FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Název:	Simulátor kolového robota
Student:	Tomáš Balihar
Vedoucí:	Ing. Miroslav Skrbek, Ph.D.
Studijní program:	Informatika
Studijní obor:	Počítačové inženýrství
Katedra:	Katedra číslicového návrhu
Platnost zadání:	Do konce letního semestru 2018/19

Pokyny pro vypracování

Proveďte rešerši open source simulátorů robotů a na základě rešerše vyberte nejvhodnější simulátor pro vytvoření modelu existujícího kolového robota. Model robota by měl co nejvíce odpovídat realitě, jak co do rozměrů, tak dynamických parametrů. Existující programové vybavení robota upravte, případně využijte nebo nahraďte jeho části tak, aby byla na úrovni aplikačního rozhraní zajištěna kompatibilita mezi modelem a reálným robotem. Při výběru simulačního prostředí, knihoven a nástrojů preferujte operační systém Linux. Funkčnost modelu vyzkoušejte v jednoduché demo aplikaci. Rozsah práce upřesněte po dohodě s vedoucím práce.

Seznam odborné literatury

Dodá vedoucí práce.

doc. Ing. Hana Kubátová, CSc.
vedoucí katedry

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
děkan

V Praze dne 1. prosince 2017

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Bakalářská práce

Simulátor kolového robota

Tomáš Balihar

Katedra Číslicového návrhu
Vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

13. května 2018

Poděkování

Děkuji vedoucímu práce, Ing. Miroslavu Skrbkovi, Ph.D. za všechnu poskytnutou pomoc při analýze robota i tvorbě práce samotné.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisů. V souladu s ust. § 46 odst. 6 tohoto zákona tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 13. května 2018 .....

Odkaz na tuto práci

Balihar, Tomáš. *Simulátor kolového robota.* Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2018.

Abstrakt

Tato bakalářská práce se zabývá vytvořením modelu kolového robota, pro účely počítačové simulace. Model v simulátoru je použit pro testování, bez potřeby reálného robota. K realizaci celé simulace je použit simulátor jménem Gazebo, který je společně s operačním systémem Linux nutností pro spuštění celé simulace. Dále je pro realizaci simulace využit framework Robot Operating System, který vytváří komunikační vrstvu mezi modelem a softwarem robota.

Díky této práci se urychlí vývoj pro robota a zároveň může práce posloužit i jako návod pro převedení reálného robota do simulátoru.

Klíčová slova Kolový robot, simulace robota, model robota, Robot FIT, Robot Operating System, Gazebo

Abstract

This bachelor thesis addresses creation of a wheeled robot model for the purpose of computer simulation. The model in the simulator is used for testing without the need of the real robot. Simulator named Gazebo is used to implement this simulation and together with Linux operating system is essential for it. Also Robot Operating System framework is used to create communication layer between model and the software of real robot.

This thesis will speed up development for the robot and it may also serve as a guide for creating model of any real robot.

Keywords Wheeled robot, robot simulation, robot model, Robot FIT, Robot Operating System, Gazebo

Obsah

Úvod **1**

1 Cíl práce **3**

2 Analýza **5**
2.1 Robot FIT 5
2.1.1 Software 6
2.2 Robot Operating System 8
2.3 Rešerše simulátorů 9
2.3.1 OpenRAVE 9
2.3.2 MORSE 9
2.3.3 ARGoS 10
2.3.4 V-REP 10
2.3.5 Gazebo 11
2.3.6 Vyhodnocení 12

3 Návrh **13**
3.1 Model robota 14
3.2 Ovládací moduly 17
3.3 Úprava programového vybavení 19

4 Realizace **21**
4.1 Model robota 21
4.1.1 Vizuální stránka 21
4.1.2 Kolizní model 23
4.1.3 Fyzikální vlastnosti 24
4.1.4 Výsledek 25
4.2 Rozhýbání modelu 26
4.2.1 Modul `libgazebo_ros_control` 26
4.2.2 Moduly simulátoru Gazebo 27

4.3 Napojení na software 29
4.3.1 Program `firmware-board` 29
4.3.2 Knihovna `libstereo` 31
4.3.3 Program `robot-server` 31

Závěr **33**

Literatura **35**

A Seznam použitých zkratek **37**

B Návod pro spuštění **39**
B.1 Instalace 39
B.2 Spuštění 40

C URDF vizualizace **41**

D Obsah přiloženého CD **43**

Seznam obrázků

0.1 Fotografie použitého robota[1] 1

2.1 Popis komponent robota[1] 5
2.2 Schéma ovládacích prvků robota 6

3.1 Schéma programů a knihoven na robotovi 14
3.2 Model vytvořený z kódu výše 17
3.3 Návrh propojení modelu s existujícím softwarem 19

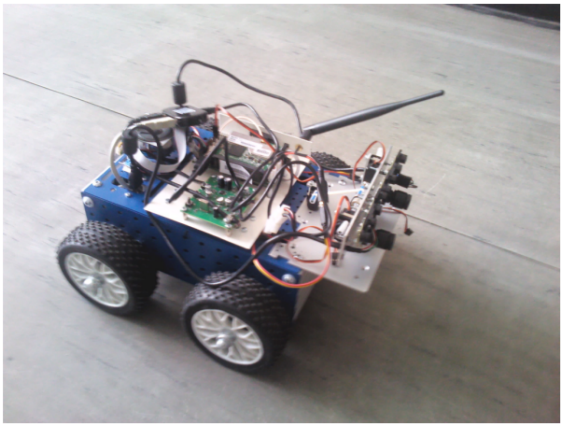
4.1 Vizualizace modelu v Rviz 23
4.2 Model robota v simulátoru Gazebo 25
4.3 Diagram všech uzlů a jejich komunikace 32

Úvod

Robotika je v dnešní době velmi diskutované téma, a to hlavně díky zvýšené medializaci a filmové tvorbě. I když je tento obor v povědomí lidí spíše futuristický, už nyní se rychle rozšiřuje a napomáhá lidem v každodenním životě. Zájem o robotiku každým dnem narůstá a po celém světě se stále objevují nové technologie.

Robot FIT, kterého můžete vidět na obrázku 0.1, je robotická platforma, vytvořená na Fakultě informačních technologií ČVUT v Praze. Na robotovi pracovalo mnoho studentů již přede mnou a já navazuji na jejich práce, které formovali robota do podoby jakou má nyní.

Tato práce se zabývá analýzou systému robota a simulačních programů, pomocí kterých by bylo možné napodobit jeho chování v počítači. Poté na tyto části navazuje implementací tohoto robota do vybraného simulátoru. V implementační části je pro propojení robota se simulátorem použit framework Robot Operating System, díky kterému lze lehce přesunout robota i na jiné simulátory, bez nutnosti velkých úprav.



Obrázek 0.1: Fotografie použitého robota[1]

Výsledek práce pomůže v dalším vývoji tohoto robota tím, že zjednoduší přístup dalším zájemcům k jeho programování. Studenti budou moci otestovat své programy pro robota bez toho, aby ho mohli nějakou chybou poškodit a zastavit vývoj ostatních. Navíc nebude třeba mít reálného robota neustále u sebe a studenti budou moci tvořit i doma.

Zároveň tato práce poslouží i jako návod, podle kterého bude moci čtenář vytvořit model vlastního robota a ten v simulátoru rozhýbat. Takové návody jsou na internetu často zastaralé, a proto doufám, že tato práce někomu pomůže.

Kapitola 1

Cíl práce

Cíle rešeršní části bakalářské práce je možné shrnout do těchto bodů:

- Provést rešerši vhodných simulátorů a vybrat nejvhodnější z nich pro kolového robota.
- Prozkoumat systém robota, a to jak po stránce hardwaru, tak i jeho programového vybavení a knihoven.
- Zjistit nejlepší způsob napojení existujícího softwaru robota na jeho model v simulátoru.

Cíle implementační části práce je možné shrnout takto:

- Vytvořit 3D model robota s fyzickými parametry podobnými reálnému robotovi.
- Vytvořit ovládací rozhraní pro model robota.
- Upravit existující program robota pro napojení na model v simulátoru.
- Vytvoření návodu pro spuštění simulace ostatním studentům.

Výstup implementační části musí být spustitelný na počítačích s operačním systémem Linux, a zároveň musí podporovat jednoduchý přenos kódu ze simulovaného modelu robota na robota reálného. Také je třeba dělat co nejmenší zásahy do existujícího vybavení, jelikož na robotovi pracuje najednou více studentů.

Kapitola 2

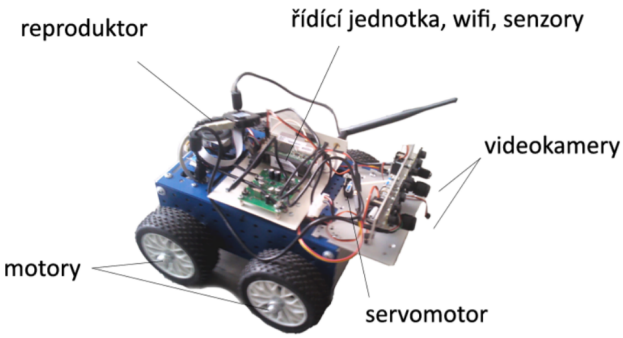
Analýza

V této kapitole se věnuji průzkumu existujícího softwaru a hardwaru robota, a poté rešerši a výběru vhodného simulačního programu, pro vytvoření modelu tohoto robota. Také zde popisuji funkci a použití frameworku jménem Robot Operating System, pro účely tvorby ovládacích programů robotů.

2.1 Robot FIT

Robot FIT je robotická platforma, vytvořená pro akademické účely na Fakultě informačních technologií ČVUT v Praze. Na robotovi vzniklo již několik bakalářských prací, které ho dotvářeli do podoby, kterou má nyní. Jedná se o čtyřkolového robota osazeného dvěma kamerami a několika různými senzory. Jeho popis můžete vidět na obrázku 2.1.

