehlivé ukládání/přehrávání proudů dat přenášené mezi procesy. Tuto službu zajišťuje nástroj jménem *rosbag* (viz kapitola 4.1.14).

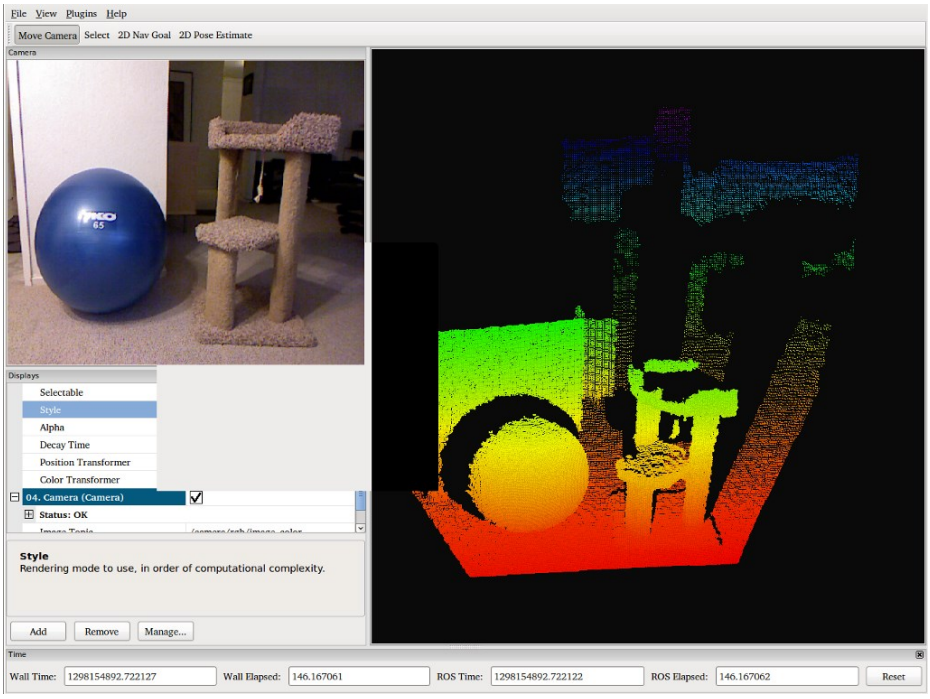
Distribuovaný parametr server je služba, která slouží jako globální skladovací prostor pro parametry různých datových typů. Každý aktivní uzel může uložit nebo získat hodnotu parametru na určité adrese. Server je převážně používán na konfiguraci uzlů za jejich běhu. Klíčové slovo je zde *rosparam* (viz kapitola 4.1.11).

Vzdálené volání procedur neboli RPC (Remote Procedure Calls) je možností jak provádět synchronní interakci typu požadavek/odpověď mezi procesy. Díky službě RPC vzniká komunikace typu klient/server (viz kapitola 4.1.9).

Primární způsob, jak ROS ovládat, je použití příkazové řádky jako terminál operačního systému. Je možno využít více než 45 tzv. *konzolových nástrojů*, které dovolují např. spouštět mnoho uzlů najednou, ovládat parametr server, provádět diagnostiku aktivních témat a služeb, zobrazovat senzorická data do grafů atd. Způsob ovládání konzolových nástrojů je popsán v kapitole 4.1.3.

Další možnosti ovládání ROS poskytují tzv. *grafické nástroje,* které lze spouštět v rámci grafického uživatelského rozhraní RQT (viz kapitola 4.1.12), které umožnuje např. zobrazit propojení všech spuštěných uzlů v systému, přehrávat data z logování nebo různě vykreslovat senzorická data do grafů v aktuálním čase. Příkladem může být zobrazení dat z enkodérů.

Hlavním představitelem z grafických nástrojů je RVIZ, který je schopen provádět 3D vizualizaci robotů popsaných v URDF (Unified Robot Description Format) formátu a např. zobrazit obraz z kamer a 3D obraz z laserových skenerů. [14]



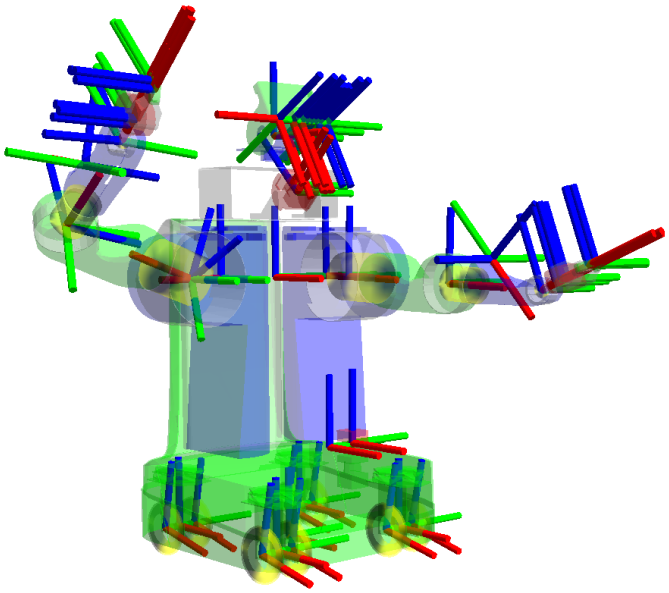

*Obr. 9 - Zobrazení 3D hloubkové mapy ze sensoru Microsoft Kinect pomocí grafického nástroje RVIZ. [14]*

### *2.1.2.2 Robotický software*

Z hlediska mobilní robotiky poskytuje prostředí ROS důležité robotické knihovny, které jsou fyzicky velké sady balíčků. Tyto knihovny pomáhají při řešení základních robotických problémů, jako je např. lokalizace v mapě, vytváření map, navigace, odhad 3D polohy, diagnostika atd.

Součástí je také knihovna obsahující standardizované robotické zprávy pro senzory, geometrii, navigaci atd. Smyslem je zpřehlednit komunikaci a vytvořit bezproblémovou spolupráci vlastního projektu s celým ekosystémem ROS.

Dále je k dispozici knihovna geometrie, která umožnuje sledovat polohu různých částí robota. Použití nastává, když je např. potřeba kombinovat data z kamery a laser skeneru, kde je nezbytné vědět vzájemnou polohu senzorů. Tato knihovna poskytuje zpracovávat aktuální transformaci mnoha souřadnicových soustav z dat senzorů. Praktický příklad zobrazuje Obr. 10. [14]



*Obr. 10 - Sledování mnoha souřadnicových soustav robota pomocí knihovny geometrie. [14]*

Další užitečný software je sada knihoven dovolující popis a modelování robotů ve formátu URDF. Charakteristika robota je dána XML (Extensible Markup Language) dokumentem, ve kterém lze popsat jednotlivé součásti robota z hlediska velikosti a polohy. Výhodou je, že model robota vytvořený dle URDF lze následně využít v RVIZ vizualizaci, v simulaci, při plánování pohybu a také s knihovnou geometrie. [14]

ROS jako komplexní systém poskytuje integraci dalších samostatně existujících robotický nástrojů. V systému lze využít možností knihoven jako je Gazebo, OpenCV, Poitn Cloud Library (PCL) a MoveIt.

Gazebo je robotický 3D grafický simulátor, který umožnuje nasimulovat vlastní robotickou aplikaci v plné míře. Gazebo nabízí robustní fyzikální engine, kvalitní grafické a programové rozhraní. Propojení mezi ROS a Gazebo je vytvořeno pomocí zásuvných modulů (plugins), kde tyto moduly poskytují mnoho existujících robotů a sensorů. Díky tomu, že zásuvné moduly mají stejný MPI (Message Passing Interface) jako celý ROS ekosystém, můžeme vytvářet ROS uzly, které jsou kompatibilní se simulací, daty a hardwarem. Gazebo umožnuje nejdříve virtuálně navrhnout a otestovat svoji robotickou aplikaci, kterou je posléze možné jednoduše implementovat do reálného světa.

PLC je knihovna zaměřena na vnímání objektů v prostoru. Využívá se při manipulaci s předměty nebo při zpracování 3D dat. Poskytuje mnoho algoritmů pro filtrování, registraci nebo detekci prvků v troj-rozměrném obrazu. Vstupní data do této knihovny mohou zajistit např. 3D laser skenery.

OpenCV je světově známý a výkonný nástroj na zpracování obrazu. Systém ROS dovoluje uživatelům jednoduše přenést data z různých kamerových modulů do OpenCV algoritmů takových jako segmentace a sledování. Framework ROS poskytuje knihovnu *image_pipeline*, která může být použita pro kalibraci kamery, monokulární, stereo nebo pro hloubkové zpracování obrazu.

MoveIt je knihovna pro plánování pohybu robotů na bázi tzv. *art* plánovacích algoritmů. Data plánování mohou být zobrazena v ROS pomocí vizualizačních nástrojů

jako RVIZ nebo RQT. Knihovnou lze ihned použít na robotech, které jsou oficiálně skupinou ROS podporovány. [15]


*Obr. 11 – Softwarové knihovny integrované v ROS.[15]*

## 2.2 ROS-Industrial

Zajímavý směr, do kterého se projekt ROS ubírá, je oblast průmyslových robotů. Toto odvětví se nazývá ROS-Industrial.


*Obr. 12 - Logo projektu ROS-Industrial. [16]*

ROS-Industrial je program, který podléhá BSD licenci a který v podstatě rozšiřuje možnosti normální verze softwaru ROS. Navíc obsahuje knihovny, nástroje a ovladače zaměřené na průmyslové roboty. Tento projekt je zvlášť vyvíjen konsorciem, ve kterém zaujímá členství mnoho mezinárodních společností a výzkumných institucí, viz Obr. 13. Členství je podmíněno vstupním poplatkem. [16]

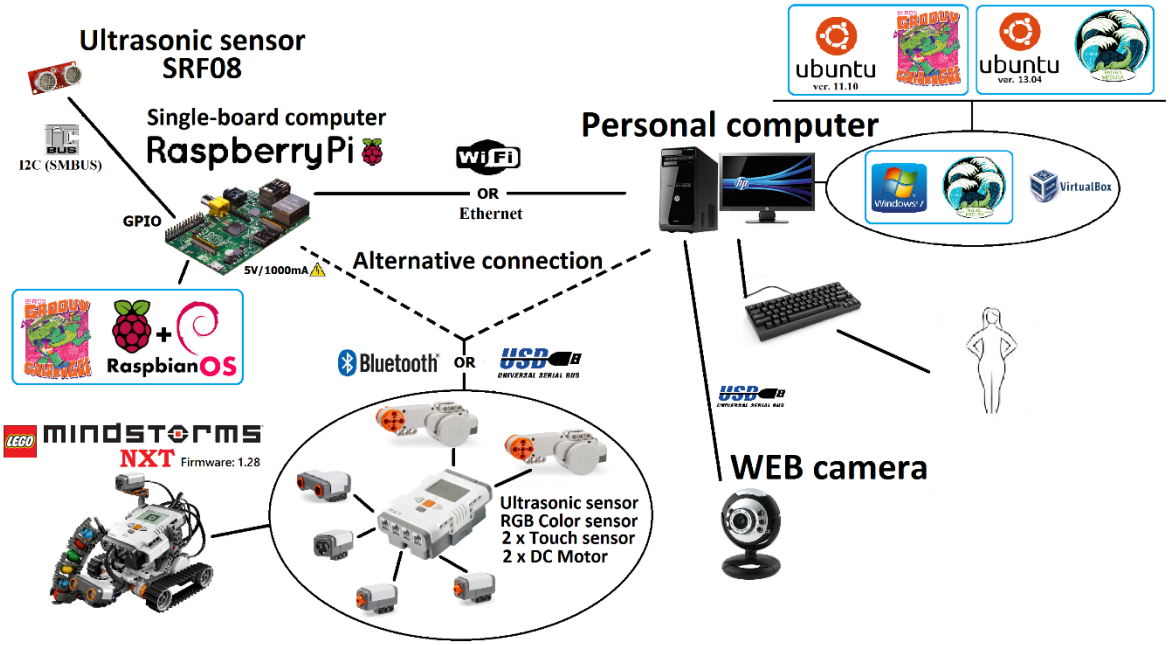

*Obr. 13 – Firmy a instituty zapojené do projektu ROS-Industrial. [16]*

# 3. Specifikace – hardware

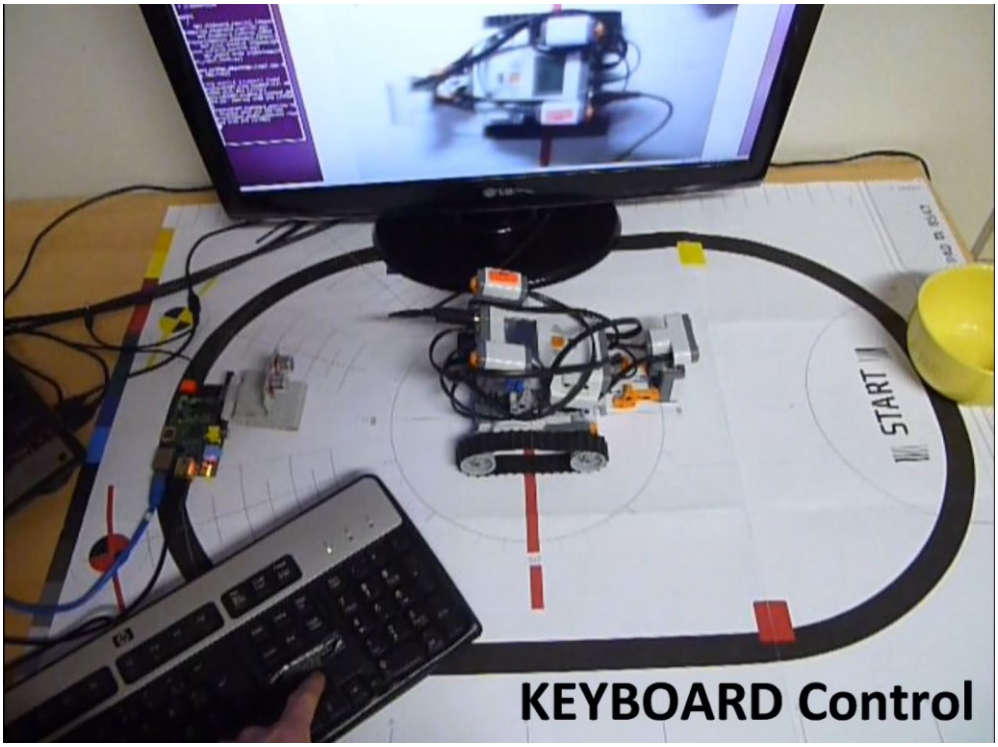
Tato kapitola se zabývá popisem jednotlivých hardware zařízeních, které byly použity v testovací robotické aplikaci této diplomové práce.

Navržená testovací aplikace s mobilním robotem podléhala požadavkům zadání diplomové práce. Kompletní design byl navržen tak, aby co nejvíce demonstroval možnosti ROS frameworku.

Architektura vzájemného propojení jednotlivých komponent je znázorněna na Obr. 14 a reálné rozmístění komponent je zřejmé z Obr. 15. Mobilní robot LEGO NXT byl primárně připojen do mini-počítače Raspberry Pi a sekundárně mohl být připojen do nadřazeného počítače. Text práce je zaměřen na primární zapojení robota.




*Obr. 14 - Architektura senzorického systému (testovací robotické aplikace).*




*Obr. 15 - Fotografie reálného rozmístění hardware komponent testovací robotické aplikace.*

## 3.1 Ultrazvukový dálkoměr SRF

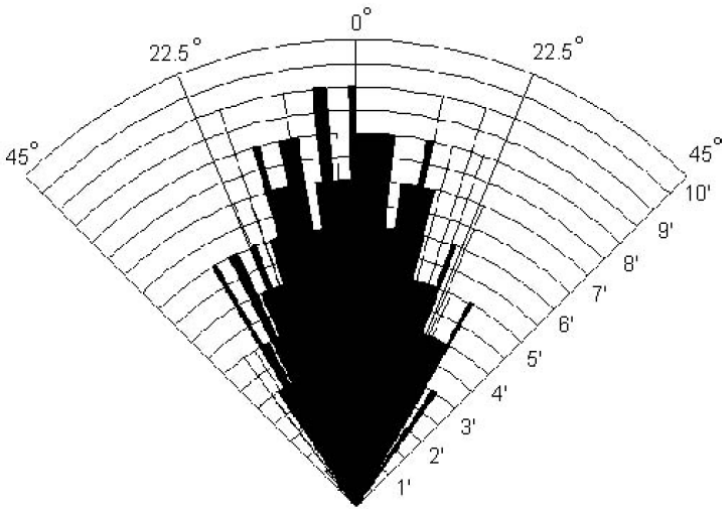
V testovací aplikaci byl použit jako hlavní sensor ultrazvukový dálkoměr SRF ve specifikaci SRF08. Sloužil k detekci překážek a byl připojen do mini-počítače Raspberry Pi pomocí $I^2C$ sběrnice.

Tento snímač je schopen měřit vzdálenost objektů, a také světelnou intenzitu. Při drátování sběrnice $I^2C$ je nutné, aby kanály SCL a SDA byly propojeny tzv. *pull-up* rezistorem proti napájení. Nejdůležitější parametry a specifikace jsou uvedeny v Tab. 1. [17]

| | |
|---|---|
| Napájení napětí | 5 [V] |
| Snímací frekvence | 40 Hz |
| Komunikace | $I^2C$ sběrnice |
| Proudový odběr | 15mA, pohotovost: 3mA |
| Snímací rozsah - max | 6 m |
| Snímací rozsah - min | 3 cm |
| Světelná intezita - max | 248 (jasné světlo) |
| Světelná intezita - min | 2-3 (úplná tma) |
| Rozměry | 43x20x17mm |

*Tab. 1 – Specifikace senzoru SRF08. [17]*

*Obr. 16 - Ultrazvukový dálkoměr SRF08.[17]*




*Obr. 17 - Rozptyl paprsku ultrazvukového dálkoměru SRF08. [17]*

## 3.2 Kamera

Senzorický systém obsahoval běžnou webovou kameru disponující komunikační sběrnicí USB (Universal Serial Bus) ve verzi 2.0.

Kamera byla umístěna stacionárně a sloužila pro snímání pohybu robota. Připojena byla do nadřazeného počítače.

*Obr. 18 – USB webkamera.*

## 3.3 Nadřazený počítač

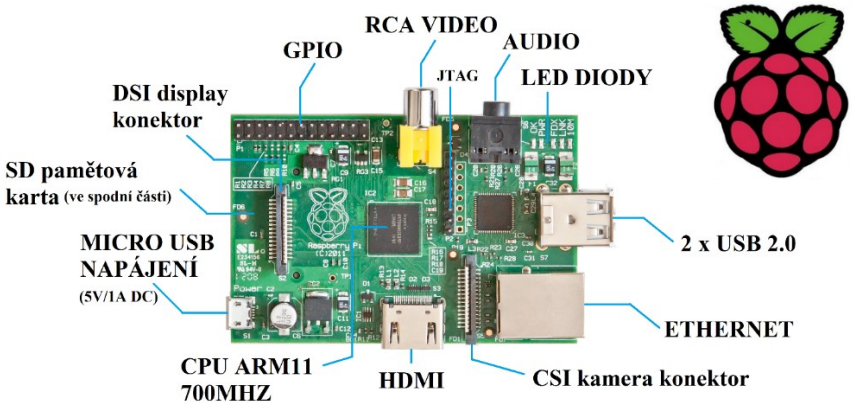
Pro hlavní výpočty byl použit běžný multimediální notebook jako nadřazený (Master) počítač, který byl osazen dostatečně výkonným dvoj-jádrovým procesorem Intel i3 380 a 4GB operační pamětí. Tento počítač disponoval třemi operačními systémy, kde dva z nich běžely na notebooku jako virtuální stroje.

