VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

FRAMEWORK ROS A JEHO VYUŽITÍ S OHLEDEM NA POHONY

ROS FRAMEWORK AND ITS USE IN REGARD TO ACTUATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — Vojtěch Starý
AUTHOR

VEDOUCÍ PRÁCE — Ing. Tomáš Hůlka
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Vojtěch Starý**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**Ing. Tomáš Hůlka**
Akademický rok:	2019/20

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

Framework ROS a jeho využití s ohledem na pohony

Stručná charakteristika problematiky úkolu:

Úkolem studenta bude seznámit se s frameworkem ROS (Robot Operating System) a zaměřit se na možnosti aktuace. Poté student navrhne a vytvoří model robotu a otestuje ho na definované demonstrační úloze.

Cíle bakalářské práce:

Stručná rešerše problematiky.
Návrh a realizace robotu v rámci frameworku ROS.
Otestování funkčnosti na definované úloze.

Seznam doporučené literatury:

QUIGLEY, M., et al. ROS: an open-source Robot Operating System. ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

QUIGLEY, M., et al. Programming Robots with ROS: a practical introduction to the Robot Operating System. O'Reilly Media, Inc., 2015.

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

V Brně, dne

L. S.

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

ABSTRAKT

Tato bakalářská práce se zabývá frameworkem ROS a jeho využitím. Software ROS je operačním systémem používaným při řízení robotů. Podstatou této práce je stručný popis frameworku ROS, postup vytvoření a zprovoznění vlastního robota a vyzkoušení robota na definovaných úkolech. Prvním úkolem je mapování simulovaného prostředí a druhým autonomní navigace v simulovaném prostředí. Střední část se zabývá pohony.

ABSTRACT

This bachelor thesis deals with the ROS framework and its use. ROS software is an operating system used to control robots. The essence of this work is a brief description of the ROS framework, the process of creating and commissioning your own robot and testing the robot on defined tasks. The first task is to map the simulated environment and the second is autonomous navigation in the simulated environment. The middle part deals with drives.

KLÍČOVÁ SLOVA

ROS Framework, ROS Development Studio, URDF formát, XML makra.

KEYWORDS

ROS Framework, ROS Development Studio, URDF format, XML macros.

BIBLIOGRAFICKÁ CITACE

STARÝ, Vojtěch. Framework ROS a jeho využití s ohledem na pohony, Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky.

PODĚKOVÁNÍ

Děkuji vedoucímu práce Ing. Tomáš Hůlka za vedení, cenné rady a vstřícný přístup při zpracování této bakalářské práce.
Dále bych chtěl poděkovat své rodině za pochopení a podporu při studiu a během psaní bakalářské práce.

ČESTNÉ PROHLÁŠENÍ

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

V Brně dne 22. 5. 2020 ……

Vojtěch Starý

OBSAH

1 ÚVOD

Tato práce se zabývá řídícím systémem pro roboty ROS (Robot Operating System), což je open-source framework sloužící pro tvorbu softwaru robota. Cílem této bakalářské práce je seznámení se se systémem ROS, vytvoření vlastního modelu robota a otestování vytvořeného modelu robota na definovaných úkolech.

Práce je rozdělena do čtyř částí, z čehož první a druhá část jsou rešeršního charakteru. Třetí a čtvrtá část jsou praktické.

První část se zaobírá frameworkem ROS, jeho distribucí, činností systému ROS a seznámením s ROS Development Studio platformou. V druhé části lze nalézt základní typy pohonů pro ROS a jejich stručnou charakteristiku, jelikož tahle část není primárním cílem této práce, tak není podrobněji rozebírána. Třetí část obsahuje popis vytvoření projektu v ROS Development Studio platformě včetně specifikování verze ROSu. Dále je zde kompletní popis vytvoření vlastního robota, včetně kinematických a dynamických vlastností, a popis funkcí jednotlivých oddílů projektu. Poslední část se zaobírá způsobem a postupem testování vytvořeného robota na definovaných úkolech (mapování a autonomní navigace v simulovaném prostředí).

2 FRAMEWORK ROS

Projekt ROS vznikl na Stanfordské univerzitě v květnu 2007 pod vedením Willow Garage. První distribuce ROSu přišla v březnu 2010 a od roku 2013 spadá vývoj ROSu pod OSRF (Open Source Robotic Foundation).[1]

ROS neboli Robot Operating System je vlastně systém pro programování robotů. ROS je pro roboty to samé, jako je Android pro telefony či Windows pro počítače. Jedná se o Open-source systém, který slouží primárně pro ovládání robotů. Funguje podobně jako operační systém, jež má svoje nástroje a knihovny, které umožnují získávání, psaní a spouštění kódu na více počítačích. Avšak není to plnohodnotný operační systém, neboť pro svou funkci potřebuje operační systém, proto je lepší označení Framework ROS.[2]

Dříve bylo největší překážkou při programování robotů skloubit hardware se softwarem. Vývojáři tak museli znát jak programovou, tak mechanickou část robota. Museli vědět, jak správně komunikovat s elektrickými komponenty robota (senzory a motory) a k tomu, jak je správně interpretovat pro samotné programování robota. Každá firma si tak vyvíjela svoje vlastní softwary a postupy pro roboty. Díky tomu byl kladen velký nátlak na vývojáře, kteří museli řešit problémy při programování softwaru robota a k tomu ještě problémy mezi komunikací softwaru s robotem. Navíc pro každého robota musel být vyvíjen zcela nový software, který uměl správně komunikovat s hardwarem daného robota. Díky ROSu lze rozdělit software od hardwaru, a navíc se samotný software se stal jednoduše modulovatelným. Díky ROS API (Application Programming Interface), což je seznam ROS topics, services, action servers a messages která nám umožnují komunikaci s daným typem robota, lze donutit dělat roboty to co chcete.[2]

Obrázek 2: Logo ROS [3]

Podobnost ROSu s ostatními systémy jako jsou MOOS, Otca, CARMEN, Microsoft Robotics Studio tu samozřejmě je avšak hlavním cílem ROSu je opakované použití již vytvořených kódů pro vývoj a výzkum robotiky. Nejedná se tedy o systém s nejvíce funkcemi, ale díky jeho rozložitelnosti na jednotlivé package se dá jednoduše sdílet. Uživatelé tak nemusí jednotlivé kódy pro ovládání pohonů, čtení senzorů a ostatní časti hardwaru psát ale mohou si je stáhnou například od odborníků v dané oblasti a upravit dle svých potřeb což značně ulehčuje a zrychluje vývoj robotiky.[4, 5]

2.1 Distribuce

Jelikož konstantním „upgradovanim“ jádra ROSu by vznikly značné potíže tak pravidelně vychází kompletní balíčky ROSu jako jeho distribuce. Jakmile je vydána nová distribuce tak se vývojáři snaží omezit změny na opravy chyb. V současné době existuje dvanáct distribucí ROSu a v květnu 2020 je plánovaná nová distribuce (ROS Noetic Ninjemys). Podporované distribuce jsou pouze ty nejnovější což jsou ROS Melodic Morenia jež vyšel 23. května 2018 a plánovanou podporu má mít do Května 2023 a ROS Kinetic Kame Jež vyšel 23. května 2016 s plánovanou podporou do Dubna 2021 (Distribuce ROS Kinetic Kame byla zvolena pro potřeby této práce). V dnešní době je ROS plně podporován na Linuxových systémech (Ubuntu, Debian) a nejnovější distribuce (ROS Melodic Morenia) dokonce na Windows 10. Experimentální podporu mají i některé další systémy, avšak není doporučeno je používat, jestliže uživatel není zběhlý s ROSem. Každá novější distribuce tedy nabízí lepší podporu pro software i hardware ale také lepší nástroje a knihovny.[3, 6]

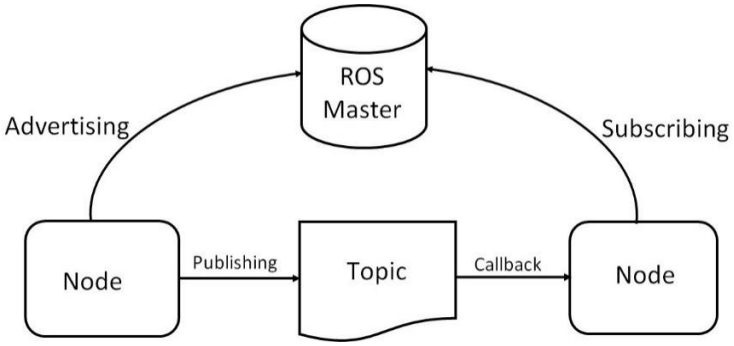
Obrázek 2.1: Distribuce ROS Loga [6]

2.2 Packages

Celý software ROS je postaven z jednotlivých packages, které slouží k uspořádání systému. Packages mohou nést třeba ROS nody, nezávislé knihovny, datasety, konfigurační soubory či cokoliv co obsahuje užitečné logické schéma. Jednotlivé packages se dají buď stáhnout nebo si můžete vytvořit vlastní pomocí funkce ROSu *catkin_create_pkg*. Samotná distribuce ROSu je touto formou dodávána. Díky tomuhle systému, který má lehce naučitelnou a užitečnou funkci, se dá celý software lehce znovu použít.[5, 7]

2.3 Komunikace ROSu

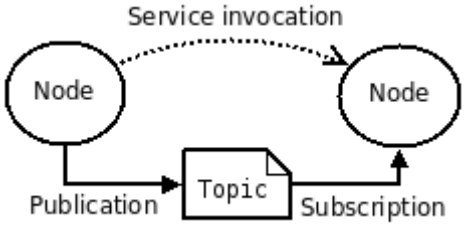
Proces, který provádí výpočty v ROSu, je nazýván „node“ a má zodpovědnost za jednotlivé úkoly. Výpočet probíhá pomocí P2P (Peer-to-peer) sítě. Samotný systém je většinou tvořen značným množstvím nodů. Aby mohli nody mezi sebou komunikovat, má každý node svůj unikátní název, díky kterému je lze snadno identifikovat vůči systému. Nody jsou ještě rozděleny do typů což zjednodušuje jejich vyhledávání. Aby šlo tohle prostředí ROS nodů dobře koordinovat je vytvořena služba ROS Master (obr.2.2), která zodpovídá za správné spojení komunikace mezi nody, avšak poskytuje pouze vyhledávací informace. Komunikace mezi jednotlivými nody může probíhat 3 způsoby, a to streamováním Topics, RPC (Remote procedure call) Services a Parameter Server.[5, 8, 9]



