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 NAVIGAČNÍHO SYSTÉMU PRO MOBILNÍ ROBOT BENDER II

DESIGN AND REALIZATION OF NAVIGATION SYSTÉM FOR MOBILE ROBOT BENDER II

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE — LUBOŠ VENC
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. STANISLAV VĚCHET, Ph.D.
SUPERVISOR

BRNO 2015

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky

Akademický rok: 2014/2015

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student (ka): Luboš Venc

který/která studuje v **bakalářském studijním programu**

obor: **Aplikovaná informatika a řízení (3902R001)**

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

Návrh a realizace navigačního systému pro mobilní robot Bender II

v anglickém jazyce:

Design and realization of navigation system for mobile robot Bender II

Stručná charakteristika problematiky úkolu:

Hlavním cílem práce je návrh a realizace software pro autonomní mobilní robot Bender II. Navržený software bude sloužit jako hlavní navigační systém pro autonomní pohyb ve vnitřních prostorách budov. Navigační systém bude postaven na populárním frameworku ROS z důvodu dostupnosti již existujících knihoven které usnadní návrh vlastního navigačního systému.

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.

Vedoucí bakalářské práce: doc.Ing. Stanislav Věchet, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2014/2015.

V Brně, dne 24. 11. 2014

L. S.

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

Abstrakt

Tato práce je návodem pro návrh a realizaci navigačního systému, který je určen pro navigaci ve vnitřních prostorách budov. Z důvodu již existujících knihoven pro práci s laserovým scannerem a předpřipraveným navigačním balíčkem byl pro realizaci navigačního systému zvolen robotický operační systém ROS. Hlavním senzorem pro orientaci v prostoru je tedy laserový scanner SICK LMS2xx.

Abstract

This project is a guide to the navigation system realization, which is designed for a navigation inside of a building. Because of already existing libraries for the work with laser scanners, and prepared navigation package, the robotic operation system, ROS, had been selected. Therefore, the main sensor for a space orientation is the laser scanner SICK LMS2xx

Klíčová slova

Navigační systém, ROS (robotický operační systém), framework, laserový scanner, Ubuntu, BeagleBoard, indoor navigace.

Keywords

Navigation systém, ROS (Robot Operating Systém), framework, laser scanner, Ubuntu, BeagleBoard, indoor navigation.

Prohlášení o originalitě

Prohlašuji, že tuto bakalářskou práci na téma Návrh a realizace navigačního systému pro mobilní robot Bender II jsem vypracoval samostatně pod vedoucím práce doc. Ing. Stanislavem Věchetem, Ph.D., a pro vypracování této práce jsem použil pouze uvedenou literaturu a informační zdroje.

V Brně dne 29. 5. 2015 ……

Bibliografická citace

VENC, L. *Návrh a realizace navigačního systému pro mobilní robot Bender II.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 47 s. Vedoucí bakalářské práce Stanislav Věchet.

Poděkování

Tímto bych chtěl poděkovat vedoucímu bakalářské práce doc. Ing. Stanislavu Věchetovi, Ph.D., za věnovaný čas a důležité informace, které mi poskytl při tvorbě této práce.

Dále bych chtěl poděkovat své rodině a mé dívce, kteří mi stáli oporou a podporovali mě při tvorbě závěrečné práce.

Obsah

1 Úvod

Navigační systém určuje aktuální polohu robota, poskytuje informace o prostředí, ve kterém se robot pohybuje, vykonává navigační úkoly, jako je vyhledání nejvhodnější cesty z aktuální polohy do polohy požadované nebo lokalizaci robota v předem známé mapě. Tyto navigační úkoly jsou často velice komplikované a jejich řešení by zabralo mnoho času. A proto je ve světě robotiky velkým trendem používání již vytvořených a připravených knihoven, které tyto úkoly dokážou řešit.

Pro navigační software, který byl vytvořen v této práci, byl zvolen populární framework ROS. Ten obsahuje rozsáhlou sbírku knihoven pro práci s laserovými dálkoměry, ultrazvukovými dálkoměry, GPS a mnoha dalšími senzory. Dále obsahuje knihovnu, která je určena pro navigaci robotů ve vnitřních prostorách budov, tato knihovna obsahuje algoritmy pro plánování tras nebo pro lokalizaci. ROS také poskytuje nástroje pro vytváření map a dále je zde k dispozici vizualizační nástroj Rviz, ve kterém je možné celý navigační systém nejen zobrazit, ale také i zároveň ovládat.

Velkou výhodou tohoto softwaru je jeho distribuovaný design, díky němuž je možné náročné výpočetní úkoly rozložit do několika platforem a zrychlit tím běh celého systému. Další výhodou je podpora uživatelů, kteří se podílí na řešení různých problémů, jenž při vytváření robotických aplikací vznikají. Podle statistik z roku 2014 ROS uvádí až 22 000 lidí, kteří jsou zapojeni do online podpory.

Tato práce vznikla ve spolupráci s kolegou Janem Meindlem, ten se staral o elektronickou část robota, řízení jednotlivých hardwarových komponentů a schopnost robota reagovat na jednotlivé příkazy, které mu byly posílány ze systému ROS.

Cílem této práce je vytvoření návrhu a samotná realizace navigačního systému pomocí frameworku ROS. Tato práce bude dále sloužit jako návod, jsou zde popsány jednotlivé kroky, jež bylo třeba splnit pro vytvoření funkčního navigačního systému pro mobilního robota Bender II.

2 Navigační systém mobilního robota

Navigační systém patří mezi nejdůležitější systémy, které tvoří autonomní chování robotů. Mezi nejčastější úkoly tohoto systému patří především poskytnutí informací o okolí, díky nimž se robot dokáže přemístit z aktuální pozice do pozice požadované. Tento úkol může být omezen několika podmínkami, z nichž asi nejdůležitější podmínkou je vyhnout se překážkám, které stojí v cestě. Další podmínky pro navigační systém mohou být například vyhledání nejkratší cesty k cílové poloze nebo průjezd v nejkratším časovém intervalu. Pro splnění těchto úkolů musí navigační systém obsahovat algoritmy pro lokalizaci robota, ty zajišťují, aby robot věděl, kde se momentálně nachází. Dále potřebuje algoritmy pro plánování trasy, pomocí kterých se dokáže vyhýbat překážkám a najít nejvhodnější cestu. Aby navigační systém správně fungoval, musí získat přesné informace ze senzorického systému, zde jde o informace, podle kterých se určuje poloha a orientace robota vůči globálnímu nebo lokálnímu souřadnému systému, a mimo jiné také rozpoznání okolí robota [1].

2.1 Globální souřadný systém

Lze hápat jako souřadný systém, který je zvolen tak, aby obsahoval celý pracovní prostor robota. Globální navigace určuje polohu a orientaci robota vůči globálnímu souřadnému systému.

2.2 Lokální souřadný systém

Tento systém se dá také nazvat jako souřadnicový systém robota, přičemž lokální navigace je nadřazena globální navigaci. Její funkce spočívá v tom, že zpracovává informace o překážkách v omezené vzdálenosti od robota. Tato vzdálenost musí být však taková, aby řídicí systém dokázal na překážky reagovat a nedošlo ke kolizi. Vzdálenost je tedy vždy individuální, s ohledem na fyzické možnosti robota. Z čehož plyne, že při volbě senzorů nemají vliv pouze jejich vlastnosti, ale také parametry robota a jeho okolí.

2.3 Relativní navigace

K určení polohy využívá měření přírůstku změny polohy a orientace od počáteční do polohy aktuální. Problémem těchto technik je, že zde při každém měření vzniká chyba, která se kumuluje při každém dalším měření, a proto je vhodné tyto metody kombinovat s absolutním systémem navigace.

2.4 Absolutní navigace

Metody absolutní navigace určují polohu robota vůči globálnímu souřadnému systému, kde jsou zaneseny významné body – tzv. referenční, díky nimž tyto metody dokážou určit polohu robota [1].

3 Lokalizace

Lokalizace je též systémem, který patří mezi nejdůležitější části navigačního systému jako celku pro autonomní roboty. Lze ji chápat jako děj, při kterém se robot snaží zorientovat a najít svojí polohu v prostředí, ve kterém se právě nachází. Problematiku lokalizace můžeme rozdělit do několika skupin.

První skupinou je **lokální lokalizace**. Může být také označována jako sledování polohy (position tracking). V tomto případě je známa počáteční poloha robota, dále však vlivem nepřesného měření při pohybu robota začíná být jeho poloha nepřesná. Úkolem této lokalizace je opravovat takto vzniklé odometrické chyby a eliminovat nepřesnost aktuální polohy.

Další skupinou je **globální lokalizace**, v tomto případě robot nezná svoji počáteční polohu v prostoru a při dalším pohybu se snaží svoji polohu najít, např. podle známe mapy a dále ji zpřesňovat.

A poslední skupinou je tzv. **problém únosu**, patří mezi nejtěžší případy lokalizace. Princip spočívá v tom, že robot je ze známého prostředí skokovou změnou přenesen do prostředí neznámého. Úkolem této lokalizace je, aby se robot přizpůsobil novému prostředí. Tato metoda se používá pro testování schopnosti navigačních systémů [2].

3.1 Odometrie

Tato metoda lokalizace patří mezi nejstarší a nejpoužívanější metody. Pomocí dat získaných z inkrementálních snímačů, (enkodéru), které jsou umístěny na kolech a poskytují informace o ujeté vzdálenosti nebo aktuální rychlosti jednotlivých kol, a zároveň díky známému kinematickému modelu robota dokáže tato metoda vypočítat změny polohy robota. Pokud chceme, aby tato metoda fungovala co nejpřesněji, je nutné mít inkrementální snímače s co největším rozlišením a výpočty provádět v co nejkratším intervalu.

S touto metodou se mnohdy pojí chyby, které vznikají při měření nebo špatně nadefinovaném kinematickém modelu robota. A proto je tato metoda často doprovázena dalšími lokalizačními metodami [1].

3.2 Metoda PCSM (Pre-Computed Scan Matching)

Jednou z mnoha metod lokalizace je metoda přepočítaných scanů. Tato metoda lokalizace je určená pro méně výkonné, z pravidla vestavěné, počítače. Výhodou této metody je její použitelnost i v reálném čase.

Principem této metody je porovnání aktuálních dat ze senzorů robota s mapou, která je tvořena předpočítanými scany. Rychlost a přesnost lokalizace závisí na počtu uložených scanů okolí, čím více dat k porovnání je k dispozici, tím je metoda lokalizace přesnější a rychlejší. Avšak nevýhodou rostoucího počtu porovnání je růst výpočetní náročnosti. Na shodě scanů s předpočítanými informacemi závisí použitelnost metody, je totiž nutné definovat funkci, která analyzuje míru shodnosti [3].