Do notebooku byla připojena web kamera a externí klávesnice pro ovládání robata. Komunikace s jednodeskovým počítačem probíhala skrze Ethernet. Hlavní funkcí počítače bylo přijímání dat ze senzorů s následným vyhodnocením a vytvořením příkazů pro ovládání mobilního robota.

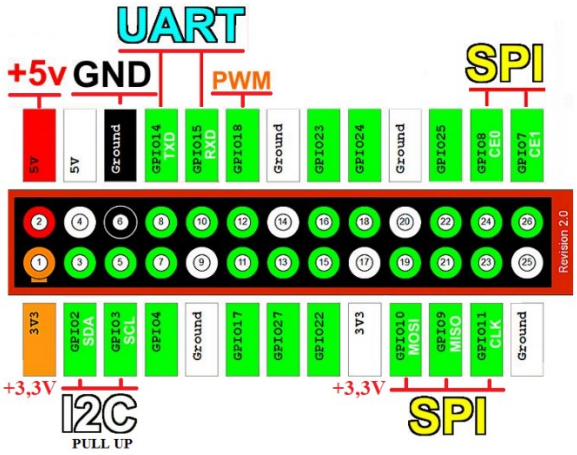
## 3.4 Jednodeskový počítač

Jako jednodeskový počítač byl použit Raspberry Pi (RPi) – model B (viz Obr. 19), který je velmi populární a snadno dostupný. Pořizovací cena se pohybuje kolem 1000 Kč. Tento mini-počítač disponuje procesorem ARM11 o taktu 700MHz s pasivním chlazením a operační pamětí 512 MB. Výkon celé sestavy je dostačují pro běžnou kancelářskou práci, internet, grafické aplikace postavené na OpenGL (Open Graphics Library) ve verzi 2.0 a přehrávání videa ve vysokém rozlišení. Jako pevný disk slouží flash SD (Secure Digital) paměťová karta o minimální velikosti 4GB. Přednosti tohoto počítače jsou vstupní/výstupní rozhraní jako 2 x USB 2.0 pro připojení např. klávesnice a myši, Ethernet pro připojení do počítačové sítě, HDMI (High-Definition Multi-media Interface) pro spojení s monitorem a zvukový výstup. Dále lze k modulu připojit nejrůznější zařízení díky GPIO (General Purpose Input Output) rozhraní. Na jednotlivých pinech najdeme $I^2C$, SPI (Serial Peripheral Interface) a UART (Universal Asynchronous Receiver Transmitter), viz Obr. 20. [18]

Počítač Raspberry Pi sloužil pro zpracování dat z SRF senzoru, která byla následně přeposílána do nadřazeného PC (Personal Computer) na vyhodnocení. Dále do RPi byl připojen robot pomocí USB sběrnice, kde data ze sensorů robota byla přeposílána na vyhodnocení do nadřazeného počítače. Zároveň mini-počítač čekal na pokyny ze strany Master stanice, které po přijmutí řídily motory robota. Touto komunikací bylo RPi nastaveno jako Slave zařízení.




*Obr. 19 – Jednodeskový počítač Raspberry Pi.*




*Obr. 20 – Rozmístění jednotlivých komunikačních sběrnic na GPIO rozhraní. [18]*

## 3.5 LEGO Mindstorms NTX kit 2.0

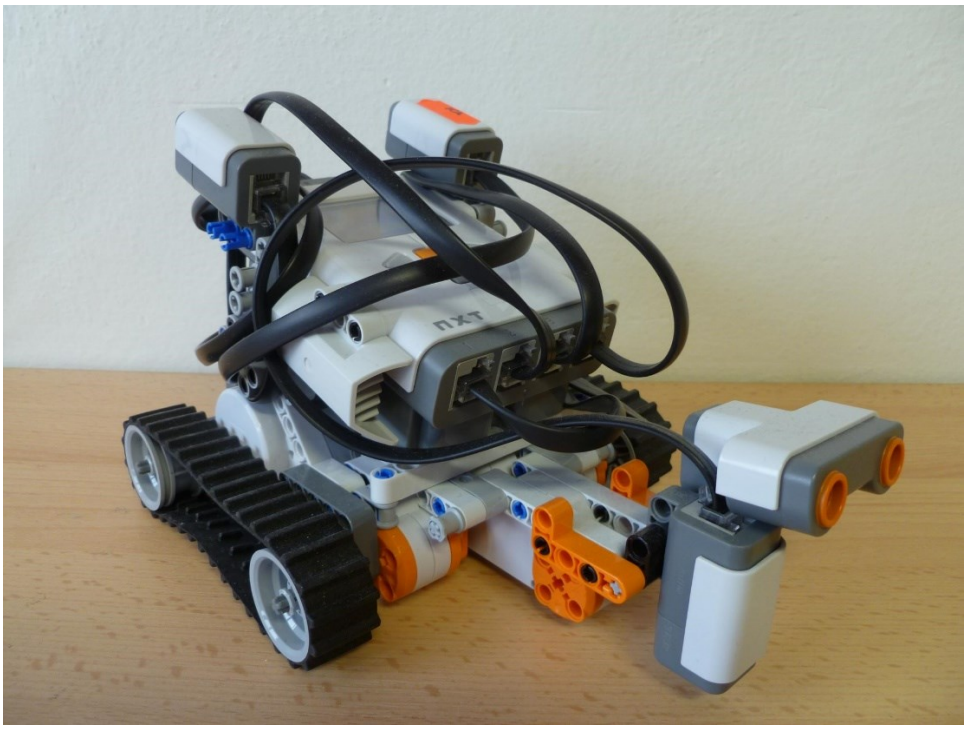
LEGO Mindstorms NTX je robotická stavebnice představená roku 2009, ze které lze sestavit různé kolové či pásové mobilní roboty. Specifikace stavebnice NXT je charakteristická svojí řídící jednotkou, která je na bázi 32-bitového mikrokontroléru Atmel AT91SAM7S256. Komunikace s řídící jednotkou je možná pomocí USB sběrnice nebo bezdrátovou technologií Bluetooth. Stavebnice nabízí různé senzory a motory.

Stavebnice poskytla postavit pásového mobilního robota (viz Obr. 24), který byl poháněn dvěma motory. Robot byl osazen ultrazvukovým sensorem pro detekci překážek, barevným RGB senzorem pro pohyb po černé čáře a dvěma koncovými spínači, které sloužily jako uživatelský interface [19]. Robot byl primárně připojen k počítači Raspberry Pi pomocí USB sběrnice, kde řídící pokyny byly zasílány do RPi z nadřazeného počítače.


*Obr. 21 – Řídící jednotka robotické stavebnice LEGO NXT 2.0. [19]*

*Obr. 22 – Sensory robotické stavebnice LEGO NXT 2.0 (Koncový, RGB barevný a ultrazvukový senzor). [19]*

*Obr. 23 – DC motor robotické stavebnice LEGO NXT 2.0. [19]*



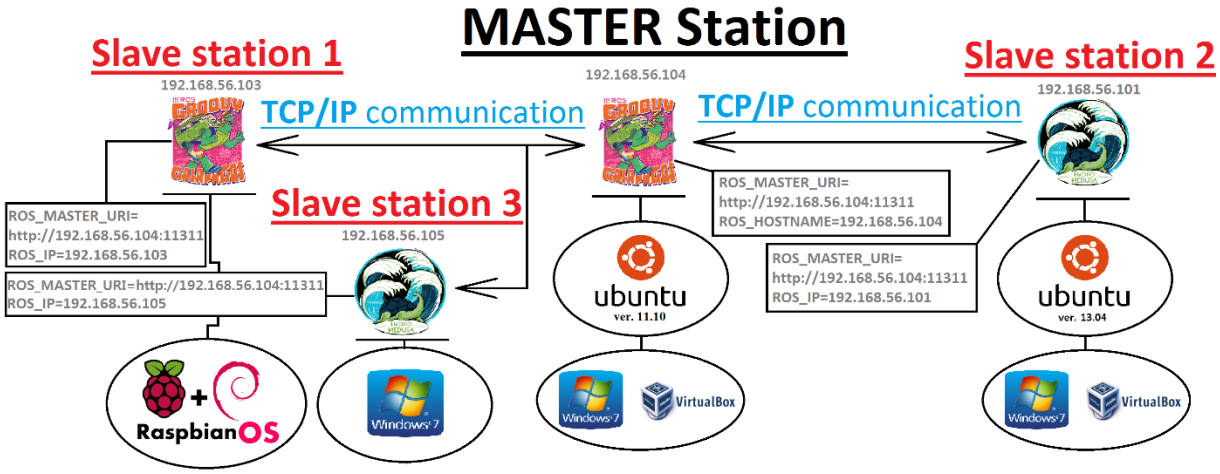

*Obr. 24 - Mobilní robot postavený z robotické stavebnice LEGO NTX 2.0.*

# 4. Specifikace – software

Touto kapitolou je charakterizován software, který byl použit při vývoji testovací robotické aplikace v této diplomové práci. Analýza je nejvíce zaměřena na robotický framework ROS, který přestavuje hlavní software (middleware) celé práce.

Design software architektury byl navrhnut s myšlenkou ověřit ROS framework jako distribuovaný systém. Proto byly navrhnuty čtyři software platformy, které plnily specifické úkoly, a které komunikovaly dle Master/Slave modelu. Tyto platformy tvořily jeden řídicí systém založený na systému ROS. Software architektura testovací aplikace se skládala ze čtyř platforem, kterým odpovídaly čtyři instalace ROS (dvě verze). Jednotlivé platformy disponovaly operačními systémy Raspbian, Windows a dvěma verzemi Ubuntu, které byly nainstalovány virtuálně na Windows pomocí nástroje Oracle VM VirtualBox.

Architektura komunikace mezi jednotlivými platformami je znázorněna na Obr. 25.




*Obr. 25 – Software architektura testovací robotické aplikace.*

## 4.1 ROS

Tato podkapitola popisuje systém ROS z praktického hlediska. Obecná charakteristika je uvedena v kapitole 2.1. V textu jsou vysvětleny základní pojmy, terminologie, vlastnosti a způsoby ovládání systému ROS, které jsou nezbytně nutné pro uživatele typu začátečník.

ROS představoval hlavní software - middleware, který umožnil spojení jednotlivých software platforem. Framework byl nainstalován dvakrát v distribuci Groovy a dvakrát ve verzi Hydro.

### 4.1.1 Verze ROS

Díky neustálému vývoji a sedmi leté historii se ROS vyskytuje v hned několika distribucích. K dnešnímu datu (květen 2014) rozlišujeme 5 verzí (Box, C, Diamondback, Electric a Fuerte), které nejsou dále vyvíjeny a 2 verze (Groovy a Hydro), které jsou aktuálně podporovány vývojáři. Vydání nadcházející verze je plánováno na jaro 2014, která se bude jmenovat Indigo.

Důležitým faktem je, že jednotlivé verze nejsou mezi sebou zcela kompatibilní a každá verze podporuje jen určitý počet operačních systémů a robotů. V praxi je volena logicky verze, co nejnovější, ale pokud operační systém nebo robot není podporován nejaktuálnější verzí, je nezbytné zvolit příslušnou předchozí distribuci. [20]


*Obr. 26 – Chronologicky seřazené jednotlivé distribuce ROS od nejstarší po nejnovější (Box, C, Diamondback, Electric, Fuerte, Groovy, Hydro, Indigo). [20]*

### 4.1.2 Instalace ROS

V té chvíli, kdy je k dispozici počítačová platforma s určitým OS, nastává možnost instalace ROS frameworku online z repositáře. Jelikož průběh instalace je dán kombinací OS a verze ROS je nutné postupovat dle pokynů uvedených na [12]. Text práce je dále zaměřen na OS Ubuntu, jelikož přestavuje hlavní operační systém, pro který je framework ROS určen.

Instalaci provádíme v příkazovém řádku terminálu OS. Hlavní příkaz zobrazený níže nainstaluje plnou verzi ROS Hydro.

```
$ sudo apt-get install ros-hydro-desktop-full
```

Dále musí být po instalaci zadán příkaz (viz níže), který provede konfiguraci systému ROS s operačním systémem.

```
$ source /opt/ros/<verze_ROS>/setup.bash //verze_ROS = hydro, groovy atd.
```

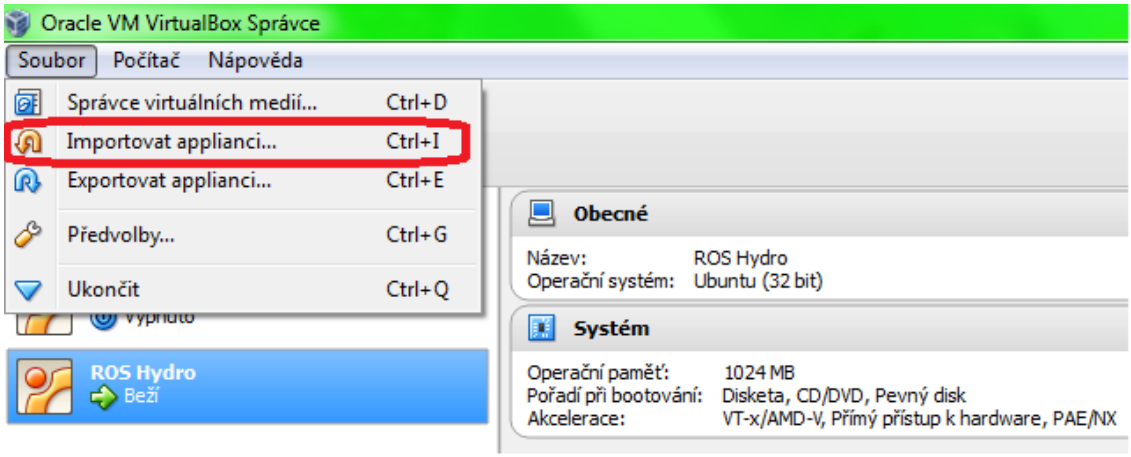
Důležitým krokem po instalaci je editace skrytého souboru *.bashrc* (skrytý soubor), který je potřebný pro konfiguraci uživatelského rozhraní (*shell*) *bash* operačního systému. Obsah souboru je dále nutné měnit v případě vytvoření tzv. *pracovního prostředí* (viz kapitola 4.1.5) nebo při vytváření sítě počítačových platforem (viz kapitola 4.1.16).

Příkaz níže provede nahrání příkazu pro konfiguraci ROS do souboru *.bashrc*. Následkem příkazu je propojení ROS s operačním systém při každém spuštění příkazového terminálu (bash).

```
$ echo "source /opt/ros/<verze_ROS>/setup.bash" >> ~/.bashrc
$ sudo nano ~/.bashrc   // Manuální editace souboru.
```

Další alternativou instalace ROS je použití předinstalovaného OS Ubuntu a ROS s kombinací nějakého virualizačního nástroje např. Oracle VM VirtualBox (viz kapitola 4.2). Instalace spočívá pouze ve stažení souboru OVA (Open Virtual Appliance) s koncovou příponou *ova*, ve kterém se nachází předinstalovaný ROS na OS Ubuntu. [12]

Nejaktuálnější OVA soubor obsahuje předinstalovaný virtuální systém Ubuntu 12.04 LTS (Precise) s ROS Hydro o velikosti 3,7 GB. Uživatelské jméno a heslo je „viki“. Po provedení importu v nástroji VirtualBox (viz Obr. 27) instalace zaujímá paměťové místo o velikosti 8 GB. Poté lze virtuální stanici už jen spustit a začít používat ROS. OVA soubor je dostupný z webových stránek zde [21]. Tato instalace je velice rychlá.




*Obr. 27 - Import souboru OVA v prostředí Oracle VM VirtualBox.*

### 4.1.3 Ovládání a spuštění ROS

Primárně k ovládání systému se používají tzv. *konzolové nástroje*. Sekundárně lze k ovládání využít možností grafického uživatelského rozhraní RQT (viz kapitola 4.1.12).

Konzolové nástroje se spouštějí pomocí příkazů, které jsou zadávány do příkazové řádky operačního systému. Příkaz je tvořen názvem nástroje a argumenty, které definují činnost nástroje. Každý nástroj má určitý počet argumentů (arg.). Velmi užitečným tipem pro práci v příkazové řádce je automatické doplňování textu příkazu při opakovaném stlačení tabulátorové klávesy na klávesnici.

Vzor příkazu pro ovládání systému ROS.

```
$ <název_nástroje> <arg1> <arg2> .. <argN>
```

Nejpoužívanější konzolové nástroje pro práci s ROS jsou zobrazeny v Tab. 2. [22]

| Název nástroje | Popis funkce |
|---|---|
| roscore | Spuštění jádra ROS. |
| roslaunch | Spouštění mnoha uzlů zároveň (lokálně, vzdáleně), správa parametr serveru, nastavení systémových proměnných ROS. |
| rosrun | Spouštění uzlů jednotlivě a lokálně. |
| rostopic | Diagnostika témat. |
| rosnode | Diagnostika uzlů. |
| rosmsg, rossrv, rosservice | Diagnostika standartních a služebních zpráv. |
| roslocate | Lokalizace balíčků. |
| roscd | Pohyb mezi balíčky. |
| rosparam | Správa parametr serveru. |
| roswtf | Ladění systému ROS. |
| rosbag | Logování dat. |

*Tab. 2 - Nejpoužívanější konzolové nástroje při práci s frameworkem ROS. [22]*

Jeden z nejzákladnějších příkazů je pokyn na spuštění systému. Tento příkaz je tvořen jen názvem nástroje, který zajistí spuštění mnoha balíčků tj. jádra. Nastartování ROS se provede, pokud do příkazové řádky napíšeme klíčové slovo (bez $):

```
$ roscore
```

Druhý způsob nastartování jádra a také ovládání systému poskytuje nástroj *roslaunch*. [23]

Vzor příkazu:

```
$ roslaunch <název_balíčku> <název_spouštěního_souboru>
```

V první argument příkazu je potřeba zadat název balíčku (viz. Kapitola 4.1.6), ve kterém je uložený tzv. *spouštěcí soubor* (druhý argument příkazu). Před použitím nástroje roslaunch je tedy nutné vytvořit spouštěcí soubor s příponou *launch*, který podléhá syntaxi značkovacího jazyka XML. Zdrojový kód souboru je tvořen značkami neboli tzv. *tagy*, kde každý má své jméno a specifikou funkci. Při konstrukci značek je nutná znalost tzv. *atributů*, které provádějí nastavení tagů. Pro základní funkci určité značky je nutná konfigurace tzv. *povinných* atributů a pro rozšířenou funkci slouží tzv. *volitelné* atributy. Detailní charakteristika značek a jejich atributů je uvedena zde: [23].