Robot je osazen deskou RoBoard RB-110 s operačním systémem Gentoo Linux, na které běží aplikace `robot-server`, vytvořená jako bakalářská práce pro vysokoúrovňové řízení robota. Také zde funguje knihovna `libstereo`, sta-



Obrázek 2.1: Popis komponent robota[1]

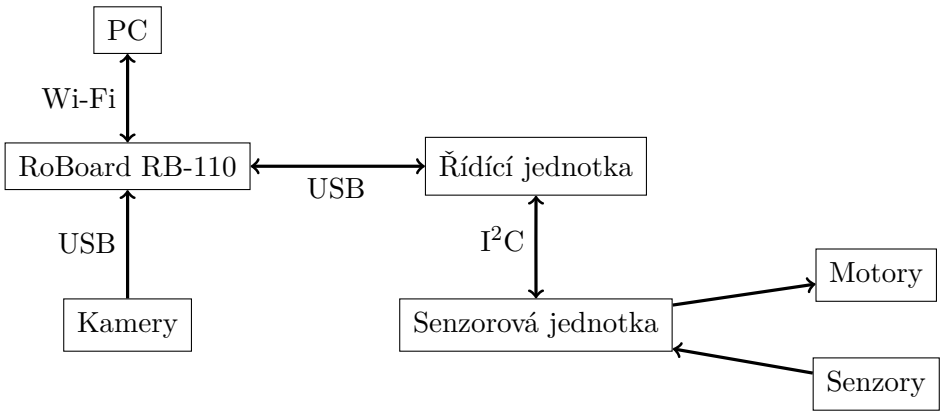
rající se o snímání obrazu z kamer a jeho zpracování, a několik dalších knihoven pro práci se zvukem. Dále je na robotovi řídící deska s procesorem dsPIC, která je propojena s deskou RoBoard RB-110 pomocí USB a zajišťuje samotné ovládání robota. Propojení součástí na robotovi lze vidět na obrázku 2.2. [2]

Řídící deska robota se stará o tyto prvky:

Pohyb robota je řízen dvěma motory na každé straně, které mají propojené přívodní vodiče. Zatáčení je dosaženo pomocí diferenciálního řízení, neboli rozdílnými rychlostmi na levé a pravé straně robota. Ovládání rychlosti motorů je řízeno pomocí PWM jednotky. [2]

Senzory jsou snímány za pomoci senzorové desky, připojené k řídící desce pomocí sběrnice I^2C. Senzory napojené na tuto desku jsou:

- IR senzory vzdálenosti na přední a zadní straně robota, pro kontrolu vzdálenosti k nejbližší překážce,
- 3osý gyroskop, pro zjištění náklonu robota,
- 3osý akcelerometr, pro zjištění akceleračních koeficientů,
- inkrementální senzory polohy na kolech, pro výpočet ujeté vzdálenosti a rychlosti jízdy,
- senzor měření proudu na baterii. [2]



Obrázek 2.2: Schéma ovládacích prvků robota

2.1.1 Software

Software robota je rozdělen do několika částí. Jak již bylo popsáno výše, na robotovi je více ovládacích desek. Deska RoBoard, na které běží vysokoúrovňové ovládání robota, komunikuje s počítačem pomocí Wi-Fi a přijímá informace

o tom, co má robot dělat. Na desce je spuštěn `robot-server`, který funguje jako hlavní ovládání robota. [3]

Aplikace `robot-server` v sobě obsahuje několik podpůrných knihoven na práci s kamerami a zvukem. Pro účely simulace je důležitá knihovna `libstereo`, která obstarává komunikaci s kamerami na přední části robota. Tato knihovna zařizuje vše od pořízení snímků z kamer až po vytvoření hloubkové mapy z těchto fotografií. [4]

Dalšími knihovnami pracujícími na hlavní desce jsou `libsoundloc`, která pracuje s mikrofony vpředu a lokalizuje původ zvuků v prostoru, a knihovna `libvoicerecog`, která rozpoznává příkazy z lidské řeči. Těmito se však v simulaci zabývat nebudu, jelikož simulátory nepodporují přímou práci se zvukem. Bylo by tedy potřeba vytvořit složitý modul, kterým by práce přesáhla daný rozsah. [3]

S deskou RoBoard a aplikací `robot-server` komunikuje i řídící deska robota pomocí USB. Mezi deskami komunikují aplikace pomocí příkazů zakódovaných do písmen. Na řídící desce běží aplikace `firmware-board`, která přijímá signály od aplikace `robot-server` a podle nich řídí motory, nebo odesílá zpět data z obstarávaných senzorů.

Deska RoBoard komunikuje pomocí sítě s počítačem, který se připojuje jako klient k aplikaci `robot-server`. Na počítači je poté možné spouštět programy pro ovládání robota za pomoci příkazů, posílaných serveru. Dále také existuje API pro ovládání robota, na kterém je postavena aplikace pro řízení robota pomocí mobilního telefonu se systémem Android a nyní vzniká webové rozhraní pro ovládání robota. [5]

2.2 Robot Operating System

Robot Operating System, zkráceně ROS, je framework pro tvorbu ovládacích programů pro roboty. Jak může z názvu vyplývat, ROS není operační systém, ale bere si z nich příklad s hardwarovou abstrakcí. ROS je vlastně jen pomocník, který je spuštěn na ovládací desce robota a pomáhá vytvářet abstrakční vrstvu mezi hardwarem reálného robota a jeho řídícím systémem. Díky tomu lze přenést ovládání na jiný hardware bez velkých obtíží. [6]

ROS je možné také využít pro simulaci. Pomocí abstrakce na úrovni hardwaru je možné vzít řízení robota a místo napojení na hardware reálného robota jej napojit na model robota v simulátoru, který podporuje spolupráci s rozhraním ROS. Toto může být velice užitečné u některých simulátorů, kde může být jednodušší a univerzálnější přidat do existujícího kódu komunikaci přes ROS, nežli specifickou komunikaci simulátoru. [6]

ROS funguje na systému uzlů (*Nodes*), propojených pomocí témat (*Topics*). Uzel je základní stavební prvek frameworku ROS. Jako uzel lze označit jakýkoliv proces komunikující pomocí frameworku ROS. Uzly mezi sebou komunikují pomocí již zmíněných témat. Téma je komunikační kanál, přes který lze anonymně posílat a přijímat zprávy mezi uzly, znající dané téma. Přes každé téma lze posílat a přijímat pouze jednu danou datovou strukturu. Na jedno téma může najednou posílat nekonečně uzlů a stejné je to i s přijímáním dat. [6]

Software pro ROS je možné psát v jazyce C++ pomocí knihovny roscpp, nebo v jazyce Python za pomoci knihovny rospy. Uzly se strukturují do balíčků, které jsou spravovány pomocí vestavěných nástrojů jako například `catkin`, který pomáhá se správným sestavováním balíčků. ROS je ovládán primárně pomocí příkazové řádky, ale obsahuje několik grafických nástrojů pro zlehčení práce s frameworkem. [6]

ROS je velice využívaný nástroj pro tvorbu a ovládání robotů a díky velké a aktivní komunitě usnadňuje práci lidem po celém světě. Jeho dokumentace je podrobně sepsaná a obsahuje mnoho různých návodů, tutoriálů a již vytvořených robotů, ze kterých mohou začínající uživatelé čerpat. [6]

Pro simulaci se systémem ROS je třeba umět použít i formáty XML a YAML. První zmiňovaný je třeba pro popis modelu robota v simulátoru, kde se využívá nástavba tohoto formátu zvaná URDF, neboli Unified Robot Description Format. YAML je poté potřebný pro popis různých parametrů uzlů. Další využití jazyka XML je ve vytváření spouštěcích skriptů, kde se popisuje jaké uzly se mají spustit, s jakými parametry a jak je propojit. [6]

2.3 Rešerše simulátorů

Tato část práce se zabývá porovnáním různých open source simulátorů, použitelných pro tento účel. Ke každému zde vypíši jeho hlavní rysy, společně s jeho výhodami a nevýhodami. Nakonec provedu vyhodnocení a výběr nejvhodnějšího kandidáta.

2.3.1 OpenRAVE

OpenRAVE je projekt udržovaný v Jouhou System Kougaku Laboratory na Univerzitě v Tokiu. Celý simulátor je psaný v jazyce C++, ale ovládání robotů může být psáno i v jazyce Python. Simulátor je zaměřen hlavně na průmyslové robotické paže a jejich vývoj. [7]

Simulátor využívá pro tvorbu modelů robotů vlastní úpravu jazyka XML, ale také podporuje načtení modelů ve formátu COLLADA, což je velice známý a využívaný formát pro přenos 3D modelů všeho druhu mezi programy, vyvinutý původně firmou Sony Computer Entertainment. Tento formát je standart podporovaný každým programem pro tvorbu 3D modelů. [7]

Vlastní ovládání modelů robotů je řízeno pomocí modulů, které vytváří uživatel a napojuje každý modul na nějakou část hlavní simulace. Moduly mohou ovládat jak roboty v simulaci, tak i simulovaný svět okolo nich. [7]

Jedním z největších problémů simulátoru OpenRAVE je jeho dokumentace, která je psána oproti některým simulátorům velice zmatečně a chybí v ní důležité údaje. To je možná důvodem, proč je komunita uživatelů u OpenRAVE tak malá, což také nepřispívá k jednoduchosti učení práce s ním. [7]

2.3.2 MORSE

MORSE je simulátor vyvíjený pro využití v akademickém prostředí. Simulátor je celý napsaný v jazyce Python a v tomto jazyce se i vytváří kód pro ovládání robota. V mém případě je toto problém, jelikož veškerý kód pro Robota FIT je psán v jazyce C++. Práce v tomto simulátoru by tedy znamenala vytvořit více programů a ty nějakým způsobem propojit. [8]