4 Hardware

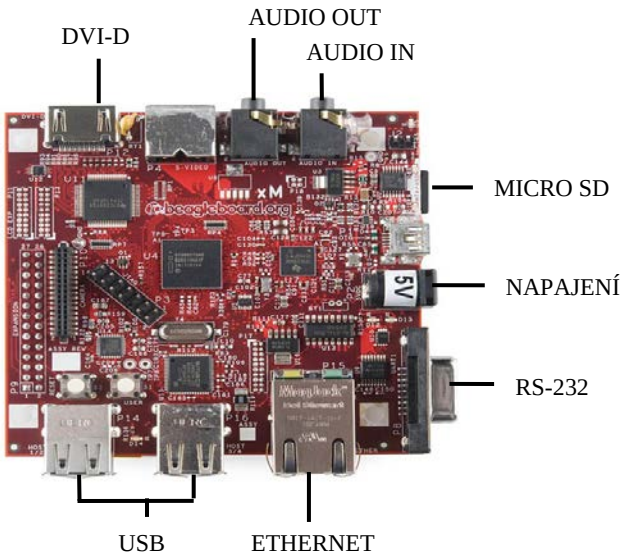
V této kapitole je popsána hardwarová část, která byla použita k realizaci navigačního systému. Dále zde popisuji rozmístění jednotlivých komponentů, které tvoří robota Bender II.

4.1 BeagleBoard

Jedná se o jednodeskový počítačový systém řady xM, který byl vytvořen firmou Texas Instruments. Počítač obsahuje procesor Texas Instrument Cortex A8 o frekvenci 1GHz, operační paměť 512 MB. Dále obsahuje 4x integrovaný USB port pro připojení externích zařízení, Micro SD slot pro microSD kartu, na které je nainstalován operační systém, Ethernet pro připojení do sítě pomocí UTP kabelu, pro sériovou komunikaci je tu konektor RS-232, zvukový vstup a výstup pro připojení zařízení jako jsou reproduktory nebo mikrofon. Pro připojení externího monitoru je poskytováno rozhraní HDMI-DVI-D konektor, zde musíme být obezřetní, výrobce totiž udává, že nesmí být připojen při zapnutém počítači, připojuje se tedy před zapnutím počítače. Celý systém je napájen zdrojem o 5V. Rozmístění jednotlivých konektorů můžete vidět na obrázku (viz Obr. 1) [4].

BeagleBoard obsahuje mnoho dalších komunikačních multifunkčních rozhraní, jako jsou GPIO, J2 konektor, kamerový port a další, které jsou dále důkladně popsány na stránkách výrobce [4].

BeagleBoard je umístěn přímo na robotu a slouží jako hlavní řídící jednotka celého robota, dále je k němu připojen laserový dálkoměr SICK a mikrokontrolér, pomocí něhož komunikuje s low-levlem robota. Na tomto počítači je nainstalován operační systém Ubuntu a framework ROS, který si více přiblížíme v softwarové kapitole a v kapitole, jež bude věnována frameworku ROS (viz kapitola 5. 2).



Obr. 1 – BeagleBoard xM[4]

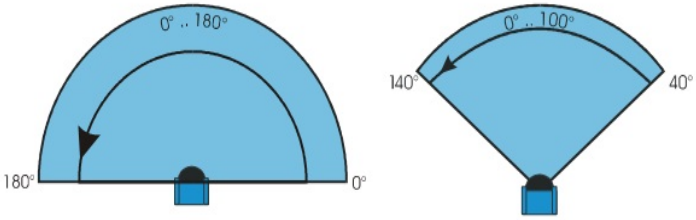
4.2 Laserový Scanner LMS 291

Jedná se o velmi přesný laserový dálkoměr, který byl vyvinut německou firmou SICK. Základním principem tohoto dálkoměru je posílání laserového paprsku do okolí a zároveň počítání času, za který se tento paprsek vrátí zpět. Na základě těchto informací vypočítá vzdálenost předmětu, který se nachází v prostoru, jež je snímán scannerem.

Scanner může pracovat v několika režimech a rozlišeních (viz Obr. 3). Jedním z nich je možnost zorného pole 0 ° až 100 °, kde může mít rozlišení 1 °, 0.5 °, 0.25 °, což znamená, že při největším rozlišení v tomto modu můžeme dostat až 400 hodnot, které nám poskytnou ucelené informace o okolí. Další možností je 0 ° až 180 ° a rozlišení 1 ° nebo 0.5 °, přičemž v tomto případě můžeme dostat až 360 hodnot. Dále je možné nastavit jednotky, ve kterých má scanner měřit a to například centimetry nebo milimetry, kde v centimetrech může měřit vzdálenosti až 81 metrů a v milimetrech to je 8.1 metrů [5].

Laserový dálkoměr je připojen do BeagleBoardu pomocí USB a je umístěn tak, aby sledoval prostor před robotem. Pro práci a zpracování dat ze skeneru je používána speciální knihovna, kterou obsahuje systém ROS a kterou dále popíši v kapitole 7.1.1.

Obr. 2 – Laserový dálkoměr SICK LMS 291[5]



Obr. 3 – Režimy snímá

4.3 Mikrokontrolér mbed LPC1768

Mbed LPC1768, můžete vidět na obrázku (viz Obr. 4), obsahuje 32bit ARM Cortex-M3 jádro, jež běží na 96 MHz, 512KB flash, 32KB RAM a spoustu dalších rozhrání jako SPI, I2C, ADC, DAC, PWM. Pro komunikaci s PC může využívat Ethernet nebo USB rozhraní [6].

Hlavní funkcí mikrokontroléru je přijímání příkazů pro motory a natočení kol, které vyhodnocuje a pomocí nichž robota dále ovládá. Také provádí výpočet odometrických dat a ty posílá jednodeskovému počítači. Všechny tyto procesy probíhají mezi mbedou a BeagleBoardem, jež spolu komunikují pomocí USB rozhraní.

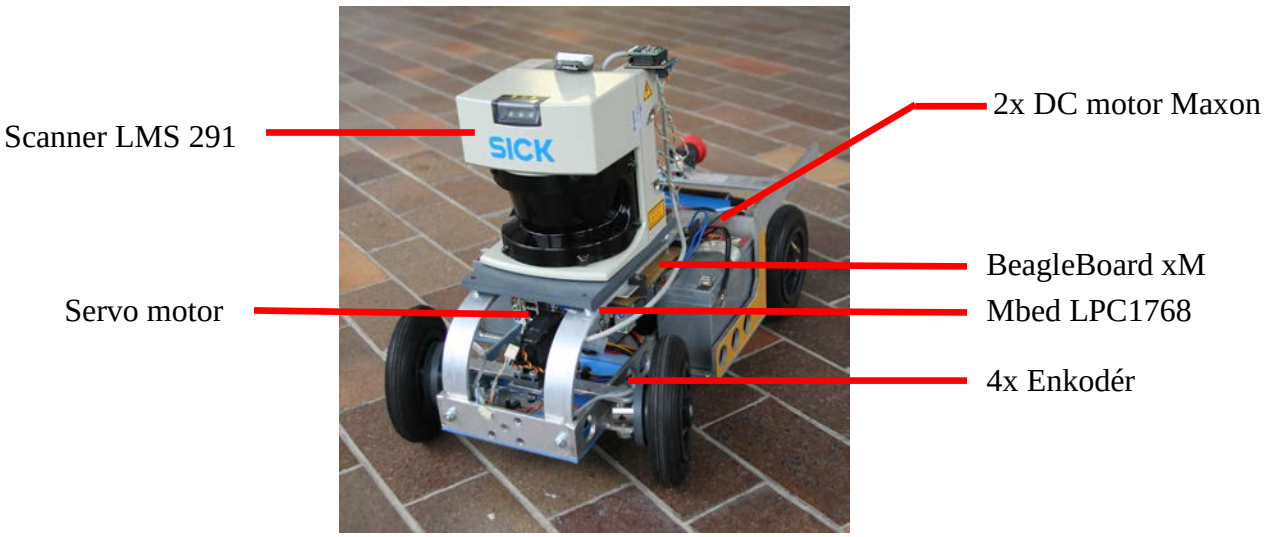
Obr. 4 – med LPC1768[6]

4.4 Ovládací počítač

Pro všechny výpočty, vizualizaci a ovládání navigačního systému byl zvolen notebook MSI GE620DX, který obsahuje 2 jádrový procesor i5-2430M, grafickou kartu NVIDIA GT 555M 2GB a 4GB RAM, což je dostačující výkon pro účely této bakalářské práce. Notebook a BeagleBoarde byl propojen do jedné lokální sítě.

4.5 Robot Bender II

Robot dále obsahuje čtyři inkrementální senzory (enkodéry), ty slouží pro výpočet odometrických dat. Jako pohon používá dva stejnosměrné motory od firmy Maxon, a pro řízení natočení kol modelářský servo motor. Jako napájení využívá dvě 12V baterie (viz Obr. 5).



Obr. 5 – Robot Bender II

5 Software

V této kapitole popíši softwarovou část, která byla použita pro tvorbu navigačního systému. Především se tato kapitola zaměřuje na popis instalace, nastavení a funkce operačního systému ROS, dále se zmíním o linuxovém operačním systému Ubuntu.

5.1 Ubuntu

Ubuntu je linuxový operační systém, jež je vyvíjen britskou společností Canonical. Tento software je ke stažení zcela zdarma. Dá se spouštět přímo z CD nebo je možná instalace například z flash disku [7].

Tento software byl v navigačním systému použit dvakrát, a to jak na notebooku, tak i na jednodeskovém počítači. V obou případech byla použita verze Ubuntu 14.04 Trusty Tahr. Tato verze byla zvolena z důvodů podporovanosti nejnovější verze systému ROS.

Obr. 6 – Logo Ubuntu [7]

5.2 ROS

V této podkapitole se budeme zabývat robotickým operačním systémem ROS, jeho instalací a nastavením, které bylo nutné provést pro účely této práce.

Obr. 7 – Logo systému ROS [8]

5.2.1 Popis

Jedná se o robotický operační systém ROS, který slouží k vytváření robotických aplikací. Tento software obsahuje rozsáhlou sbírku různých knihoven, například pro navigaci, mapování, lokalizaci, plánovaní a vizualizace robotických aplikací. Dále obsahuje ovladače pro práci s různými senzory, jako jsou např. laserové dálkoměry, ultrazvukové dálkoměry a mnoho dalších.

Cílem tohoto robotického systému je usnadnění práce svým uživatelům, díky těmto již vytvořeným knihovnám, při vytváření robotických aplikací. ROS se snaží být co nejvíce flexibilní a rozsáhlý robotický software, kde si uživatel rozhodne, kterou část software použije a kterou si vytvoří sám [8].