Seznam nejčastěji používaných značek je zobrazen v Tab. 3.

| Značka (tag) | Povinné atributy | Volitelné atributy | Funkce značky |
|---|---|---|---|
| <launch> | - | - | Hlavní značka. Definice oblasti pro ostatní tagy. |
| <node> | pkg, name, type | args, machine, respawn, required, ns, clear_params, output, cwd, launch-prefix | Spouštění uzlů. |
| <machine> | name, address,env-loader | default, user, password, timeout | Definice stanic tvořící distribuovaný systém. |
| <param> | name | value, type, textfile, binfile, command | Konfigurace parametr serveru. |
| <rosparam> | command, file, param | ns, subst_value | Nahrání parametr souboru na parametr server. |
| <env> | name, value | - | Nastavení systémových proměnných. |

*Tab. 3 – Hlavní značky a atributy pro vytvoření spouštěcího souboru. [23]*

Příklad spouštěcího souboru (jazyk XML):

```
<launch>

 <machine name="Machine1" address="10.0.0.1" evn-loader="opt/ros/hydro/en
v.sh"> </machine> <!-- Definice externí stanice.-->

 <node pkg="package" type="node1.py" name="Node1" machine="Machine1">
  <!-- Spouštění uzlu na externí stanici. -->
        <rosparam command="load" file="$(find package)/robot.yaml" />
         <!--Nahrání parametr souboru na parametr server. -->
 </node>

 <node pkg="package" type="node2" name="Node2" respawn="true" output="scr
een" >
 <!-- Spuštění uzlu na lokální stanici-->
        <param name="string_parameter" type = "string" value="none"  />
          <!-- Nahrání jednoho parametru na parametr server. -->
 </node>

 </launch>
```

### 4.1.4 Build systémy Catkin, Rosbuild

Další pojem, který je potřeba ovládat je tzv. *build systém*. Zjednodušeně řečeno build systém nám určuje, jakým způsobem jsou sestavovány balíčky v systému ROS. Aktuálně lze rozlišit dva: starší *Rosbuild* a nový tzv. *Catkin*, který byl poprvé plně implementován ve verzi ROS Groovy.

V praxi je tedy nezbytné rozlišit balíčky podle toho v jakém build systému byly vytvořeny. Rosbuild balíček je nazýván jako *dry* a Catkin balíček jako *wet*. Dle build systému je rozlišeno i tzv. *pracovní prostředí* (viz následující kapitola). Pravidla jsou taková, že dry balíček je možné umístit jen do Rosbuild pracovního prostředí a naopak wet balíček jen do Catkin pracovního prostředí.

Je možné používat oba build systémy zároveň, ale je doporučeno používat jen Catkin build systém. Z tohoto důvodu se text práce zaměřuje jen na nejnovější verzi. [24]

### 4.1.5 Pracovního prostředí (workspace)

Aby bylo možné vytvářet vlastní aplikace v ROS tj. tvorba vlastních balíčků a uzlů, je nutné nejdříve vytvořit uživatelské pracovní prostředí. To vznikne, pokud jsou provedeny následující kroky, viz níže (Catkin pracovní prostředí).

```
$ mkdir -p ~/catkin_ws/src    //Vytvoření adresářů.
$ cd ~/catkin_ws/src          //Přemístění do adresáře.
$ catkin_init_workspace       //Inicializace pracovního prostředí.
$ cd ~/catkin_ws/             //Přemístění do adresáře.
$ catkin_make                 //Sestavení pracovního prostředí.
$ source devel/setup.bash     //Konfigurace pracovního prostředí.
```

Po dokončení instalace prostředí vzniknou v adresáři *catkin_ws* další adresáře, ze kterých je nepodstatnější složka *src*, do které se umísťují balíčky.

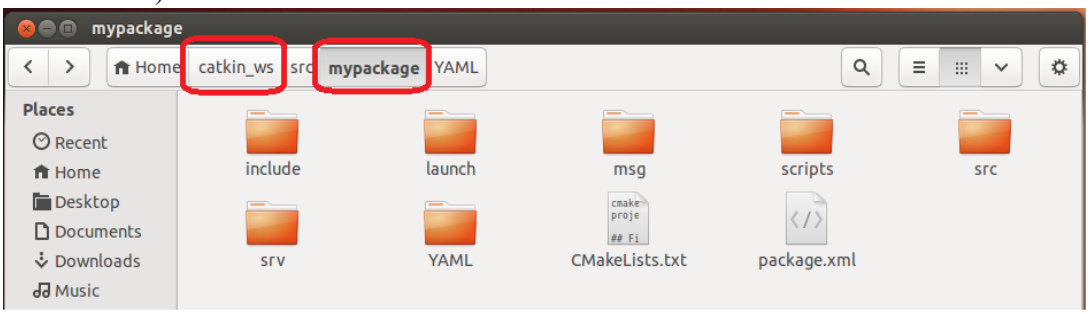
Sestavení prostředí, které provede sestavení všech balíčků v něm obsažených, je nutné provádět v případech, pokud je potřeba provést kompilaci uzlů napsaných v jazyce C++ a nebo když je provedena editace souboru *CMakeLists.txt*, který se nachází v kořenovém adresáři každého balíčku.

Další důležitou věcí je provést editaci souboru *.bashrc*. Příkazem níže proběhne vložení příkazu pro konfiguraci pracovního prostředí s jádrem ROS do zmíněného souboru. Následkem této operace je propojení pracovního prostředí s ROS při každém spuštění příkazového terminálu (bash). [25]

```
$ echo "source ~/catkin_ws/devel/setup.bash " >> ~/.bashrc
```

### 4.1.6 Balíček (package)

Balíček je hlavním prostředkem pro sdílení softwaru v rámci celé komunity ROS. Celé sady balíčků tvoří převážnou část systému ROS. Balíčky správně vytvořené a ověřené komunitou jsou umístěny v repositářích, ze kterých se mohou snadno šířit mezi ostatní uživatele. Balíček má definovanou strukturu, kterou je možné vidět na Obr. 28 (Catkin balíček).




*Obr. 28 – Struktura adresářů Catkin balíčku se jménem „mypackage“ v pracovním prostředí „catkin_ws“.*

Základem správně vytvořeného a fungujícího balíčku Catkin jsou soubory *CMakeLists.txt* a *package.xml*, které jsou umístěny v kořenovém adresáři balíčku. Soubory vzniknout automatiky při vytváření balíčku, viz kapitola 4.1.6.1.

Soubor *package.xml* obsahuje základní informace o balíčku, který podléhá XML struktuře. Je nutné zde uvést jméno, verzi, popis funkce, druh licence a autora balíčku s jeho kontaktem. Dále musí obsahovat souhrn tzv. *závislostí*, které říkají jaké jiné balíčky (knihovny) jsou potřebné pro sestavení a běh balíčků. Nejzákladnější závislost balíčku je na knihovně *roscpp*, která je potřebná pro programování uzlů v C++. Analogicky pro programování uzlů v Python je k dispozici knihovna *rospy*.

Soubor *CMakeLists.txt* je textový soubor, který definuje obsah balíčku, závislosti, jak sestavit a instalovat balíček. V souboru jsou uvedeny např. názvy uzlů (.cpp), služeb, zpráv atd. [26]

Dále balíček obsahuje adresáře, které musí mít specifická jména, do kterých se ukládají různé soubory. Popis těchto složek je zobrazen v Tab. 4.

| Název složky | Obsah složky | Koncovka souboru |
|---|---|---|
| launch | XML soubory pro spouštění projektů. | .launch |
| msg | Standartní zprávy. | .msg |
| scripts | Uzly v jazyce Python. | .py |
| src | Uzly v jazyce C++. | .cpp |
| srv | Služební zprávy. | .srv |
| YAML | Parametr soubory pro parametr server. | .yaml |

*Tab. 4 – Charakteristika složek, které jsou obsaženy v každém balíčku Catkin. [26]*

Pro zjištění, zda místo uložení balíčků je propojené se systémem ROS, je možné použít příkaz níže.

```
$ echo $ROS_PACKAGE_PATH //Výpis hodnoty systémové proměnné ROS.
```

Možný výsledek:

```
/home/<uživatel_OS>/catkin_ws/src:/opt/ros/<verze_ROS>/share:
/opt/ros/<verze_ROS> /stacks
```

Pokud nastanou nějaké problémy lze provést manuální nastavení cesty k balíčkům, viz příkaz níže.

```
$ export ROS_PACKAGE_PATH=/home/<uživatel_OS>/ros/ros-pkg:/<další_cesta>
```

#### *4.1.6.1 Vytvoření vlastního balíčku*

Vznik balíčku je podmíněn zadáním příkazů níže do příkazové řádky. [26]

```
$ cd ~/catkin_ws/src            //Přemístění do adresáře.
$ catkin_create_pkg <jméno_balíčku> std_msgs rospy roscpp //Vytvoření
balíčku se závilostmi.
$ cd ~/catkin_ws/               //Přemístění do adresáře.
$ catkin_make                   //Sestavení pracovního prostředí (balíčku).
```

#### *4.1.6.2 Instalace cizího balíčku*

Termínem cizí balíček se míní balíček vytvořený v rámci komunity ROS. Seznam těchto balíčku je dostupný zde [27]. Balíčky sloužící jako ovladače pro sensory jsou dostupné zde [28] a balíčky určené pro roboty podporované ROS jsou přístupné zde [13]. Instalaci balíčku lze provést automaticky pomocí příkazů:

```
$ apt-cache search ros-hydro  //Vyhledání dostupných balíčků.
$ sudo apt-get install ros-<verze_ROS>-<název_balíčku> //Instalace
```

Alternativou instalace balíčku je manuální stažení z příslušného repositáře s následnou manuální instalací. Nejpoužívanější jsou *GIT* nebo *HG* repositáře. Příklady příkazů pro manuální stažení balíčků:

```
$ Hg clone http://stack-nxt.foote-ros-pkg.googlecode.com/hg
$ git clone https://github/<autor_balíčku>/<název_balíčku>.git
```

### 4.1.7 Uzel (node)

Uzel představuje základní stavební jednotku celého systému. Uzel je prakticky zdrojový kód napsaný v jazyce C++ nebo Python. Uzly se umísťují do balíčků.

Existují tři základní typy uzlů tj. uzel, který data odesílá tzv. P*ublisher*, který data přijímá tzv. S*ubscriber* a uzel, který je kombinací Subscriber/Publisher. S těmito uzly se úzce váže pojem *téma*, přes které probíhá veškerá komunikace mezi uzly.

Dále existují uzly, které jsou typu Client/Server, kde komunikace neprobíhá přes téma, ale probíhá v rámci tzv. *služby* (viz kapitola 4.1.10). [29]

Pro práci s uzly slouží příkazový nástroj *rosnode* [30], který například dovoluje manuálně ukončit činnost uzlu. Ostatní možnosti tohoto nástroje jsou uvedeny v Tab. 5.

Vzor příkazu:

```
$ rosnode ping /<název_uzlu>
```

| Nástroj | 1. agr. | 2. agr. | Popis funkce |
|---|---|---|---|
| rosnode | info | / + název uzlu | Výpis informací o aktivním uzlu. |
| | kill | / + název uzlu | Ukončuje aktivní uzel. |
| | list | - | Výpis seznamu aktivních uzlů. |
| | machine | název stanice | Výpis seznamu aktivních uzlů na určité stanici. |
| | ping | / + název uzlu | Test aktivity uzlu. |
| | cleanup | - | Výpis nedostupných uzlů. |

*Tab. 5 – Seznam možností nástroje rosnode. [30]*

#### *4.1.7.1 Vytvoření uzlu (node)*

Tvorba nového uzlu znamená vytvoření souboru se zdrojovým kódem, který bude mít příponu *cpp* pro jazyk C++ nebo *py* pro jazyk Python. Soubory s C++ kódem v balíčku zaujímají místo v adresáři *src* a Python soubory se nachází ve složce *scripts*.

```
$ touch <název_souboru>       //Vytvoření souboru.
$ chmod +x <název_souboru>    //Nastavení práv pro soubor.
$ nano <název_souboru>        //Editace souboru.
```

Nezbytným krokem pro správné vložení uzlu do balíčku je editace souboru *CMakeLists.txt*, který se nachází v kořenovém adresáři každého balíčku. Tato editace je nutná pouze v případě uzlu napsaného v jazyce C++. Editace slouží k tomu, aby mohl být uzel sestaven. Níže je zobrazen vzor úpravy souboru *CMakeLists.txt*.

```
###########                    //Oblast v souboru, kde je nutná editace.
## Build ##
###########
  add_executable(<název_uzlu> src/<název_souboru_uzlu>.cpp)
  add_dependencies(<název_uzlu> <název_balíčku>_generate_messages_cpp)
  target_link_libraries(<název_uzlu> ${catkin_LIBRARIES})
```

Po správném upravení souboru CMakeLists.txt, je vše připraveno pro sestavení uzlu. Příkazy pro sestavení jsou zobrazeny níže.

```
$ cd ~/<název_pracovního_prostředí> //Přemístění do pracovního prostředí.
$ catkin_make                       //Sestavení balíčku => sestavení C++ uzlů.
```

Důležitou informací je, že zdrojový kód v jazyce C++ musí být při každé změně sestaven (příkaz `$ catkin_make`) a pak může být spuštěn. Naopak změnu kódu v jazyce Python stačí jen uložit. To má za následek daleko rychlejší vývoj v jazyce Python. [29]

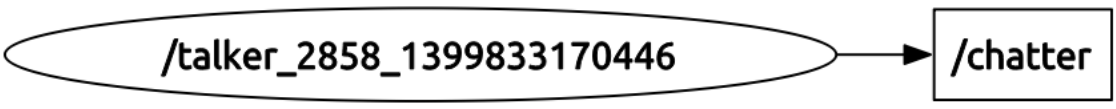
#### *4.1.7.2 Uzel typu Publisher (Talker)*

V případě, kdy je potřeba z uzlu odesílat data musí zdrojový kód obsahovat konstrukci zobrazenou níže (jazyk Python).

Příklad uzlu typu Publisher odesílající textový řetězec „Hello world“ na *téma* „chatter“:

```
#!/usr/bin/env python
import rospy                    #Nahrání knihovny pro programování v Python.
from std_msgs.msg import String #Nahrání ROS datového typu.

def talker():
    pub = rospy.Publisher('chatter', String)     #Inicializace tématu.
    rospy.init_node('talker', anonymous=True)    #Inicializace uzlu.
    r = rospy.Rate(10)       #Frekvence odesílání dat - 1Hz.
    while not rospy.is_shutdown():
        str = "hello world %s" % rospy.get_time()#Přiřazení.
        rospy.loginfo(str)  #Logování hovnoty v proměné "str".
        pub.publish(String(str)) #Publikování hodnoty v proměné "str".
        r.sleep()

if __name__ == '__main__':
    try:
        talker()            #Volání funkce "talker".
    except rospy.ROSInterruptException: pass
```

Zdrojový kód uvedený výše lze spustit (viz kapitola 4.1.7.5) a znázornit v nástroji RQT Graph (viz kapitola 4.1.12), který zobrazí spuštěný uzel v grafické podobě, viz Obr. 29. [29]




*Obr. 29 - Grafická podoba uzlu typu Publisher (nástroj RQT Graph).*

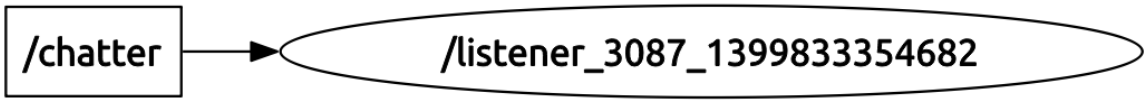
#### *4.1.7.3 Uzel typu Subscriber (Listener)*

V případě, kdy je potřeba z uzlu odesílat data musí zdrojový kód obsahovat konstrukci zobrazenou níže (jazyk Python).

Příklad uzlu typu Subscriber přijímající textový řetězec z *tématu* „chatter“:

```
#!/usr/bin/env python
import rospy                    #Nahrání knihovny pro programování v Python.
from std_msgs.msg import String #Nahrání ROS datového typu.

def callback(data):
    rospy.loginfo(rospy.get_name() + "I heard %s" % data.data)#Logování.

def listener():
    rospy.init_node('listener', anonymous=True)   #Inicializace uzlu.
    rospy.Subscriber("chatter", String, callback) #Inicializace tématu.
    rospy.spin()                  #Start naslouchání.

if __name__ == '__main__':
    listener()                    #Volání funkce “listener”.
```

Zdrojový kód uvedený výše lze spustit (viz kapitola 4.1.7.5) a zobrazit v nástroji RQT Graph (viz kapitola 4.1.12), který zobrazí spuštěný uzel v grafické podobě, viz Obr. 30. [29]




*Obr. 30 – Grafická podoba uzlu typu Subscriber (nástroj RQT Graph).*

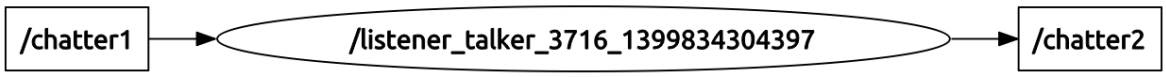
#### *4.1.7.4 Uzel typu Subscriber/Publisher (Listener/Talker)*

V případě, kdy je potřeba z uzlu přijímat a zároveň odesílat data musí zdrojový kód obsahovat konstrukci zobrazenou níže (jazyk Python).

Příklad uzlu typu Subscriber přijímající textový řetězec z *tématu* „chatter“:

```
#!/usr/bin/env python
import rospy                    #Nahrání knihovny pro programování v Python.
from std_msgs.msg import String #Nahrání ROS datového typu.
msg = String()                  #Definice proměnné msg.

def talker():
   pub = rospy.Publisher('chatter2', String ) #Inicializace tématu 2.
   r = rospy.Rate(1)           #Frekvence odesílání dat - 1Hz.
   while not rospy.is_shutdown():
        rospy.loginfo(msg)     #Logování.
        pub.publish(msg)       #Publikování hodnoty v proměné "msg".
        r.sleep()

def callback(data):
   rospy.loginfo(rospy.get_caller_id()+"I heard %s",data.data) #Logování.
   global msg                  #Globalizace proměnné msg.
   msg = data

def listener():
   rospy.init_node('listener_talker', anonymous=True)#Inicializace uzlu.
   rospy.Subscriber("chatter1", String, callback)#Inicializace tématu 1.
   talker()                    #Volání funkce "talker".
   rospy.spin()                #Start naslouchání.

if __name__ == '__main__':
    listener()                 #Volání funkce "listener".
```

Zdrojový kód uvedený výše lze spustit (viz kapitola 4.1.7.5) a zobrazit v nástroji RQT Graph (viz kapitola 4.1.12), který zobrazí spuštěný uzel v grafické podobě, viz Obr. 31. [29]




*Obr. 31 – Grafická podoba uzlu typu Subscriber/Publisher (nástroj RQT Graph).*

### *4.1.7.5 Spouštění uzlů*

Naprogramované uzly lze spustit buď jednotlivě, nebo hromadně. V té chvíli, kdy je uzel v systému ROS spuštěn, stává se procesem, který může být zpracováván na předem zvolené platformě.