Modely robotů jsou v MORSE vytvářeny v 3D modelovacím programu Bledner, který se zároveň stará o vykreslování prostředí a objektů v tomto simulátoru. Blender je open source projekt pro modelování 3D objektů. Nejen díky tomu, že je zdarma je velice používaný a uznávaný. [8]

Ovládání modelů robotů je prováděno skripty v jazyce Python, kterými lze ovládat jak model, tak i svět okolo něho. Simulátor samotný je ovládaný z příkazové řádky a prostředí do kterého se robot simuluje je popisováno také v jazyce Python. [8]

V simulátoru jsou již vytvořené moduly pro kamery, senzory, ovládání kol a další často využívané doplňky. Toto je velké plus, které urychlí tvorbu

simulace. Nevýhodou MORSE je však malá komunita a řešení problémů při vývoji je proto složitější. [8]

Dokumentace MORSE je velice dobrá a na jejich stránkách se dá najít mnoho návodů, které usnadní učení a vývoj v tomto simulátoru. MORSE navíc podporuje několik druhů frameworků pro roboty, jako například Robot Operating System. O tom více v části 2.2. [8]

2.3.3 ARGoS

ARGoS je projekt zaměřený na vytvoření modulárního simulátoru pro akademické účely. Jeho části jako fyzikální, či vizuální engine je možné nahradit jiným podle přání uživatele. Simulátor je zaměřen na simulaci spolupráce více robotů v prostoru, což je funkce, kterou většina ostatních simulátorů vynechává. V mém případě tuto funkci nevyužiji, ale ARGoS je vhodný i na simulaci jednoho robota. [9]

Konfigurace celé simulace se vytváří jako XML soubor, kde se definuje svět, roboti a jejich ovládání. Samotné ovládání je realizováno pomocí programů v jazyce C++, které se v konfiguraci přiřazují k robotům, či k prostředí ve kterém se pohybují. [9]

Tvorba modelu robota v simulátoru ARGoS je kvůli jeho modularitě mnohem složitější než u ostatních simulátorů. Model robota se popisuje třídami v jazyce C++ a je třeba vytvořit 3 různé třídy pro popis jednoho robota. Jedna třída definuje strukturu robota, další popisuje jeho vizuální stránku a poslední určuje jeho fyzikální vlastnosti. [9]

Dokumentace není zrovna silná stránka tohoto simulátoru a ani jeho komunita není veliká. Tento problém částečně vynahrazuje hlavní vývojář tohoto simulátoru, který je neustále aktivní na fóru a pomáhá každému s problémy na které narazí. Další zdroje, ze kterých se dá čerpat jsou tutoriály a ukázkové projekty od vývojářů. Navíc existuje několik vědeckých publikací o tomto simulátoru napsaných zakladatelem celého projektu. [9]

2.3.4 V-REP

V-REP je uznávaný simulátor, který přešel na model open source v roce 2013. Simulátor je zdarma pouze pro akademické účely, jinak je placený. Jádro simulátoru je psáno v jazyce C++, ale nyní podporuje programování modulů i v dalších jazycích jako Python, Java, či skriptovací jazyk Lua. V-REP také podporuje hned několik fyzikálních jader, ze kterých si může uživatel vybrat ten nejvhodnější pro svoji potřebu. [10]

Model robota lze v tomto simulátoru vytvořit hned několika způsoby. Dokáže pracovat s formáty jako již zmiňovaný COLLADA, nebo také známé 3DS a STL, či formát pro Robot Operating System jménem URDF, viz 2.2. Simulátor také obsahuje vestavěný editor modelů, díky kterému lze upravit model přímo v simulaci. V-REP má také již předdefinované části jako kamery, sen-

zory, motory a další podobné, které lze jednoduše přidat k modelu a není třeba je vytvářet od začátku. [10]

K ovládání modelů robotů i jejich okolí podporuje V-REP více možností přístupu. Mezi ně patří například tvorba ovládacích modulů v jazyce C++, ovládacích skriptů v jazyce LUA, použití vnější API pro jeden ze 6 programovacích jazyků, či napojení na Robot Operating System, viz 2.2. Všechny tyto možnosti lze navíc volně kombinovat mezi sebou. Simulátor má zároveň zabudované často využívané algoritmy pro rychlejší tvorbu nového robota. [10]

V-REP má také dlouhou a podrobnou dokumentaci všech svých částí, takže učení se s ním je snadnější než s doposud zmiňovanými simulátory. Komunita je zde také veliká, což je vidět hlavně na oficiálním fóru simulátoru. Jediná nevýhoda tohoto simulátoru je velká náročnost na výkon počítače, což může být problém. [10]

2.3.5 Gazebo

Gazebo je simulátor vyvíjený společností Open Source Robotics Foundation, ta samá firma vyvíjí také robotický framework Robot Operating System, viz 2.2. Díky tomuto spojení je vzájemná integrace obou programů dobře propracovaná a použití ROS značně zjednodušuje práci se simulátorem. Gazebo je již od začátku vývoje v roce 2002 zcela zdarma. Původně vznikl jako studentská práce na univerzitě v Jižní Kalifornii. Simulátor je psán v jazyce C++. [11]

Pro vytvoření modelu robota v simulátoru Gazebo lze využít jeden z popisových formátů postavených na XML, nebo editor modelů přímo v grafickém rozhraní simulátoru. Z popisových formátů lze vybrat mezi formátem SDF, který byl vytvořen přímo pro Gazebo, nebo formátem URDF, který byl vytvořen pro kombinaci ROS a Gazebo. V simulátoru existuje také mnoho předdefinovaných součástí jako kamery, senzory, či ovládání motorů, které se při přidání automaticky napojí na ROS. [11]

Ovládání robotů v simulátoru je řízeno přímo pomocí modulů psaných v jazyce C++, nebo za pomoci ROS. Při použití ROS lze použít jak jazyk C++, tak i jazyk Python. Pro podrobnější popis jak funguje ROS viz sekce 2.2. Simulátor také obsahuje mnoho předpřipravených ovládacích modulů, které dokáží připojit části robota na systém ROS, například při vytvoření několika kol k ním lze pomocí pár řádků kódu přidat diferenciální řízení napojené na ROS. [11]

Gazebo je velice rozšířený simulátor a nejen díky tomu, že je od počátku své existence zdarma, je jeho komunita obrovská. Společně s kvalitní dokumentací, kterou má, je řešení problémů o dost jednodušší než u ostatních simulátorů. Avšak kvůli propojení s frameworkem ROS je potřeba studovat dvě různé dokumentace, které se občas obsahově překrývají, a občas trochu rozcházejí. Na takovéto kolize je ale vždy nějaká diskuze na fóru, která navede zmatené uživatele správným směrem. [11]

2.3.6 Vyhodnocení

Při vyhodnocování simulátorů je třeba brát v potaz hlavně tyto parametry:

- spustitelnost na operačním systému Linux,
- možnost programování v jazyce C++,
- jednoduchost tvorby modelu,
- kvalita dokumentace,
- velikost aktivní komunity.

Operační systém Linux a možnost programování v jazyce C++ jsou důležité parametry hlavně kvůli tomu, že veškerý již vytvořený kód pro ovládání robota je psán v jazyce C++ a byl vytvořen a vyladěn pro operační systém Gentoo Linux. Proto je třeba, aby vybraný simulátor podporoval tyto dvě věci bezpodmínečně. Tomuto, bohužel nevyhovuje simulátor MORSE.

Jednoduchost tvorby modelu je zde hlavně kvůli rychlosti vývoje a případně jednoduššímu pochopení kódu někým, kdo by kód potřeboval upravit. Zde by mohl být problém se simulátorem ARGoS, který má ze všech vybraných simulátorů nejsložitější popis modelu, viz 2.3.3.

Kvalitní dokumentace je v každém případě velice důležitým parametrem. V těchto případech je však rozsáhlost a čitelnost dokumentace velice různorodá. Nejlepší dokumentace mají simulátory V-REP, Gazebo a MORSE. MORSE však nesedí již na předchozí kritérium. Zbylé dva zmíněné však zatím vedou a na dokumentaci vyřazují simulátor OpenRAVE, který téměř žádnou dokumentaci nemá.

Ohledně velikosti komunity jsou největší znovu simulátory V-REP a Gazebo, které patří mezi nejvíce používané simulátory. Oba mají aktivní fórum, kde je mnoho uživatelů připravených pomoci každému s problémem. Mezi nimi se tedy budu rozhodovat, jelikož nejlépe odpovídají požadavkům, popsaným výše.

Jak Gazebo, tak i V-REP jsou velice populární simulátory, a oba dva jsou neustále aktivně vyvíjeny. Proto mají velice podobné specifikace a je těžké mezi nimi vybrat. Oba dva splňují moje požadavky stejně a jediný možný problém, na který jsem narazil byl, že V-REP je mnohem náročnější na hardware počítače. Proto jsem nakonec jako vítěze zvolil simulátor Gazebo.