Z názvu robotický operační systém se může zdát, že se skutečně jedná o operační systém. ROS jím však sám o sobě není, a proto je lépe ho nazývat jako framework, pro svůj běh potřebuje, aby v zařízení, na kterém má fungovat, byl nainstalovaný plnohodnotný operační systém. V této bakalářské práci byl použit již zmíněný operační systém Ubuntu.

5.2.2 Historie

Framework ROS vznikl jako známý akademický Stanford AI projekt. Tento projekt následně převzala v roce 2008 společnost Willow Garage. ROS byl od začátku svého vývoje označován jako open-source systém, tudíž veškerá jeho distribuce je zcela zdarma a díky tomu jeho použití s každým rokem roste. Je mimo jiné využíván na předních světových univerzitách po celém světě. Tomu nasvědčuje i velmi aktivní ROS komunita, tvořena ros – uživateli, kterých je přes 1500. Dále více než 3300 uživatelů, spolupracujících na tvorbě online dokumentací a asi 5700 uživatelů, kteří se podílejí na řešení online problémů [9].

5.2.3 Instalace

Poté, co máme připravenou platformu s plnohodnotným operačním systémem, můžeme přistoupit k instalaci frameworku ROS. Protože na notebooku i na jednodeskovém počítači beagleboard jsou nainstalovány stejné verze operačního systému, byla pro obě platformy zvolena i stejná verze ROS systému, a tou je verze Indigo Igloo. Tato verze vyšla 22. července 2014 a momentálně patří mezi druhou nejnovější verzi [10]. Pro přesné informace, nejnovější verzí je verze ROS Jade.

Obr. 8 – ROS Indigo Igloo [10]

Instalace frameworku ROS se provádí pomocí zadávání příkazů do příkazového řádku v terminálu. Instalace probíhá online z repozitáře, je zapotřebí si přidat zdroj, z kterého bude ROS instalován pomocí příkazu:

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

Nyní když máme přidaný zdroj, z kterého se ROS instaluje, můžeme přejít přímo k instalaci. Byla zvolena plná verze, která obsahuje všechny nástroje ROS. Příkaz pro instalaci vypadá takto:

```
  sudo apt-get install ros-indigo-desktop-full
```

Poté co se provede instalace, je ještě nutné před spuštěním systému přepsat skrytý soubor, který nalezneme v kořenovém adresáři našeho OS a jmenuje se *.bashrc*. Do tohoto souboru na konec připíšeme následující příkaz, který zajistí propojení našeho OS s frameworkem ROS při každém spuštění terminálu:

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

5.2.4 Nastavení pracovního prostředí

Před samotným používáním systému ROS je třeba vytvořit pracovní prostředí, ve kterém budeme vytvářet vlastní balíčky a uzly. Protože používáme verzi Indigo Igloo, musíme zvolit prostředí *Catkin*, které vznikne pomocí zadání níže uvedených příkazů do příkazového řádku:

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

Těmito třemi příkazy si vytvoříme adresář *src*, poté se do něj přesuneme a inicializujeme si pracovní prostředí.

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

Dále se přesuneme do adresáře *catkin_ws* a sestavíme si pracovní prostředí.

```
source devel/setup.bash
```

Nakonec provedeme konfiguraci pracovního prostředí pomocí posledního příkazu.

Nutností je propojit pracovní prostředí se systémem ROS. Aby se toto propojení provedlo při každém novém spuštění terminálu, musíme znovu přepsat již zmíněný skrytý soubor *.bashrc*. Na konec tohoto souboru znovu přidáme příkaz uvedený níže.

```
source ~/catkin_ws/devel/setup.bash
```

Poté co jsme vytvořili pracovní prostředí, bychom měli zkontrolovat, zda je správně nastavena proměnná *ROS_PACKAGE_PATH*, ve které jsou uloženy cesty k našemu pracovnímu prostředí. Tu zkontrolujeme pomocí tohoto příkazu:

```
echo $ROS_PACKAGE_PATH
```

Pokud je proměnná správně nastavena, příkazový řádek by nám měl vypsat například toto:

```
/home/<uživatel>/catkin_ws/src
/opt/ros/indigo/share
/opt/ros/indigo/stacks
```

A pokud tomu tak není, je nutné do této proměnné přidat cestu k našemu pracovnímu prostředí. Přidání cesty do proměnné *ROS_PACKAGE_PATH* se provádí pomocí příkazu:

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

5.2.5 Catkin Balíčky

Základními prvky, které tvoří celý ROS systém, jsou tzv. balíčky. Tyto balíčky se vytvářejí ve složce *src*. Strukturu pracovního prostředí a balíčku můžete vidět níže.

```
workspace_folder/        -- pracovní prostředí
  src/                   -- adresář pro balíčky
    CMakeLists.txt       -- 'Toplevel' CMake soubor
    balíček_1/
      CMakeLists.txt     -- CMakeLists.txt soubor pro package_1
      package.xml        -- Soubor s informacemi o package_1
    ...
    balíček_n/
      CMakeLists.txt     -- CMakeLists.txt soubor pro package_n
      package.xml        -- Soubor s informacemi o package_n
```

Ve složce src se nachází soubor CMakeLists.txt, který si vytváří catkin prostředí automaticky, při každém sestavení pracovního prostředí, a jsou v něm informace o balíčcích, které jsou momentálně nainstalovány a připraveny k použití. Dále každý balíček musí obsahovat složku CMakeLists.txt ve které jsou informace o závislostech na jiné baličky a další informace [11]. A v neposlední řadě musí obsahovat soubor package.xml, tento soubor je psaný ve značkovacím jazyce XML a obsahuje informace o autorovi, funkci, jménu a další informace o balíčku [12].

Jsou dva způsoby jak tyto balíčky získat. První způsob je, že si ho vytvoříme sami. Vytvoření balíčku se provádí pomocí příkazů:

```
cd ~/catkin_ws/src
catkin_create_pkg <jméno_balíčku> [závislost_1] [závislost_n]
```

Prvním příkazem se přesuneme do složky *src*, ve které se balíčky vytvářejí. A druhým příkazem si balíček vytvoříme, tento příkaz má dva argumenty. Do prvního argumentu zadáváme jméno balíčku, který chceme vytvořit. Do druhého zadáváme balíčky (knihovny), které bude vytvořený balíček potřebovat – tzv. závislosti. Nejčastější balíčky, které se zadávají do druhého argumentu, jsou knihovna *rospy*, pomocí které můžeme v balíčku vytvářet uzly v jazyce Python a knihovna *roscpp*, pomocí které můžeme vytvářet uzly v jazyce C++.

Druhým způsobem jak získat balíček je stáhnutí již vytvořeného a připraveného balíčku. ROS poskytuje velkou sbírku těchto balíčků a všechny je můžete najít zde [13] a zde [14]. Instalace se může provádět příkazem:

```
apt-cache search ros-indigo
sudo apt-get install ros-indigo-<jméno_balíčku>
```

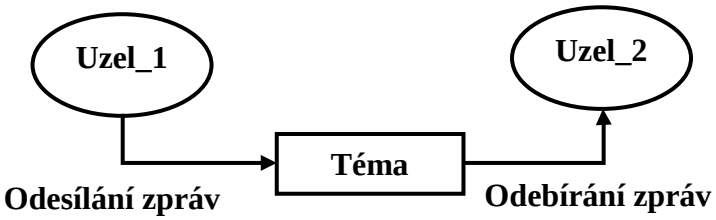
Pomocí prvního příkazu si najdeme dostupné balíčky pro verzi Indigo a druhým příkazem si tento balíček nainstalujeme. Další možností pro stažení cizího balíčku je stažení balíčku z repozitáře. To se provádí pomocí příkazu:

```
git clone https://github/<autor_balíčku>/<název_balíčku>.git
```

Po každém instalování balíčku bychom se měli přesunout do složky *catkin_ws* a provést sestavení pracovního prostředí pomocí příkazu *catkin_make*.

5.2.6 Komunikační systém

Komunikace v systému ROS je založena na anonymním publikování/odebíraní zpráv přes tzv. *téma*. Zprávy jsou odesílány/přijímány pomocí tzv. *uzlů*, které lze chápat jako procesy běžící na jednotlivých platformách robota, tyto procesy jsou na sobě nezávislé. ROS v současné době podporuje komunikaci pomocí protokolu TCP/IP a UDP [15].



Obr. 9 – Systém komunikace v ROS

5.2.7 Uzel

Uzly patří mezi základní prvky, které tvoří systém ROS. Můžeme si je představit jako programy nebo skripty, které se spouštějí a ukončují při běhu systému ROS. Máme tři druhy, prvním je uzel typu *Publisher*, tento uzel zprávy odesílá. Dalším je uzel typu *Subscriber*, ten zprávy přijímá. A posledním typem je kombinace těchto dvou uzlů, tento uzel je schopen zprávy přijímat a zároveň je i odesílat. Tyto uzly mezi sebou komunikují přes tzv. *témata*. Pro psaní jednotlivých uzlů je možno si vybrat ze dvou programovacích jazyků, a to jazyk *C++*, ty mají příponu *cpp* nebo jazyk *Python*, a ty mají příponu *py*. Uzly musí být uloženy v balíčcích v určitých složkách. Uzly, které jsou psané v jazyce *C++* musí být uloženy ve složce *src* a uzly psané v jazyce Python musí být uloženy ve složce *scripts*. Pokud balíček neobsahuje tyto složky je nutné si je vytvořit. Podrobnější popis programování uzlů naleznete zde [16].

Při vytváření uzlů v programovacím jazyce *C++* je nutné editovat soubor *CMakeLists.txt* v oblasti uvedené níže, který nalezneme v kořenovém adresáři balíčku.

```
############
## Build ##
############
  add_executable(<jméno_uzlu> src/<jméno_uzlu>.cpp)
  target_link_libraries(<název_uzlu> ${catkin_LIBRARIES})
  add_dependencies(<název_uzlu>
<název_balíčku>_generate_messages_cpp)
```

Dále po každém vytvoření nebo úpravě uzlů psané v *C++* je nutné se přemístit do adresáře *catkin_ws* a provést sestavení pracovního prostředí příkazem *catkin_make*. Pro uzly psané v jazyce Python tato editace souboru *CMakeLists.txt* neplatí a stačí provést sestavení pracovního prostředí pouze po vytvoření uzlu, tato operace se nemusí provádět po každé úpravě uzlu.

5.2.8 Téma

Téma slouží jako komunikační prvek, přes který spolu uzly mohou komunikovat, a také komunikují. V každém uzlu se definuje, z jakého téma má uzel zprávy přijímat nebo naopak na jaké je má odesílat. Na téma je možné zprávy odesílat z několika uzlů a zároveň z něj i několik uzlů může zprávy odebírat [15].