Při psaní příkazů na spouštění uzlů se nezadává koncovka u názvu uzlu napsaného v jazyce C++, naopak pokud je použit jazyk Python, je nutné zadat koncovku *py* oddělenou od názvu uzlu tečkou.

Pokud je potřeba spustit pouze jeden uzel, použije se nástroj *rosrun*. Nutnou podmínkou pro spuštění uzlu je běžící systém ROS (příkaz `$ roscore`). [29]

Vzor příkazu:

```
$ rosrun <název_balíčku> <název_uzlu>
```

V případě požadavku na spouštění mnoha uzlů najednou, se použije nástroj *roslaunch*. Zde není potřeba použít nástroj *roscore*, jelikož *roslaunch* spouští jádro systému ROS automaticky. Před použitím nástroje *roslaunch* je nutné vytvořit tzv. *launch soubor* s příponou *launch*, který podléhá syntaxi značkovacího jazyka XML. [23]

Vzor příkazu:

```
$ roslaunch <název_balíčku> <název_launch_souboru>
```

Jednoduchý vzor *launch* souboru pro spuštění dvou uzlů zároveň (jazyk XML):

```
<launch>
 <node pkg="název_balíčku" type="název_souboru_uzlu" name="název_uzlu" >
 </node>
 <node pkg="název_balíčku" type="název_souboru_uzlu" name="název_uzlu" >
 </node>
</launch
```

### 4.1.8 Téma (topic)

Tzv. *téma* přestavuje přístupný bod mezi komunikací uzlů (viz Obr. 8). Definice probíhá ve zdrojovém kódu uzlu, kde se definuje jméno tématu a datový typ zprávy, kterou téma přijímá nebo odesílá.

Konstrukce pro uzel typu Publisher (jazyk Python):

```
rospy.Publisher('<jméno_tématu>', <název_zprávy>) //Inicializace tématu.
```

Konstrukce pro uzel typu Subscriber (jazyk Python):

```
rospy.Subscriber('<jméno_tématu>', <název_zprávy>, <název_funkce_pro
logování>) //Inicializace tématu.
```

Téma dovoluje pomocí nástroje *rostopic* např. odposlouchávat nebo odesílat vlastní data do komunikace mezi uzly. Možnosti, které lze s tématy vykonávat jsou zobrazeny v Tab. 6. [31]

| Nástroj | 1. agr. | 2. agr. | Popis funkce |
|---|---|---|---|
| rostopic | bw | / + název tématu | Zobrazení propustnosti tématu. |
| | echo | / + název tématu | Odposlouchání komunikace v tématu. |
| | find | název balíčku /název zprávy | Nalezení témat dle datového typu. |
| | hz | / + název tématu | Zobrazení frekvence publikování dat do tématu. |
| | info | / + název tématu | Výpis informací o aktivním tématu. |
| | list | - | Výpis informací o aktivních tématech. |
| | type | / + název tématu | Výpis datového typu v tématu. |

*Tab. 6 – Seznam možností nástroje rostopic. [31]*

Další možností nástroje *rostopic* je příkaz se čtyřmi argumenty (není zobrazen v tabulce), díky kterému lze publikovat data (zprávy) do tématu.

Vzor příkazu a následný příklad příkazu:

```
$ rostopic pub /<název_tématu> <náze_balíčku>/<název_zprávy> <data>
$ rostopic pub /my_topic std_msgs/String "hello there"
```

### 4.1.9 Zpráva (message)

Dle komunikačního modelu (viz Obr. 8) v ROS probíhá datová komunikace pomocí tzv. *zpráv*. Tato komunikace vzniká standardně mezi uzly přes téma (standartní zpráva) nebo je vedena v rámci tzv. *služby* (viz kapitola 4.1.10), kde je definována tzv. *služební zpráva*. Každá takováto zpráva je určena svým jménem a obsahuje libovolný počet dat. Tyto data (proměnné) lze definovat standartními datovými typy jako: int8, int16, int32, int64, uint8, uint16, uint32, uint64, float32, float64, string, time, duration, navíc je možné přidat existující jiné zprávy. Dále lze vytvořit z jednotlivých datových typů statické nebo dynamické pole.

Standartní zpráva vznikne vytvořením souboru příponou *msg*, kde název zprávy je roven názvu souboru zprávy. Tento soubor v balíčku zaujímá místo v adresáři *msg*. Příklad obsahu takovéto zprávy je zobrazen níže.

```
string data_1              // data_1 = proměnná
uint32 constant=123        // Nastavení konstanty.
std_msgs/Point 3D_Point    // Zápis jiné zprávy jako datový typ.
float64[] data_2           // Zápis dynamického pole dat.
int16 [5] data_3           // Zápis statického pole dat o velikosti 5.
```

Služební zpráva vznikne vytvořením souboru příponou *srv*, který v balíčku patří do adresáře *srv*. Název souboru je současně názvem služby. Příklad obsahu této zprávy je zobrazen níže.

```
int8 data_IN               // Data při požadavku.
---                        // Povinný oddělovač.
float64 data_1_OUT         // Data při odpovědi.
uint64 data_2_OUT          // Data při odpovědi.
```

Dále lze provádět určitou diagnostiku standartních zpráv pomocí nástroje *rosmsg*. V případě služebních zpráv je k dispozici nástroj *rossrv*. Možnosti zmíněných nástrojů jsou zobrazeny v Tab. 7, přičemž argumenty u obou nástrojů jsou stejné. [32]

| Nástroj | 1. agr. | 2. agr. | Popis funkce |
|---|---|---|---|
| rosmsg<br>rossrv | show | název balíčku /název zprávy | Zobrazení definice zprávy. |
| | list | - | Zobrazení všech aktivních zpráv. |
| | package | název balíčku /název zprávy | Zobrazení zpráv v balíčku. |
| | packages | - | Zobrazení zpráv vše balíčků. |
| | users | název balíčku /název zprávy | Zobrazení souborů používající zprávu. |

*Tab. 7 – Seznam možností nástroje rosmsg a rossrv. [32]*

Příklad příkazu:

```
$ rosmsg show geometry_msgs/Point
```

Konečným krokem po vytvoření standartních nebo služebních zpráv v balíčku je nutná editace souboru *CMakeLists.txt*, který se nachází v kořenovém adresáři každého balíčku. Soubor musí obsahovat text zobrazený níže. [32]

```
find_package(catkin REQUIRED COMPONENTS   message_generation)
catkin_package( CATKIN_DEPENDS message_runtime )

add_add_message_files(
  FILES
  <název_zprávy_1>.msg
  <název_zprávy_2>.msg
)
add_service_files(
  FILES
  <název_služby_1>.srv
  <název_služby_2>.srv
)
//V případě, že balíček je závislý na jiném balíčku obsahující zprávy.
generate_messages( DEPENDENCIES <název_msgs_balíčku> )
```

Důležitou informací je, jak ve vlastním programování uzlů zprávy používat. Demonstrační příklad (viz níže) v jazyce Python zobrazuje použití zprávy *2D_point*, která je definovaná proměnnou *int8 x* a *uint8 y*.

```
from mypackage.msg import 2D_point // Import zprávy z balíčku mypackage.
Point = 2D_point () // Vytvoření objektu Point dle zprávy 2D_point.
Point.x = 10    // Pomocí tečkové notace přiřazení hodnoty do proměnné x.
Point.y = -20
```

ROS framework v základní verzi obsahuje mnoho předefinovaných druhů dat (zpráv). K dispozici jsou definice zpráv určené např. pro senzory, diagnostiku, navigaci nebo lze využít zprávy typu geometrie. [33]

Příkladem zprávy typu geometrie je g*eometry_msgs/Point.msg* představující trojrozměrný bod. Definice je zobrazena níže.

```
float64 x
float64 y
float64 z
```

### 4.1.10 Služba (service)

Služba v ekosystému ROS dovoluje tzv. *vzdálené volání procedur* model komunikace (viz Obr. 8), který je užitečný v distribuovaných systémech. Služba přestavuje dva uzly, kde první odesílá požadavek na nějaký výpočet a druhý výpočet provede a výsledek odešle ihned zpět prvnímu uzlu, tím vzniká komunikační model Client/Server.

Aby mohla tato komunikace v systému ROS vzniknout, je potřeba uzel typu Client, uzel typu Server a služební zprávu. [34]

Při používání služeb je možno využít nástroje *rosservice*, kde jeho možnosti jsou zobrazeny v Tab. 8. [35]

| Nástroj | 1. agr. | 2. agr. | 3.arg. | Popis funkce |
|---|---|---|---|---|
| rosservice | args | název služební zprávy | - | Zobrazení argumentů služby |
| | call | název služební zprávy | argumenty služby | Zavolání služby s argumenty. |
| | find | název balíčku / název služební zprávy | - | Nalezení služby. |
| | list | - | - | Zobrazení seznamu dostupných služeb. |
| | info | název služební zprávy | - | Zobrazení informací o službě. |
| | node | název služební zprávy | - | Zobrazení uzlu, který poskytuje službu. |
| | type | název služební zprávy | - | Zobrazení definice služby. |
| | uri | název služební zprávy | - | Zobrazení URI adresy, kterou služba používá. |

*Tab. 8 – Seznam možností nástroje rosservice. [35]*

### 4.1.11 Parametr server

Parametr server je součást jádra frameworku ROS, která poskytuje možnost uložení dat nejrůznějších datových typů. Podporovány jsou standartní datové typy jako: string, integer, float, boolean, seznam, slovník, iso8601 data a base64-encoded data. Při multiplatformní aplikaci je výhodou, že server je globálně dostupný pro všechny propojené platformy.

ROS framework nabízí tři možnosti jak parametr server ovládat. Tyto možnosti dovolují nahrát na server tzv. *parametr soubor*, který podléhá syntaxi značkovacího jazyka YAML. Soubor musí být zakončen koncovkou *yaml*. Příklad obsahu takového souboru je zobrazen níže.

```
<název_proměnné> : <hodnota> // Vzor konstrukce parametru
hexadecimal: 0xC      // Hexadecimální hodnota.
true: true            //Pravdivostní hodnota (místo true lze použít yes).
list: [1, 2, 3, 4, 5] // Seznam hodnot.
string: '12345'       //Textový řetězec.
```

První způsob ovládání nám poskytuje příkazová řádka, ve které je k dispozici nástroj *rosparam*. Jednotlivé konstrukce příkazů jsou znázorněny na Tab. 9. [36]

Vzor příkazu:

```
$ rosparam set /<název_parametru> "<libovolný_řetězec>" //Nahrání
textového řetězce na server.
```

| Nástroj | 1. agr. | 2. agr. | 3. agr. | Popis funkce |
|---|---|---|---|---|
| rosparam | set | / + název parametru | hodnota parametru | Nahrání parametru s hodnotou na server. |
| | get | název parametru | - | Získání hodnoty parametru. |
| | load | název parametr souboru | - | Nahrání souboru s mnoha parametry. |
| | dump | název parametr souboru | - | Smazání souboru. |
| | delete | název parametru | - | Smazání parametru. |
| | list | - | - | Výpis všech parametrů. |

*Tab. 9 – Seznam možností nástroje rosparam. [36]*

Další možností jak získat přístup k severu je použití konstrukce (viz níže) přímo ve zdrojovém kódu uzlu. [36]

```
// jazyk Python
rospy.set_param('<název_parametru>',<hodnota_odpovídající_datovému_typu>)
#Náhrání parametru s hodnotou.
rospy.get_param("<název_parametru>")    # Získání hodnoty parametru.
rospy.delete_param('<název_parametru>') # Smazání parametru.
```

Třetí alternativa je použití nástroje *roslaunch* (viz kapitola 4.1.3), kde ve spouštěcím souboru lze použít tag <*param*> nebo tag <*rosparam*>. [23]
Příklad XML konstrukce pro nahrání parametru:

```
<param name="publish_frequency" type="double" value="10.0" />
```

Příklad XML konstrukce pro nahrání souboru s parametry:

```
<rosparam command="load" file="$(find
<název_balíčku>)/<název_parametr_souboru>.yaml"/>
```

### 4.1.12 Uživatelské rozhraní RQT

RQT je GUI (Graphical User Interface) neboli grafické uživatelské rozhraní, které umožnuje spouštět a ovládat tzv. *grafické nástroje*. V tomto rozhraní lze nalézt některé repliky konzolových nástrojů jako např. nástroje pro diagnostiku zpráv, služeb, témat atd. RQT GUI se spouští příkazem níže.

```
$ rqt
```

Jedním z grafických nástrojů je nástroj RQT Graph, který je schopen zobrazit síť komunikace mezi uzly graficky. Tento graf lze vyvolat v samotném RQT nebo pomocí příkazu níže.

```
$ rosrun rqt_graph rqt_graph
```

Dalším užitečným grafickým nástrojem je RQT plot, který umožnuje vykreslovat různá data (zprávy) do 2D grafů v aktuálním čase. Spuštění aplikace lze provést přímo v RQT nebo příkazem níže. [37]

```
$ rqt_plot
```

Jeden z předních prvků frameworku ROS je grafický nástroj RVIZ, který má schopnost provádět 3D vizualizaci robotů popsaných v URDF formátu a dále např. zobrazit obraz z kamer nebo 3D obraz z laserových skenerů. Detailní specifikace a popis ovládání nástroj je dostupný zde [38].

RVIZ lze přímo spustit v RQT nebo je možné tuto aplikaci spustit pomocí příkazu níže:

```
$ rosrun rviz rviz
```

### 4.1.13 Editor souborů

Další užitečný nástroj, který je v systému ROS dostupný je souborový editor *rosed*. Dle vzorového příkazu níže, lze provést jednoduše editaci souborů v určitém balíčku [39].

```
$ rosed <název_balíčku> <název_souboru>
```

### 4.1.14 Logování dat

ROS framework umožnuje kompletní záznam komunikace mezi uzly a zpětného přehrání pomocí nástroje *robag*. Například po připojení nějaké kamery lze jednoduše uložit a přehrát záznam pořízeného videa. Základní ovládání nástroje je zobrazeno níže. [40]

```
$ rosbag record -a all <název_tématu> //Záznam komunikace na všech tématech.
$ rosbag play <název_log_souboru> // Zpětné přehrání komunikace, soubor s koncovkou bag.
```

Dále lze logování dat provádět v RGT GUI. Grafickou aplikaci lze přímo spustit v RQT nebo pomocí příkazu níže.

```
$ rqt_bag
```

### 4.1.15 Odstraňování problémů

ROS framework poskytuje konzolový nástroje *roswtf*, který slouží k diagnostice běžícího systému a k odstraňování nejrůznějších problémů. Kontroluje balíčky, systémové proměnné a hledá potenciální problémy. Nástroj se spouští příkazem zobrazeným níže.

```
$ roswtf
```

Dále co *roswtf* nabízí je kontrola spouštěcího souboru. Vzor příkazu pro kontrolu je ukázán níže. [41]

```
$ roswtf <název_spouštěcího_souboru>.launch
```

### 4.1.16 Distribuovaný systém

Jednou z hlavních předností systému ROS je distribuovaný design, který dovoluje provádět výpočty na libovolné počítačové platformě, která je zapojena do celého systému. Například je možné uzel uložený na jedné stanici nastavit tak, aby byl při spuštění zpracováván na jiné platformě, aniž by byl fyzicky přesunut (viz Tab. 3 – značka <*machine*>).

První věcí, pro vznik řídicího systému spojeného z mnoha stanic, je nutnost spojit všechny platformy do jedné počítačové sítě dle protokolu TCP/IP. ROS dovoluje vytvořit síť dle tzv. *Master/Slave* architektury, která je tvořena z jedné hlavní stanice (Master), ke které je možno připojit libovolný počet podřízených (Slave) platforem.

Předpoklad pro vznik sítě je nainstalovaný ROS framework na každé platformě. Dále musí být na každé stanici provedeno nastavení tzv. *systémových proměnných* ROS, které zajistí propojení Slave stanic s Master stanicí.

Nastavení systémových proměnných na Master stanici:

```
$ export ROS_MASTER_URI=http://<IPv4_adresa_Master_stanice>:11311
$ export ROS_HOSTNAME=<IPv4_adresa_Master_stanice>
```

Nastavení systémových proměnných na Slave stanici:

```
$ export ROS_MASTER_URI=http://<IPv4_adresa_Master_stanice>:11311
$ export ROS_IP=<IPv4_adresa_Slave_stanice>
```

Další důležitou věcí je provést editaci souboru *.bashrc*. Do souboru se vloží zmíněné příkazy výše. To zajistí nastavení stanice do režimu Master nebo Slave při každém spuštění příkazového terminálu (bash). [42]

Systém vzniknu, společného řídicího systému spočívá ve spuštění jádra ROS na Master stanici, dále jen stačí, aby Slave stanice měly správně nastaveny systémové proměnné a fyzicky byly spojeny s hlavní stanicí. Aniž by bylo spuštěno jádro na podřízené platformě, je možné využívat všechny služby ROS na jakékoli stanici. Tím vznikne např. společný parametr server nebo možnost globálního použití konzolových nástrojů na všechny připojené stanice. V případě spuštění RQT Graph na libovolné stanici budou zobrazeny všechny spuštěné uzly v rámci celé sítě. To napovídá tomu, že síť spojených stanic je z „venku“ viděna jako jeden systém spuštěný na jediné platformě. Výhodou je také to, že za běhu systému na Master stanici je možné ostatní stanice připojit/odpojit v libovolném časovém okamžiku.

V případě potřeby vrátit jakoukoliv stanici na lokální provoz je potřeba provést příkazy níže.

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

Respektive je nutné zeditovat soubor *.bashrc*, kde je potřeba odstranit nastavení systémových proměnných (ROS_HOSTNAME, ROS_IP a ROS_MASTER_URI).

## 4.2 Oracle VM VirtualBox

VM VirtualBox je multiplatformní nástroj pro virtualizaci operačních systémů, který je produktem firmy Oracle Corporation. Podpora je vnímána ze dvou stran. Strana hostitelských OS a strana OS, které lze virtualizovat. Obě skupiny mají mnoho zástupců jako Microsoft Windows, Mac OS X, Solaris, Linux, Android atd.

V praxi lze například nainstalovat na OS Windows 7 libovolný počet různých OS, kdy host a virtuální stroje lze používat zároveň na jednom počítači.

V testovací aplikaci této práce byl jako host použit Windows 7, který poskytoval virtualizaci dvěma verzím OS Ubuntu. [43]


*Obr. 32 - Logo multiplatformního virtualizačního nástroje Oracle VM VirtualBox. [43]*

## 4.3 Windows

Windows je jeden z nejpoužívanějších operačních systému, který je produktem firmy Microsoft Corporation. [44]

K realizaci testovací robotické aplikace byla použita verze Windows 7 Home Edition.


*Obr. 33 – Logo operačního systému Windows 7. [44]*

## 4.4 Ubuntu

Ubuntu přestavuje distribuci operačního systému Linux, která je velmi populární. Velkou výhodou je, že tento OS je vyvíjen zdarma. [45]

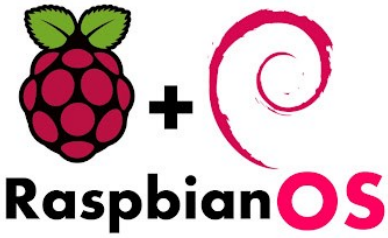
Ve výsledné testovací aplikaci byly použity hned dvě verze tohoto systému a to Ubuntu 11.10 Oneiric Ocelot a 13.04 Raring Ringtail.