Kapitola 3

Návrh

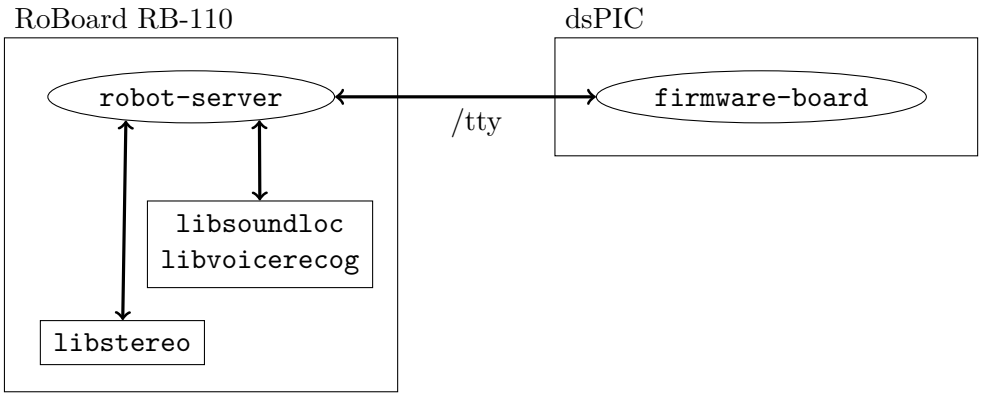
Pro vytvoření simulace byl v předchozí kapitole vybrán simulátor Gazebo. Práci s ním velice ulehčuje použití Robot Operating System, pro řízení modelu a proto zde ROS použiji. Pro kompletní převod existujícího robota i s jeho softwarem do tohoto simulátoru, je třeba vytvořit několik částí, které je navíc třeba mezi sebou propojit. Tyto části zahrnují:

- Vytvořit model robota, s popisem fyzikálních a vizuálních parametrů v jednom z podporovaných formátů.
- Vytvořit, či vybrat již existující moduly pro propojení simulovaného modelu s Robot Operating System.
- Upravit existující verzi programu `firmware-board` pro práci s modelem přes ROS.
- Upravit knihovnu `libstereo`, aby pracovala se simulovanými kamerami místo reálných.
- Upravit řídící program `robot-server` pro možnost spuštění mimo desku robota.

Možnosti provedení výše uvedených popíši detailněji dále v této kapitole. Také se zde budu zaobírat možnými problémy, na které bych mohl narazit při implementaci těchto částí.

Na obrázku 3.1 můžete vidět komunikaci probíhající v robotovi na úrovni softwaru. Z toho lze usoudit, že hlavní místa úprav budou již zmíněný program `firmware-board` a knihovna `libstereo`.

Knihovny `libsoundloc` a `libvoicerecog`, které jsou na reálném robotovi také přítomny, nebudou převedeny do simulace, jelikož simulátor nepodporuje přímou práci se zvukem. Vytvoření modulu pro svět simulace by bylo velice náročné a přesáhlo by rozsah práce.



Obrázek 3.1: Schéma programů a knihoven na robotovi

3.1 Model robota

Pro vytvoření modelu robota je třeba využít jeden z popisovacích formátů, podporovaných simulátorem Gazebo. Tyto formáty jsou dva a to URDF a SDF. Oba formáty jsou nástavby XML a oba je možné také upravovat ve vestavěném grafickém editoru. [11]

URDF je popisový formát používaný pro Gazebo již od začátků. Používá se na popis jednoho modelu robota a je také používán systémem ROS pro vizualizaci pohybu. URDF popisuje nejen vizuální stránku robota, ale také jeho kolizní model a fyzikální vlastnosti jeho částí. To bohužel nestačí pro simulaci a proto je třeba pomocí dodatečných parametrů simulátoru přidat i tření některých částí a jejich tuhost. Tento formát lze konvertovat na formát SDF, ale zpětně to bohužel možné není. Navíc většina nástrojů pro ROS umí pracovat pouze s URDF modely a proto je lepší tvořit v tomto formátu. [12]

SDF je formát, který měl nahradit URDF. To se však nepovedlo, jelikož neexistuje převod z formátu SDF na formát URDF a proto tento formát nelze použít se systémem ROS. SDF však napravuje různé nedostatky URDF, jako možnost popsání více modelů v jednom souboru, či popsání světa a pozic modelů v něm. Tento formát je navíc již základním formátem pro Gazebo, a pokud by nebyl použit systém ROS, zvolil bych tento formát. [13]

Při tvorbě samotného modelu je třeba popsat robota po částech. Jak URDF tak i SDF používají k popisu modelu dva základní tagy: `<link>` a `<joint>`. Zde je popsána jejich funkce v URDF:

`<link>` definuje jednu část modelu, v mém případě například kolo, či hlavní krabici modelu. Tento element má několik dalších parametrů, které se zadávají jako vnořené tagy. Těmi jsou:

- `<visual>`, který popisuje jak bude daná část vypadat pomocí základních geometrických obrazců zadaných tagem `<geometry>`,
- `<collision>`, který popisuje kolizní model, pro výpočet nárazů do objektů, zadávaný podobným způsobem jako vizuální stránka,
- `<inertial>`, který popisuje hmotnost a setrvačnost dané části pomocí momentů setrvačnosti, zadanými maticí o rozměrech 3x3. [12]

Všechny tyto tagy mají zároveň parametr `<origin>`, který definuje jejich střed, či těžiště. [12]

`<joint>` definuje spojení dvou `<link>` elementů, a to jak místo jejich spoje tagem `<origin>`, tak i způsob spojení. Zde je možné použít různé typy spojů. Z nich je na výběr:

- `fixed`, neboli pevný spoj,
- `continuous`, neboli spoj otáčející se okolo jedné osy bez omezení,
- `revolute`, neboli spoj otáčející se okolo jedné osy s omezením úhlů,
- `floating`, neboli spoj otáčející se volně ve všech osách,
- `prismatic`, neboli spoj posouvající se v jedné ose s omezením vzdáleností,
- `planar`, neboli spoj posouvající se volně v jedné rovině. [12]

V URDF se také dále používá element `<gazebo>`, kterým se definuje barva části a její další fyzikální veličiny jako tření, nebo tuhost. Oba výše uvedené formáty také obsahují i možnost definovat senzory, jako například čidla vzdálenosti, či kamery, pomocí jejich elementů. [12]

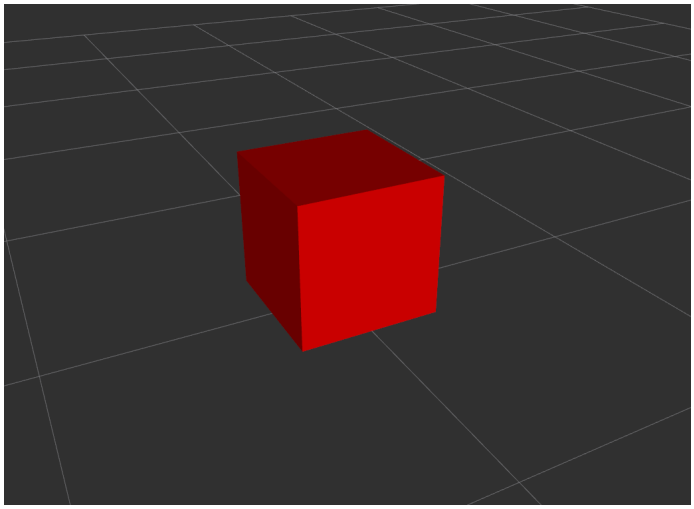
Formát URDF také podporuje použití `xacro`, což je pomůcka pro zjednodušení přehlednosti velkých modelů robotů. Jedná se o rozšiřující formát URDF, který používá makra k usnadnění práce s modely. Díky němu lze zadefinovat proměnné, používat matematické konstanty, vypočítávat jednoduché matematické vzorce a tvořit vlastní makra. [14]

Definování proměnných v `xacro` zjednoduší přehlednost a možnost úprav rozměrů robota. Za pomoci maker lze vytvořit například všechny kola bez zdlouhavého kopírování a přepisování pár parametrů.

Použití matematických konstant zase zjednoduší výpočty úhlů zadávaných v radiánech a pomocí jednoduché matematiky mohu místo zadávání čísel vypočítat pozice z naměřených dat uložených v proměnných.

Na další stránce můžete vidět ukázku kódu v URDF. V ukázce je vidět složitost kódu při popisu jediné krychle o rozměru půl metru. Tento model je pak možné vidět na obrázku 3.2.

```
<link name="base">
</link>
<!-- Vytvoření krychle -->
<link name="box">
  <!-- Popis vizuální stránky -->
  <visual>
    <!-- Posunutí středu modelu o půl výšky nahoru -->
    <origin rpy="0 0 0" xyz="0 0 0.25"/>
    <geometry>
      <!-- Krychle o rozměru 0.5 metru -->
      <box size="0.5 0.5 0.5"/>
    </geometry>
    <!-- Barva části v RGBA -->
    <material name="red">
      <color rgba="1 0 0 1"/>
    </material>
  </visual>
  <!-- Popis kolizního modelu, stejně jako vizuální -->
  <collision>
    <origin rpy="0 0 0" xyz="0 0 0.25"/>
    <geometry>
      <box size="0.5 0.5 0.5"/>
    </geometry>
  </collision>
  <!-- Popis setrvačnosti a těžiště -->
  <inertial>
    <!-- Umístění těžiště -->
    <origin rpy="0 0 0" xyz="0 0 0.25"/>
    <!-- Hmotnost v kilogramech -->
    <mass value="1"/>
    <!-- Popis momentů setrvačnosti -->
    <inertia ixx="0.5" ixy="0" ixz="0"
             iyy="0.5" iyz="0" izz="0.5"/>
  </inertial>
</link>
<!-- Vytvoření pevného spoje se základnou -->
<joint name="box_to_ground" type="fixed">
  <!-- Linky, které chceme spojit -->
  <parent link="base"/>
  <child link="box"/>
  <!-- Umístění spoje -->
  <origin rpy="0 0 0" xyz="0 0 0"/>
</joint>
```



Obrázek 3.2: Model vytvořený z kódu výše

3.2 Ovládací moduly

Po vytvoření modelu je třeba zprovoznit jeho komponenty. V mém případě se jedná o tyto:

- motory na kolech,
- gyroskop,
- akcelerometr,
- IR čidla vzdálenosti,
- kamery,
- servo pro naklánění kamer.

Pro každou z těchto částí je naštěstí již existující modul, který nám ulehčí práci s vytvořením softwarově řiditelného rozhraní. Gazebo má mnoho vlastních modulů, které se definují přímo v URDF souboru s popisem robota a vytváří ovládací *topic* ve frameworku ROS. Ty je možné vidět zde [15]. Další možností je použít modul `ros_control`, který dovolí použít moduly přímo od frameworku ROS. Ty se definují pomocí konfiguračního YAML souboru.