Obrázek 2.2: ROS Master [5]

2.3.1 Topics

Jedná se o jednosměrnou komunikaci, která funguje pomocí anonymní Publish/Subscribe struktury. Nody se přihlašují do Topics jako Subscribers, kteří data přijmají, nebo jako Publishers, kteří data odesílají, popřípadě obojí. Topics (obr. 2.3) fungují jako schránky pro přenos informací mezi nody. Jednotlivé nody nevědí odkud data jsou, zajímá je pouze jestli data přijímat nebo odesílat. Tento typ komunikace je kontinuální, tzn. jakmile se node přihlásí jako Publisher/Subscriber bude neustále odesílat/přijímat všechna nová data.[10]



Obrázek 2.3: Topic komunikace [11]

2.3.2 Services

Další způsobem komunikace mezi nody jsou tzv. Services, které komunikují pomocí Server/Client struktury. Jedná se o jedno účelovou komunikaci mezi nody, kde Client node pošle požadavek na Service node a ten pošle zpětnou vazbu. Poté se komunikace zase uzavře.[12]

2.3.3 Parameter Server

Tento typ komunikace slouží pouze pro přenos malého objemu dat, jako jsou konfigurační parametry a binární data. Samotný server se nachází uvnitř ROS Master a je přístupný díky síti API. Používá se jako globální server pro nástroje ke kontrole či konfiguraci systému. Nody jej využívají pro ukládání a načítání parametrů za provozu.[13]

2.4 ROS Development Studio (ROSDS)

ROSDS je vývojovou platformou pro ROS. Největší výhodou ROSDS je, že pro samotné programování robotů je potřeba pouze internetové připojení. Odpadá zde totiž potřeba řešit, jestli máme dostatečně výkonný hardware pro ROS, jakou verzi ROSu použít a jaká verze a typ operačního systému pro danou distribuci ROSu je potřeba. Nemusí se zde nic instalovat ani připravovat. Vše je připraveno pro samotné programování robota, což výrazně ulehčuje a zrychluje vývoj robotiky. Celé prostředí ROSDS je velice uživatelsky přívětivé a umožnuje i laikovi bez jakýchkoliv znalostí linuxových systémů velice rychlé osvojení programování robotů za pomoci ROSu. Framework ROS zde přitom funguje stejně jako na linuxových platformách včetně všech příkazů. Všechny projekty vytvořené v ROSDS se dají velice jednoduše sdílet jen pomocí internetového odkazu a kdokoliv tak může duplikovat naprosto stejné výsledky. Tato platforma je primárně vytvořena, aby zjednodušila a ulehčila vývoj robotiky.[14, 15]

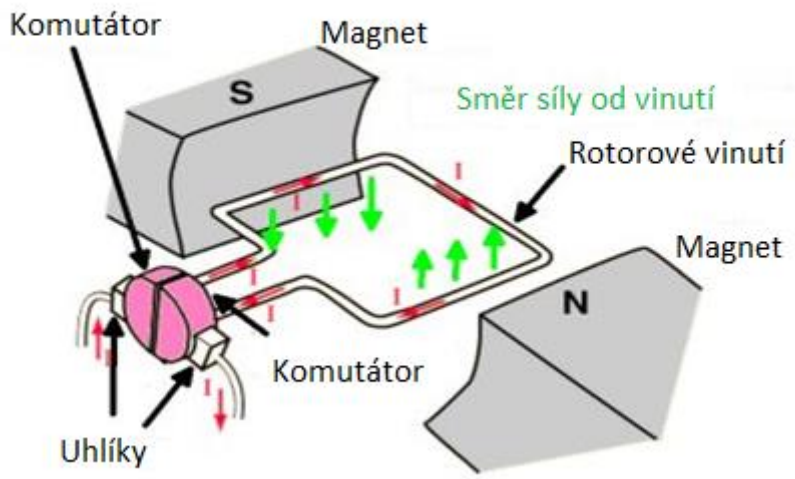
Obrázek 2.4: Logo ROSDS [16]

3 TYPY POHONŮ V ROSU

V Rosu není omezení pro užití typu pohonu, ovšem jen pokud jej dokážeme s ROSem správně spojit. Jestliže se však člověk v tomto odvětví nevyzná, je lepší použít pohony, pro které je zpracovaná podpora například od firmy ROBOTIS (http://en.robotis.com/). Protože typ pohonu není omezen, zmíním pouze pár základních (nejpoužívanějších) typů, jelikož tato kapitola není primárním cílem předkládané práce. Základním principem všech elektromotorů je měnit elektrickou energii na mechanickou práci.[17]

3.1 DC elektromotory

DC elektromotory jsou motory pracující na stejnosměrném elektrickém proudu. Skládají se z rotoru s cívkami vyvedenými k mechanickému komutátoru, k němuž přiléhají grafitové kartáče, a statoru s póly. DC elektromotory fungují na principu proudové smyčky umístněné v magnetickém poli a jejich výhodou je velký točivý moment při nižších otáčkách, snadná změna smyslu otáčení a jednoduché řízení motoru. Nevýhodou je nutnost údržby komutátoru, větší rozměry než střídavé motory a klesání výkonu s navyšujícími se otáčkami.[18, 19]

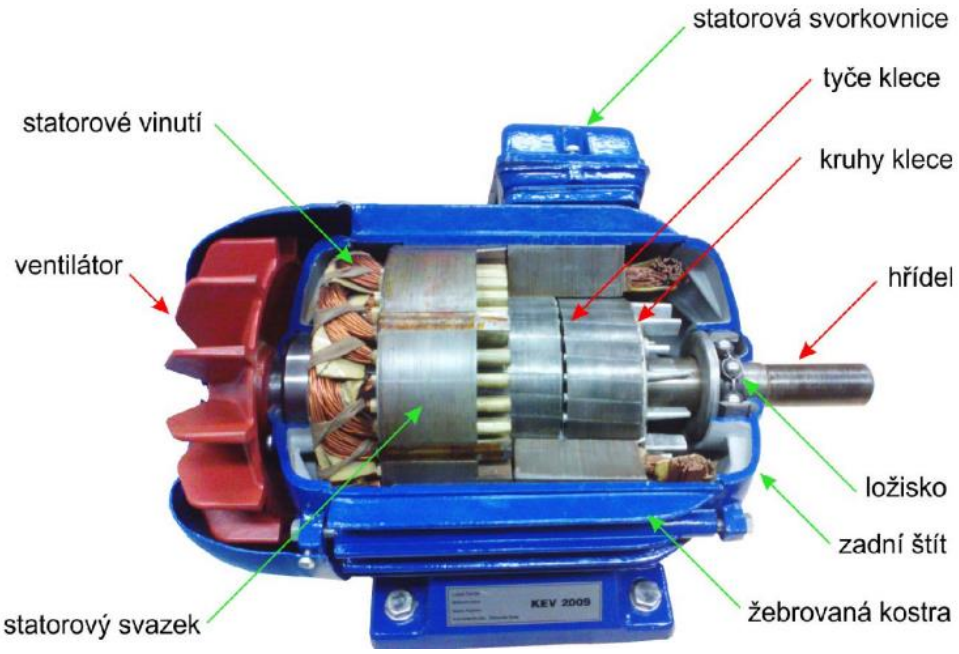


Obrázek 3.1: DC elektromotor [20]

3.2 Asynchronní motory

Asynchronní motory se staly nejrozšířenějšími elektromotory pracujícími na střídavém proudu. Skládají se ze statoru a rotoru, kde stator je pevnou částí motoru s vinutím z izolovaných vodičů, a rotor je otočnou částí motoru s klecí se zešikmenými neizolovanými tyčemi spojenými na koncích spojovacími kruhy. Rotor a stator však

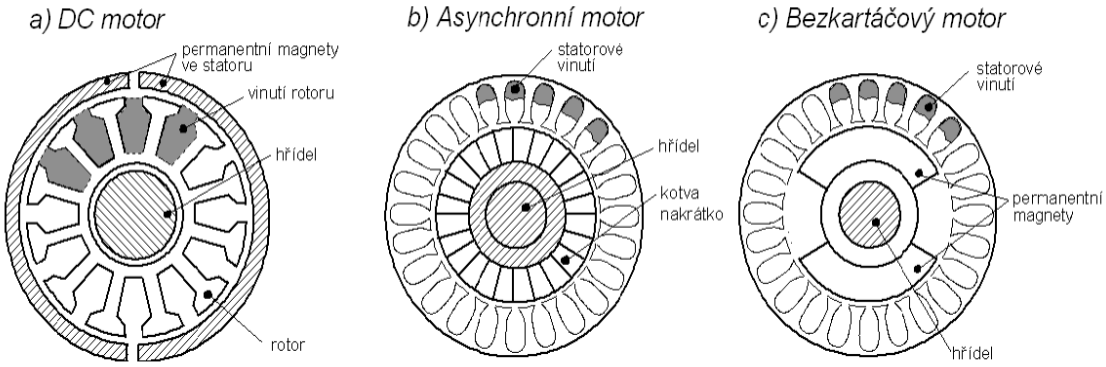
nejsou v přímém kontaktu, kromě ložisek. K přenosu energie tedy dochází pomocí elektromagnetické indukce. Výhodou těchto motorů je, že jsou prakticky bezúdržbové, relativně levné a mohou pracovat jako motor či generátor. Nevýhodou je složitější řízení motoru.[18, 19, 21]



Obrázek 3.2: Asynchronní motor s kotvou na krátko [22]