*Obr. 34 – Logo linuxového operačního systému Ubuntu. [45]*

## 4.5 Raspbian

Raspbian je operační systém určený pro jednodeskový počítač Raspberry Pi. Tento systém je na bázi linuxové distribuce Debian Wheezy. [46]

V praktické části práce byl na Raspberry Pi nainstalován Raspbian s datem vydání leden 2014.




*Obr. 35 – Logo linuxového operačního systému Raspbian. [46]*

# 5. Instalace a nastavení senzorického systému

Kapitola instalace a nastavení popisuje metody, které vedly k úspěšnému oživení hardware/software architektury (viz Obr. 14 a Obr. 25) testovací robotické aplikace této diplomové práce. Po provedení instalace a nastavení bylo možné začít realizovat vlastní řídící software v testovací aplikaci.

## 5.1 Raspberry Pi, SRF08 a robot LEGO NXT

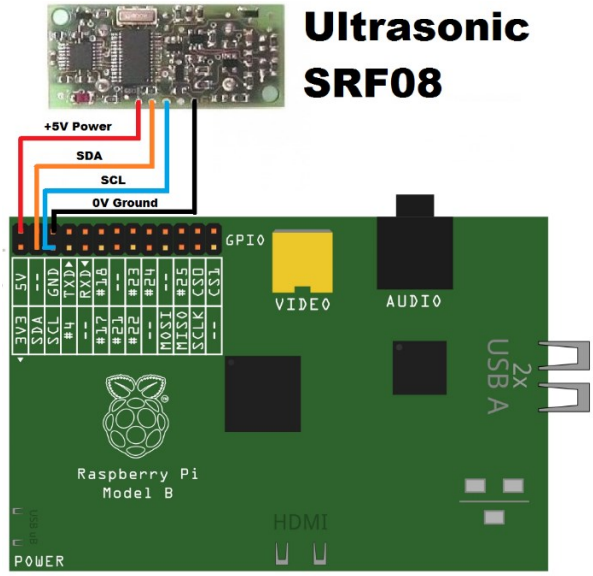
Jednodeskový mini-počítač Raspberry Pi (RPi), ultrazvukový sensor SRF08 a robot NXT tvořily jednu počítačovou platformu (Slave stanice 1). V této kapitole jsou popsány metody, které uvedly tuto Slave stanici do stavu, kdy se mohly stát součástí distribuovaného řídicího systému ROS.

### 5.1.1 Instalace hardware

Raspberry Pi poskytlo rozhraní GPIO, kde přes $I^2C$ komunikační sběrnici byl připojen sensor SRF08. Drátování těchto dvou zařízení bylo provedeno dle Obr. 36. Dráty SDA a SCL byly propojeny přes pull-up rezistory 1,8kΩ oproti napájení.

Počítač RPi byl napájen dostatečně silným zdrojem 5V/1000mA přes rozhraní micro-USB a na ultrazvukový sensor bylo přivedeno napájení +5V z rozhraní GPIO.

Jako pevný disk byla použita dostatečně veliká 16GB paměťová karta, která musela být nejprve zformátována do souborové systému FAT32.




*Obr. 36 – Schéma drátování mezi počítačem Raspberry Pi a sensorem SRF08.*

Robota po sestavení stačilo již jen připojit do RPi pomocí sběrnice USB. Jednotlivé sensory byly zapojeny to řídící jednotky takto: dotykový sensor 1 = port 1, dotykový sensor 2 = port 2, ultrazvukový sensor = port 3 a barevný snímač = port 4. Dále motory byly připojeny: pravý motor = port B, levý motor = port C.

### 5.1.2 Instalace software

Nejprve bylo potřeba nainstalovat na Raspberry Pi operační systém, ovladače pro připojené periferie a nakonec byl nainstalován ROS framework. Toto uskupení bylo označeno jako Slave stanice 1.

#### *5.1.2.1 Raspbian*

Raspbian představoval OS pro mini-počítač Raspberry Pi, který byl instalován dle návodu zde [18]. Výchozí uživatelské jméno (pi) a heslo (raspberry) bylo možné změnit po zadání příkazu:

```
$ sudo raspi-config
```

Po přihlášení uživatele bylo možné spustit grafické rozhraní systému příkazem:

```
$ startx
```

Po spuštění systému byla provedena konfigurace síťového nastavení, která spočívala v editaci souboru *interfaces* příkazem:

```
$ sudo nano /etc/network/interfaces
```

Síťová konfigurace byla nastavena na dynamické přidělování adres pomocí DHCP (Dynamic Host Configuration Protocol). V případě potřeby byla změněna na statické přidělení dle příkladu:

```
iface eth0 inet static
address 192.168.1.2
netmask 255.255.255.0
gateway 192.168.1.1
```

Aby mohl SRF08 komunikovat s RPi bylo nutné nainstalovat ovladače pro $I^2C$ sběrnici a komunikační protokol SMBUS (System Management Bus). Kompletní nastavení proběhlo dle návodu uvedeného zde [47]. Provedená konfigurace dovolila programování komunikace s ultrazvukovým sensorem v jazyce Python a to díky knihovně *smbus*.

Dalším krokem bylo připravit systém pro připojení robota s mini-počítačem. Níže jsou uvedeny příkazy, které musely být vykonány. [48]

```
$ lssub                              // Prvotní ověření přípojení.
$ sudo apt-get install python-nxt  //Instalace knihoven pro programování
v jazace Python.
$ sudo apt-get install python-usb  // Instalace knihoven pro USB
komunikaci pomocí jazyka Python.
$ sudo groupadd lego               //Vytvoření skupiny lego.
$ sudo usermod -a -G lego <uživatelské_jméno_raspbian> //Přídání
uživatele do skupiny lego.
$ echo "SUBSYSTEM=="usb", ATTRS{idVendor}=="0694",
ATTRS{idProduct}=="0002", SYMLINK+="legonxt-%k", GROUP="legonxt",
MODE="0666"" > /tmp/70-lego.rules && sudo mv /tmp/70-lego.rules
/etc/udev/rules.d/70-lego.rules // Vytvoření souboru pro indentifikaci
zařízení a umožnění komunikace.
```

Pro ověření instalace mohl být spuštěn zdrojový kód (viz níže, jazyk Python), který po se spuštění příkazem `$ python <název_souboru>.py` připojil k řídící jednotce a získal hodnot True/False z dotykového sensoru připojeného na port č. 1.

```
!/usr/bin/env python     # Identifikace Python kódu.
import nxt.locator       # Knihovny pro komunikaci s řídící jednotkou NXT.
from nxt.sensor import * # Knihovny pro komunikaci se sensory NXT.
brick = nxt.locator.find_one_brick()  # Nalezení řídicí jednotky.
print 'Touch:', Touch(brick, PORT_1).get_sample()# Získání hodnoty.
```

#### *5.1.2.2 ROS Groovy*

Výběr verze ROS podléhal operačnímu systému Raspbian, který byl nejvýše podporován verzí ROS Groovy. Instalace proběhla dle návodu zde [12]. Hlavní příkaz:

```
$ sudo apt-get install ros-groovy-full
```

Po instalaci bylo vytvořeno pracovního prostředí Catkin, ve kterém byl vytvořen prázdný balíček dle kapitoly 4.1.6.1. Balíček byl doplněn o potřebné soubory v kapitole 6.

Finální nastavení spočívalo v úpravě souboru *.bashrc*, příkazem:

```
$ sudo nano ~/.bashrc
```

Na konec tohoto souboru byly vloženy příkazy, viz níže. Tyto příkazy nastavily příkazovou řádku pro pohodlnou práci s ROS.

```
source /opt/ros/groovy/setup.bash   //Propojení OS s ROS.

source ~/catkin_ws/devel/setup.bash //Propojení pracovního prostředí
s ROS.

export ROS_MASTER_URI=http://192.168.56.104:11311 //Přípojení do

export ROS_IP=192.168.56.103               //počítačové sítě jako Slave.
```

Tato platforma byla nastavena jako Slave stanice, tudíž spouštění uzlů bylo možné až po předchozím spuštění Master stanice.

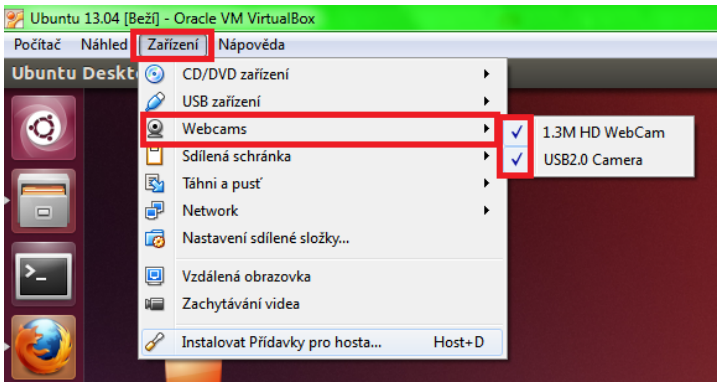
## 5.2 PC, webkamera, klávesnice

Personální počítač (notebook) představoval tři počítačové platformy, kterým poskytoval svůj hardware. Počítač byl ovládán standardně nainstalovaným operačním systémem, nad kterým byly nainstalovány další dva virtuální OS, a tím bylo docíleno tří běžících systémů na jednom počítači. Virtuální uspořádání platforem dovolilo elegantním způsobem vytvořit síť různých stanic, a tím poskytlo možnost otestovat multiplatformní vlastnost frameworku ROS, aniž by bylo fyzicky potřeba dalších počítačů. V rámci testovací aplikace notebook reprezentovat Master stanici a dvě Slave stanice. Externí klávesnice byla připojena do nadřazené stanice a webkamera do jedné z podřízených stanic.

### 5.2.1 Instalace hardware

Periférie webkamera a klávesnice byly připojeny do PC pomocí USB sběrnice. Obě zařízení byly připojeni do virtuálních stanic. Aby virtuální stanice měla přístup k webkameře nebylo ji možné v rámci virtualizačního nástroje VirtualBox připojit jako standartní USB zařízení. Připojení proběhlo dle Obr. 37. Klávesnice byla do virtuální stanice připojena automaticky.

Notebook s Raspberry Pi byl spojen standardním síťovým kabelem.




*Obr. 37 - Připojení webkamery do virtuální stanice v rámci nástroje VirtualBox.*

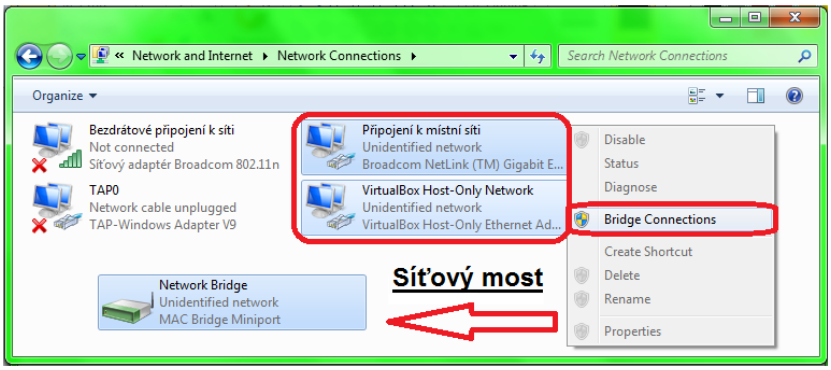
### 5.2.2 Instalace software

Instalace softwaru spočívala v nainstalování tří operačních systémů a nástroje Oracle VM VirtualBox.

#### *5.2.2.1 Windows 7 a VirtualBox*

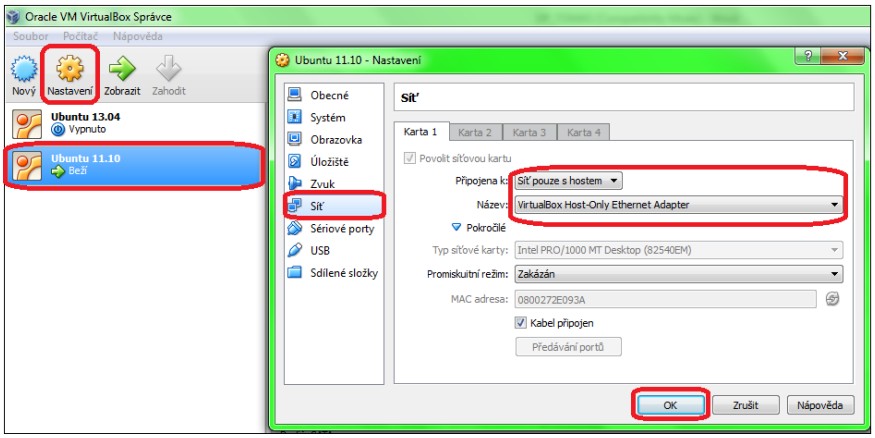
Primární operační systém na počítači reprezentoval Microsoft Windows 7, na kterém byl nainstalován virtualizační nástroj VirtualBox ve verzi 4.3.

V první řadě bylo nutné nastavit síťové připojení ve Windows, aby virtuální stanice mohly být ve společné počítačové síti spolu s Raspberry Pi. Nastavení spočívalo ve vytvoření síťového mostu mezi síťovou kartou notebooku a síťovou kartou virtuálního stroje, viz Obr. 38.



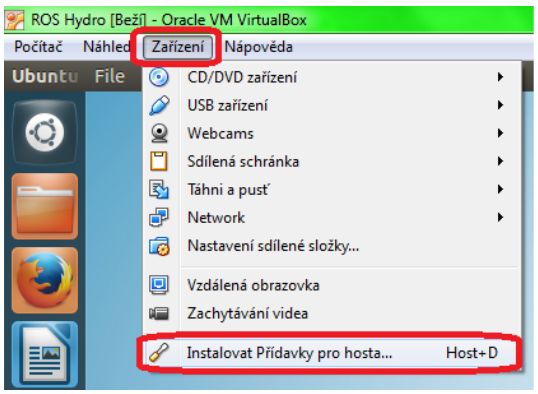

*Obr. 38 - Vytvoření síťového mostu ve Windows 7.*

Po instalaci virtuálních stanic bylo dále nutné pro každý systém nastavit síťovou kartu. Nastavení proběhlo dle Obr. 39.




*Obr. 39 – Nastavení síťového připojení v nástroji VirtualBox.*

Po spuštění virtuálních operačních systémů byly na každé stanici nainstalovány tzv. *přídavky pro hosta*, které umožnily pohodlnější práci s OS.




*Obr. 40 – Instalace přídavků pro hosta v nástroji VirtualBox.*

#### *5.2.2.2 Ubuntu 11.10 a ROS Groovy*

Pro ověření funkčnosti komunikace mezi různými verzemi ROS byla jako Master stanice zvolena verze ROS Groovy, kterému odpovídal operační systém Ubuntu 11.10. Po instalaci OS Ubuntu jako virtuální stroj byl nainstalován framework ROS Groovy dle návodu zde [12]. Hlavní příkaz:

```
$ sudo apt-get install ros-groovy-full
```

Po instalaci ROS bylo vytvořeno pracovního prostředí Catkin, ve kterém byl vytvořen prázdný balíček dle kapitoly 4.1.6.1. Balíček byl doplněn o potřebné soubory v kapitole 6.

Finální nastavení spočívalo v úpravě souboru *.bashrc*. Na konec tohoto souboru byly vloženy příkazy níže. Tyto příkazy nastavily příkazovou řádku pro pohodlnou práci s ROS.

```
source /opt/ros/groovy/setup.bash   //Propojení OS s ROS.
source ~/catkin_ws/devel/setup.bash //Propojení pracovního prostředí
s ROS.
export ROS_MASTER_URI=http://192.168.56.104:11311 //Přípojení do
export ROS_HOSTNAME=192.168.56.104        //počítačové sítě jako Master.
```

Master stanice představovala hlavní výpočetní jednotku, která vykonávala kód většiny uzlů. Do této stanice byla připojená externí klávesnice.

#### *5.2.2.3 Ubuntu 13.04 a ROS Hydro*

Další seskupení OS Ubuntu 13.04 a frameworku ROS Hydro bylo označeno jako Slave stanice 2. Hlavním úkolem této stanice bylo přeposílání obrazu z webkamery do Master stanice.

Instalace ROS Hydro proběhla dle instrukcí zde [12]. Hlavní příkaz:

```
$ sudo apt-get install ros-hydro-full
```

Stejně jako v předešlé kapitole 5.2.2.2 bylo potřeba vytvořit pracovního prostředí Catkin s prázdným balíčkem. Nakonec byl upraven soubor *.bashrc* podobně jako v předchozí kapitole.

```
source /opt/ros/hydro/setup.bash    //Propojení OS s ROS.
source ~/catkin_ws/devel/setup.bash //Propojení pracovního prostředí
s ROS.
export ROS_MASTER_URI=http://192.168.56.104:11311 //Přípojení do
export ROS_IP=192.168.56.101               //počítačové sítě jako Slave.
```

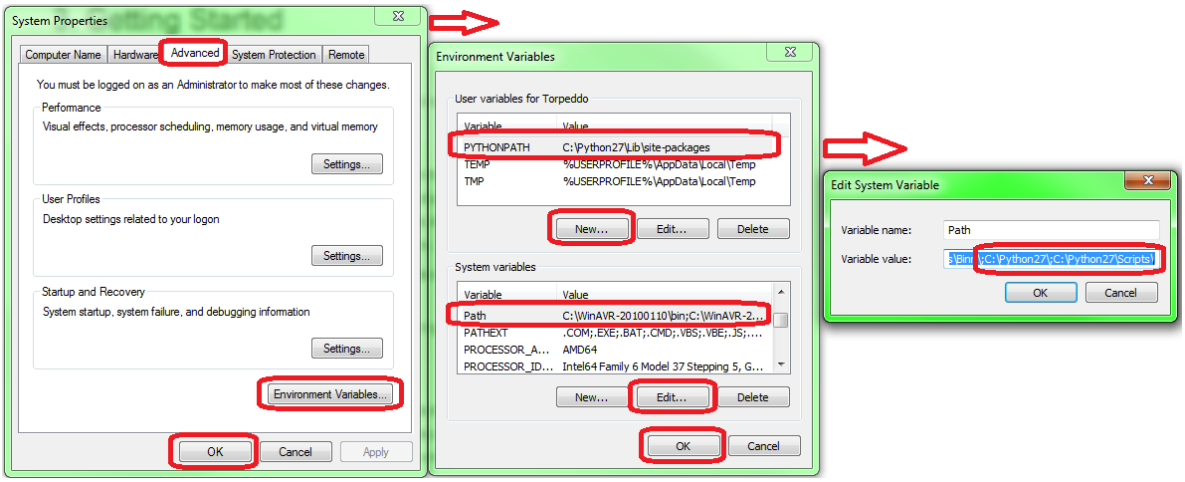
Jelikož byla do této stanice připojena webkamera, bylo nutné nainstalovat ovladač pro komunikaci s kamerou. Byl vybrán standartní ovladač pro USB webkamery. [49] Instalace proběhla dle příkazu níže.

```
sudo apt-get install ros-hydro-usb-cam
```

#### *5.2.2.4 Windows 7 a ROS Hydro*

Tato počítačová platforma byla označena jako Slave stanice 3 a byla tvořena OS Windows 7 a frameworkem ROS Hydro. Jako Slave stanice sloužila jen pro diagnostiku celého řídicího systému.