V obou případech se v parametrech modulu definuje několik potřebných údajů. Mezi ně patří:

- část, které se týká, ať už `<link>`, nebo `<joint>`,
- obnovovací frekvence modulu,

- specifikace částí, například rozlišení kamery,
- název pro *topic*, kterým se bude ovládat, či na který se budou publikovat data.

Parametry jsou u každého modulu jiné a proto je třeba řádně nastudovat dokumentaci každého z nich. Některé moduly frameworku ROS jsou však hůře dokumentované a poté je třeba studovat přímo zdrojový kód daného modulu.

Pro ovládání kol je možné využít několik různých modulů. Nejvhodněji vypadá modul pro diferenciální řízení frameworku ROS, kterému se definuje jako parametr pouze *topic*, na kterém má naslouchat a které spoje patří kolům. Tento modul po spuštění čeká na zprávy publikované na zadaný *topic* a ovládá kola podle přijatých zpráv. [16]

Zprávy pro ovládání řízení jsou typu `Twist` a mají dvě hlavní složky. První je lineární rychlost, zadaná v metrech za sekundu a druhou je úhlová rychlost v radiánech za sekundu. Tato rychlost je zadána celému robotovi a modul podle ní nastaví rychlosti jednotlivých kol. [16]

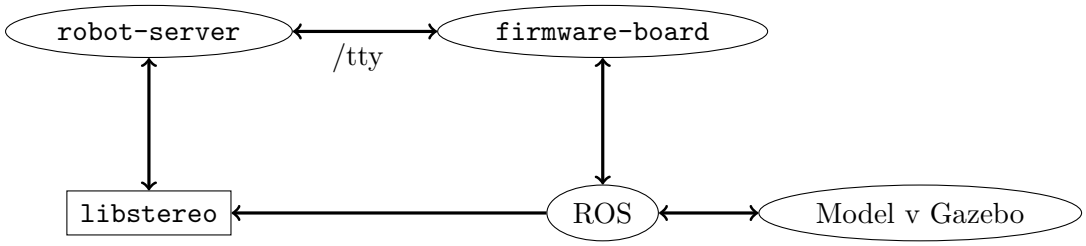
Pro senzory je také možné využít existující moduly. Ty lze vybrat ze široké nabídky Gazeba.

Kamery lze vytvořit pomocí přidání tagu `<sensor>` s nastavením parametru `type="camera"`. Tomuto tagu se poté nastaví další parametry podle specifikace kamery jako rozlišení, rychlost snímání, nebo formát obrazu. Poté se části, která byla zvolena kamerou přidá ovládací modul, a ten zařídí zbytek za nás. Kamera vysílá obraz společně s informacemi o formátu a rozlišení na *topic* jako zprávu typu `Image`. Tu můžeme v kódu číst a posílat dále.

IR čidla lze vytvořit podobně jako kameru pomocí tagu `<sensor>` s parametrem `type="ray"`. Tento typ senzoru vysílá paprsek nastaveným směrem a na nastavený *topic* posílá zprávy typu `LaserScan`, které obsahují vzdálenost k nejbližšímu objektu. Tento senzor využívá více paprsků v zadaném úhlu. Pro můj případ postačí pouze jeden paprsek pro každý senzor.

Gyroskop a akcelerometr je možné vytvořit pomocí modulu s názvem `Imu`, neboli Inertial Motion Unit, který zachycuje jak akceleraci, tak i náklon. Tento modul vysílá zprávy typu `Imu`, které obsahují potřebná data.

Servo pro naklánění platformy s kamerami je možné ovládat pomocí dalšího modulu ROS s názvem `JointPositionController`, který se stará o ovládání spojů s omezenou rotací. Ten odposlouchává nastavený *topic* a upravuje natočení podle poslední přijaté zprávy.



Obrázek 3.3: Návrh propojení modelu s existujícím softwarem

3.3 Úprava programového vybavení

Pro správnou funkčnost softwaru robota mimo něj je třeba udělat úpravy v jeho programech. Bez úprav se neobejde hned několik částí, naštěstí jsou však některé již připraveny na konverzi do simulátoru. Na obrázku 3.3 můžete vidět návrh propojení modelu s programovým vybavením robota.

Úpravy se týkají těchto aplikací, či knihoven:

firmware-board je jak již bylo řečeno, nízkoúrovňová ovládací deska robota. Pro ní je však připraven HAL, neboli Hardware Abstraction Layer. To znamená, že je v mém případě potřeba implementovat pouze ovládací metody a komunikace mezi touto částí a zbytkem je převzata z původního provedení. Z toho vyplývá, že je třeba vytvořit pouze metody pro nastavení motorů, získání dat z čidel a podobně.

libstereo je knihovna využívaná aplikací `robot-server` k získávání data z kamer, která poté zpracovává. Zde je třeba udělat úpravy v načítání obrazů z kamer. To znamená úpravu metod pro zachycení obrazu, kde narážím na možný problém. Data z reálných kamer jsou získávána ve formátu YUY2, a v tomto formátu s nimi i knihovna dále pracuje. Simulované kamery však tento formát nepodporují a proto bude třeba napsat konverzi z formátu RGB, který používají simulované kamery, na již zmíněný formát YUY2. Tento proces popíši dále v implementační části.

robot-server obstarává hlavní vysokoúrovňové řízení robota. Zde by nemělo být třeba dělat velké úpravy, jelikož tato aplikace nepracuje s robotem přímo, ale využívá již zmíněné programy a knihovny. Proto bude jediný možný důvod k úpravám odpojení knihoven pro práci se zvukem, které z již zmíněných důvodů nebudu v modelu implementovat.

Kapitola 4

Realizace

V této kapitole se věnuji implementačním detailům, které jsem při realizaci simulace řešil. Implementace je provedena v simulátoru Gazebo, za pomoci Robot Operating System.

4.1 Model robota

Pro vytvoření modelu je využit formát URDF, kterým popisuji části robota postupně. Pro funkční model je třeba popsat vizuální stránku robota, jeho kolizní model a momenty setrvačnosti všech komponent. V této části je všechny vysvětlím.

4.1.1 Vizuální stránka

Prvním krokem ve tvorbě modelu bylo vytvořit statický vizuální model. Ten zahrnuje rozdělení robota na části, které zapíši v popisu jako `<link>` a do těchto tagů zapíši jejich vizuální popis jako `<visual>`.

Před začátkem tvorby modelu bylo však třeba robota změřit a zapsat přesné rozměry jeho částí. Po zapsaní všech potřebných rozměrů jsem mohl začít se zapisováním částí do modelu. Prvním krokem bylo vytvořit proměnné pomocí maker typu `property` programu Xacro. V nich jsem definoval názvy proměnných a naměřené rozměry, které jsem později využil k popisu. Pro zápis vizuální stránky robota jsem využil tag `<visual>`, viz kapitola 3.1.

Rozdělení jednotlivých částí reálného robota na části modelu jsem provedl následujícím způsobem:

Hlavní základna se skládá pouze z jednoho kvádru a je nastavena jako hlavní součást modelu. Součást je nazvána `main_box`.

Kola jsou vytvořeny dynamicky pomocí makra v podpůrném programu Xacro, díky kterému je možné definovat parametry makra udávající, zda je

kolo vpředu, či vzadu a zda je vlevo, nebo vpravo. V makru poté vypočítám pozici kola podle zadaných argumentů. Každé kolo je tvořeno pouze jedním válcem a je připevněno pomocí otočného spoje k přidruženému zavěšení. Součástky jsou nazvány `wheel` s prefixem podle umístění na modelu (například `left_front_wheel`).

Zavěšení kol je namodelováno také, aby kola nevyseli ve vzduchu. Proto je na každé straně mezi kolem a základnou vymodelován válec, spojující kolo se stranou základny. Toho je dosaženo pomocí stejného makra v programu Xacro. Zavěšení je připevněno pevným spojem k základně modelu. Součástky jsou nazvány `wheel_hinge` s prefixem jako u kol.

IR senzory jsou definovány jako kvádry na přední a zadní části robota. Ty jsou také vytvořeny pomocí makra programu Xacro. Pro model jsou potřeba hlavně k připevnění modulu na snímání vzdálenosti. Tyto senzory jsou pevným spojem připevněny k základně. Součástky jsou nazvány `ir` s prefixem podle toho, zda jsou vpředu, či vzadu.

Spodní kamerová základna je vymodelována ze dvou částí. První část je tenký kvádr, který slouží jako podstavec a druhá část je vysoký kvádr uprostřed prvního, tyčící se vzhůru. Tato část je připevněna pevným spojem k hlavní základně modelu. Součást je nazvána `camera_base`.

Horní kamerová základna je vymodelována jako tenký kvádr, ze kterého vede spojující válec do vztyčené části spodní základny. V místě kde se dotýkají jsou obě části spojeny otočným spojem s omezením, který simuluje servomotor ovládající náklon kamer. Součást je nazvána `camera_platform`.

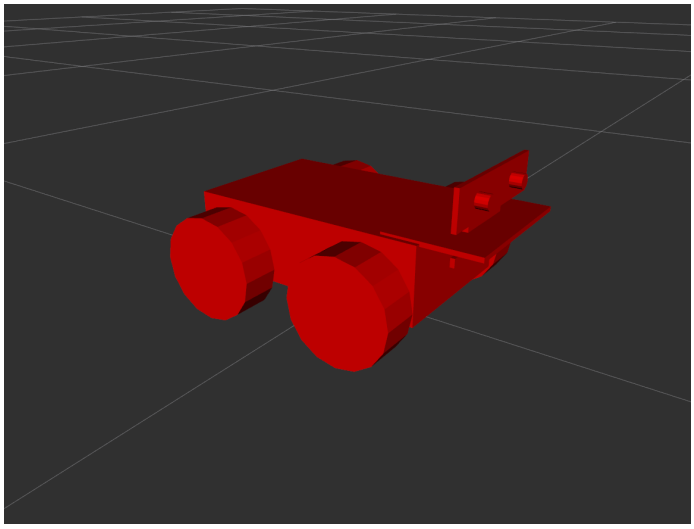
Kamery jsou vytvořeny jako válce, připevněné pevným spojem na horní kamerovou základnu. Opět jsou generovány makrem za pomoci programu Xacro. K těmto částem je v pozdějším kroku přidán modul na snímání obrazu. Součástky jsou nazvány `camera` s postfixem podle strany, na které jsou umístěny.