Zdrojový kód v jazyce *Python* pro vytvoření téma pro uzel typu *Publisher* vypadá takto:

```
rospy.Publisher(<jméno_téma>,<typ_zpravy>, queue_size=10)
```

Tato funkce má tři argumenty. První je jméno tématu, druhý je typ zprávy a poslední je hodnota, pomocí které si definujeme kolik zpráv si má dané téma uchovávat zpráv, pokud z něj žádný uzel zprávy neodebírá.

Zdrojový kód v jazyce *Python* pro vytvoření téma pro uzel typu *Subscriber* vypadá takto:

```
rospy.Subscriber(<jméno_téma>,<typ_zprávy>, callback)
```

Tato funkce má opět tři argumenty, kde první je jméno tématu, druhý je typ zprávy a třetí je funkce, do které se ukládají příchozí zprávy z daného tématu.

5.2.9 Zprávy

Jak už bylo zmíněno, uzly spolu komunikují přes témata pomocí tzv. *zpráv*. Tyto zprávy mohou být různého datového typu jako je: *int8, int16, int32, int64, uint8, uint16, uint32, uint64, float32, float64, string, time*. Navíc systém ROS používá vlastní struktury zpráv, které lze použít například *geometry_msgs* [17], jež obsahuje mnoho možností pro volbu struktury zprávy. Jednou z možností, která byla použita v této bakalářské práci, je struktura zprav typu *Twist*. Ta vypadá takto:

```
Linear:
  float64 x
  float64 y
  float64 z
Angular:
  float64 x
  float64 y
  float64 z
```

Pomocí této struktury lze popsat pohyb v osách x, y, z a rotace kolem os x, y, z.

5.2.10 Nástroje

Pro ovládání a práci v prostředí ROS tento systém poskytuje více než 45 příkazových nástrojů, ty se zadávají do příkazového řádku v terminálu. Každý nastroj je tvořen svým jménem a za ním je většinou jeden nebo více argumentů, pomocí kterých definujeme, jak se má daný nástroj chovat [18].

Roscore je jedním z nejzákladnějších nástrojů, je to příkaz pro spuštění jádra systému, který se zadává bez argumentů. Tento příkaz musí být vždy spuštěn jako první, před spouštěním jiných uzlů. Jedinou výjimkou je, pokud spouštíme více uzlů najednou pomocí příkazu *roslaunch* [19].

Příkaz vypadá takto:

```
roscore
```

Roslaunch je dalším nástrojem, pomocí kterého je možné spustit jádro systému ROS. Dále pak spouští více uzlů najednou, tento příkaz jako první spustí jádro systému, a poté spouští uzly, které jsou nakonfigurovány ve spouštěcím souboru.

A vypadá takto:

```
roslaunch  <jméno_balíčku>   <spouštěcí_soubor.launch>
```

Do argumentů tohoto příkazu se zadává jméno balíčku a název spouštěcího souboru, který musí být umístěn v zadaném balíčku. Před spuštěním tohoto příkazu je nutné si vytvořit spouštěcí soubor s příponou *.launch*. Tyto soubory jsou psány ve značkovacím jazyce XML, pomocí něhož popisují uzly, které by měly být spuštěny, parametry, které by měly být nastaveny a další atributy. Specifikace tohoto XML formátu pro ROS naleznete zde: [20].

Rosrun pomocí tohoto nástroje spouštíme jednotlivé uzly. Podmínkou před zadáním tohoto příkazu je nutnost spuštěného jádra systému. Příkaz vypadá takto:

```
rosrun  <jméno_balíčku>  <jméno_uzlu>
```

U tohoto příkazu se zadává jméno uzlu, který chceme spustit a jméno balíčku ve kterém se uzel nachází [21].

Rosnode je nástroj pro práci s uzly a jeho možnosti jsou uvedeny níže [22].

```
rosnode info    Vypíše informace o spuštěných uzlech.
rosnode kill    Ukončení aktivního uzlu.
rosnode list    Zobrazí spuštěné uzly.
rosnode machine Výpiše spuštěné uzly na stanici.
rosnode ping    Testpřipojení uzlu.
rosnode cleanup Vipíše informace o nedostupných uzlech.
```

Za tento nástroj se udává ještě jméno uzlu, pro který chceme operaci provést. Kromě příkazu *list*, se zde neudává žádný další argument, u příkazu *machine* se zadává název stanice, z které chceme vypsat aktivní uzle a dále příkaz *cleanup* je také bez dalšího parametru.

Rostopic je nástroj pro práci s tématy a jeho možnosti jsou uvedeny níže [23].

```
rostopic bw     Zobrazí propustnost tématu.
rostopic echo   Odposlech tématu.
rostopic find   Nalezne téma.
rostopic hz     Zobrazí frekvenci publikování dat do tématu.
rostopic info   Vypíše informace o aktivním tématu.
rostopic list   Vypíši seznam aktivních témat.
rostopic pub    Publikování dat do téma.
rostopic type   Výpis datového typu tématu.
```

U tohoto nástroje se také za příkaz zadává jméno uzlu, kromě příkazu *find*, kde se zadává *jméno_balíčku* nebo *jméno_zprávy*, kterou chceme nalézt. Příkaz *list* je zadáván bez dalšího argumentu.

Rqt je software frameworku ROS, který implementuje různé nástroje grafického uživatelského rozhraní v podobě zásuvných modulů. Rqt obsahuje přes 20 zásuvných modulů pro správu uzlů, témat, diagnostiku zpráv, zobrazení zpráv a mnoho dalších. Více informací o tomto software naleznete zde [24]. Spuštění tohoto grafického rozhraní se provádí pomocí příkazu do příkazového řádku terminálu příkazem:

```
rqt
```

Další možností je spouštění jednotlivých modulů samostatně. Mezi nejužitnější patří například modul *rqt_graph*, který zobrazí komunikační graf aktivních uzlů a používaných témat. Spouští se příkazem:

```
rosrun rqt_graph rqt_graph
```

Rviz patří mezi jeden z modulů, které obsahuje software *rqt*. Slouží především k 2D, či 3D vizualizaci. Je zde možné zobrazení 3D modelů robotů, simulace pohybu robota, zobrazení obrazu z kamer, nebo vizualizaci ať už 2D, či 3D laserových scannerů. Jeho softwarové specifikace a možnosti použití naleznete zde [25]. Rviz lze spustit přímo z grafického rozhraní rqt nebo příkazem uvedeným níže.

```
rosrun rviz rviz
```

Roswtf je poslední nástroj, o kterém v této kapitole bude zmínka. Tento nástroj slouží k odstraňování problémů a chyb. Používá se za běhu systému ROS, kde diagnostikuje právě běh systému a pokud zde vzniká chyba, chybu nalezne a vypíše nám do terminálu, jaká chyba vzniká a další potřebné informace. Nástroj se spouští pomocí příkazu:

```
rosrun roswtf
```

5.2.11 Distribuovaný design

Framework ROS je navržen tak, že jednotlivé uzly mohou být spuštěny na předem zvolené platformě, což je velká výhoda toho systému, protože pokud robot nemá dostatečný výpočetní výkon pro složité výpočty, například zpracování obrazu nebo práci s daty z laserového scanneru, stačí připojit do systému ROS výkonnější výpočetní jednotku a provádět tyto operace na ní.

Pro připojení do systému ROS musí být splněno několik požadavků. Prvním z nich je nainstalování frameworku ROS na jednotlivých platformách. Dále musí být všechny platformy připojeny do jedné lokální sítě dle protokolu TCP/IP.

Protože systém ROS pracuje na principu *Master/Slave*, musíme si předem zvolit, která platforma bude jako *Master* a ostatní se k ní připojují jako *Slave*. To se provádí přepsáním systémových proměnných frameworku ROS. Aby se nastavování těchto systémových proměnných nemuselo provádět po každém spuštění terminálů, přidají se jednotlivé příkazy pro nastavení do skrytého souboru *.bashrc*. To nám zaručí nastavení proměnných při každém spuštění nového terminálu [26].

Editace souboru *.bashrc* pro Master:

```
export ROS_MASTER_URI=http://<IPv4_adresa_Master >:11311
export ROS_HOSTNAME=<IPv4_adresa_Master >
```

Editace souboru *.bashrc* pro Slave:

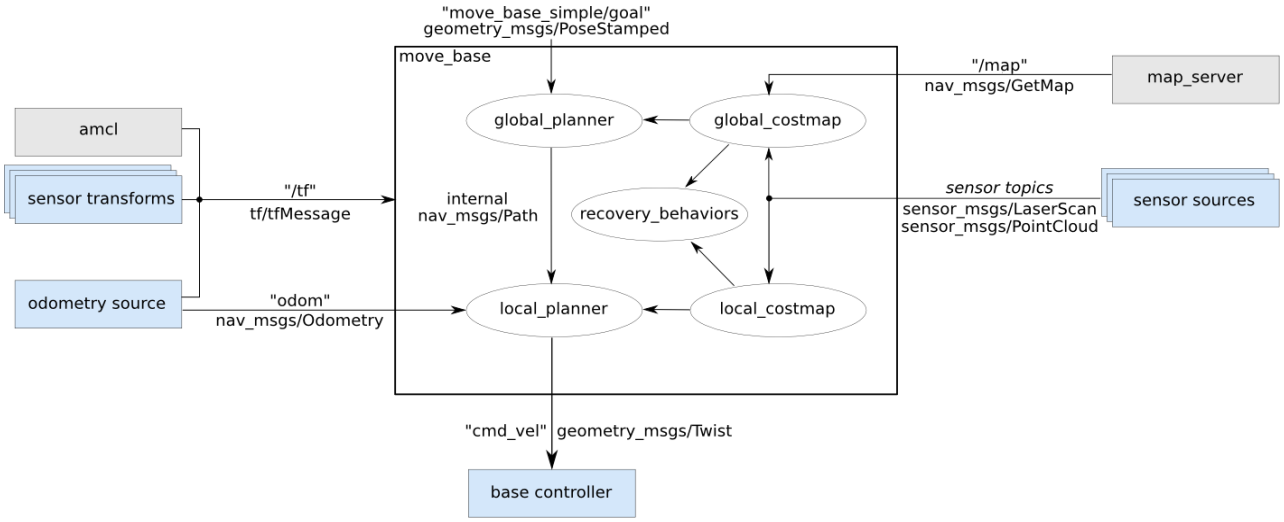
```
export ROS_MASTER_URI=http://<IPv4_adresa_Master >:11311
export ROS_IP=<IPv4_adresa_Slave>
```

Spuštění takto vzniklého jednotného systému ROS se provádí na platformě, která je zvolena jako *Master*. Spustí se nejprve jádro systému, poté co je spuštěno, se mohou spouštět další jednotlivé uzly na platformách, které si zvolíme.