Instalace ROS byla provedena dle návodu zde [12]. Při instalaci bylo nutné nainstalovat Python na Windows. Po instalaci bylo důležité správně nastavit systémové proměnné Windows, aby Python fungoval. Postup nastavení je zobrazen na Obr. 41.




*Obr. 41 - Instalace Python na Microsoft Windows 7.*

Po správné instalaci bylo provedeno připojení do testovací aplikace. Příkazy níže připojily stanici k Master stanici. Příkazy byly zadávány do příkazové řády Windows.

```
$ cd C:\Windows                                  // Přesun do adresáře.
$ call C:\opt\ros\hydro\x86\setup.bat      //Propojení Windows s ROS.
$ set ROS_MASTER_URI=http://192.168.56.104:11311 // Přípojení do
$ set ROS_IP=192.168.56.105                      // počítačové sítě jako Slave.
$ rosnode list                 // Test přípojení - zobrazení aktivních uzlů.
```

# 6. Realizace testovací robotické aplikace

Tato kapitola se zabývá charakteristikou řídícího softwaru testovací robotické aplikace, která vznikla v této diplomové práci.

Předpokladem pro vývoj řídícího softwaru bylo kompletní provedení kapitoly *5. Instalace a nastavení senzorického systému*.

Obecným úkolem testovací aplikace bylo vytvořit řídící signály pro motory robota na základě přijatých dat ze sensorů a to v rámci multiplatformního počítačového systému. Jednotlivé platformy byly rozděleny dle funkce na Master a tři Slave stanice. V hierarchii platforem figuroval jeden nadřazený počítač (Master), který přijímal a vyhodnocoval senzorická data, ze kterých vytvářel a následně odesílal příkazy do podřízené stanice (Slave 1), která zprostředkovala komunikaci s řídící jednotkou robota.

V terminologii ROS frameworku bylo úkolem vytvořit síť vzájemně komunikujících uzlů, každého se specifickou funkcí, s cílem řídit mobilního robota.

Navrhnutý řídicí systém umožnil ovládat robota v režimu manuálním nebo automatickém. Uživatel robota měl k dispozici manuální řízení z externě připojené klávesnice. Automatické řízení spočívalo v autonomní jízdě robota po černé čáře. Přepínání režimů probíhalo pomocí tlačítek umístěných na konstrukci robota.

V následujících kapitolách jsou popsány funkce všech stanic v řídicím systému se zaměřením na obsahy balíčků stanic a programový kód jednotlivých uzlů.

## 6.1 Slave stanice 1

Tato podřízená stanice představovala mini-počítač Raspberry Pi s připojeným sensorem SRF08 a robotem LEGO NXT, na které běžely dva uzly.

### 6.1.1 Definice balíčku

V pracovním prostředí Slave stanice 1 byl vytvořen balíček typu Catkin s pojmenováním *package_slave_1*. Balíček obsahoval následující soubory, viz Tab. 10.

| Balíček – package_slave_1 (catkin) | |
|---|---|
| **Typ souboru** | **Název souboru** |
| Uzel (Python) | RPI_sensor.py, nxt_API.py |
| Zpráva | Motors_values_NXT.msg, Sensors_values_NXT.msg |
| Parametr soubor | robot_NXT.yaml |
| Spouštěcí soubor | slave_station_1.launch |

*Tab. 10 – Seznam souborů ve vlastním balíčku na Slave stanici 1.*

#### *6.1.1.1 Uzly, témata a zprávy*

Prvním úkolem této stanice bylo získat a přeposílat data z ultrazvukového sensoru SRF08 do Master stanice. Řešení spočívalo ve vytvoření uzlu (*RPI_sensor.py*) typu *Publisher,* který odesílal naměřená data na téma (*topic_RPI_Ultrasonic*). Data byla přenášena ve zprávě (*UInt16*), která obsahovala jedinou proměnnou standartního datového typu uint16. Tato standartní zpráva byla dostupná z ROS prostředí, konkrétně z balíčku *std_msgs*. Naměřené hodnoty následně nadřízený počítač z tématu odposlouchával. Data ze sensoru reprezentovala celočíselné hodnoty, které udávaly vzdálenost objektu v centimetrech. Běžící uzel v grafické podobě je zobrazen na Obr. 42.

Nejdůležitější části kódu uzlu *RPI_sensor.py* jsou zobrazeny níže (jazyk Python).

```
import smbus   # Knihovna pro komunikaci se sensorem (SMBUS protokol).
import rospy   # Knihovna pro vytváření uzlů v jazyce Python.
std_msgs.msg import UInt16 # Zpráva pro data ze sensoru.

def distance():# Funkce na získání hodnoty vzdálenosti ze sensoru.
        range1 = bus.read_byte_data(address, 2)
        range2 = bus.read_byte_data(address, 3)
        range3 = (range1 << 8) + range2
        return range3

def talker():  # Hlavní funkce vytvářející uzel typu Publisher.
    pub = rospy.Publisher(' topic_RPI_Ultrasonic ',UInt16) # Vytvoření
    tématu s nastavením zprávy.
    rospy.init_node(' RPI_sensor ') # Vytvoření uzlu.
    while not rospy.is_shutdown():
      write(0x51)      # Příkaz pro nastavení vzdálenosti v centimetrech.
      time.sleep(1.0)  # Frekvence čtení dat ze sensoru (1Hz).
      rng = distance()
      rospy.loginfo(rng)# Logování hodnot vzdálenosti.
      pub.publish(UInt16(rng)) # Odesílání hodnot vzdálenosti na téma.
      rospy.sleep(1.0) # Frekvence odesílání dat na téma.
```

Druhým úkolem stanice bylo poskytnout rozhraní mezi systémem ROS a robotem NXT. Rozhraní reprezentoval uzel (*nxt_API.py*) typu *Subscriber/Publisher.* Uzel přijímal z témat (*topic_nxt_API_motor_L, topic_nxt_API_motor_R*) příkazy na řízení pravého a levého motoru. Tyto příkazy byly přenášeny ve zprávě (*Motors_values_NXT*). Dále uzel odesílal data ze všech sensorů v jedné zprávě (*Sensors_values_NXT*) na téma (*topic_nxt_API_sensors*). Senzorická data z tématu byla následně odebírána nadřazeným počítačem. Běžící uzel v grafické podobě je zobrazen na Obr. 42.

V kapitole 5.1.2.1 proběhla instalace důležité knihovny NXT-Python, která poskytla funkce na komunikaci s řídící jednotkou NXT robota. Při použití této knihovny nebylo nutné jinak zasahovat do řídící jednotky robota. Řídící jednotka robota byla jen zapnuta a připojena do počítače RPi, na kterém běžel uzel, který zajistil přenos dat ze sensorů robota a naopak přenos příkazů na ovládání motorů do robota.

Uzel používal dva typy zpráv. První zpráva (*Sensors_values_NXT*), kterou uzel odesílal, obsahovala data ze sensorů robota. Definice zprávy je zřejmá z Tab. 11.

| Zpráva - Sensors_values_NXT.msg | | | |
|---|---|---|---|
| **Datový typ** | **Název proměnné** | **Hodnota** | **Popis dat ze sensorů robota** |
| bool | Touch_1 | True/False | Dotykový sensor 1: stlačen = True. |
| bool | Touch_2 | True/False | Dotykový sensor 2: stlačen = True. |
| uint8 | Ultrasonic | 0 až cca 70 | Ultrazvukový sensor: vzdálenost v centimetrech. |
| uint8 | Color20 | 1,2,3,4,5,6 | RGB sensor: černá = 1, modrá = 2, zelená = 3, žlutá = 4, červená = 5, bílá = 6. |
| uint64 | Tacho_L | 0 až $2^{64}$ | Zpětná vazba o natočení levého motoru. |
| uint64 | Tacho_R | 0 až $2^{64}$ | Zpětná vazba o natočení pravého motoru. |

*Tab. 11 – Definice zprávy pro senzorická data robota.*

Druhá zpráva (*Motors_values_NXT*), kterou uzel přijímal, obsahovala příkazy na řízení motorů robota. Definice zprávy je patrná z Tab. 12.

| Zpráva - Motors_values_NXT.msg | | | |
|---|---|---|---|
| **Datový typ** | **Název proměnné** | **Hodnota** | **Popis příkazů pro motory** |
| string | mode | run | Roztočení motoru. |
| | | turn | Roztočení motoru s určitým počtem otáček. |
| | | idle | Vypnutí motoru bez brždění. |
| | | brake | Zabrzdění motoru. |
| uint16 | distance | 10 až $2^{16}$ | Vzdálenost posunu pásu (100 = cca 3cm). |
| int8 | power | 60 až 128 | Rychlost otáčení dopředu (mode = run). |
| | | 15 až 128 | Rychlost otáčení dopředu (mode = turn). |
| | | -60 až -128 | Rychlost otáčení dozadu (mode = run). |
| | | -15 až -128 | Rychlost otáčení dozadu (mode = turn). |

*Tab. 12 – Definice zprávy pro řízení motorů robota.*

Příklady zdrojového kódu uzlu (*nxt_API.py*) na získání hodnoty ze senzoru a řízení motoru (jazyk Python).

```
import nxt.locator # Knihovna na detekci řídící jednotky.
import nxt.sensor  # Knihovna na získání dat ze sensorů.
import nxt.motor   # Knihovna na řízení motorů.
from mypackage.msg import Sensors_values_NXT # Import zprávy.
from mypackage.msg import Motors_values_NXT  # Import zprávy.
msg = Sensors_values_NXT()
brick = nxt.locator.find_one_brick()         # Nalezeni řídící jednotky.
motor_right = Motor(brick, PORT_A)
 # Funkce na získání hodnoty z dotykového sensoru a nahrání do zprávy.
def get_touch1(self):
   msg.Touch_1 = Touch(brick, PORT_1).get_sample()
   return msg
 # Část funkce na řízení pravého motoru.
def motor_control(self,data):
 if data.mode == "run":
    motor_right.run(data.power) #Rozběh motoru s určitou rychlostí.
 elif data.mode == "brake":
    motor_right.brake()         #Zabrzdění motoru.
```

Motory robota bylo možné řídit nezávisle a to ve čtyřech režimech (*mode*). První režim *„run“* rozběhl motor s určitou rychlostí (*power*) dokud nebyl zastaven, zpráva tedy musela obsahovat hodnoty *mode* a *power*. Druhý režim *„turn“* rozběhl motor také určitou rychlostí, ale navíc s možností do zvolené vzdálenosti (*distance*), ve zprávě tedy musely být naplněny proměnné *mode, power* a *distance*. Třetí režim *„brake“* motor prudce kdykoliv zabrzdil a poslední režim *„idle“* také motor zabrzdil, ale bez nárazového zastavení.

Dále kód obsahoval konstrukce na vytvoření tří témat a samotného uzlu (viz níže).

```
rospy.init_node('NXT_API_talker_listener')
rospy.Subscriber("topic_nxt_API_motor_L", Motors_values_NXT, callback)
rospy.Subscriber("topic_nxt_API_motor_R", Motors_values_NXT, callback)
rospy.Publisher('topic_nxt_API_sensors', Sensors_values_NXT)
```

Problémem při vývoji tohoto uzlu bylo to, že knihovna NXT-Python neumožnila použít režim motoru *„turn“* ve stejný časový okamžik na oba motory. Řešení spočívalo v rozdělení příkazů na řízení pravého motoru do jednoho vlákna a levého motoru do druhého vlákna. Vlákna byla vytvořena pomocí knihovny *threading*.

#### *6.1.1.2 Parametr soubor*

Parametr soubor v balíčku obsahoval konfiguraci zapojení motorů a sensorů do řídící jednotky robota. Pokud například byl změněn port zapojení motoru, bylo nutné jen změnit parametr soubor. Nebylo tedy potřeba zasahovat do zdrojovém kódu uzlu.

Příklad nastavení dvou parametrů v souboru *robot_NXT.yaml*

```
motor_right: yes // Nastavení zda je pravý motor připojen.
port_MR: PORT_C  // Definice portu pravého motoru.
```

Příklad vyhodnocení parametrů ve zdrojovém kódu uzlu *nxt_API.py* (jazyk Python).

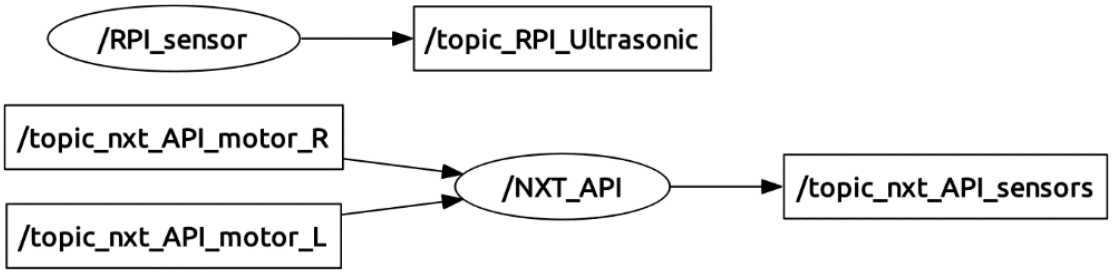
```
devices = rospy.get_param("/NXT_API")#Získání všech parametrů ze souboru.
if (devices['motor_right'] == True and devices['port_MR'] == 'PORT_A'):
 motor_right = Motor(brick, PORT_A)
elif devices['port_MR'] == 'PORT_B':
 motor_right = Motor(brick, PORT_B)
```

#### *6.1.1.3 Spouštěcí soubor*

Spouštěcí soubor zajistil spuštění obou uzlů a nahrání souboru s parametry na parametr server. Spuštění stanice proběhlo jediným příkazem níže:

```
$ roslaunch package_slave_1 slave_station_1.launch
```

Po spuštění stanice nástroj RQT Graph poskytl zobrazit uspořádání uzlů a témat, viz Obr. 42.




*Obr. 42 – Grafické zobrazení uzlů ve Slave stanice 1 (nástroj RGT Graph).*

## 6.2 Slave stanice 2

Druhá podřízená stanice v balíčku pouze obsahovala spouštěcí soubor (*slave_station_2.launch*), který spustil ovladač (uzel), který byl po nainstalování (viz kapitola 5.2.2.3) součásti systému ROS. Tento uzel bylo možné konfigurovat pomocí parametrů (značka <*param*> viz zdrojový kód níže).

Zdrojový kód spouštěcího souboru (jazyk XML):

```
<launch>

<node name="usb_cam1" pkg="usb_cam" type="usb_cam_node" output="screen" >
  <param name="video_device" value="/dev/video0" />//Připojovací adresa.
  <param name="image_width" value="640" />       //Šířka obrazu.
  <param name="image_height" value="480" />      //Výška obrazu.
  <param name="pixel_format" value="mjpeg" />    //Formát obrazu.
  <param name="camera_frame_id" value="usb_cam" /> //Identifikační název.
  <param name="io_method" value="mmap"/>
</node>

</launch>
```

Spuštění stanice proběhlo jediným příkazem (viz níže).

```
$ roslaunch package_slave_2 slave_station_2.launch
```

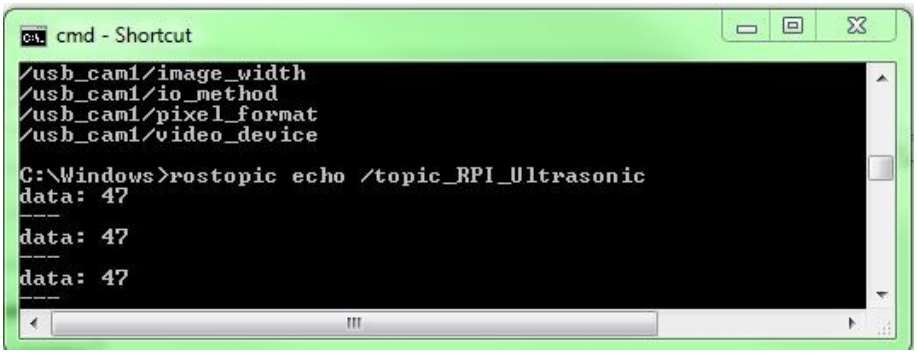
Po provedení příkazu bylo možné zobrazit grafickou reprezentaci uzlu, viz Obr. 43. Uzel poskytoval mnoho témat, kde nedůležitější bylo téma (*usb_cam1/image_raw*), do kterého uzel posílal nezpracovaný obraz. Z tohoto tématu Master stanice odposlouchávala obraz z webkamery.


*Obr. 43 – Grafické zobrazení uzlu komunikující s webkamerou (nástroj RQT Graph).*

## 6.3 Slave stanice 3

Jelikož verze ROS na této stanici byla experimentální, dovolila pouze diagnostiku celého řídicího systému pomocí konzolových nástrojů. Příkladem diagnostiky je odposlouchání naměřených hodnot z ultrazvukového sensoru, viz Obr. 44. Odposlouchání dat zajistil nástroj *rostopic*. Tvar příkazu je níže.

```
$ rostopic echo /topic_RPI_Ultrasonic
```




*Obr. 44 – Odposlouchání dat ze sensoru SRF08 na Slave stanici 3.*

## 6.4 Master stanice

Nadřazená stanice představovala hlavní výpočetní jednotku, která zpracovávala dohromady čtyři uzly. Do této stanice byla připojena externí klávesnice.

### 6.4.1 Definice balíčku

V pracovním prostředí Master stanice byl vytvořen balíček typu Catkin s pojmenováním *package_master*. Balíček obsahoval následující soubory, viz Tab. 13.

| Obsah balíčku - package_master (catkin) | |
|---|---|
| **Typ souboru** | **Název souboru** |
| Uzel (Python) | nxt_robot_mode.py, nxt_keyboard_control.py, nxt_color20_control.py |
| Uzel (C++) | nxt_keyboard_control_input.cpp |
| Zpráva | Motors_values_NXT.msg, Robot_mode_NXT.msg, Sensors_values_NXT.msg |
| Spouštěcí soubor | master_station.launch |

*Tab. 13 – Seznam souborů ve vlastním balíčku na Master stanici.*

Každý uzel na této stanici měl svoji specifickou funkci, a tím mohla být úloha stanice rozdělena na čtyři části. Hlavními úkoly byly: získat vstupní data z připojené klávesnice, zajistit přepínání provozních režimů robota, řídit robota manuálně pomocí klávesnice a automaticky při jízdě po černé čáře.