3.3 Bezkartáčové elektromotory

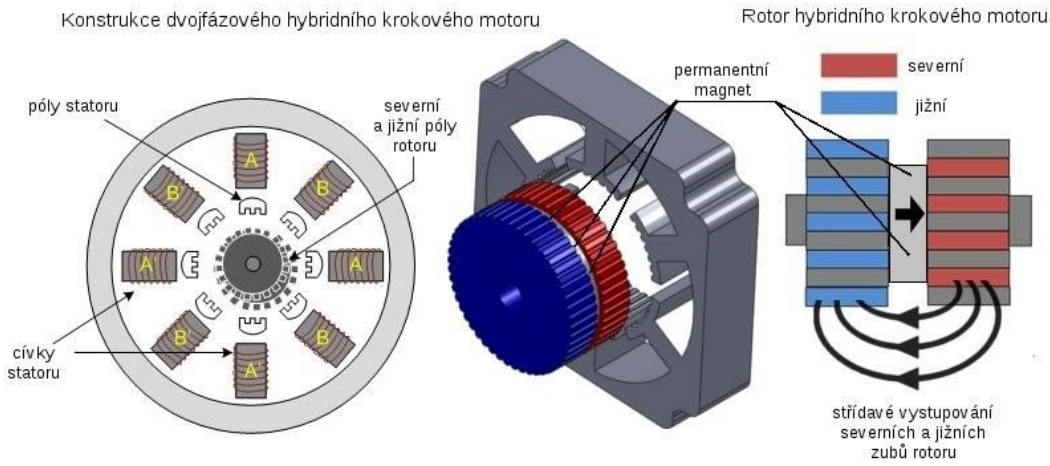
Bezkartáčové elektromotory jsou jedny z nejmodernějších pohonů současné doby, jež kombinují nejlepší vlastnosti DC motorů a asynchronních motorů. Princip funkce vychází z DC motorů, ale se záměnou funkce rotoru a statoru. Výhodami jsou tedy vyšší otáčky, lepší dynamické vlastnosti, zmenšení rozměrů motorů a zvýšení spolehlivosti a životnosti. Nevýhodou je složitější řízení motoru.[18]



Obrázek 3.3: Princip konstrukce elektromotorů [18]

3.4 Krokové motory

Krokové motory jsou synchronní motory poháněné pulzy stejnosměrného proudu. Skládají se ze statoru s cívkami a rotoru z magneticky měkké oceli. Magnetické pole je zde generováno napájením cívkových dvojic. Jejich postupným zapínáním se tvoří točivé magnetické pole, přičemž rotor se vždy pootočí do nejbližší magnetické klidové polohy. Výhodou je vysoká polohová přesnost, vysoký točivý moment pří nízkých otáčkách a výborné dynamické vlastnosti. Nevýhodou je citlivost na vyšší otáčky, pří ztrátě kroku je ztracena i poloha, a točivý moment je menší než u stejnosměrných a střídavých elektromotorů.[19, 23]



Obrázek 3.4: Konstrukce krokového motoru [23]

4 REALIZACE PROJEKTU

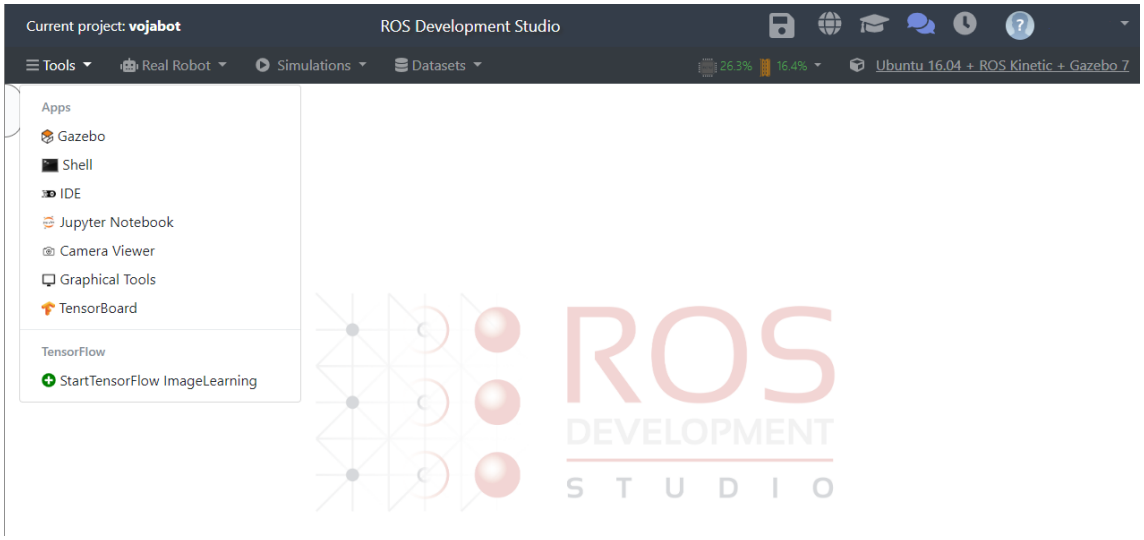
Praktická část této bakalářské práce je dělána za použití platformy ROSDS s ROS Kinetic Kane distribucí a CAD softwaru Autodesk Inventor 2020.

4.1 Vytvoření ROSjectu

Pro použití platformy ROSDS, musí být uživatel přihlášen na jejich stránkách. Základní účet, který nabízí 8 hodin denně v pracovním prostředí ROSjetu s cloudovým hardwarem 2 CPU a 8 GB RAM, je zdarma. Avšak je zde možné vytvářet pouze veřejné ROSjecty a není zde možnost komunikace s reálným robotem. Dají se však zakoupit balíčky a licence, které ruší tahle omezení plus navyšují hardwarové parametry až na 32 CPU, 4 GPU a 60 GB RAM. Poté už je vytvoření ROSjectu velice jednoduché. Stačí kliknout na ikonu New ROSject, vybrat si konfiguraci ROSu, typ ROSjectu (soukromý/veřejný) a pokud chcete tak si vybrat model robota na programování z 24 předpřipravených modelů. Momentálně jsou nabízeny 2 konfigurace ROSu (Ubuntu 16.04 + ROS Kinetic + Gazebo 7 a Ubuntu 18.04 + ROS Melodic + Gazebo 9). Mnou zvolené konfigurace byla Ubuntu 16.04 + ROS Kinetic + Gazebo 7 bez modelu robota. Jsou zde ještě možnosti konfigurace pro ROS2, ale tímhle tématem se bakalářská práce nezaobírá.[24]

4.2 Seznámení s prostředím

Po otevření ROSjectu se zobrazí hlavní pracovní prostředí, které je plně konfigurovatelné, což umožnuje měnit i oddělit se od nezávislých oken. Na horním panelu se nachází čtyři kategorie (Tools, Real Robot, Simulation, Datasets) a informace o daném ROSjectu. Kategorie Tools obsahuje aplikace potřebné k programování robota např. Linux Shells, cloud 9 IDE, simulátor Gazebo, knihovny OpenAI, Rviz, aj. Kategorie Real Robot umožnuje připojit se k reálnému robotu pomocí vytvoření sítě VPN (Virtual Private Network). Kategorie Simulation umožnuje načíst uživatelem vytvořené simulační prostředí nebo si vybrat ze seznamu již před vytvořených simulačních prostředí. Poslední kategorií je Datasets, které slouží pro ukládání velkého množství informací, například balíčků obrázků pro učení umělé inteligence.



Obrázek 4.1: ROSDS screenshot

4.2.1 Tools - Nástroje

Gazebo

Gazebo je 3D simulátor prostředí pro roboty včetně fyzikálního enginu. Jedná se o open-source aplikaci, která umožnuje testování robotů. Poskytuje realistické vykreslování prostředí včetně osvětlením, stínů a textur.[25]

Shell

Shell je příkazový řádek který funguje stejně jako v Linuxových operačních systémech.

IDE

IDE (Itegrated development environment) je vývojové prostředí které ulehčuje orientaci při programování. ROSDS požívá svoje vlastní IDE, které je přizpůsobené pro ROS.

Jupyter Notebook

Jupyter notebook je open-source aplikace, která umožňuje vytvářet a sdílet dokumenty, které obsahují živý kód, rovnice, vizualizace a vysvětlující text. Tedy umožnuje propojit výpočetní informace s multimédii a grafy.

Camera Viewer

Tato aplikace umožnuje zobrazení pohledu kamery robota.

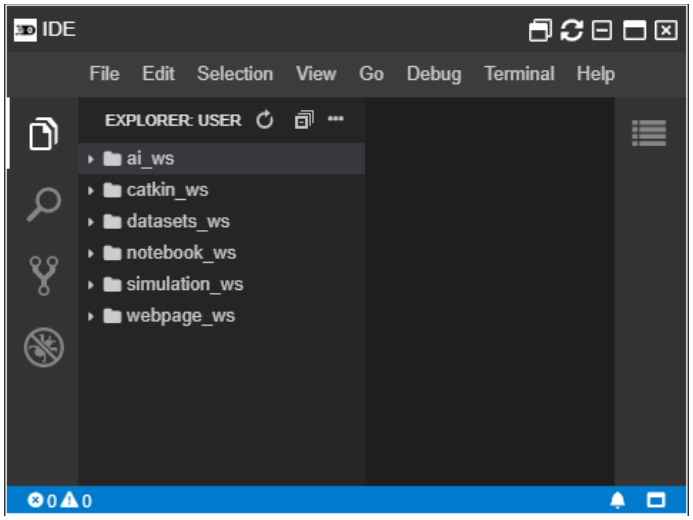
Graphical Tools (Rviz)

Rviz je trojrozměrné vizualizační prostředí, které nám umožnuje vidět co robot dělá, vidí a jak přemýšlí.

Tensor Board
Tensor board je aplikace jež poskytuje nástroje potřebné pro strojové učení a vizualizaci.