Akcelerometr a gyroskop potřebují také svoji součástku, ke které se později přidá potřebný modul. Tato část je schována uvnitř hlavní základny modelu a není vidět. Přesto však musí být definována. Součást je nazvána `imu_link` podle použitého modulu.

`base_link` je základní součást každého modelu. Gazebo požaduje existenci této části, ale část zároveň nesmí obsahovat žádný popis. Proto je tato část definována jako prázdná a k ní je pevně připojena hlavní základna modelu.

Po vytvoření vizuální stránky modelu lze použít program frameworku ROS s názvem Rviz, který dokáže model zobrazit. Tento program byl velice užitečný

při sestavování celého modelu pro průběžnou vizualizaci. Na obrázku 4.1 můžete vidět model robota v tomto programu. Pro přidání barev do modelu je třeba využít URDF tag `<gazebo>`, ve kterém lze definovat tag `<material>`. Zde je možné vybrat z předdefinovaných barev a simulátor poté barvy vykreslí.



Obrázek 4.1: Vizualizace modelu v Rviz

V příloze C můžete také vidět vizualizaci celého URDF souboru, kterou vytvořil program `urdf_to_graphiz`. Ten je součástí pomocných programů frameworku ROS. Zde můžete vidět jak jednotlivé části robota, tak i jejich spoje.

4.1.2 Kolizní model

Když je vizuální stránka robota hotova, můžu pokračovat další potřebnou částí. Tou je popis kolizního modelu robota. Simulátor Gazebo požaduje pro každou část definovaný kolizní model, jinak nelze spustit simulaci s tímto modelem. Přidání kolizního modelu k částem typu `<link>` je možné pomocí tagu `<collision>`, ve kterém se definují stejné parametry jako při popisu vizuální stránky robota.

Přidávání kolizí k částem je v mém případě jednoduché. Pro mé rozdělení částí není třeba vytvářet jiné kolizní modely nežli ty, které jsou stejné jako vizuální modely. K vytvoření kolizního modelu tedy stačí použít kód vložený v tagu `<visual>`, který se pouze zkopíruje a vloží do tagu `<collision>`.

Možnost vytvoření kolizního modelu nezávisle na vizuálním je důležité v případě, že vizuální model byl vytvořen detailně a je třeba zmenšit zátěž při výpočtech kolizí. V takovém případě se složité modely nahrazují základními geometrickými tělesy.

4.1.3 Fyzikální vlastnosti

Další částí, kterou je třeba popsat jsou některé fyzikální vlastnosti modelu. Pro spuštění v simulátoru Gazebo je třeba definovat pro každou část typu `<link>` tyto vlastnosti:

- hmotnost části,
- těžiště části,
- moment setrvačnosti části, zadaný pomocí tzv. tenzoru setrvačnosti.

To se provádí přidáním tagu `<inertial>` ke každé části typu `<link>`. V něm se definuje těžiště stejně, jako se definuje střed modelu ve vizuální části. Pro zadání hmotnosti modelu je třeba zadefinovat tag `<mass>`.

Definování tenzoru setrvačnosti je již složitější. Pro funkčnost simulace je třeba definovat každé části matici o rozměrech 3x3, která definuje moment setrvačnosti pro danou součástku, neboli zachování energie při pohybu v prostoru. Tenzor setrvačnosti má tento formát:

$$I = \begin{pmatrix} I_{x,x} & I_{x,y} & I_{x,z} \\ I_{y,x} & I_{y,y} & I_{y,z} \\ I_{z,x} & I_{z,y} & I_{z,z} \end{pmatrix}$$

Tato matice je však souměrná a proto je třeba simulaci dodat pouze těchto 6 konstant z matice uvedené výše: $I_{x,x}$, $I_{x,y}$, $I_{x,z}$, $I_{y,y}$, $I_{y,z}$, $I_{z,z}$.

Pro výpočet tenzoru setrvačnosti existuje několik různých způsobů. V mém případě lze využít vzorce pro základní geometrická tělesa, a pomocí nich dopočítat tyto konstanty. V modelu využívám pouze dva druhy těles, a to kvádr a válec. Tenzory setrvačnosti pro tyto dva základní tvary vypadají následovně:

Kvádr o šířce w, délce d, výšce h a hmotnosti m má tenzor setrvačnosti:

$$I_{kvádr} = \begin{pmatrix} \frac{1}{12}m(h^2+d^2) & 0 & 0 \\ 0 & \frac{1}{12}m(w^2+d^2) & 0 \\ 0 & 0 & \frac{1}{12}m(w^2+h^2) \end{pmatrix} [17]$$

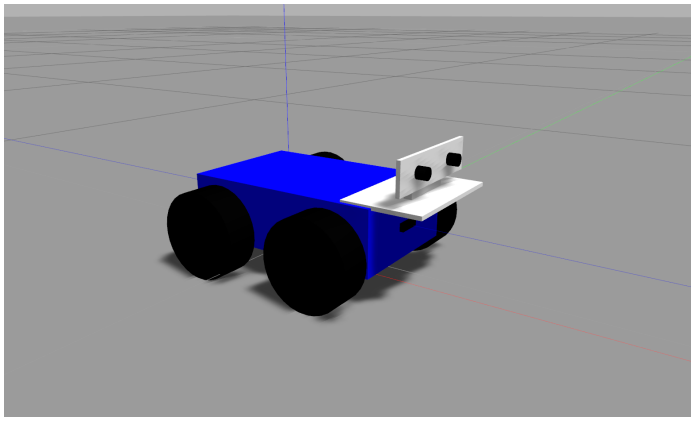
Válec o průměru r, výšce h a hmotnosti m má tenzor setrvačnosti:

$$I_{válec} = \begin{pmatrix} \frac{1}{12}m(3r^2+h^2) & 0 & 0 \\ 0 & \frac{1}{12}m(3r^2+h^2) & 0 \\ 0 & 0 & \frac{1}{2}mr^2 \end{pmatrix} [17]$$

Z toho vyplývá, že je do modelu třeba dodefinovat pouze konstanty $I_{x,x}$, $I_{y,y}$, $I_{z,z}$. Ty lze navíc vypočítat pomocí matematického vzorce, který lze zadat jako makro programu Xacro.

4.1.4 Výsledek

Po definování všech těchto částí je možné spustit simulaci s modelem robota. Ten zatím nebude možné ovládat, ale bude již reagovat na podněty v simulátoru. Výsledný model v simulátoru po takovéto úpravě můžete vidět na obrázku 4.2.



Obrázek 4.2: Model robota v simulátoru Gazebo

4.2 Rozhýbání modelu

Vytvořením modelu se sice podařilo zobrazit robota v simulátor, ale zatím ho nelze řídit. Nyní je třeba přidat k modelu ovládací rozhraní. Toho lehce dosáhnu pomocí Robot Operating System a modulů simulátoru Gazebo.

Simulátor Gazebo má velice kvalitně propracovanou integraci s frameworkem ROS, pomocí připravených modulů. Použití těchto modulů se definuje přímo v souboru URDF s popisem robota. Modul je možné přidat k jakékoliv části, nebo i k celému robotovi. Tento modul má poté přístup k ovládání robota.

Použití modulu se definuje uvnitř tagu `<gazebo>`, ve kterém se poté definuje tag `<plugin>`. V tomto tagu je třeba popsat název modulu, který chci použít. Takto definovaným modulům se vkládají další vnořené tagy, které definují parametry předané tomuto modulu. Tyto parametry se však liší podle použitého modulu. Takto definované moduly se poté spouští automaticky při vložení modelu do simulátoru.

V mnou vytvořeném modelu jsem použil dva různé způsoby napojení modulů. Prvním z nich je použití modulů simulátoru Gazebo. Ty je možné definovat pouze v URDF souboru s popisem modelu, kde se definují všechny parametry, jako i *topic* frameworku ROS, se kterým budou pracovat. Dalším způsobem je použít modul s názvem `libgazebo_ros_control`, který zpřístupní celého robota frameworku ROS. Poté je možné za pomoci konfiguračního souboru ve formátu YAML definovat moduly vytvořené pro ROS. Tyto moduly používám na ovládání spojů v simulátoru, přesněji pro kola a servo ovládající náklon kamer.

4.2.1 Modul `libgazebo_ros_control`

Tento modul slouží pro přemostění ovládání celého robota do frameworku ROS. Po jeho zadefinování v souboru s popisem modelu je třeba vytvořit konfigurační soubor formátu YAML, ve kterém se poté definují parametry a další podpůrné moduly. Tyto moduly nejsou stejné jako moduly simulátoru Gazebo. Tento přístup jsem zvolil kvůli lépe provedeným ovládacím modulům na řízení spojů v simulátoru.

První modul, který je třeba definovat v konfiguračním souboru má název `joint_state_controller`. Ten zpracovává data o pohyblivých spojích na modelu robota a umožňuje s nimi dále pracovat ostatním modulům. Jeho použití je nezbytné pro funkčnost.