Pokud chceme takto nastavené proměnné vrátit do původního nastavení, stačí tyto dva řádky smazat ze souboru *.bashrc*.

6 Návrh navigačního systému

Pro tento návrh byl použit model navigačního systému, který poskytuje framework ROS. Schéma navigačního systému můžete vidět na obrázku (viz Obr. 10).



Obr. 10 – Schéma navigačního systému [27]

Tento model obsahuje tři druhy uzlů. Uzly bílé jsou nutnou součástí, ty jsou již vytvořeny a implementovány. Šedé uzly jsou volitelné a jsou již realizovány a připraveny k použití. Nakonec zbývají uzly modré, ty musí být vytvořeny a implementovány pro zvolenou platformu robota. Jednotlivé požadavky pro tento model navigačního systému budou popsány v následujících podkapitolách [27].

6.1 Move_base

Balíček *move_base* je jádro celého navigačního systému. Obstarává plánování a řízení robota, pomocí informací ze senzorů a v předem známé mapě. *Move_base* spojuje globální a lokální plánovač trasy pro splnění navigačních úkolů. Kde se globální plánovač stará o omezení, která vzniknou v mapě a robot by je nedokázal projet. A lokální plánovač se stará o prostor kolem robota, hlídá, aby nenarazil do nějaké nečekané překážky. *Move_base* dále obstarává dvě mapy, a to lokální mapu a globální mapu robota.

V tomto balíčku framework ROS implementuje kombinaci různých plánovacích algoritmů, které naleznete zde [28]. Jediné co je nutné povést pro tuto navigační část, je vytvoření souboru, kde budou nadefinovány proměnné, jež popisují robota a jeho omezení.

Ovládání navigačního systému je v prostředí Rviz, které je nutno po spuštění navigačního systému nastavit. Ovládání se provádí pomocí funkce Goal, kde poté, co se robot lokalizuje v mapě pomocí této funkce, zadáme navigační úkol a balíček *move_base* tento navigační úkol provede.

6.2 Transformační konfigurace (sensor transforms)

Protože se robot skládá z několika souřadnicových systémů, je nutné si vytvořit uzel, který bude popisovat vztahy mezi jednotlivými souřadnicovými systémy. Bude také provádět transformaci polohy robota vůči globálnímu souřadnicovému systému.

6.3 Informace ze senzorů (sensor sources)

Navigační systém využívá informace ze senzorů, aby zabránil srážkám robota s překážkami. Tento uzel musí poskytovat zprávy z laserového scanneru nebo z jiných senzorů, které popisují okolí robota.

6.4 Odometrická data (odometry source)

Dále je zapotřebí do balíčku *move_base* poskytnout odometrická data, ta popisují pohyb robota. Tato data musí být posílána pomocí transformační funkce, která transformuje souřadnicový systém robota vůči mapě, v níž se pohybuje.

6.5 Ovládání robota (base controller)

Navigační systém odesílá pro řízení robota zprávy typu *Twist* na téma *cmd_vel*. To znamená, že musí existovat uzel, který posílá robotu příkazy typu rychlost v ose x *(cmd_vel.linear.x)*, rychlost v ose y *(cmd_vel.linear.y)* a natočení robota kolem osy z *(cmd_vel.angular.z)* a dále se předpokládá, že robot dokáže tyto příkazy přijímat a na základě nich se pohybovat.

6.6 Mapa (map_server)

V neposlední řadě je zapotřebí poskytnou mapu, ve které se bude robot pohybovat. Tuto mapu je nutno vytvořit a dále poskytnout, pomocí knihovny *map_server*, navigačnímu systému. Tato knihovna je už vytvořena a poskytována systémem ROS.

6.7 Amcl

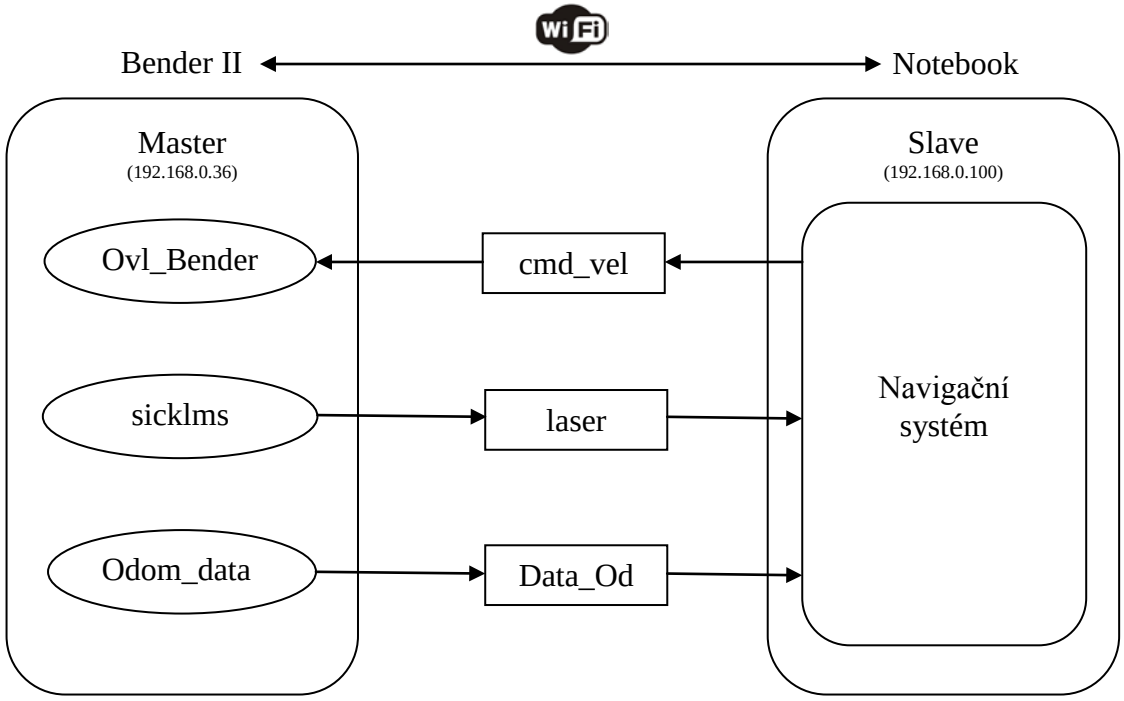
Tato knihovna je již také předem vytvořena systémem ROS a je připravena k implementaci. Amcl je pravděpodobnostní lokalizační systém pro robota, který se pohybuje ve 2D známé mapě. Jsou zde použity tyto lokalizační algoritmy: *sample_motion_model_odometry, beam_range_finder_model, likelihood_field_range_finder_model, Augmented_MCL, and KLD_Sampling_MCL.* Tyto algoritmy jsou důkladně popsány v této knize [2].

7 Realizace navigačního systému

V této kapitole bude popsán postup realizace navigačního systému. Pro realizaci bylo nutné provést instalaci a nastavení frameworku ROS jak na jednodeskový počítač BeagleBoard, tak i na stolní notebook MSI. Postup instalace a nastavení frameworku ROS je popsán v kapitole *5.2.3 Instalace* a *5.2.4 Nastavení pracovního prostředí*. Dále bylo nutné oba dva počítače připojit do jedné lokální sítě, což bylo řešeno vytvořením Wifi sítě pomocí routeru, do které se připojily oba dva počítače.

Po splnění těchto požadavků byl vytvořen jednotný systém ROS, který obsahoval dvě připojené platformy a to jednodeskový počítač BeagleBoard, který byl nastaven jako *Master* a jeho funkcí bylo odesílání dat z laseru a dat odometrických, dále přijímání příkazů, podle kterých se poté řídil. K jednodeskovému počítači se připojil stolní notebook MSI, který byl nastaven jako *Slave*, na němž byl spuštěn navigační systém, jež přijímal data z robota a na základě těchto dat zpětně robota řídil. Postup pro vznik jednotného systému ROS je popsán v kapitole *5.2.11 Distribuovaný design*. Takto navržený systém byl zvolen proto, aby umožňoval připojení jiného počítače, který bude mít správně nainstalovaný navigační systém a také bude moct ovládat robota.

Reálné schéma celého navigačního systému můžete vidět níže:



Obr. 11 – Reálné schéma navigačního systému

V následujících podkapitolách budou popsány jednotlivé stanice a jejich balíčky, uzly a části kódu programu některých uzlů. Dále bude popsán postup, pomocí kterého si lze vytvořit vlastní mapu prostředí, ve kterém se robot pohybuje.

7.1 Master stanice

Master stanicí byl zvolen robot Bender II, fungoval jakožto nadřazená stanice, na které běželo jádro systému. Na této stanici nebyly prováděny žádné složité výpočty nebo operace, sloužila pouze k poskytování dat potřebných pro řízení robota. Obsahovala dva druhy balíčků. První balíček se nazýval *sicktoolbox_wrapper*, který obsluhoval laserový scanner. Druhým balíčkem byl vytvořený balíček *bender_nav*, tento balíček obsahoval dva uzly, jež se staraly o posílání odometrických dat do systému ROS a o přijímání příkazu ze systému ROS.

7.1.1 Balíček 1

Tento balíček se jmenoval *sicktoolbox_wrapper* a byl již předem vytvořen a připraven k používání komunitou ROS. V tomto balíčku je vytvořený uzel *sicklms*, který slouží k práci s laserovým scannerem *SICK LMS2xx*, přijímá data z tohoto scanneru a posílá je na téma *laser*.

Stažení balíčku se provede příkazem:

```
git clone https://github.com/ros-drivers/sicktoolbox_wrapper
```

Tento balíček je nutné umístit do složky *catkin_ws/src*.

Dále se provede instalace balíčku, nastavení závislostí a sestavení pracovní plochy pomocí příkazu:

```
rosdep install sicktoolbox_wrapper rviz
rosmake sicktoolbox_wrapper rviz
```

Pro čtení dat z laserového scanneru je nutné nastavení práv pro USB rozhraní, přes které je laser připojen. To se provádí pomocí příkazu:

```
sudo chmod a+rw /dev/ttyUSB0
```

Tuto operaci je nutné provést před každým spuštěním uzlu *sicklms*.

Dále je nutné spustit jádro systému pomocí příkazu *roscore*.

A posledním příkazem spustíme uzel *sicklms*, kterému nastavíme do parametrů port, na kterém je laser připojen a přenosovou rychlost pomocí příkazu:

```
rosrun sicktoolbox_wrapper sicklms _port:=/dev/ttyUSB0
_baud:=38400
```

Uzel defaultně s laserem pracuje v režimu, kde laser je schopen měřit do dálky až 81 metrů v rozmezí 0 ° až 180 °a rozlišení 0,5 °.