4.2.2 Pracovní prostor (_ws)

V ROSDS máme několik pracovních prostorů (ai_ws, catkin_ws, datasets_ws, notebook_ws, simulation_ws a webpage_ws), ovšem důležitými prostory pro tuhle práci jsou pouze *catkin_ws* a *simulation_ws*.



Obrázek 4.2: IDE screenshot

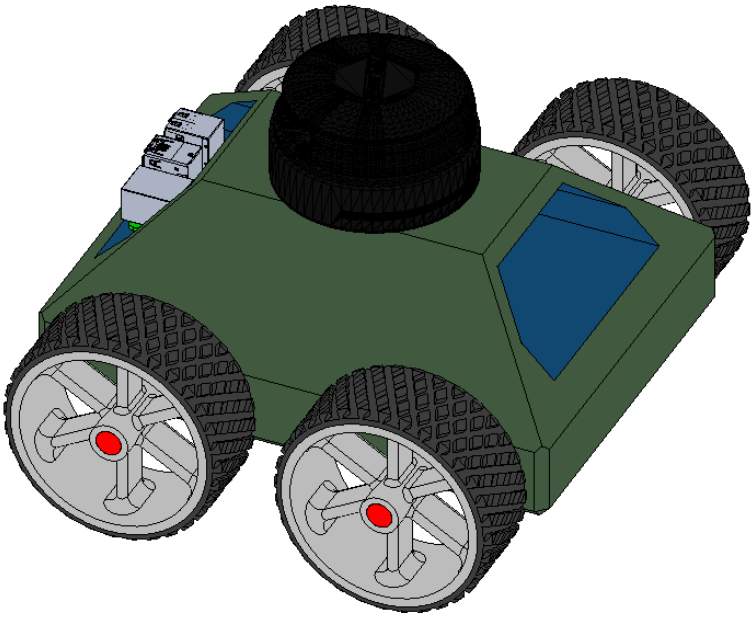
catkin_ws
Catkin workspace je hlavní pracovní prostor sloužící pro upravování, vytváření a instalování Catkin packages. Obsahuje 3 složky a to *Build*, *Devel* a *Src* která je klíčovou pro tuhle práci.

simulation_ws
Simulation_ws je použit jako pomocný pracovní prostor pro definování robota a simulačního světa.

4.3 CAD model robota vojabot

Pro vytvoření modelu robota byl zvolen program Autodesk Inventor 2020. Modely se poté vkládají do URDF formátu (Unified Robot Description Format), který komunikuje pouze s Colada a STL soubory. Colada soubory jsou lepší variantou, jelikož přenášejí i informace o barvě a texturách modelu od STL souborů. Inventor však umožnuje exportovat pouze soubory do STL formátu. Jelikož však barva modelu není podstatná pro funkci robota, byl použit STL formát.

Byly vytvořeny dva modely, model kola (vojabot_kolo.stl) a model těla robota (vojabot_telo.stl). Pro model laserového scanneru byl použit model skutečného scanneru RPLIDAR A2 (rplidar.dae) včetně jeho parametrů.



Obrázek 4.3: Model vojabot

4.4 Package vojabot_description (definování robota)

Package vojabot_description nese veškeré informace o robotovi včetně vizuálních, dynamických a kinematických vlastností. Pro vytvoření package použijeme příkazy:

```
$ cd/simulation_ws/src/
$ catki_kreate_pkg vojabot_description urdf
```

První příkaz nás posune do složky src v simulation_ws a druhý příkaz vytvoří package vojabot_description závyslý na urdf. Poté vytvoříme ve vojabot_description package čtyři složky Config, Daeimg, Launch a Urdf pomocí IDE nebo příkazu:

```
$ mkdir Název_složky
```

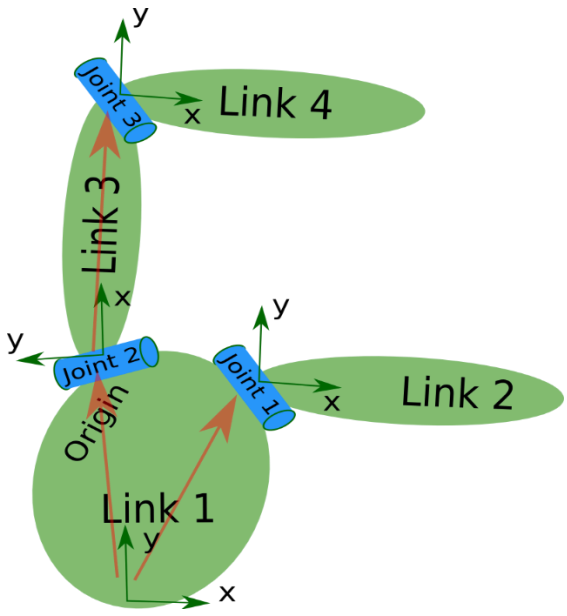
Po vytvoření všech čtyř složek rovnou nahrajeme všechny modely robota do složky *Daeimg*. Dále do složky *config* stáhneme základní konfiguraci pro pohyb a navigování robota v prostoru z přílohy. Do složky *Launch* stáhneme *vojabot_gazebo.launch* soubor, který se stará o načtení robota do rvizu a gazeba z přílohy. Samotné vytvoření těla robota se odehrává ve složce urdf.

4.5 URDF Formát

URDF je vlastně XML formát sloužící pro reprezentaci modelu robota. XML (eXtensible Markup Language) je softwarově a hardwarově nezávislý nástroj sloužící pro ukládání a přenos dat. URDF nese kinematické a dynamické informace o robotovi, vizuální popis robota a kolizní model robota (délkovou jednotkou je zde metr a úhlovou jednotkou radián). Tento formát však dokáže reprezentovat pouze stromové struktury, což vylučuje všechny paralelní struktury robotů a neumožnuje užití flexibilních kloubů. Struktura URDF modelu je tvořena dvěma prvky a to tzn. Linky a Jointy, kde Linky jsou jednotlivé části robotu a Jointy jsou vazby mezi linky.[26, 27] Schéma URDF modelu vypadá takhle:

```
<?xml version="1.0"?>
<robot name="vojabot">
    <link> ... </link>
    <link> ... </link>

    <joint> .... </joint>
    <joint> .... </joint>
</robot>
```

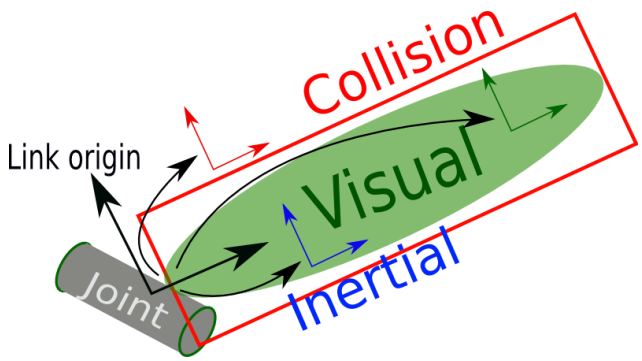


Obrázek 4.4: Schéma modelu [27]

4.5.1 Link

Link udává vlastnosti tělu robota pomocí tří vlastností *Inertial*, *Visual*, *Collision*. Linky jsou rozděleny pomocí atributu *Name*.[28]

```
<?xml version="1.0"?>
<robot name="vojabot">
    <link name="Nazev_linku">
      <inerial>
        ...
      </inertial>
      <visual>
        ...
      </visual>
      <collision>
        ...
      </collision>
    </link>
</robot>
```



Obrázek 4.5: Schéma linku [28]

Inertial

Inertial obsahuje kinematické a dynamické vlastnosti daného linku pomocí tří parametrů *Origin*, *Mass* a *Inertia*.

- Origin

Obsahuje informace o těžišti daného linku.

```
<origin xyz="0 0 0" rpy="0 0 0" />
```

- Mass

Obsahuje informaci váhy daného linku v kilogramech.

```
<mass value="1"/>
```

- Inertia

Obsahuje informace o tenzoru setrvačnosti (3x3 matice) daného linku. Skládá se však jen z ixx, ixy, ixz, iyy, iyz, izz atributů díky symetrii matice.

```
<inertia ixx="100"  ixy="0"  ixz="0" iyy="100" iyz="0" izz="0" />
```

Visual

Visual osahuje vzhledové vlastnosti linku včetně vizuálního tvaru linku za pomocí tří parametrů *Origin*, *Geometry* a *Material*. Vlastnost visual se v jednom linku může vyskytovat vícekrát.

- Origin

Obsahuje souřadnice posunutí a natočení souřadnicového systému vizuálního prvku vůči počátku souřadnicového systému daného linku.

```
<origin xyz="0 0 0" rpy="0 0 0" />
```

- Geometry

Obsahuje informace tvaru vizuálního objektu pomoci geometrických objektů (box cylinder a sphere) nebo 3D modelu.

Box

Vytvoří krabici pomocí atributu size, který obsahuje tři boční délky krabice s počátkem ve středu.

```
<geometry>
  <box size="1 1 1" />
</geometry>
```

Cylinder

Vytvoří válec pomocí atributů radius a length které obsahují poloměr a výšku válce s počátkem ve středu.

```
<geometry>
  <cylinder radius="1" length="0.5"/>
</geometry>
```

Sphere

Vytvoří kouli pomocí atributu rádius, který obsahuje poloměr koule s počátkem ve středu.

```
<geometry>
  <sphere radius="1"/>
</geometry>
```

Mesh

Nahraje 3D model pomocí atributů Filename (umístnění souboru) a Scale (měřítko pro 3 osy)

```
<mesh filename="package://Package/cesta/soubor.stl" scale="1 1 1"/>
```

- Material

Obsahuje informace o barvě či textuře linku.

Color

Určuje barvu materiálu pomoci RGBA.

```
<material>
  <color rgba="0 1.0 1.0 1.0"/>
</material>
```

Texture

Textura materiálu je specifikována stejně jako Mesh jenom bez atributu Scale.