Dalším krokem je definovat ostatní moduly pro ovládání robota. V mém případě jsem použil tyto dva:

`diff_drive_controller` je modul ovládající řízení kol. Jak již název napovídá jedná se o diferenciální řízení, tedy stejné jako používá reálný robot. Modul odposlouchává definovaný *topic* a přijímá zprávy typu `Twist`. Ty

obsahují data o rychlosti jakou se má robot pohybovat dopředu, a jakou rychlostí se má otáčet. V konfiguračním souboru se tomuto modulu definují jako parametry `left_wheel` a `right_wheel` pole s názvy pohyblivých spojů, odpovídajících kolům na jednotlivých stranách. Dále se zde definují limity rychlosti a zrychlení, a to jak pro lineární pohyb, tak i pro otáčení.

`position_controllers` je modul ovládající servo pro naklánění kamer. Tomuto modulu se definuje pouze parametr `joint`, který udává spoj, se kterým má pracovat.

Pro rozhýbání spojů v modelu je však potřeba zadefinovat ještě jeden tag. Tím je `<transmission>`. Tento tag se definuje ke každému pohyblivému spoji a uvnitř něho je možné vybrat, zda chci použít nějaký převod a jakým způsobem má být spoj ovládán. Důležitý je hlavně způsob ovládání spoje, zadávaný tagem `<hardwareInterface>`. Tím se definuje, jak má ovládací modul se spojem pracovat. Pro kola je zde zadán parametr `VelocityJointInterface`, který modulu říká, že bude ovládat spoj zadáváním rychlosti otáčení. U servomotoru by toto ovládání bylo velice nepraktické, a proto zde má parametr `PositionJointInterface`, který modulu říká, že spoj bude ovládán zadáváním úhlu náklonu.

Tímto je ovládání spojů nastaveno a oba dva moduly si vytvoří *topic* podle svého názvu, na kterém naslouchají a ovládají dané spoje dle přijatých zpráv.

4.2.2 Moduly simulátoru Gazebo

Pro všechny senzory jsem využil základní moduly připravené v simulátoru Gazebo. Ty se definují přímo v popisovém souboru modelu, uvnitř tagů `<gazebo>`. Zde se ke každému modulu definují různé další parametry. Některé moduly se zde navíc zabalují do tagu `<sensor>`, ve kterém se popisuje jak modul, tak i parametry senzoru.

Zde je seznam implementovaných senzorů v modelu robota a jejich popis.

Akcelerometr a Gyroskop jsou implementovány jako jeden modul. Tento modul nese název `libgazebo_ros_imu` a je definován jako samotný tag `<plugin>`, přiřazený k části robota s názvem `imu_link`. Tento modul má mezi parametry *topic*, na který vysílá zprávy typu `Imu`. Ty obsahují mnoho různých dat, ale hlavně obsahují data označená `orientation`, což jsou data ze simulovaného gyroskopu a data typu pojmenovaná `linear_acceleration`, obsahující hodnoty z akcelerometru.

IR senzory jsou implementovány již složitějším způsobem. Pro napojení těchto senzorů na framework ROS je třeba zadefinovat nejen modul, ale i již zmíněný tag `<sensor>`. Tomu je třeba definovat typ `ray`. Poté je možné dodefinovat několik dalších tagů uvnitř. Hlavním je tag `<resolution>`,

kterým nastavíme kolik paprsků má senzor vysílat. Tuto hodnotu pro mé potřeby nastavuji na 1. Senzory tedy snímají pouze přímo před sebou.

Po zadefinování tagu `<sensor>` je ještě třeba určit modul, který má použít. Ten se nastavuje znovu tagem `<plugin>`, kterému se předá název `libgazebo_ros_laser`. Tomuto modulu se určí *topic*, na který má odesílat sbíraná data a tím mám IR senzor hotov. Celý tento kód je však třeba přidat do makra pro tvorbu tohoto senzoru, jelikož v popisovém souboru generuji senzory dva.

Kamery jsou implementovány podobným způsobem jako IR senzory. I zde je třeba definovat tag `<sensor>`, tentokrát však s typem `camera`. Kamery mají však mnohem více parametrů, které je třeba dodat. Nejdůležitější tagy, které je třeba definovat jsou tyto:

- `<updateRate>`, udávající rychlost snímání obrazu,
- `<horizontal_fov>`, udávající zorné pole kamery, které je třeba zadat podle reálné kamery robota,
- `<width>` a `<height>`, udávající rozlišení kamery,
- `<format>`, udávající barevný model, v jakém kamera snímá.

Poté co zadefinuji tag `<sensor>` můžu nastavit také modul, který chci použít. Zde se používá modul jménem `libgazebo_ros_camera`, kterému je třeba definovat hlavně *topic*, na který má vysílat obraz z kamer a frekvenci, s jakou má posílat obraz z kamer.

Tento kód je opět třeba přidat do makra pro tvorbu kamer, které se jím generují.

4.3 Napojení na software

Nyní mám vytvořené ovládací rozhraní pro model robota ve frameworku ROS. Dalším krokem je upravení softwaru robota tak, aby bylo možné jej spustit na počítači a ovládat jím model. Toho lze docílit úpravou knihovny `libstereo`, a programů `firmware-board` a `robot-server`.

ROS obsahuje také knihovnu `roscpp`, která umožňuje tvorbu ovládacích programů pro roboty. Tato knihovna je psána pro jazyk C++, čímž jsem narazil na první problém. Většina softwaru robota je psána v jazyce C, a proto bylo třeba zabalit kód pro přístup ke knihovně `roscpp`.

4.3.1 Program `firmware-board`

Program `firmware-board` má již připravenou abstrakční vrstvu. Díky ní je zde třeba implementovat jen funkce, které získávají data ze senzorů a řídí motory robota. Zbytek programu tedy zůstává nezměněn, čímž se zachová jeho komunikace s programem `robot-server`.

Knihovnu `roscpp` bohužel nelze zahrnout přímo do kódu v jazyce C. Proto jsem zde vytvořil další soubor s názvem `gazebo_ros_control.cpp`, který již pracuje s touto knihovnou přímo. V souboru definované metody odpovídají vždy jedné metodě v základním souboru `gazebo_hal.c`, se kterým pracuje zbytek programu. Metody volané při přijetí požadavku z programu `robot-server`, tedy pouze volají příslušné metody z již zmiňovaného souboru `gazebo_ros_control.cpp`.

První potřebnou metodou k implementaci byla `ros_init`, která inicializuje uzel v programu a vytvoří spojení s frameworkem ROS. Poté je již program registrován a lze přistupovat na jakýkoliv *topic*. Také se zde inicializují *callback* metody, které se budou volat, když na nastavený *topic* dorazí nová zpráva. Tyto metody se zde nastavují tři. První je metoda `imu_callback`, která obsluhuje akcelerometr a gyroskop a další dvě metody jsou `ir_front_callback` a `ir_back_callback`, které obsluhují přední a zadní IR senzor pro měření vzdálenosti. Všechny *callback* metody mají jeden parametr, a to zprávu která právě přišla na odposlouchávaný *topic*. Uvnitř těchto metod se pouze uloží důležité informace a skončí. Dále se v inicializační metodě nastavuje a ukládá *publisher* pro kola. Ten je uložen jako globální proměnná, pomocí které lze ovládat motory kol.

Zde jsem však narazil na další problém. Pro obsluhu motorů přes rozhraní ROS je třeba pravidelná obsluha. Připravené funkce se však volali pouze když je bylo potřeba. Naštěstí je však v programu implementován časovač, který spouští každých 10 ms metody, které se mu definují. Vytvořil jsem tedy metodu `ros_in_loop`, která se stará o obsluhu motorů a tu přidal k tomuto časovači.

Samotné ovládání motorů je řízeno pomocí dvou globálních proměnných s nastavením rychlostí jednotlivých stran. Do těchto proměnných zapisují metody `motor_left_set` a `motor_right_set`, které volá komunikační část pro-

gramu. Tyto proměnné čte každých 10 ms metoda `ros_in_loop` a podle nich posílá na *topic* motorů požadovanou rychlost a směr. Ty jsou posílány v jedné zprávě typu `Twist`, která však obsahuje data o lineární a úhlové rychlosti. Jelikož mi je zadána rychlost jednotlivých stran, musím provést přepočet. Ten je proveden pomocí těchto rovnic:

$$linear = \frac{motor\_right + motor\_left}{2} * \frac{MAX\_LINEAR}{MAX\_MOTOR}$$

$$angular = (\frac{motor\_left}{MAX\_MOTOR} - \frac{motor\_right}{MAX\_MOTOR}) * MAX\_ANGULAR$$

Kde $motor\_right$ a $motor\_left$ jsou nastavené hodnoty pro rychlost příslušných motorů od 0 do 1000, MAX_MOTOR je konstanta udávající maximální hodnotu motorů, tedy 1000, MAX_LINEAR udává maximální rychlost, kterou může robot vyvinou v m/s a $MAX\_ANGULAR$ udává maximální úhlovou rychlost v rad/s.

Získávání informací o senzorech je prováděno pomocí již zmíněných *callback* metod, které ukládají data o jimi obsluhovaném senzoru do globální proměnné. Z té si poté příslušná metoda, volaná z komunikační části programu, vezme data a předá je dál.

Zde je krátká ukázka inicializace předního IR senzoru a jeho obsluhy:

```
//globální proměnná s nejnovějšími daty ze senzoru
static float ir_front = 5;
//globální proměnná s odkazem na topic
ros::Subscriber ir_front_sub;
//callback metoda pro obsluhu příchozích zpráv
void ir_front_callback(const sensor_msgs::LaserScan::ConstPtr& m)
{
   ir_front = m->ranges[0];
}
void control_init()
{
   ...
   //inicializace programu jako uzlu ROS
   ros::init(argc, argv, "robot_fit_controll_board",
             ros::init_options::NoSigintHandler);
   //vytvoření rozhraní pro přístup k ROS a napojení na topic
   ros::NodeHandle nh;
   ir_front_sub = nh.subscribe("fitrobot/laser/front",
                               1, ir_front_callback);
   ...
}
//metoda která vrací data o senzoru zbytku programu
float get_front_ir() { return ir_front; }
```

4.3.2 Knihovna `libstereo`

Knihovna `libstereo` obstarává práci s kamerami a je součástí programu `robot-server`. Aby tato knihovna pracovala se simulovanými kamerami, stačí upravit tři metody v souboru `capture.c`. První je metoda `capture_open`, která inicializuje práci s kamerami a kontroluje nastavené parametry pro snímání. Druhou metodou je `capture_stereo_pair`, která zajišťuje snímání fotografií z kamer. Poslední metodou je `capture_close`, která uzavírá komunikaci s kamerami.