7.1.2 Balíček 2

Tento balíček byl vytvořen s názvem *bender_nav* a obsahoval dva uzly, první uzel posílal odometrická data do systému ROS, kdy tato data byla přijímána od robota. A druhý uzel přijímal data ze systému ROS a následně je posílal robotovi, podle kterých se poté robot řídil. Oba dva uzly byly napsány v jazyce *Python*.

Uzel 1

Tento uzel byl typu *Publisher* a odesílal data na téma *Data_Od*. Tato data byla typu *Twist* a vypadala takto:

```
  linear:
    float64 x   #aktuální rychlost v ose x
    float64 y   #aktuální rychlost v ose y
  angular:
    float64 z   #aktuální natočení robota kolem osy z
```

Nejdůležitější část programového kódu a vysvětlení funkce tohoto uzlu je uvedeno níže:

```
def talker():   #Funkce talker
     #Vytvoření tématu a nastavení typu zpráv
    pub = rospy.Publisher('Data_Od', Twist,queue_size=5)
    rospy.init_node('Odom_data')      #Inicializace uzlu
    rate = rospy.Rate(100) #Nastavení frekvence
    twist = Twist()  #Inicializace funkce Twist
    while not rospy.is_shutdown():
     moduly.naplnpaket()#volaní funkce, která plni paket ze
     sériové komunikace

     if moduly.paket[3]==0x05:  #Kontrola příchozích dat a po
          splnění podmínky předání hodnoty do proměné Twist
          actual_x=moduly.rychlost_x()
          twist.linear.x=actual_x


     pub.publish(twist)    #Odeslání dat do systému Ros
     rate.sleep()    #Frekvence odesílání dat
```

Tento uzel sloužil k tomu, že přijímal data ze sériové komunikace. Data dále roztřídil a naplnil s nimi proměnné *Twist*, po naplnění je dále odeslal do systému ROS na předem nadefinované téma.

Uzel 2

Tento uzel byl typu *Subscriber* a přijímal data z téma *cmd_vel*. Tato data byla opět ve formátu *Twist*. Bylo tedy nutné je opět roztřídit a postupně je odesílat robotu, aby se podle nich mohl následně řídit. Nejdůležitější část programového kódu můžete vidět níže:

```
from geometry_msgs.msg import Twist #Import knihovny Twist
import rospy #Import knihovny pro psaní uzlu v Python
import moduly #Import knihovny pro komunikaci přes USB
def callback(data): #Definice funkce, která kontrolovala, jaké
data přicházejí do téma cmd_vel a následně je posílala robotovi
    if data.linear.x != 0:
         moduly.send_x(data.linear.x)
    elif data.linear.x == 0:
         moduly.send_x(data.linear.x)
    if data.linear.y != 0:
         moduly.send_y(data.linear.y)
    elif data.linear.y == 0:
         moduly.send_y(data.linear.y)
    if data.angular.z != 0:
         moduly.send_z(data.angular.z)
    elif data.linear.z == 0:
         moduly.send_z(data.linear.z)
def listener():#Definice funkce listener
    rospy.init_node('Ovld_bender')#Inicializace uzlu

    pub=rospy.Subscriber("cmd_vel", Twist, callback) #Vytvoření
uzlu typu subscriber
rospy.spin()
if __name__ == '__main__':
    listener()#Volí funkce listener
```

7.2 Slave stanice

Slave stanicí byl zvolen stolní notebook, fungoval jako hlavní výpočetní platforma, na které se prováděly skoro všechny výpočty, které byly potřebné k navigaci robota. Dále sloužil pro vizualizaci a ovládaní celého navigačního systému. Tato stanice obsahovala celkem 5 balíčků, které si popíšeme v následujících podkapitolách.

7.2.1 Balíček 1

Tento balíček se jmenoval *teleop_twist_keyboard* a sloužil k manuálnímu ovládání robota. Obsahoval uzel *teleop_twist_keyboard*, který byl typu Publisher a fungoval na principu čtení stisků předem nadefinovaných znaků na klávesnici a podle nich odesílal na téma *cmd_vel* data ve formátu *Twist*. Podle těchto dat se dále robot řídil. Tento balíček je dostupný zde [29]. K používání stačí jeho stažení a následná instalace do systému ROS.

7.2.2 Balíček 2

Dalším balíčkem byl balíček *scan_tools*, tento balíček opět poskytuje framework ROS a obsahuje rozsáhlou knihovnu pro práci s laserovým scannerem. Celkem je v této knihovně k dispozici 8 různých nástrojů pro práci s laserovým scannerem. V této práci byl používán nástroj pro vypočítávání polohy robota na základě porovnávání starých a aktuálních snímků ze scanneru. Podle míry odlišnosti těchto snímků a jednotlivých bodů mezi těmito snímky se odhadovala poloha robota v prostoru. Tento balíček je dostupný zde [30]. Používaný nástroj se jmenoval *laser_scan_matcher* a spouštěl se pomocí příkazu:

```
rosrun laser_scan_matcher laser_scan_matcher_nodelete
```

7.2.3 Balíček 3

Tento balíček byl nejdůležitějším balíčkem celého navigačního systému a jmenoval se *navigation*, opět je poskytován frameworkem ROS, kdy bylo třeba tento balíček stáhnout a nainstalovat. Z tohoto balíčku se spouštěly jednotlivé uzly s nadefinovanými parametry, jež byly potřebné pro navigační systém. Tento balíček je dostupný zde [31]. A obsahuje již zmíněný uzel pro plánování *move_base*, uzel pro lokalizaci robota *amcl* a uzel, který pracuje s mapou *mape_sever*.

7.2.4 Balíček 4

Dalším balíček byl *senzor_bender*, tento balíček patří mezi balíčky, které byly vytvořeny v rámci této práce. Tento balíček obsahoval dva uzly. První uzel pracoval s funkcí *tf*, pomocí které se definovaly vztahy mezi jednotlivými souřadnicovými systémy robota. A druhý uzel přijímal odometrická data, pomocí kterých vypočítával přírůstky vzdáleností v jednotlivých osách a natočení robota. Pomocí funkce *tf* prováděl transformaci polohy robota v prostoru. Oba dva uzly byly psané v jazyce *C++*.

Uzel 1

Tento uzel se jmenoval *tf_broadcaster* a používal funkci *tf* , pomocí které se definují vztahy mezi jednotlivými souřadnicovými systémy robota. Pomocí této funkce se staví tzv. *transformační strom*, který se skládá z tzv. *rodičů* a *dětí*, každý transformační strom může mít jen jednoho *rodiče*, který může mít mnoho *dětí*, a ty můžou mít také mnoho *dětí*. Podrobnější popis *tf* funkce naleznete zde [32]. Pro tuto funkci je zapotřebí nadefinovat jména jednotlivých souřadnicových systému. V této práci jsme měly tři souřadnicové systémy a to základnu robota, která se jmenovala *base_link* k této základně byl připevněn laserový scanner, jež byl nadefinován jako *laser*. Dále prostor, ve kterém se robot pohyboval, souřadnicový systém *map*. V tomto uzlu bylo zapotřebí si nadefinovat vztah mezi *base_link* a *laser*. Transformaci a vztahy mezi *base_link* a *map* prováděl uzel 2, který transformoval polohu robota v mapě pomocí odometrických dat.

Popis nejdůležitější části programového kódu je uveden níže:

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h> //Načtení tf funkce

int main(int argc, char** argv){
  ros::init(argc, argv, "robot_tf_publisher"); //Vytvoření uzlu,
pomocí kterého odesíláme data o transformaci a vztazích mezi
jednotlivými souřadnicovými systémy na téma tf
  ros::NodeHandle n;
  ros::Rate r(100); //Frekvence odesílání dat
  tf::TransformBroadcaster broadcaster;
  while(n.ok()){ //Cyklus ve kterém se prováděla
                 transformace
    broadcaster.sendTransform( //Transformace natočení a posunu
      tf::StampedTransform(
        tf::Transform(tf::Quaternion(0, 0, 0, 1),
tf::Vector3(0.17, 0.0, 0.2)),
        ros::Time::now(),"base_link", "laser")); //Definice
base_link jako rodiče a laser jako dítě
    r.sleep();
  }
}
```

Uzel 2

Tento uzel byl typu *Subscriber/Publisher* a jmenoval se *odometry_publisher*. Přijímal data z téma *Data_Od*, které mu posílal robot, tato data byla o aktuální rychlosti v osách x a y a aktuální natočení kolem osy z. A na základě těchto dat vypočítává změnu polohy v čase.

Část programového kódu pro výpočet je uveden níže:

```
//Výpočet přírůstku změny polohy v jednotlivých osách
  double dt = (current_time - last_time).toSec();
  double delta_x = (vx * cos(th) - vy * sin(th)) * dt;
  double delta_y = (vx * sin(th) + vy * cos(th)) * dt;
  double delta_th = vth * dt;
```

A dále pomocí těchto dat prováděl transformaci souřadnicového systému *base_link* vůči prostředí *map*. Podrobnější popis jak provádět transformaci souřadnicových systému pomocí systému ROS naleznete zde [33].

7.2.5 Balíček 5

Posledním balíčkem, který obsahovala tato stanice, byl balíček *slam_gmmaping*. Tento balíček obsahoval soubor knihoven pro tvorbu mapy, pomocí laserového scanneru. Tento nástroj pro tvorbu mapy je poskytován frameworkem ROS a je k dispozici zde [34].

7.3 Mapa

Před spuštěním navigačního systému bylo zapotřebí si vytvořit mapu, která byla vytvořena pomocí nástroje *slam_gmmaping*, jež byl nainstalován na *Slave* stanici. Postup vytvoření mapy bude popsán níže.

Na *Master* stanici se jako první musel spustit jádro systému pomocí příkazu:

```
roscore
```

Dále bylo zapotřebí poskytnout systému ROS data z laserového scanneru a to pomocí nástroje *sicktoolbox_wrapper*, spuštění tohoto nástroje je popsáno (viz kapitola 7. 1. 1). A poslední uzel, který byl spuštěn na této stanici, byl uzel pro ovládání robota (viz kapitola 7. 1. 2), jaž se spouštěl pomocí příkazu:

```
rosrun bender_nav Ovld_Bender
```

Na *Slave* stanici bylo zapotřebí spustit uzel *teleop_twist_keyboard*, pomocí kterého lze manuálně ovládat robota. Příkaz pro spuštění vypadal takto:

```
rosrun teleop_twist_keyboard teleop_twist_keyboard
```

Dále bylo zapotřebí spustit uzel, který pomocí *tf* funkce definoval jednotlivé souřadnicové systémy v našem případě mezi základnou robota a laserovým scannerem. Uzel se spouštěl pomocí příkazu:

```
rosrun senzor_bender tf_broadcaster
```

Poté se spustil nástroj *laser_scan_matcher*, popis a spuštění tohoto nástroje je popsán (viz kapitola 7. 2. 2.)

Po spuštění těchto uzlů se mohl spustit nástroj pro tvorbu mapy *slam_gmmaping*, který se spouštěl pomocí příkazu, kde bylo nutné zadat do parametru *scan* název základny laserového scanneru:

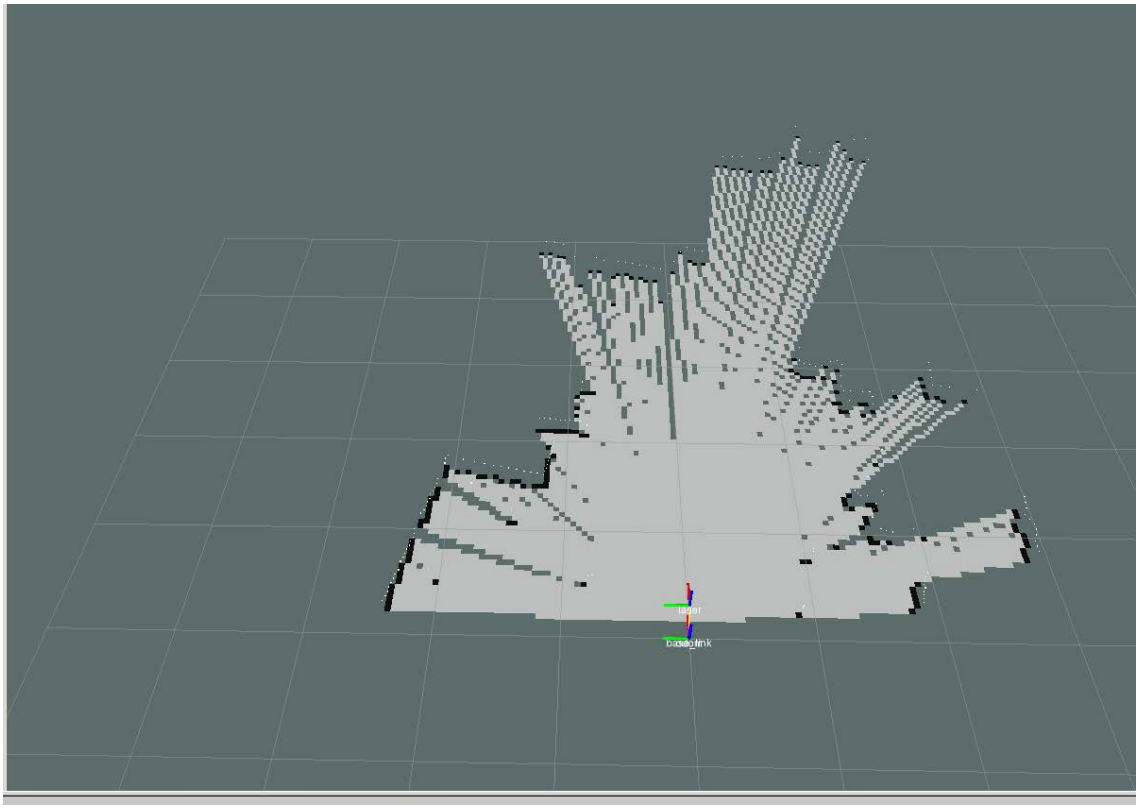
```
rosrun gmapping slam_gmapping scan:=laser
```

Když byly splněny všechny tyto požadavky, systém ROS prováděl výpočet změny polohy robota pomocí *laser_scan_matcher* a pomocí dat z laserového scanneru nástroj *slam_gmmaping* vytvářel mapu prostředí, ve kterém se robot pohyboval.

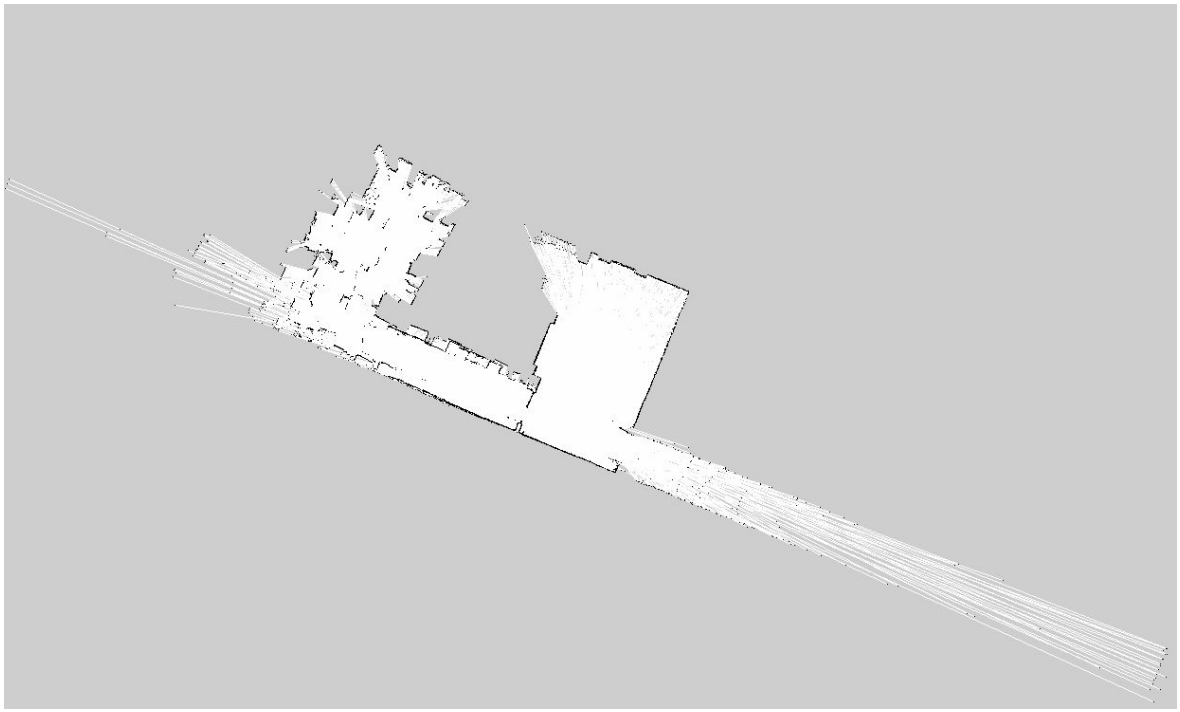
Nakonec je zapotřebí provést uložení vytvořené mapy. Provádí se pomocí příkazu uvedenému níže. Mapa se uloží do adresáře, ve kterém se momentálně nacházíme.

```
rosrun map_server map_saver
```

Vizualizace tvorby mapy a sledování pohybu robota se prováděla pomocí nástroje Rviz (viz kapitola 5. 2. 10), kterou můžete vidět na (viz Obr. 11a), kde je snímek z vizualizace pomocí nástroje Rviz. Výsledná vytvořená mapa laboratoře je na (viz Obr. 11b). Celý průběh vytvoření mapy je možno si prohlédnou na videu, které je součástí této práce (viz Příloha č. 1).



Obr. 12a – Snímek z vizualizace pomocí nástroje Rviz



Obr. 12b – Mapa laboratoře

8 Spuštění navigačního systému

Pro spuštění navigačního systému byl vytvořen samostatný balíček *launch_nav*, který se nacházel na *Slave* stanici a měl v závislostech balíčky *senzor_bender* a *move_base*. Tento balíček obsahoval soubory *base_local_planner_params, costmap_common_params, global_costmap_params* a *local_costmaps_params*, pomocí těchto souborů se definují omezení a parametry robota a dále parametry použité mapy. Podrobný popis jak správně nastavit tyto parametry jsou k dispozici zde [26]. Dále tento balíček obsahoval dva spouštěcí soubory.

První spouštěcí soubor *robot_configuration.launch* zajišťoval spuštění všech potřebných uzlů, které zajišťovali odometrická data a definici *tf* funkce. Spouštěcí soubor vypadal takto:

```
<launch>
     //Definice stanice, na které jsou spouštěny uzly
     <machine name="MSI" address="192.168.1.100" user="lubos">
     </machine>
     //Spuštění uzlu tf_broadcaster
      <node machine="MSI" name="tf_broadcaster"
pkg="senzor_bender" type="tf_broadcaster" />
     //Spuštění uzlu odometry_publisher
      <node machine="MSI" name="odometry_publisher"
pkg="senzor_bender" type="odometry_publisher" />
</launch>
```

Druhým spouštěcím souborem byl *move_base.launch* a ten zajišťoval spuštění uzlů *map_server*, pomocí kterého se nahrála mapa, dále spouštěl uzel *amcl* a nakonec spouštěl uzel *move_base*, kterému bylo zapotřebí nadefinovat cestu k jednotlivým souborům, ty popisovaly parametry pro navigaci robota Bender II. Tento spouštěcí soubor je součástí přílohy (viz Příloha č. 1).

Dále byl na *Master* stanici vytvořen spouštěcí soubor *senzor_launch.launch*, který zajišťoval spuštění tří uzlů, uzel, který poskytoval data z laserového scanneru (viz kapitola 7. 1. 1), uzel, který zajišťoval posílání příkazů, podle kterých se robot řídil (viz kapitola 7. 1. 2) a uzel, který posílal odometrická data do systému ROS (viz kapitola 7. 1. 2). Tento soubor je také součástí přílohy (viz Příloha č. 1)

Spuštění celého systému se provádělo pomocí těchto tří spouštěcích souborů. Jako první musel být spuštěn spouštěcí soubor na *Master* stanici pomocí příkazu:

```
roslaunch bender_nav senzor_launch.launch
```

Dále se spustil spouštěcí soubor, který poskytl odometrická data a definici *tf* funkce příkazem:

```
roslaunch launch_nav robot_configuration.launch
```

A poslední příkaz spouštěl navigační systém. Příkaz je uveden níže:

```
roslaunch launch_nav move_base.launch
```

Ovládání a vizualizace navigačního systému byla opět prováděna v prostředí Rviz. Instrukce pro nastavení a ovládání navigačního systému v prostředí Rviz naleznete zde [35].

9 Závěr

Hlavním cílem této bakalářské práce bylo vytvoření návrhu a následná realizace navigačního systému pro mobilního robota Bender II pomocí frameworku ROS. Textová část je koncipována jako návod a je dále rozdělena do několika částí.

V první části této práce jsme se zabývali teoretickým úvodem, který pojednává o tom, co vlastně navigační systém znamená, jaké úkoly by měl zvládat a co je nutné vyřešit pro jeho realizaci. Také je zde zmíněn nejtěžší navigační úkol a tím je lokalizace robota v prostředí, ve kterém se nachází. Tato lokalizace se dělí do několika skupin, přičemž byly zmíněny dvě lokalizační metody. V další části je popsán použitý hardware s příslušnými technickými parametry. V části třetí je popsán postup, jak správně systém ROS nainstalovat, dále se zabývá nastavením pracovního prostředí tohoto systému a následným propojením jednotlivých platforem do jednoho společného systému. V této kapitole byly také zmíněny nástroje, jež poskytuje ROS, a které byly používány při vývoji tohoto navigačního systému (viz kapitola 5. 2). Dále se zabýváme modelem návrhu, který poskytuje systém ROS a byl použit v této práci. Postupně jsou popsány jednotlivé části toho modelu. Samotná realizace, jako poslední část práce, celého navigačního systému byla řešena tak, aby co nejméně zatěžovala výpočetní výkon jednodeskového počítače, který byl umístěn na robotu. Řešením bylo navržením robota Bender II jako *Master* stanice, k níž byl připojen stolní notebook jako *Slave* stanice (viz kapitola 7). *Master* stanice sloužila pouze k poskytování informací o svém pohybu a k přijímání jednotlivých příkazů, které byly posílány ze systému ROS, zatímco veškeré výpočetní a vizualizační procesy byly vykonávány na stolním notebooku, jenž disponoval dostatečným výkonem. Realizovaný navigační systém byl tedy navržen tak, aby bylo možné ovládat robota Bender II z více počítačů.

Za největší výhodu systému ROS tedy považujeme distribuovaný design, který umožňuje provádět jednotlivé výpočetní operace na různých platformách, jež jsme si předem zvolili a dále zajišťuje stabilní komunikaci mezi těmito platformami pomocí protokolu TCP/IP. Bezesporu další velkou výhodou je rozsáhlá sbírka knihoven, kterou obsahuje, například pro ovládání různých motorů, senzorů nebo propojení mobilních telefonů, obsahujících systém Android a možnost používání senzorů zabudovaných v telefonu. Těmito senzory můžou být snímače intenzity světla, gyroskopy, akcelerometry a jiné.

Naopak za nevýhodu považujeme náročnost pro začínající uživatele. Tento framework by se mohl zdát z hlediska ovládání nepřehledný, což je způsobeno především spouštěním nových terminálů pro každý jednotlivý uzel při testování jejich správnosti a funkčnosti.

Závěrem bych chtěl podotknout, že po seznámení se s u nás méně známým frameworkem ROS, jsem byl mile překvapen jeho již avizovanou zahraniční popularitou pro tvorbu robotických aplikací, a pro velice silné postavení ve světě robotiky. Chtěl bych dále pokračovat v rozvoji toho systému a aplikovat ho i na jiné roboty. Například vytvoření autonomního konvoje v čele s robotem Bender II, nebo přidání na robota Bender II senzor Kinect, pomocí kterého by bylo možné vytvářet 3D mapu.

Seznam použité literatury

[1] NOVÁK, Petr. *Mobilní roboty: pohony, senzory, řízení*. 1. vyd. Praha: BEN - technická literatura, 2005, 243 s. ISBN 80-7300-141-1.

[2] THRUN, Sebastian, Wolfram BURGARD a Dieter FOX. *Probalistic Robotics* [online]. c1999–2000 [cit. 11. 3. 2015]. Dostupné z: http://people.ufpr.br/~danielsantos/ProbabilisticRobotics.pdf

[3] VĚCHET, Stanislav. *Reaktivní řízení autonomního mobilního robotu v dynamickém vnitřním prostředí: Reactive control of autonomous mobile robot in dynamic indoor environment: zkrácená verze habilitační práce v oboru Aplikovaná mechanika*. Brno: VUTIUM, 2014, 34 s. ISBN 978-80-214-4932-9.

[4] CREATIVE COMMONS. BeagleBoard-xM Systém Reference Manual. In: *BeagleBoard.org* [online]. 4. 4. 2010 [cit. 15. 3. 2015]. Dostupné z: http://beagleboard.org/static/BBxMSRM_latest.pdf

[5] SICK AG WALDKIRCH. Laser Measurement Systems LMS200/211/221/291. In: *Sick.com* [online]. c2006 [cit. 15. 3. 2015]. Dostupné z: http://sicktoolbox.sourceforge.net/docs/sick-lms-technical-description.pdf

[6] Mbed LPC1768. *Mbed* [online]. c2014 [cit. 16. 3. 2015]. Dostupné z: https://developer.mbed.org/platforms/mbed-LPC1768/

[7] CANONICAL LTD. *Ubuntu* [online]. 17. 4. 2014 [cit. 17. 3. 2015]. Dostupné z: http://www.ubuntu.cz

[8] About ROS. *Robot Operating Systém* [online]. [cit. 2. 4. 2015]. Dostupné z: http://www.ros.org/about-ros/

[9] Is ROS For Me?. *Robot Operating Systém* [online]. [cit. 2. 4. 2015]. Dostupné z: http://www.ros.org/is-ros-for-me/

[10] ROS Indigo Iglo. *Robot Operating Systém* [online]. [cit. 4. 4. 2015]. Dostupné z: http://www.ros.org/is-ros-for-me/

[11] CMakeLists.txt. *Wiki.Robot Operating Systém* [online]. Edit. 21. 2. 2015 [cit. 5. 4. 2015]. Dostupné z: http://wiki.ros.org/catkin/CMakeLists.txt

[12] Package.xml. *Wiki.Robot Operating Systém* [online]. Edit. 14. 4. 2014 [cit. 5. 4. 2015]. Dostupné z: http://wiki.ros.org/catkin/package.xml

[13] Browsing packages for indigo. *Robot Operating Systém* [online]. [cit. 4. 4. 2015]. Dostupné z: http://www.ros.org/browse/list.php

[14] Sensors supported by ROS. *Wiki.Robot Operating Systém* [online]. Edit. 15. 4. 2015 [cit. 7. 4. 2015]. Dostupné z: http://wiki.ros.org/Sensors

[15] Topics. *Wiki.Robot Operating Systém* [online]. Edit. 1. 6. 2014 [cit. 7. 4. 2015]. Dostupné z: http://wiki.ros.org/Topics

[16] Writing a Simple Publisher and Subscriber (Python). *Wiki.Robot Operating Systém* [online]. Edit. 4. 3. 2015 [cit. 9. 4. 2015]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28python%29

[17] Package Summary. *Wiki.Robot Operating Systém* [online]. [cit. 9. 4. 2015]. Dostupné z: http://wiki.ros.org/geometry_msgs

[18] Core Components. *Robot Operating Systém* [online]. [cit. 12. 4. 2015]. Dostupné z: http://www.ros.org/core-components/
[19] Roscore. *Wiki.Robot Operating Systém* [online]. Edit. 31. 12. 2013 [cit. 9. 4. 2015]. Dostupné z: http://wiki.ros.org/roscore
[20] XML. *Wiki.Robot Operating Systém* [online]. Edit. 8. 1. 2015 [cit. 10. 4. 2015]. Dostupné z: http://wiki.ros.org/roslaunch/XML
[21] Rosrun. *Wiki.Robot Operating Systém* [online]. Edit. 2. 5. 2015 [cit. 11. 4. 2015]. Dostupné z: http://wiki.ros.org/rosbash#rosrun
[22] Rosnode. *Wiki.Robot Operating Systém* [online]. Edit. 2. 5. 2015 [cit. 11. 4. 2015]. Dostupné z: http://wiki.ros.org/rosnode
[23] Rostopic. *Wiki.Robot Operating Systém* [online]. Edit. 23. 4. 2015 [cit. 12. 4. 2015]. Dostupné z: http://wiki.ros.org/rostopic
[24] Rqt. *Wiki.Robot Operating Systém* [online]. Edit. 11. 2. 2015 [cit. 12. 4. 2015]. Dostupné z: http://wiki.ros.org/rqt
[25] Rviz. *Wiki.Robot Operating Systém* [online]. Edit. 2. 6. 2014 [cit. 14. 4. 2015]. Dostupné z: http://wiki.ros.org/rviz
[26] Multiple Machine. *Wiki.Robot Operating Systém* [online]. Edit. 7. 4. 2015 [cit. 15. 4. 2015]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/MultipleMachines
[27] Robot Setup. *Wiki.Robot Operating Systém* [online]. Edit. 2. 9. 2014 [cit. 19. 4. 2015]. Dostupné z: http://wiki.ros.org/navigation/Tutorials/RobotSetup
[28] Base_local_planner. *Wiki.Robot Operating Systém* [online]. Edit. 5. 10. 2014 [cit. 25. 4. 2015]. Dostupné z: http://wiki.ros.org/base_local_planner
[29] Teleop_twist_keyboar. *Wiki.Robot Operating Systém* [online]. Edit. 22. 1. 2015 [cit. 30. 4. 2015]. Dostupné z: http://wiki.ros.org/teleop_twist_keyboard
[30] Scan_tools. *Wiki.Robot Operating Systém* [online]. Edit. 31. 8. 2012 [cit. 1. 5. 2015]. Dostupné z: http://wiki.ros.org/scan_tools
[31] Navigation. *Wiki.Robot Operating Systém* [online]. Edit. 7. 10. 2012 [cit. 1. 5. 2015]. Dostupné z: http://wiki.ros.org/navigation
[32] TF. *Wiki.Robot Operating Systém* [online]. Edit. 5. 7. 2014 [cit. 2. 5. 2015]. Dostupné z: http://wiki.ros.org/navigation/Tutorials/RobotSetup/TF
[33] Odom. *Wiki.Robot Operating Systém* [online]. Edit. 13. 7. 2013 [cit. 2. 5. 2015]. Dostupné z: http://wiki.ros.org/navigation/Tutorials/RobotSetup/Odom
[34] Gmmaping. *Wiki.Robot Operating Systém* [online]. Edit. 6. 7. 2013 [cit. 4. 5. 2015]. Dostupné z: http://wiki.ros.org/gmapping
[35] Using rviz with the Navigation Stack. *Wiki.Robot Operating Systém* [online]. Edit. 6. 7. 2013 [cit. 4. 5. 2015]. Dostupné z: http://wiki.ros.org/navigation/Tutorials/Using%20rviz%20with%20the%20Navigation%20Stack

Seznam příloh

Příloha č. 1 – CD, které obsahuje:

- Elektronickou podobu bakalářské práce
- Soubor balíčku a knihoven, které tvoří navigační systém
- Video z jízdy robota Bender II