Collision

Collision obsahuje tvarové informace používané při výpočtech kolizí. Geometrické informace zde bývají často zjednodušovány, a to kvůli zrychlení výpočtů. Skládá se ze dvou parametrů *Origin* a *Geometry*, které fungují stejně jako u vlastnosti Visual.

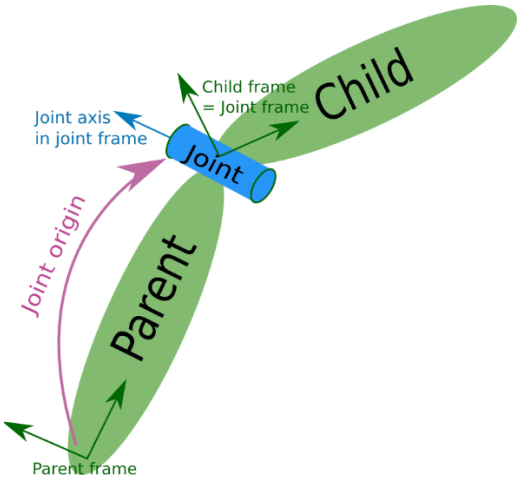
4.5.2 Joint

Joint slouží jako spojovací kloub mezi linky. Specifikuje jeho kinematické a dynamické vlastnosti a udává bezpečnostní limity pro pohyb kloubu. K tomu může použít až devět komponent *Origin*, *Parent link*, *Chield link*, *Axis*, *Calibration*, *Dynamics*, *Limit*, *Mimic* a *Safety_controller*. Jointy jsou dále rozděleny pomocí atributů *Name* a specifikovány pomocí atributu Type. Atribut Type určuje typ spoje z následujících možností: [29]

- **revolute** Kloubový spoj, který umožnuje rotaci kolem své osy a je omezen horním a dolním limitem.
- **continuouse** Kloubový spoj, který se může otáčet pouze kolem své osy.
- **prismatic** Spoj, který se posouvá podél své osy s omezeným rozsahem pohybu určeným horním a dolním limitem.
- **fixed** Pevný typ spoje, který neumožnuje pohyb.
- **floating** Spoj se šesti stupni volnosti.
- **planar** Spoj umožňuje pohyb v rovině kolmé k ose.

```
<?xml version="1.0" encoding="UTF-8"?>
<robot name="vojabot">
```

```
        <joint name="rear_right_wheel_joint" type="Continuouse">
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <parent link="base_link"/>
            <child link="rear_right_wheel"/>
            <axis xyz="0 1 0" rpy="0 0 0"/>
            <limit effort="1" velocity="10"/>
            <joint_properties damping="5.0" friction="1.0"/>
        </joint>
    </robot>
```



Obrázek 4.6: Schéma joint [29]

Origin

Obsahuje informace polohy těžiště *Child link* vůči *Parent link*.

```
    <origin xyz="0 0 0" rpy="0 0 0" />
```

Parent link

Obsahuje Name linku, jež je nadřazený vůči linku připojovanému.

```
    <parent link="Link_1"/>
```

Chield link

Obsahuje Name linku, jež je připojován k nadřazenému linku.

```
    <child link="Link_2"/>
```

Axis
Používá se pro specifikování stupně volnosti. Pro type revolute a continuous to jsou osy rotace, pro planar normály rovin a pro prismatic osa translace. Pro type fixed a planar se tato komponenta nepoužívá.

```
<axis xyz="0 1 0" rpy="0 0 0"/>
```

Calibration
Obsahuje referenční pozici kloubu pro určení absolutní pozice kloubu.

```
<calibration rising="0.0"/>
```

Dynamics
Obsahuje fyzikální vlastnosti používané pro simulace robota. Používá dva atributy damping pro velikost tlumení (pro prismatic type v N*s/m a pro revolute type v N*m*s/rad) a friction pro statické tření spoje (pro prismatic type v N a pro revolute type v N*m).

```
<dynamics damping="0.0" friction="0.0"/>
```

Limit
Obsahuje informace o limitním zatížení, rychlosti a pohybu kloubu.

lower	Udává spodní limit pohybu kloubu.
upper	Udává horní limit pohybu kloubu.
effort	Udává maximální zatížení.
Velocity	Udává maximální rychlost pohybu kloubu.

```
<limit effort="1" velocity="10"/>
```

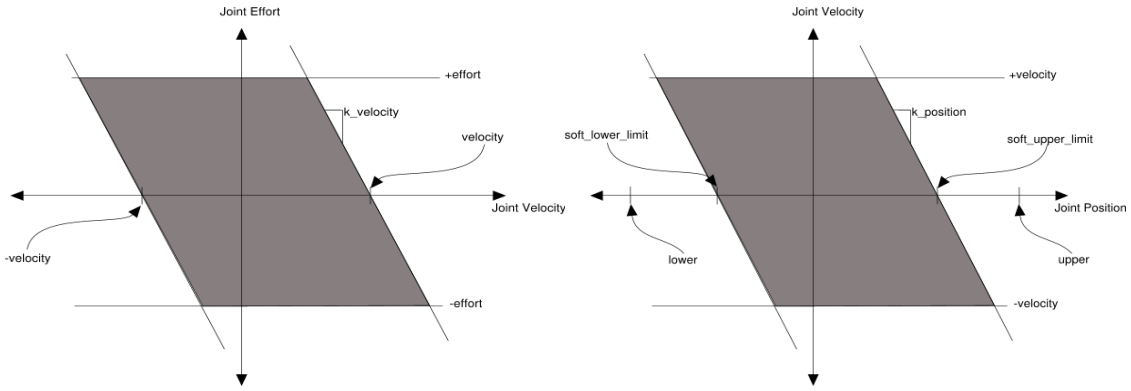
Mimic
Slouží pro definování pohybu jointu pomocí napodobení již existujícího jointu pomocí atributu joint name.

Multiplier	Určuje směr a velikost pohybu násobkem výchozího jointu.
offset	Určuje hodnotu základního natočení.

```
<mimic joint="rear_right_wheel_joint" multiplier="1" offset="0"/>
```

Safety_controller
Obsahuje *k_velocity*, *k_position*, *soft_lower_limit* a *soft_upper_limit*, jejichž funkci vystihuje obrázek 4.7.

```
<safety_controller k_velocity="10" k_position="15" soft_lower_limit="-2.0" soft_upper_limit="0.5" />
```



Obrázek 4.7: Bezpečnostní limity [30]

4.6 XML Macros

XML Macros, neboli xacro jazyk, pomáhá zjednodušit a zlepšit čitelnost XML souborů a to nadefinováním maker. Toto se hodí zejména u obsáhlejších XML dokumentů například při popisu robota, kde se vyskytují často se opakující pasáže. V případě této práce bylo použito v package URDF.[31]

Použití xacro
Pro vytvoření souboru stačí přidat za název koncovku .xacro a do hlavičky vložit:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
    ...
</robot>
```

Makra
Makro lze vytvořit pomocí atributu Macro, který obsahuje název makra (name) a v případě potřeby parametry (params) pro dané makro. Parametry se v makru používají pomocí výrazu ${} a nemusí nabývat pouze číselných hodnot.[31] Vytvořené makro pro link pak může vypadat následovně:

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:macro name="link_wheel" params="name">
    <link name="${name}">
      ...
    </link>
  </xacro:macro>
</robot>
```

Vytvořené makro se dá poté použít pomocí:

```
<xacro:Název_makra Název_parametru_makra="Parametr" />
```

Xacro se dá dále využít pro nadefinování dalších vlastností například konstant, a to pomocí atributu *Property* následovně:

```
<xacro:property name="radius" value="2" />
<xacro:property name="length" value="5.5" />
<xacro:property name="main_origin">
  <origin xyz="0 1 1" rpy="0 0 0" />
</xacro:property>
```

Hodnoty Value však nemusí nabývat pouze číselných hodnot. Property lze poté použít vložením do výrazu ${}, který může obsahovat i základní matematické operace.

```
<geometry type="cylinder" radius="${radius}" length="${2 * length}" />
```

Lze vkládat i celé bloky a to pomocí:

```
<xacro:insert_block name="main_origin" />
```

4.7 Složka URDF

Pro lepší orientování v kódu a zjednodušení je popis robota rozdělen do čtyř souborů Vojabot.xacro, Macros.xacro, Materials.xacro a Vojabot.gazebo. Pro zachování funkčnosti jsou všechny soubory připojeny k Vojabot.xacro souboru pomocí:

```
<xacro:include filename="$(find
vojabot_description)/urdf/Název_souboru.xacro" />
```

- vojabot.xacro
Tento soubor obsahuje základní definici tvaru a funkčnosti robota pomocí link/joint struktury a maker.
- macros.xacro
Tento soubor obsahuje makra použitá k tvorbě robota.
- materials.xacro
Tento soubor obsahuje barvy robota pro rviz vizualizér.
- vojabot.gazebo
Tento soubor obsahuje informace pro simulaci robora v Gazebo včetně popisu funkčnosti laserového scanneru.

4.8 Package vojabot_simulation (definování simulačního světa)

Package vojabot_simulation nese informace o simulačním světě v gazebo a spouštěcí soubor simulace. Vytvoříme package *vojabot_simulation* a do něj vložíme dvě složky *Launch* a *Worlds* podobným způsobem, jak tomu bylo u kapitoly vojabot_description, jen místo závislosti na Urdf použijeme Roscpp. Jedná se o nejpoužívanější knihovnu, která implementuje jazyk C++ do ROSu a umožnuje programátorům rychlejší propojování Topics a Services (Rospy funguje stejně jenom pro jazyk Python).

```
$ catkin_create_pkg vojabot_simulation roscpp
```

Složka Launch nese soubor Simulation_mapping.launch, který slouží pro načtení parametrů robota do rviz vizualizéru, gazebo světa a umístnění robota do světa.