#### *6.4.1.1 Uzel 1*

První uzel (*nxt_keyboard_control_input.cpp*) typu *Publisher* detekoval snímání stisků kláves na klávesnici, konkrétně ze šipek [50]. Získaná data byla publikována pomocí zprávy (*Twist.msg*) na téma (*topic_keyboard_outputs)*. Definici zprávy poskytlo prostředí ROS, a to z balíčku *geometry_msgs*. Specifikace dat ve zprávě je zobrazena v Tab. 14. Při tvorbě zdrojového kódu uzlu bylo využito hlavní myšlenky projektu ROS a to konkrétně znovupoužitelnost kódu. Grafická reprezentace uzlu je znázorněna na Obr. 45.

| Zpráva - Twist.msg | | | |
|---|---|---|---|
| **Datový typ** | **Název proměnné** | **Hodnota** | **Popis dat z klávesnice.** |
| geometry_msgs/Vector3 | linear.x | -1 | Stisknutá šipka dolu. |
| | | 0 | Nestisknutá šipka dolu ani nahoru. |
| | | 1 | Stisknutá šipka nahoru. |
| | angular.z | -1 | Stisknutá šipka doprava. |
| | | 0 | Nestisknutá šipka doleva ani doprava. |
| | | 1 | Stisknutá šipka doleva. |

*Tab. 14 – Definice zprávy pro přenos dat z klávesnice.*

#### *6.4.1.2 Uzel 2*

Druhý uzel (*nxt_robot_mode.py*) typu *Subscriber/Publisher* zajistil přepínání mezi provozními režimy robota (manuální a automatický). Přepínání probíhalo pomocí dvou dotykových sensorů (tlačítek) umístěných na robotovi, kde jedno tlačítko sloužilo pro přepnutí do manuálního módu a druhé do automatického módu. Hodnoty z těchto senzorů byly odebírány z tématu *topic_nxt_API_sensors* ze zprávy *sensors_values_NXT.msg* (viz Tab. 11), která periodicky přicházela ze Slave stanice 1. Získaná data uzel vyhodnotil a v jednoduché zprávě *Robot_mode_NXT.msg* odeslal výsledek na téma *topic_nxt_Robot_mode*. Proměnná v odchozí zprávě byla datového typu *string* a nabývala buď hodnoty *„manu“* pro manuální režim nebo *„auto“* pro automatický režim. Při vyhodnocování dat byly výsledky nahrávány na parametr server, jelikož požadavky na změnu režimu přicházely impulsově. Toto řešení např. dovolilo výběr režimu ovládání při startu celé stanice a to díky nadefinování spouštěcího souboru, kde byla použita značka <*param*> pro nahrávání dat na parametr server. Uzel v grafické podobě je zobrazen na Obr. 45.

Vyhodnocení provozního režimu robota, viz níže (jazyk Python).

```
import rospy # Knihovna pro programování uzlů v Python.
from mypackage.msg import Sensors_values_NXT #Nahrání zprávy pro sensory.
from mypackage.msg import Robot_mode_NXT     #Nahrání zprávy pro režimy
robota.
msg1 = Sensors_values_NXT()
msg3 = Robot_mode_NXT()

def get_robot_mode():# Funkce pro výběr provozního režimu robota.
  if (msg1.Touch_1 == True and msg1.Touch_2 == False):     # Detekce
zmáčknutí dotykového sensoru 1.
    rospy.set_param('NXT_Robot_mode/string_NXT_mode',"manu")# Nahrání
hodnoty na parametr server - manuální režim.
  elif (msg1.Touch_2 == True and msg1.Touch_1 == False):   # Detekce
zmáčknutí druhého dotykového sensoru.
    rospy.set_param('NXT_Robot_mode/string_NXT_mode',"auto")# Nahrání
hodnoty na parametr server - automatický režim.
  msg3.mode = rospy.get_param("NXT_Robot_mode/string_NXT_mode")# Získání
hodnoty provozního režimu z parametr serveru.
  return msg3.mode
```

#### *6.4.1.3 Uzel 3*

Třetí uzel (*nxt_keyboard_control.py*) typu *Subscriber/Publisher* vytvářel příkazy pro řízení robota pomocí klávesnice. Podmínkou pro vydávání příkazů bylo přepnutí robota do manuální režimu. Tento uzel měl čtyři vstupy, neboli odebíral data ze čtyř různých témat (*topic_keyboard_outputs, topic_nxt_Robot_mode, topic_RPI_Ulstrasonic, topic_nxt_API_sensors*). Uzel tedy přijímal zprávy s daty od čtyř uzlů. První zpráva (viz Tab. 14) obsahovala hodnoty z klávesnice, druhá hodnoty ohledně provozního režimu, třetí hodnoty z SRF sensoru a čtvrtá hodnoty ze všech sensorů robota (viz Tab. 11). Data byla vyhodnocena a z výsledků byly vytvořeny příkazy (viz Tab. 12) pro řízení robota. Příkazy byly zvlášť posílány zprávou (*Motors_values_NXT*) na téma pravého motoru (*topic_nxt_API_motor_R*) a totožnou zprávou, ale s jinými hodnotami na téma levého motoru (*topic_nxt_API_motor_L*). Grafické zobrazení uzlu je vyobrazeno na Obr. 45.

Část zdrojové kódu uzlu pro řízení robota pomocí klávesnice, viz níže (jazyk Python).

```
msg1 = Sensors_values_NXT(), msgR = Motors_values_NXT()
msgL = Motors_values_NXT(), msg4 = UInt16()
msg2 = Twist()          # Přiřazení objektu zprávy klávesnici.

def keyboard_control(): # Funkce pro řízení robota klávesnicí (část).
if (msg1.Ultrasonic < 15 or msg2.angular.z == 0 and msg2.linear.x == 0 or
msg2.linear.x == -1 and msg4.data < 15 and msg4.data > 0 ): # Podmínka
pro zastavení robota, reakce na ultrazvukové sensory a klávesnici.
   msgR.mode = "idle"       #Zastavení motoru.
   msgL.mode = "idle"
 elif msg2.linear.x == 1: ")# Detekce stisku přední šipky.
   msgR.mode = "run"         # Roztočení motoru.
   msgR.power = 85           # Rychlost pro jízdu vpřed (pravý motor).
   msgL.mode = "run"
   msgL.power = 85
```

#### *6.4.1.4 Uzel 4*

Čtvrtý uzel (*nxt_color20_control.py*) typu *Subscriber/Publisher* prováděl řízení robota automaticky. Robot byl schopen jízdy po černé čáře díky barevnému RGB sensoru. Uzel odebíral zprávy ze tří témat (*topic_nxt_Robot_mode, topic_RPI_Ulstrasonic, topic_nxt_API_sensors*), respektive od tří jiných uzlů. První příchozí zpráva obsahovala data ohledně provozního režimu, druhá data z SRF sensoru a třetí hodnoty ze všech sensorů robota (viz Tab. 11). Odebraná data z témat byly uzlem následně vyhodnoceny. Z výsledků po vyhodnocení vznikly příkazy pro řízení motorů robota. Vzniklé příkazy byly odesílány ve zprávách stejným způsobem jako v třetím uzlu (viz kapitola 6.4.1.3). Zobrazení uzlu v grafické formě je znázorněno na Obr. 45.

Část zdrojové kódu uzlu, který řídil robota automaticky při jízdě po černé čáře, viz níže (jazyk Python).

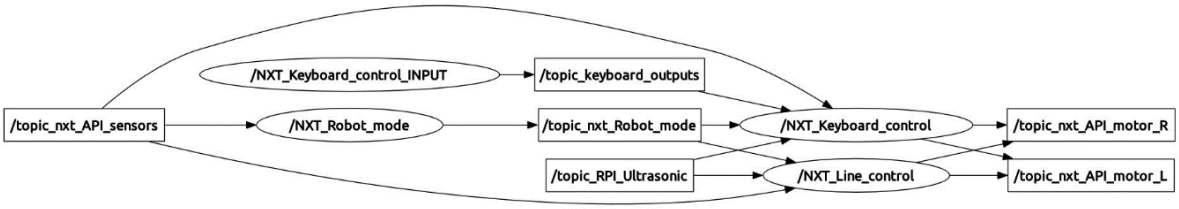
```
msg1 = Sensors_values_NXT()# Přiřazení objektu zprávy pro sensory robota.
msgR = Motors_values_NXT() # Přiřazení objektu zprávy pro pravý motor.
msgL = Motors_values_NXT() # Přiřazení objektu zprávy pro levý motor.
msg4 = UInt16()            # Přiřazení objektu zprávy pro SRF sensor.

def color20_control():     # Funkce pro jízdu po černé čáře.
 if (msg1.Ultrasonic < 15 or msg4.data < 15 and msg4.data > 0): #
Podmínka pro zastavení robota, reakce na ultrazvukové sensory.
   msgR.mode = "idle"      # Zastavení motoru.
   msgL.mode = "idle"
 elif msg1.Color20 == 1:  # Detekce černé barvy z RGB sensoru.
   msgR.mode = "run"       # Roztočení motoru.
   msgR.power = 70          # Dopředná rychlost.
   msgL.mode = "run"
   msgL.power = 70
 elif msg1.Color20 == 6:  # Detekce bílé barvy z RGB sensoru.
   msgR.mode = "run"
   msgR.power = 65
   msgL.mode = "run"
   msgL.power = -65       # Zpětná rychlost.
 return msgR, msgL
```

#### *6.4.1.5 Spouštěcí soubor*

Spouštěcí soubor zajistil spuštění všech uzlů obsažených v balíčku *package_master*. Spuštění stanice proběhlo jediným příkazem níže:

```
$ roslaunch package_master master_station.launch
```

Po spuštění stanice nástroj RQT Graph poskytl zobrazit uspořádání uzlů a témat, viz Obr. 45.

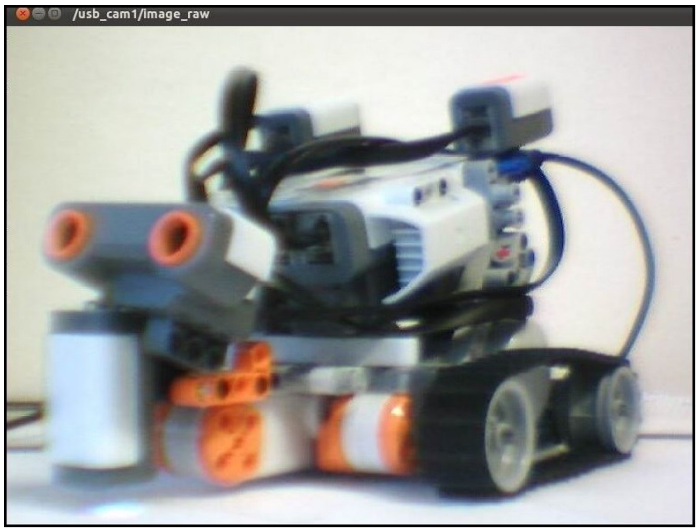



*Obr. 45 – Grafické zobrazení uzlů v Master stanici (nástroj RQT Graph).*

Sekundárním úkolem Master stanice bylo zobrazovat obraz z webkamery připojené ve Slave stanici 2. Tato podřízená stanice poskytla téma (*usb_cam1/image_raw*), ze kterého bylo možné přenášet obraz. Přenos a zobrazení obrazu v Master stanici zajistil uzel *image_view*, který je obsažen v balíčku *image_view*, který je standardně dostupný v prostředí ROS. Spuštění uzlu bylo nastaveno ve spouštěcím souboru, viz XML kód níže.

```
<node name="image_view" pkg="image_view" type="image_view"
respawn="false" output="screen">
    <remap from="image" to="/usb_cam1/image_raw"/> //Název tématu
    <param name="autosize" value="true" />
</node>
```

Po spuštění uzlu byl zobrazen obraz z připojené webkamery, viz Obr. 46. Grafické znázornění uzlu zobrazující obraz je ukázáno na Obr. 48.




*Obr. 46 – Zobrazení obrazu z webkamery pomocí uzlu image_view.*

## 6.5 Testování

### 6.5.1 Popis funkce výsledného řešení

Po spuštění celého řídicího systému byl pásový robot automaticky nastaven do manuálního režimu. V tomto režimu uživatel mohl robota ovládat pomocí šipek na klávesnici. Horní a dolní šipky odpovídaly jízdě vpřed a vzad. Při jízdě vpřed byl robot chráněn proti nárazu do překážky NXT ultrazvukovým sensorem a při jízdě vzad SRF ultrazvukovým sensorem. Robot se mohl přiblížit k překážce maximálně na 15 cm. Při stisku levé nebo pravé šipky se robot začal na místě otáčet. Uživatelské rozhraní robota reprezentovaly dvě tlačítka, kde při stisknutí prvního tlačítka byl nastaven manuální režim a při stlačení druhého byl robot uveden to automatického režimu. V automatickém režimu jezdil robot vpřed po černé čáře a to díky barevnému RGB snímači. Pokud robot vybočil z trajektorie černé čáry, začal se otáčet a hledat černou čáru. Při jízdě vpřed byl robot chráněn dvojicí ultrazvukových dálkoměrů.

Funkce celé testovací robotické aplikace byla zaznamenána videem (viz kapitola 6.5.6).

### 6.5.2 Různé verze ROS

Při testování multiplatformního systému bylo zjištěno, že verze ROS Groovy a Hydro nedokáží mezi sebou plně komunikovat, nejsou schopně přenášet data ve zprávě. Nicméně v rámci navrhnutého řešení bylo ukázáno, že přenos obrazu mezi stanicí Master (ROS Groovy) a stanicí Slave 2 (ROS Hydro) bylo uskutečněno. Ačkoliv v některých případech lze používat různé verze ROS při vzájemné komunikaci, je prioritně doporučeno pro vytváření multiplatformních systémů používat na každé stanici stejnou verzi ROS.

### 6.5.3 Uzel *nxt_API.py*

Pro testování robota z hlediska funkčnosti motorů nebo sensorů bylo možné spustit pouze uzel *nxt_API.py* na libovolné stanici, do které byl robot připojen, a na které byla provedena instalace připojení robota. (viz kapitola 5.1.2.1 nebo [48]). Poté bylo možné použít konzolové nástroje ROS pro diagnostiku robota. Příklady příkazů jsou zobrazeny níže.

Příkaz na roztočení levého motoru s rychlostí 70, viz níže.

```
$ rostopic pub /topic_nxt_API_motor_L package_slave_1/Motors_values_NXT
"power: 70 distance: 0 mode:'run'"
```

Příkaz na zobrazení hodnot ze sensorů robota.

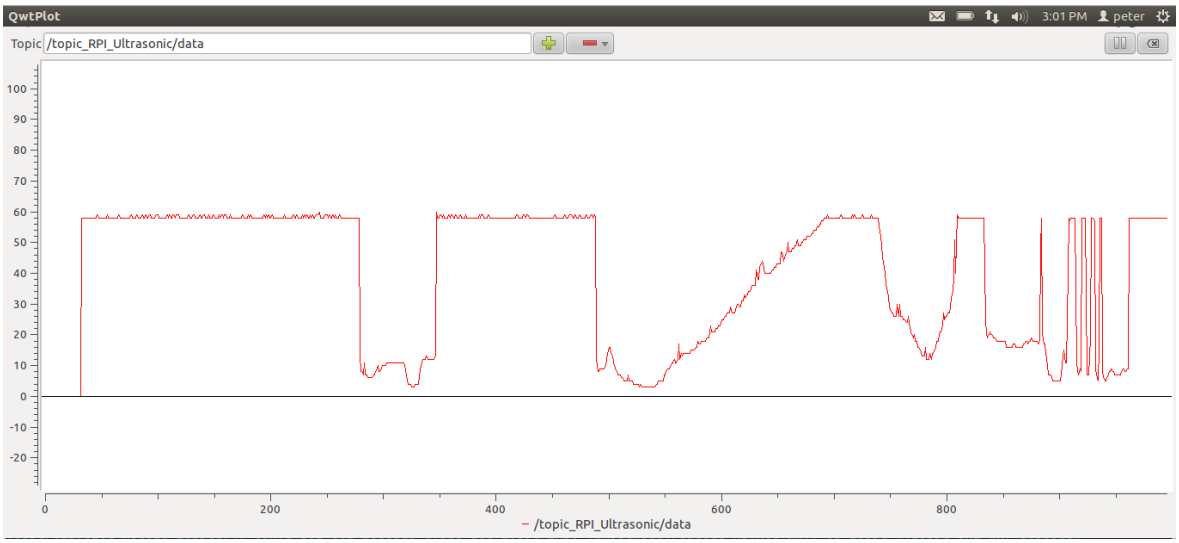
```
$ rostopic echo /topic_nxt_API_sensors
```

Možná odpověď (specifikace proměnných viz Tab. 11):

```
Touch_1: True, Touch_2: False
Ultrasonic: 27, Color20: 4
Tacho_L: 2020, Tacho_R: 2010
```

### 6.5.4 RQT plot

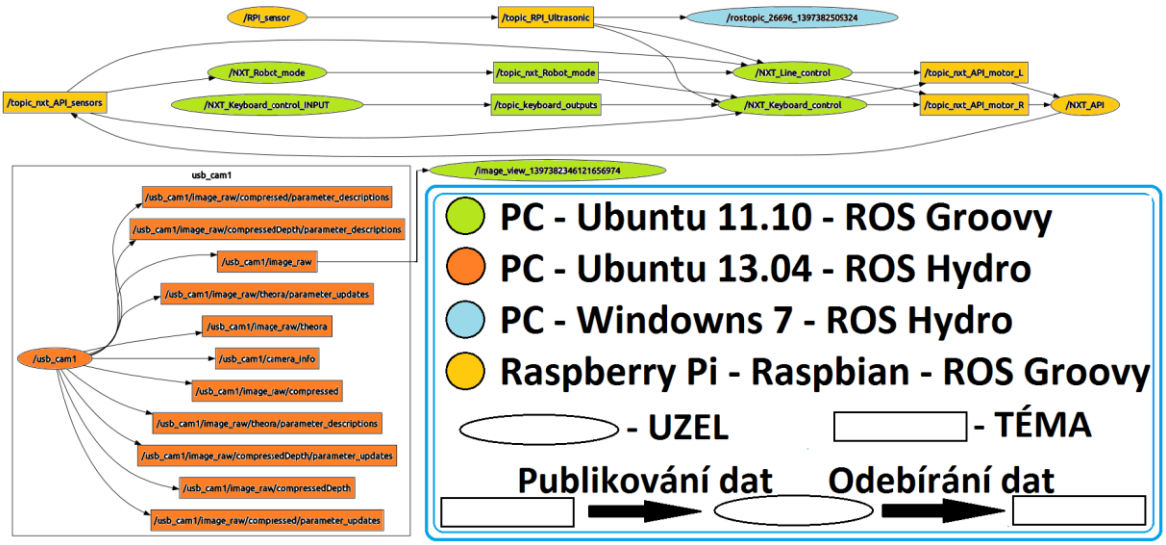
Grafický nástroj RQT plot odposlouchával datovou komunikaci z tématu */topic_RPI_Ultrasonic*, do kterého byla vysílána data (zprávy) z ultrazvukového dálkoměru SRF08. Ze zprávy byla zobrazena v grafu data datového typu *int8* v závislosti na čase, které obsahovaly aktuální hodnotu vzdálenosti ze sensoru v centimetrech. Průběh grafu z testování je zobrazen na Obr. 47.




*Obr. 47 - Testování dálkoměru SRF08 pomocí nástroje RQT plot.*

### 6.5.5 RGT Graph

Po spuštění celého řídicího systému bylo možné graficky zobrazit vzájemné propojení všech uzlů. Zobrazení zajistil nástroj RQT Graph, viz Obr. 48.