4.8.1 Svět v gazebo

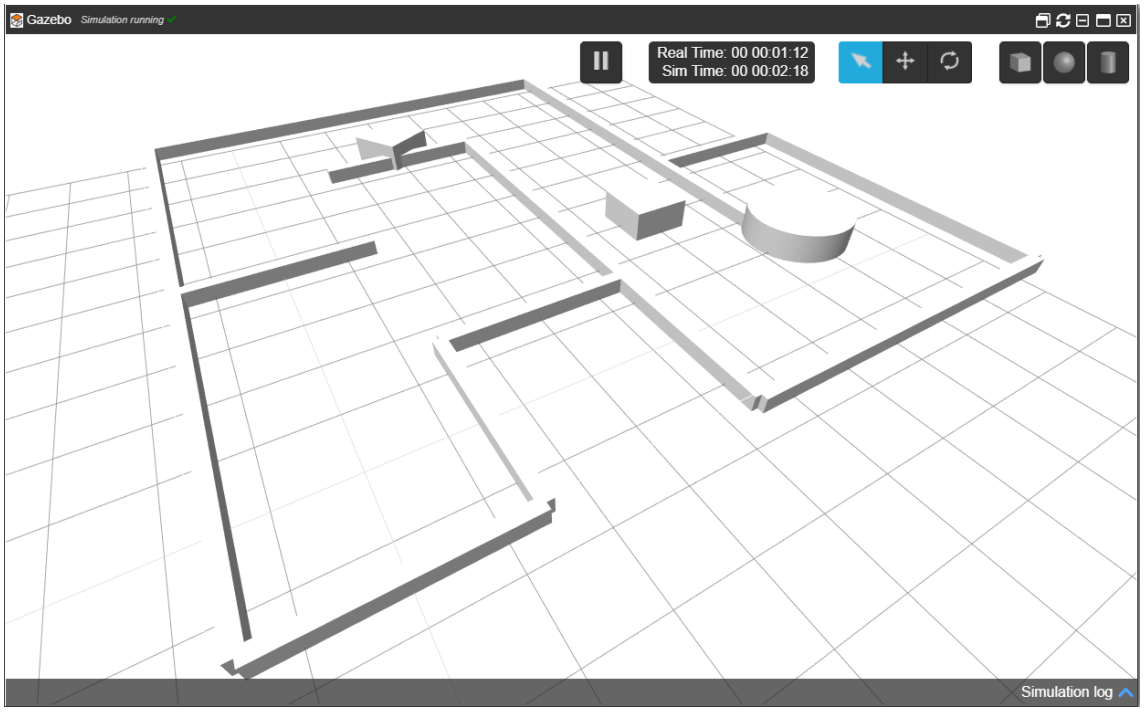
Modelování simulačního světa lze provést několika způsoby například pomocí Gazebo simulátoru, Google SketchUp Modeling software či SDF souboru. Pro účely této práce bylo použito modelování světa pomocí SDF souboru, který je založený na XML formátu. Vytvoření soubor světa se provede stejně jako u souborů URDF modelu robota, ale s .world koncovku. Jedná se zde o SDF soubor (The Spatial Data) což jsou geografická data, která specifikují svět. Tento soubor funguje stejně jako XML, ale navíc specifikuje vlastnosti potřebné pro simulační svět.[32, 33]

Vytvoření jednotlivých částí modelu světa funguje stejně jako u URDF částí robota, a to pomocí linků, které mají stejné vlastnosti, ale navíc mají přidány některé atributy potřebné pro simulaci a některé atributy nejsou potřeba. Jelikož tohle téma není hlavní myšlenkou téhle práce tak nebude rozebíráno podrobněji.[33]

```
<sdf version='1.6'>
  <world name='default'>
    <light name='sun' type='directional'>
          ...
    </light>
    <scene>
          ...
    </scene>
    <physics name='default_physics' default='0' type='ode'>
          ...
    </physics>
    <model name='ground'>
      <static>1</static>
      <link name='link'>
        <collision name='collision'>
              ...
        </collision>
        <visual name='visual'>
              ...
        </visual>
        <velocity_decay>
              ...
        </velocity_decay>
        <self_collide>0</self_collide>
        <kinematic>0</kinematic>
        <gravity>1</gravity>
        <pose frame=''>0 0 0 0 0 0</pose>
        <velocity>0 0 0 0 0 0</velocity>
        <acceleration>0 0 0 0 0 0</acceleration>
        <wrench>0 0 0 0 0 0</wrench>
      </link>
    </model>
  </world>
</sdf>
```

model.world

Tato složka nese simulační model světa vytvořený pomocí SDF pro testování modelu robota.



Obrázek 4.8: Gazebo svět model.world

4.9 Sestavení pracovního prostředí

Po vytvoření package vojabot_description a vojabot_simulation je potřeba sestavit pracovní prostředí, a to pomocí příkazů:[34]

```
$ cd/simulation_ws/
$ catki_make
```

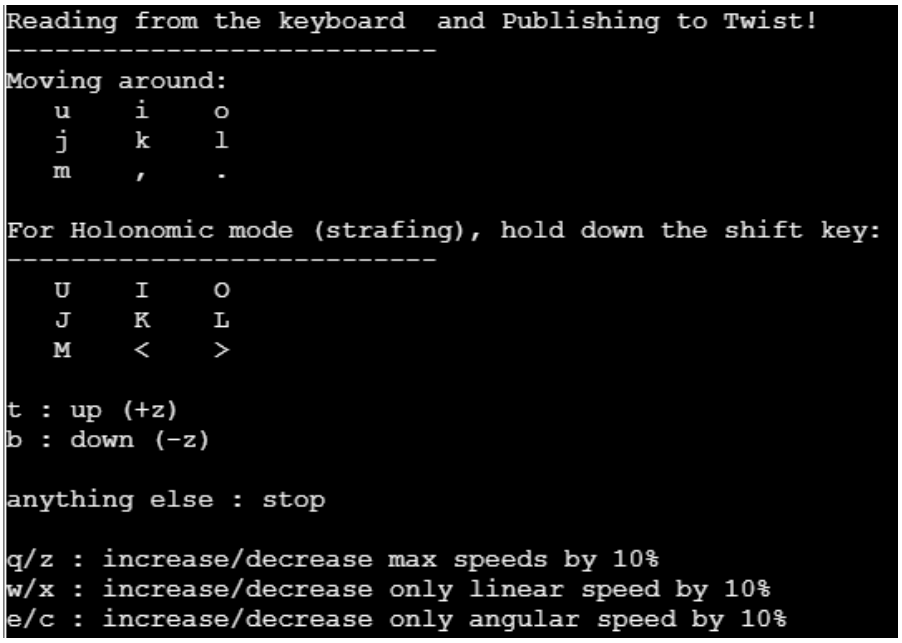
Robot je tedy nadefinovaný, simulační prostředí a načítací soubor (simulation_mapping.launch) také. Pro ověření funkčnosti spustíme simulaci pomocí karty Simulations a vybráním načítacího souboru *Simulation_mapping.launch* a kliknutím na tlačítko *Launch*, nebo otevřením nástroje Gazebo a příkazem:[35]

```
$ roslaunch vojabot_simulation simulation_mapping.launch
```

Nástroj roslaunch slouží pro snadné spouštění více nodů současně pomocí cesty a názvu souboru (Název.launch).

4.10 Testování simulačního prostředí

Po spuštění gazebo simulátoru můžete otestovat robota pomocí Teleop_twist_keyboard package, který je již importovaný v načítací složce. Funkci Teleop_twist_keyboard package vystihuje obrázek 4.9.[36]



Obrázek 4.9: Shell teleop screenshot

Pokud by teleop_twist_keyboard package nebyl zahrnut v načítací složce lze jej spustit pomocí příkazu:

```
$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

4.11 Package vojabot (definování funkcí robota)

Tento package obsahuje informace týkající se mapování prostředí a navigace robota v prostředí. Package je vytvořen stejným způsobem jako Vojabot_simulation, s tím rozdílem, že umístnění je v *Catkin_ws/src* (které slouží pro definování funkcí robota) pomocí příkazů:

```
$ cd catkin_ws/src/
$ catkin_create_pkg vojabot_simulation roscpp
```

Do tohoto package byli poté vloženy složky *Config*, *Launch* a *Maps* způsobem popsaným u Vojabot_description. Do složky *Config* je vložen soubor

Global_costmap_params.yaml, který osahuje parametry pro pohyb robota. Složka Launch obsahuje načítací soubory pro mapování světa a navigování robota.

move_base

Package Move_base je potřebnou částí pro navigaci robota. Umožnuje vytvořit mapu a přimět robota, aby se podle ní pohyboval a zároveň vyhýbal neznámým překážkám. To obsahuje soubor *Move_base_only.launch*.[37]

Mapování světa

Mapování světa je zprostředkováno pomocí package Gmapping, který pomocí laserového scaneru (RPLIDAR) sbírá data prostředí, kterým robot projel, a poté pomocí Slam_gmapping z nich vytváří 2D mapu světa. Tato mapa byla poté uložena do složky *Maps*. Samotné mapování obsahuje soubor *Gmapping_only.launch*.[38]

Amcl navigování vojabota

Navigace robota probíhá pomocí package AMCL (Adaptive Monte Carlo Localization), což je pravděpodobnostní lokalizační systém pro robota pohybujícího se ve 2D prostředí. Používá adaptivní Monte Carlo lokalizační přístup, který pomocí částicového filtru sleduje pozici robota na známé mapě. K tomu je použita uložená mapa z Gmapping. Samotnou navigaci robota obsahuje soubor *Amcl_only.launch*.[39] Po vytvoření package vojabot je potřeba sestavit pracovní prostředí, a to pomocí příkazů:

```
$ cd/catkin_ws/
$ catki_make
```

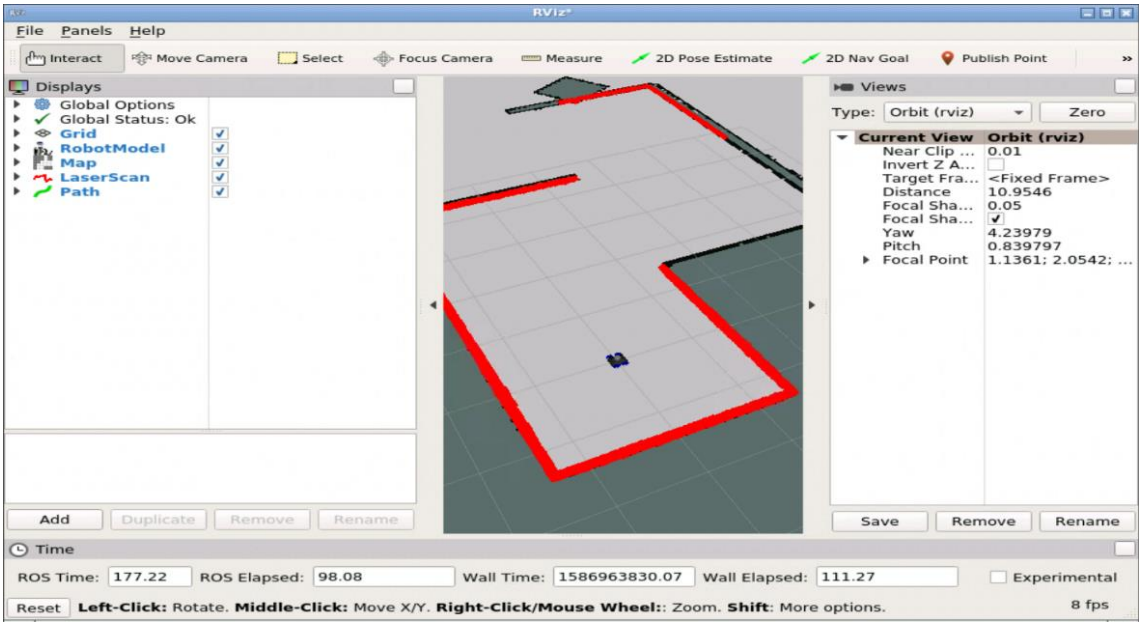
Vše je tedy připraveno pro testování robota.

5 TESTOVÁNÍ ROBOTA VOJABOT

Testování proběhlo pomocí nástrojů Gazebo simulace a Rviz vizualizéru. Byli použity dvě testované funkce, mapování světa a autonomní navigace.

5.1 Seznámení s Rviz prostředím

Po spuštění Rviz vizualizéru se zobrazí jeho hlavní okno, které je rozděleno do tří částí a horní lišty. V levé časti se nachází Display panel, který umožnuje zobrazovat prvky a témata potřebné pro práci s robotem. Pro přidávání prvků stačí kliknout tlačítko Add a vybrat si ze seznamu. Pro potřeby této práce byly použity prvky Map, Grid, Laser scan a Robot model. V pravé části se nachází panel Views, který slouží pro typ pohledu a prvku, na které se má zaměřit (bylo použito defaultní nastavení). Uprostřed se nachází samotná vizualizace robota s námi zvolenými prvky z části Display. Horní lišta obsahuje nástroje pro upravení pohledu, viz obrázek níže.[40]



Obrázek 5.1: Rviz screenshot

5.2 Mapování

Pro mapování je zapotřebí spustit simulaci dále spustit samotné mapování a Rviz pro zjištění, zdali mapování funguje. Každý příkaz musí být spouštěn v novém příkazovém řádku.[41] Pro spuštění simulace je otevřen nástroj Gazebo a použit příkaz:

```
$ roslaunch vojabot_simulation simulation_mapping.launch
```

Mapování se spustí pomocí příkazu:

```
$ roslaunch vojabot running_gmapping.launch
```

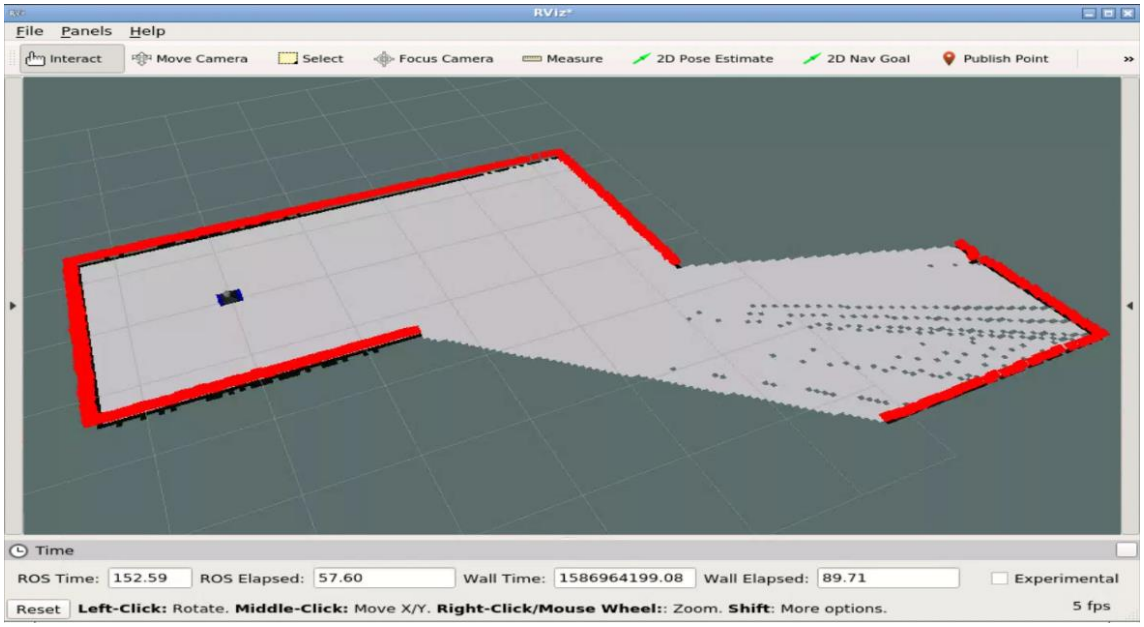
Spuštění Rvizu se provede otevřením nástroje Graphical tool a zadáním příkazu:

```
$ rviz
```

Poté byla pomocí aplikace Teleop_twist_keyboard mapa projeta robotem pro naskenování. Ověřit, že robot opravdu skenuje prostor lze ve Rvizu. Aby naskenovaná data nepřišla vniveč, byla pomocí následujících příkazů uložena do souboru *Maps*.[41]

```
$ cd catkin_ws/src/vojabot/maps/
$ rosrun map_server map_saver -f rosbot_map
```

Pro zadání těchto příkazů musí být otevřen nový příkazový řádek, aby nebyly vypnuty již běžící programy.



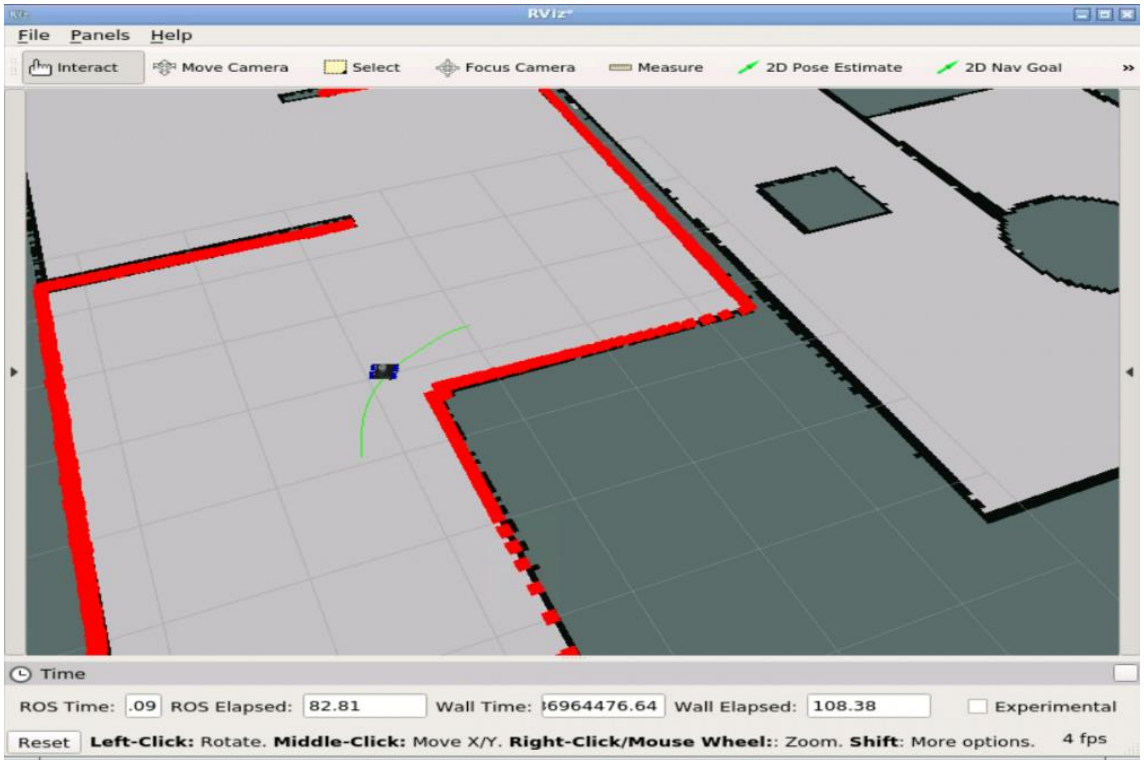
Obrázek 5.2: Mapování Rviz screenshot

5.3 Navigování

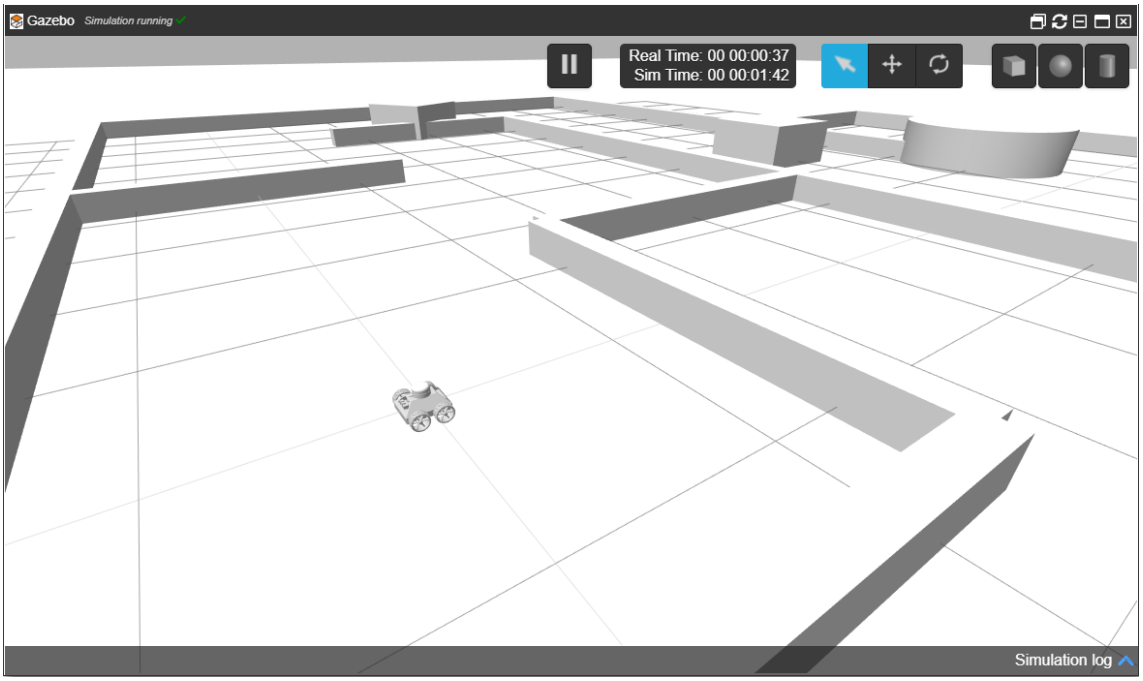
Pro navigování je zapotřebí spustit simulaci, dále navigační soubor a Rviz. Spuštění simulace a Rvizu se provede stejně jako v kapitole Mapování.[41] Navigační soubor se spustí pomocí příkazu:

```
$ roslaunch vojabot nav_vojabot.launch
```

Pro určení místa, kam má robot dojet, se použije Rviz, a to pomocí nástroje z horní lišty *2D Nav Goal* a kliknutím na určené místo. Průběh navigace robota je přiložen jako nahrávka.



Obrázek 5.3: Navigování Rviz screenshot



Obrázek 5.4: Gazebo screenshot

6 ZÁVĚR

Cílem této práce bylo seznámení se s frameworkem ROS, vytvoření vlastního modelu robota, testování vytvořeného modelu robota na definovaných úkolech a pomocí získaných informací sepsat návod, podle něhož by bylo možné vytvořit model robota.

V kapitole 2 byl představen framework ROS a jeho funkce. Dále zde byla představena platforma ROS Development Studio.

Kapitola 3 byla věnována stručné rešerši základních typů pohonů pro ROS. Jelikož tato kapitola nebyla primárním cílem této práce, nebyla podrobněji rozebírána.

V praktické kapitole 4 bylo rozepsáno vytvoření projektu, seznámení s prostředím ROS Development Studio platformy, vytvoření modelu robota a kompletní popis všech oddílů projektu.

Praktická kapitola 5 byla věnována samotnému testování vytvořeného robota. Dále zde byl představen Rviz vizualizér.

Jako první aplikaci Vojabota bylo vybráno mapování prostředí pomocí Package gmapping (running_gmapping.launch) a ručního ovládání pomocí teleop popsané v kapitole 4.10. Tato aplikace proběhla bez větších problémů a mapu se povedlo téměř bezchybně naskenovat. Scaneru dělaly problémy pouze hrany mezi stěnami jež svírali velmi ostrý úhel.

Jako druhá aplikace vojabota bylo zvoleno autonomní navigování pomocí package amcl (nav_vojabot.launch). Zde došlo k problému s výpočetním výkonem platformy ROSDS, která nezvládala plánování cesty pro delší vzdálenosti, kdy čas výpočtu plánované trasy mnohonásobně převyšoval čas pohybu robota. A by byla zachována plynulost a funkčnost chodu robota, byla plánovaná cesta zkrácena na poloměr jednoho metru kolem robota (testované poloměry byli v rozmezí od 10 m až 0.5 m). Dále zde byli problémy při autonomním řízení robota kdy robot špatně korigoval natáčení kolem své osy, což bylo způsobeno špatným nastavení parametrů drsností kol.

Obě dvě aplikace byli zdárně splněny. Záznam mapování i navigace jsou přiloženy v příloze.

7 SEZNAM POUŽITÉ LITERATURY

[1] ROS History. ROS [online]. [cit. 2020-06-12]. Dostupné z: https://www.ros.org/history/

[2] ROS. The Construct [online]. [cit. 2020-06-13]. Dostupné z: https://www.theconstructsim.com/what-is-ros/

[3] ROS Definitions. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: https://www.ros.org/

[4] ROS Introduction. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/ROS/Introduction

[5] What, Why and How of ROS [online]. [cit. 2020-06-13]. Dostupné z: https://towardsdatascience.com/what-why-and-how-of-ros-b2f5ea8be0f3

[6] ROS Distributions. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Distributions

[7] ROS Packages. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Packages

[8] ROS Concepts. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/ROS/Concepts

[9] ROS Nody. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Nodes

[10] ROS Topics. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Topics

[11] ROS Concept. In: Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/ROS/Concepts?action=AttachFile&do=view&target=ROS_basic_concepts.png

[12] ROS Services. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Services

[13] ROS Parameter Server. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Parameter%20Server

[14] ROSject. The Robot Report [online]. [cit. 2020-06-13]. Dostupné z: https://www.therobotreport.com/speed-robotics-development-cloud-ros/

[15] ROS Enviroment. The Construct [online]. [cit. 2020-06-13]. Dostupné z: https://www.theconstructsim.com/the-ros-development-studio-by-the-construct/

[16] The Construct. The Construct [online]. [cit. 2020-06-13]. Dostupné z: https://www.theconstructsim.com/

[17] ROS Drivers. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/Motor%20Controller%20Drivers

[18] Elektrické Pohony. KSR [online]. [cit. 2020-06-13]. Dostupné z: http://www.ksr.tul.cz/ksr/podklady/ARS-5.Elektropohony-5.pdf

[19] Přehled Elektromotorů. Elektromotory [online]. [cit. 2020-06-13]. Dostupné z: https://www.spslan.cz/files/1cc2eb91316dd52bf50dd77e977caa60.pdf

[20] Basics of dc motors. Test and Measurement Tips [online]. [cit. 2020-06-13]. Dostupné z: https://www.testandmeasurementtips.com/baasics-of-dc-motors/

[21] Asynchronní Stroje. VUT [online]. [cit. 2020-06-13]. Dostupné z: https://moodle.vutbr.cz/pluginfile.php/179983/mod_resource/content/0/Prezentace.pdf

[22] Asynchronní Motor. Digitální Knihovna [online]. [cit. 2020-06-13]. Dostupné z: https://dspace5.zcu.cz/bitstream/11025/10501/1/BP_Jan_Pikous.pdf

[23] Krokové motory. ELUC [online]. [cit. 2020-06-13]. Dostupné z: https://eluc.kr-olomoucky.cz/verejne/lekce/809

[24] ROS Workspace. The Construct [online]. [cit. 2020-06-13]. Dostupné z: https://www.theconstructsim.com/rosds-docs-rosject-workspaces/

[25] Gazebo Simutalion. Gazebo [online]. [cit. 2020-06-13]. Dostupné z: http://gazebosim.org/

[26] ROS URDF. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/urdf

[27] URDF Model. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/urdf/XML/model

[28] URDF Link. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/urdf/XML/link

[29] URDF Joint. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/urdf/XML/joint

[30] Safety Limits. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/pr2_controller_manager/safety_limits

[31] URDF Xacro. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/xacro

[32] Gazebo World. *Robot Operating System* [online]. [cit. 2020-06-13]. Dostupné z: http://gazebosim.org/tutorials?tut=ros_roslaunch#CreatingaCustomWorldFile

[33] Simulation World. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: https://wiki.ros.org/cob_gazebo_worlds/Tutorials/Create%20your%20own%20world

[34] Commands. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/catkin/commands/catkin_make

[35] Commands. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/roslaunch

[36] ROS Teleop. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/teleop_twist_keyboard

[37] ROS Move Structure. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/move_base

[38] ROS Gmapping. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/gmapping

[39] ROS Amcl. Robot Operating System [online]. [cit. 2020-06-13]. Dostupné z: http://wiki.ros.org/amcl

[40] RVIZ. Rviz/UserGuide [online]. [cit. 2020-06-13]. Dostupné z: http://library.isr.ist.utl.pt/docs/roswiki/rviz(2f)UserGuide.html

[41] ROSbot. The Construct [online]. [cit. 2020-06-13]. Dostupné z: https://www.theconstructsim.com/wp-content/uploads/2020/02/ros_developers_live_class_n80.pdf

8 SEZNAM ZKRATEK A OBRÁZKŮ

Zkratky:

ROS	Robot Operating System
ROSDS	ROS Development Studio
URDF	Unified Robot Description File
OSRF	Open Source Robotic Foundation
API	Application Programming Interface
P2P	Peer-to-peer komunikace
RPC	Remote procedure call
IDE	Itegrated development environment
XML	Extensible Markup Language
SDF	Standard Database Format
CAD	Computer Aided Design
STL	Stereolithography
AMCL	Adaptive Monte Carlo Localization

Obrázky:

9 SEZNAM PŘÍLOH

Příloha 1 Vojabot : Kompletní projekt se všemi programy.

Příloha 2 Autonomní_navigace_a_mapování : Videa z průběhu autonomní navigace a mapování