*Obr. 48 – Grafické zobrazení uzlů celé testovací robotické aplikace.*

### 6.5.6 Video

V rámci testování vzniklo video, které zobrazuje např. průběh testování SRF sensoru nebo demonstraci manuálního a automatického režimu ovládání mobilního robota. Toto video je dostupné z webového portálu YouTube zde [51] a nebo je ho možné shlédnout v příloze na CD (Compact Disc) této práce.

# 7. Tipy a triky s ROS

## 7.1 Učení ROS

Uživatelé, kteří s ROS začínají úplně od „nuly“ by měli nejdříve detailně nastudovat tuto diplomovou práci. Poté je doporučeno začít praktickým testováním systému, jelikož jedině praxí lze fungování celého systému nejlépe pochopit. K tomu jsou k dispozici užitečné tutoriály, které jsou umístěné zde [52]. Dalším způsobem zasvěcení do frameworku ROS je studium dostupných knih zde [53].

## 7.2 Dvě verze ROS a jeden OS

V případě potřeby používat ROS Groovy a ROS Hydro v jednom OS je nutné nainstalovat Ubuntu ve verzi 12.10, na které se provede instalace obou verzí ROS dle návodů zde [12]. Trik spočívá v nastavení souboru *.bashrc* (otevření pro editaci: `sudo nano ~/.bashrc`), kde se realizuje nastavení:

```
source /opt/ros/<groovy nebo hydro>/setup.bash
```

## 7.3 Záznam a přehrání videa

Pomocí konzolového nástroje *rosbag (4.1.14)* lze v ROS zaznamenat a přehrát video z připojené kamery. Použití nástroje lze demonstrovat na testovací aplikaci této diplomové práce. Ve Slave stanici 2 (viz 6.2) byl použit ovladač kamery, který zasílal obraz na téma *usb_cam1/image_raw*. Následuje použití nástroje *rosbag*, viz níže.

```
$ rosbag record /usb_cam1/image_raw //Ukládání videa.
```

Po ukončení záznamu je nutné se přemístit příkazem níže do adresáře, kde je soubor uložen.

```
cd /home/<uživatelské_jméno_v_OS>
```

Název souboru je zakončen příponou *.bag*. Tento soubor lze např. přemístit do jiné ROS stanice a tam použít příkaz pro přehrání videa, viz níže.

```
$ rosbag play <název_souboru>.bag
```

Tímto způsobem lze jednoduše např. vzdáleně zaznamenávat video mezi dvěma stanicemi. Kde maximální vzdálenost stanic je daná topologií počítačové sítě.

## 7.4 Dynamixel servo

Dalším tipem je možnost rychlého použití serva Dynamixel [54]. Do ROS je nutné nainstalovat ovladač příkazem:

```
$ sudo apt-get install ros-<verze_ROS>-dynamixel-motor
```

Po instalaci je nutné provést sestavení balíčku ovladače příkazem:

```
 $ rosmake dynamixel_motor
```

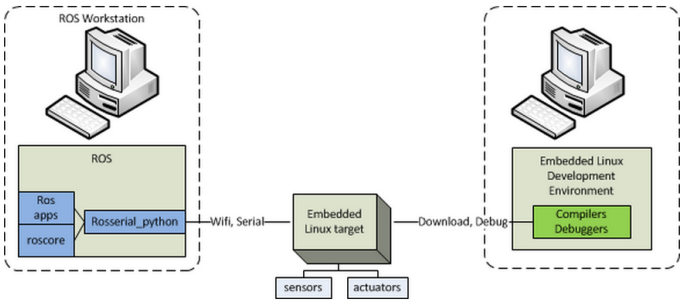
Poté následuje přemístění do adresáře ovladače příkazem:

```
$ roscd dynamixel_motor
```

V adresáři jsou hledány spouštěcí sobory, které ovládají servo motor. Pro roztočení serva je nutné nastudovat tutoriály zde [55].

## 7.5 ROS Serial

ROS Serial (rosserial) je velká sada balíčků, která představuje obecný protokol pro posílání ROS zpráv skrze sériovou linku. Tento protokol umožnuje použití ROS do tzv. *embedded systémů* s malou pamětí. Například ROS podporuje populární jednodeskový mikrokontrolér Arduino. [56]



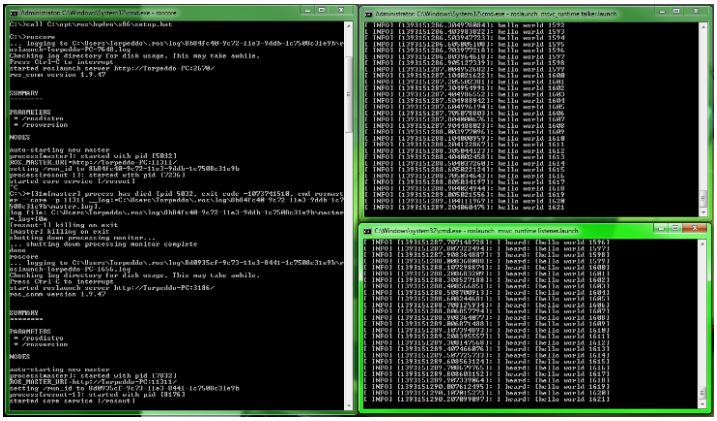

*Obr. 49 – Komunikace frameworku ROS s embedded systémem. [56]*

## 7.6 ROS na OS Windows

Pro testování základní komunikace, přenosu zpráv mezi dvěma uzly ve verzi ROS Hydro na OS Windows lze provést následující příkazy:

```
> call C:\opt\ros\hydro\x86\setup.bat     //Propojení ROS a Windows.
> roscore                                 //Spuštění jádra ROS.
> roslaunch msvc_runtime talker.launch  //Spouštění uzlu typu Publisher.
> roslaunch msvc_runtime listener.launch//Spouštění uzlu typu Subscriber.
```

Pokud je ROS na Windows správě nainstalován, výsledek po provedení příkazů je zobrazen na Obr. 50. [57]



*Obr. 50 – Test základní komunikace v ROS Hydro na OS Windows.*

# 8. ZÁVĚR

Prvotním cílem této diplomové práce bylo provedení podrobné analýzy robotického frameworku ROS. Zpracovaná analýza byla rozdělena na část teoretickou (viz kapitola 2.1) a praktickou (viz kapitola 4.1).

Z teoretické analýzy vyplývá, že framework ROS hraje v oblasti mobilní robotiky velkou roli. Svým postojem se snaží nastavit globálně v celosvětovém měřítku nový standart pro vytváření robotických aplikacích. A to se daří. Z oboru mobilní robotika dokázal oslovit ty nejlepší skupiny expertů zabývající se vědou a výzkumem robotů. I když ROS není real-time systém, tak svými možnostmi a kapacitami je použitelný k řešení velkého počtu robotických problémů.

Hlavními výhodami a silnými stránkami tohoto systému je modularita, distribuovaný design a možnost multiplatformního použití. Jádro sytému například poskytuje stabilní komunikační systém postavený na systému zasílání zpráv dle TCP/IP protokolu, což řeší prvotní úlohy při vytváření jakéhokoliv robotického projektu.

Za nevýhody lze označit to, že systém plně podporuje pouze jeden operační systém. Dále, že vývoj celého systému je velmi rychlý. Zhruba každých půl roku a až rok jsou vydávány nové verze, které přinášejí nové konvence, a které mezi sebou nejsou zcela kompatibilní.

Z hlediska kapacit ROS poskytuje nepřeberné množství robotického softwaru, který se neustále rozrůstá díky celé jeho komunitě. Tento software přestavuje ovladače pro sensory, motory a i celé roboty. Dále jsou k dispozici knihovny zaměřené na navigaci, lokalizaci, vytváření map, zpracování obrazu a mnoho dalších.

Praktická část analýzy frameworku ROS vysvětluje základní terminologii, způsob ovládání atd., objasňuje to, jak se systémem začít pracovat. Tím představuje důležitého průvodce (manuál) pro uživatele, kteří nemají žádné zkušenosti s tímto frameworkem. Například vysvětluje, jak vytvořit počítačovou síť různých platforem, která je z vnějšku viděna jako jeden distribuovaný řídicí systém.

Dle vykonané komplexní analýzy byla navrhnuta praktická testovací aplikace této diplomové práce. Podle zadání práce byla realizována komunikace ultrazvukového dálkoměru SRF s jednodeskovým počítačem pomocí $I^2C$ sběrnice, kde získaná data byla následně přeposílána na vyhodnocení do nadřazeného počítače pomocí protokolu TCP/IP. Tento senzorický systém byl otestován na navrhnutém mobilním robotu LEGO Mindstorms NXT. Výsledně tedy vznikla testovací robotická aplikace s mobilním robotem, která představovala distribuovaný multiplatformní řídicí systém, postavený na frameworku ROS. Kompletní vývoj systému je popsán v kapitole 5 a 6 a použitý hardware a software je popsán v kapitole 3 a 4.

Testovací systém disponoval čtyřmi stanicemi (čtyři operační systémy), kde každá stanice měla spoji specifickou funkci. V hierarchii stanic figurovala jedna nadřazená (Master) stanice, která vyhodnocovala data ze sensorů a vytvářela příkazy pro řízení robota. Zbylé podřízené (Slave) stanice zprostředkovaly senzorická data nebo vykonávaly příkazy dle Master stanice. Výsledné řešení poskytlo provozovat robota manuálně pomocí klávesnice nebo automaticky jízdou po černé čáře.

Výhodou celého řešení je to, že umožnuje jednoduše připojit do běžícího systému úplně novou stanici, ve které bude například běžet pouze uzel (*nxt_keyboard_control_input.cpp*) na snímání dat z klávesnice, a tím vznikne možnost ihned řídit robota úplně z jiného místa, kde omezení bude jenom podléhat síle rádiového signálu.

Testovací robotická aplikace dobře ukázala možnosti a kvality frameworku ROS. Vývoj takového řídicího systému by na stejné úrovni bez frameworku trval mnohonásobně déle. Dle zkušeností z realizace aplikace vzniká silné doporučení používat skriptovací jazyk Python, jelikož tento jazyk umožnuje daleko jednodušší a rychlejší vývoj uzlů.

O použitelnosti tohoto frameworku v mobilní robotice není pochyb. Dle mého názoru by měla být tomuto systému věnována velká pozornost, jelikož má obrovský potenciál a dokáže vývoj různorodých robotických aplikací rapidně urychlit. Mimo jiné je volně dostupný a zdarma. Považuji za velmi důležité se připojit do projektu ROS, který má podle mého názoru velkou budoucnost. Jako je dnes standartní používat Windows pro provoz osobních počítačů, tak stejně by se ROS mohl v budoucnu používat pro provoz robotů. Je potřeba naučit se ROS používat a brát ho jako nový standart při vývoji robotických aplikacích. Dále v budoucích vizích by bylo na místě navázat spolupráci a expandovat tento systém do jiných českých/zahraničních universit či výzkumných institucích zaměřených na robotiku.

Díky velkému rozsahu celého projektu ROS nebylo možné v této diplomové práci vše zcela detailně popsat a vysvětlit. Proto čtenář a nový uživatel ROS po nastudování této práce by měl postupovat dle doporučení v kapitole 7.1.

# Seznam použité literatury


[1] WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-16]. Dostupné z: http://www.ros.org/.

[2] Indroduction. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/ROS/Introduction.

[3] SOETENS, Peter. EURON. *The Orocos Project* [online]. 2002 [cit. 2014-05-16]. Dostupné z: http://www.orocos.org/.

[4] FEDOR, Christopher a Reid SIMMONS. *CARMEN Robot Navigation Toolkit* [online]. 1991 [cit. 2014-05-16]. Dostupné z: http://carmen.sourceforge.net/.

[5] STØY, Kasper. *The Player Project* [online]. 2001 [cit. 2014-05-16]. Dostupné z: http://playerstage.sourceforge.net/.

[6] MIT / LIRA-LAB COLLABORATION. *YARP middleware* [online]. 2010 [cit. 2014-05-16]. Dostupné z: http://eris.liralab.it/yarp/.

[7] MICROSOFT. *Microsoft Robotics Developer* [online]. 2012 [cit. 2014-05-16]. Dostupné z: http://msdn.microsoft.com/en-us/library/bb648760.aspx.

[8] OXFORD MOBILE ROBOTICS GROUP. *The MOOS* [online]. 2008 [cit. 2014-05-16]. Dostupné z: http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php.

[9] OROCOS@KTH. *Orca framework* [online]. 2004 [cit. 2014-05-16]. Dostupné z: http://orca-robotics.sourceforge.net/orca_doc_overview.html.

[10] Is ROS For Me?. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://www.ros.org/is-ros-for-me/.

[11] OSRF. *ROS Kong 2014* [online]. 2014 [cit. 2014-05-16]. Dostupné z: https://events.osrfoundation.org/ros-kong-2014/.

[12] Installation. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/ROS/Installation.

[13] Robots. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/Robots.

[14] Core Components. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://www.ros.org/core-components/.

[15] Integration. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://www.ros.org/integration/.

[16] ROS-INDUSTRIAL CONSORTIUM. *ROS Industrial* [online]. 2007 [cit. 2014-05-16]. Dostupné z: http://rosindustrial.org/.

[17] SRF08 UltraSonic Ranger. UNIVERSITY OF YORK. [online]. 2012. vyd. [cit. 2014-05-16]. Dostupné z: http://www.cs.york.ac.uk/micromouse/Docs/SRF08UltraSonicRanger.pdf.

[18] RASPBERRY PI FOUNDATION. *Raspberry Pi* [online]. 2012 [cit. 2014-05-16]. Dostupné z: http://www.raspberrypi.org/.

[19] LEGO. *LEGO Mindstrorms* [online]. 2009 [cit. 2014-05-16]. Dostupné z: http://www.lego.com/cs-cz/mindstorms/.

[20] THOMAS, Dirk. Distributions. WILLOW GARAGE. *Robot Operating System* [online]. 2007. vyd. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/Distributions.

[21] ROS VIRTUAL MACHINES. *Nootrix* [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://nootrix.com/downloads/#RosVM.

[22] CONLEY, Ken. Tools. WILLOW GARAGE. *Robot Operating System* [online]. 2007. vyd. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/Tools

[23] THOMAS, Dirk. Roslaunch. WILLOW GARAGE. *Robot Operating System* [online]. 2007. vyd. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/roslaunch.

[24] AXELROD, Ben. From Rosbuild to Catkin. WILLOW GARAGE. *Robot Operating System* [online]. 2007. vyd. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/catkin/migrating_from_rosbuild.

[25] WONNACOTT, Dereck. Create a Workspace. WILLOW GARAGE. *Robot Operating System* [online]. 2007. vyd. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/InstallingandConfiguringROSEnvironment.

[26] COLEMAN, Davet. Create a Package. WILLOW GARAGE. *Robot Operating System* [online]. 2007. vyd. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/CreatingPackage.

[27] Packages. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://www.ros.org/browse/list.php.

[28] TOSSELL, Ken. Sensors. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/Sensors.

[29] WOODALL, Willam. Publisher and Subscriber. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29.

[30] SAITO, Isaac. Rosnode. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/rosnode.

[31] CONLEY, Ken. Topics. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/Topics.

[32] CROSS, Cory. Messages. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/CreatingMsgAndSrv.

[33] FOOTE, Tully. Common msgs. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/common_msgs.

[34] WOODALL, William. Writing a Service and Client. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/WritingServiceClient%28python%29.

[35] CONLEY, Ken. Services. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/rosservice.

[36] KONLEY, Ken. Parametr Server. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/Parameter%20Server.

[37] THOMAS, Dirk. RQT. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/rqt.

[38] POOLEY, Acorn. RVIZ. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/rviz/.

[39] CROSS, Cory. Rosed tool. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/UsingRosEd.

[40] DIRK, Thomas. Rosbag tool. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/rosbag.

[41] KRUSE, Thibault. Roswtf. WILLOW GARAGE. *Robot Operating System* [online]. 2007 [cit. 2014-05-17]. Dostupné z: http://wiki.ros.org/roswtf.

[42] FOOTE, Tully. Multiple Machines. WILLOW GARAGE. *Robot Operating System* [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/MultipleMachines.

[43] ORACLE. *Virtual Box* [online]. 2014 [cit. 2014-05-25]. Dostupné z: https://www.virtualbox.org/.

[44] Windows 7. MICROSOFT. [online]. 2014. vyd. [cit. 2014-05-25]. Dostupné z: http://www.microsoftstore.com/store/msca/en_CA/pdp/Windows-7-Home-Premium/productID.246627500.

[45] CANONICAL LTD. *Ubuntu* [online]. 2014 [cit. 2014-05-25]. Dostupné z: http://www.ubuntu.com/.

[46] THOMPSON, Mike a Peter GREEN. *Raspbian* [online]. 2014 [cit. 2014-05-25]. Dostupné z: http://www.raspbian.org/.

[47] ANTMAN232. Raspberry Pi I2C (Python). *Instructables* [online]. 2013 [cit. 2014-05-17]. Dostupné z: http://www.instructables.com/id/Raspberry-Pi-I2C-STEPS

[48] WANNERS., Marcus. Instalation. *NXT-Python* [online]. 2013 [cit. 2014-05-17]. Dostupné z: https://code.google.com/p/nxt-python/wiki/Installation.

[49] AARON MARTINEZ, Enrique Fernández. *Learning ros for robotics programming* [online]. S.l.: Packt Publishing Limited, 2013 [cit. 2014-05-22]. ISBN 978-178-2161-448. Dostupné z: http://www.it-ebooks.info/read/3183/

[50] SIEBER, Max. Nxt_apps. *GitHub* [online]. 2014. vyd. [cit. 2014-05-22]. Dostupné z: https://github.com/maxsieber/nxt_apps/blob/master/nxt_teleop/src/nxt_key.cpp

[51] Master Thesis - Robot application based on the ROS. GOOGLE INC. *YouTube* [online]. 2014. vyd. [cit. 2014-05-22]. Dostupné z: http://www.youtube.com/watch?v=Ilzz5Qx8m4c.

[52] BOHREN, Jonathan. Tutorials. WILLOW GARAGE. *Robot Operating System* [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/ROS/Tutorials.

[53] OPITZ, Tobias. Books. WILLOW GARAGE. *Robot Operating System* [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/Books.

[54] Dynamixel motors. *ROBOTIS* [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://www.robotis.com/xe/dynamixel_en.

[55] CRUMPTON, Joe. Turorials Dynamixel motors. WILLOW GARAGE. *Robot Operating System* [online]. 2013 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/dynamixel_controllers/Tutorials/.

[56] BOUCHIER, Paul. ROS Serial. WILLOW GARAGE. *Robot Operating System* [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/rosserial.

[57] FONTANA, Flavio. ROS on Windows. WILLOW GARAGE. *Robot Operating System* [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/win_ros/hydro/Msvc%20SDK.

# Seznam obrázků

# Seznam tabulek

# Přílohy

## A - Optický disk CD-ROM

Obsah:

- ✓ Elektronická forma diplomové práce.
- ✓ Video z testování výsledného praktického řešení.