Tato knihovna je psána v jazyce C, proto je třeba znovu vytvořit soubor, který zabalí volání knihovny `roscpp` do jazyka C++. Zde jsem vytvořil soubor `roswrapper.cpp`, který obsahuje metody pro práci s frameworkem ROS. Ve výsledku každá metoda ze souboru `capture.c` pouze volá příslušné metody v souboru `roswrapper.cpp`.

Metoda `capture_open`, volá metodu `ros_setup`, která inicializuje práci s rozhraním ROS a je podobná inicializační metodě v předchozí části. Metoda `capture_close` je v tomto případě úplně prázdná, jelikož rozhraní ROS se uzavírá samo. Nejzajímavější je metoda `capture_stereo_pair`. Ta obstarává kamery na robotovi a je volána pouze když je třeba. Zde však není možné použít *callback* metody a proto jsem vybral variantu s metodou `waitForMessage`. Tato metoda zastavím běh programu a čeká dokud nepřijde další zpráva na zadaný *topic*. V případě knihovny `libstereo` však tato prodleva nevadí, jelikož reálný program počítá i s větší, kvůli náročnosti stažení obrazů z kamer.

Dalším krokem po získání obrazových dat z kamer je jejich převod. Jak již bylo popsáno výše, kamery na reálném robotovi pracují s barevným modelem YUY2. Simulované kamery však takovýto model neumí, a proto provádím konverzi z RGB na YUY2 až v kódu. YUY2 je formát rodiny YUV, který používá k popisu jasovou složku Y a dvě barevné složky U a V. Formát YUY2 navíc používá k popisu dvou po sobě jdoucích pixelů stejné barevné složky, a mění pouze jasovou složku.

Konverze se provádí pomocí těchto vzorců:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B$$

$$U = -0.147 * R - 0.289 * G + 0.436 * B$$

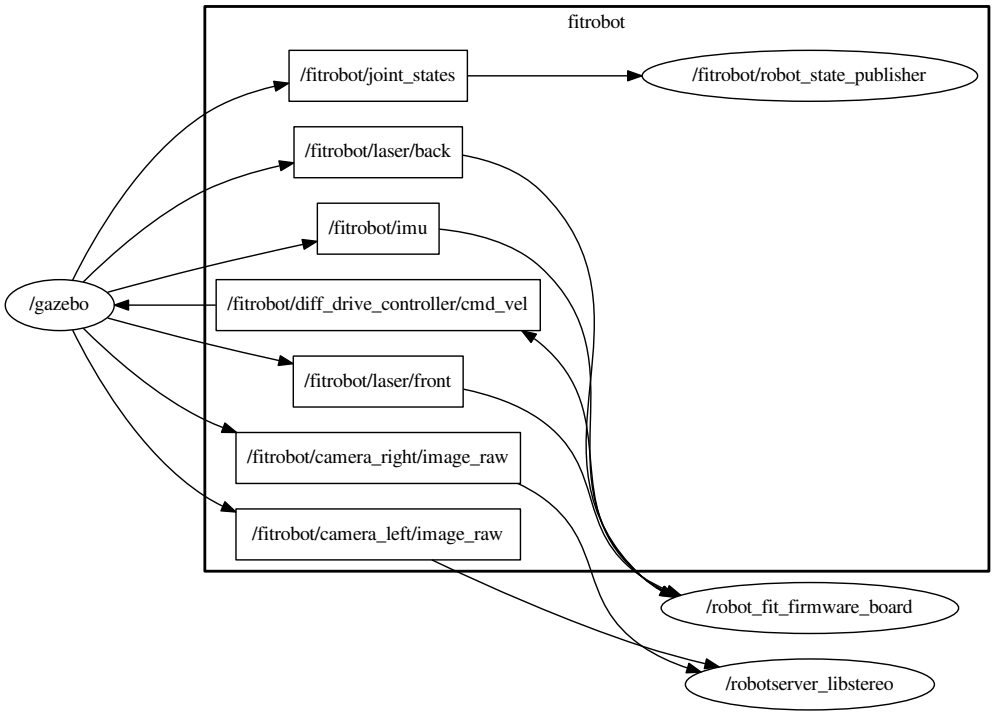
$$V = 0.615 * R - 0.515 * G - 0.100 * B$$

4.3.3 Program `robot-server`

V programu `robot-server` bylo třeba udělat jednu hlavní úpravu. Tou bylo odstranění knihoven pro práci se zvukem. Ten není možné, při dodržení rozsahu bakalářské práce, implementovat, a proto bylo třeba odebrat veškeré vazby na knihovny `libsoundloc`, `libsay` a `libvoicerecoq`. Toho bylo docíleno odebráním souboru `ears.c` z kompilace a odstraněním všech jeho zmínek ve zbytku programu.

Posledním krokem bylo upravit `Makefile` v obou výše zmíněných programech a knihovně. V nich jsem přidal odkazy na knihovny frameworku ROS a soubory vytvořené mnou.

Na obrázku 4.3 můžete vidět propojení jednotlivých uzlů programů i modelu a každý *topic*, pomocí kterého komunikují. Tento diagram je generován automaticky pomocí nástroje `rqt_graph`, který ho vytváří z běžícího prostředí ROS. Obdélník v grafu znázorňuje jeden *topic* a elipsa znázorňuje uzel.



Obrázek 4.3: Diagram všech uzlů a jejich komunikace

Závěr

Cílem práce bylo provedení rešerše open-source simulátorů a vytvoření modelu robota ve simulátoru, vybraném jako nejlepší. Model musel být navíc ovladatelný stejným softwarem, jako robot reálný. Tento cíl byl v práci splněn.

Po provedení rešerše jsem jako nejvhodnější vybral simulátor Gazebo, který splňoval všechny požadavky a navíc vynikal výbornou dokumentací a velkou aktivní komunitou uživatelů. Po prozkoumání hardwarového i softwarového vybavení robota jsem v simulátoru vytvořil model s přesnými rozměry podle reálného robota. Tento model byl vytvořen pomocí popisovacího formátu URDF, kterým bylo možné popsat jak vizuální stránku robota, tak i jeho fyzikální vlastnosti.

K vytvořenému modelu jsem poté dodal programově ovládatelné rozhraní pomocí frameworku Robot Operating System. Ten ulehčil tvorbu řídícího rozhraní modelu díky připraveným modulům. Tyto moduly zjednodušili ovládání modelu na posílání zpráv mezi simulátorem a programem.

V dalším kroku jsem upravil existující programové vybavení robota tak, aby bylo možné jej spustit na počítači a ovládat s ním model. Vše se povedlo a s modelem v simulátoru lze pracovat jako s opravdovým robotem pomocí klientských připojení na jeho server.

V příloze B najdete návod, podle kterého můžete spustit simulátor s modelem robota na počítači se systémem Linux.

Reálný robot má také mikrofony pro práci se zvukem. Ty však v této práci nebyli implementovány, jelikož simulátor nepodporuje snímání a vydávání zvuků. Bylo by tedy třeba vytvořit rozsáhlý modul pro simulátor, kterým by tato práce dalece přesáhla daný rozsah. Na tuto část může navázat kdokoliv další s jinou bakalářskou prací.

Literatura

[1] KOVÁŘÍK, K.: *Scénáře pro čtyřkolového robota.* Bakalářská práce, České vysoké učení technické v Praze, 2013, vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

[2] KOPECKÝ, M.: *Modul pro zpracování dat z inkrementálních čidel čtyřkolového robota.* Bakalářská práce, České vysoké učení technické v Praze, 2014, vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

[3] PETROUŠ, P.: *Řídící síťový server pro robotickou platformu.* Bakalářská práce, České vysoké učení technické v Praze, 2012, vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

[4] DAVID, R.: *Portování programového vybavení pro 3D vidění na vestavnou Linuxovou platformu.* Bakalářská práce, České vysoké učení technické v Praze, 2012, vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

[5] RICHTER, V.: *Řídící aplikace pro robotickou platformu.* Bakalářská práce, České vysoké učení technické v Praze, 2012, vedoucí práce: Ing. Miroslav Skrbek, Ph.D.

[6] *Robot Operating System homepage [online].* [cit. 2018-04-18]. Dostupné z: `http://www.ros.org/`

[7] *OpenRAVE homepage [online].* [cit. 2018-04-18]. Dostupné z: `http://openrave.org/`

[8] *MORSE homepage [online].* [cit. 2018-04-18]. Dostupné z: `https://www.openrobots.org/wiki/morse`

[9] *ARGoS homepage [online].* [cit. 2018-04-18]. Dostupné z: `http://www.argos-sim.info/`

[10] *V-REP homepage [online].* [cit. 2018-04-18]. Dostupné z: `http://www.coppeliarobotics.com/index.html`

[11] *Gazebo homepage [online].* [cit. 2018-04-18]. Dostupné z: http://gazebosim.org/

[12] *URDF documentation [online].* [cit. 2018-04-25]. Dostupné z: http://wiki.ros.org/urdf/XML

[13] *SDFormat specification [online].* [cit. 2018-04-25]. Dostupné z: http://sdformat.org/spec

[14] *Xacro documentation [online].* [cit. 2018-04-25]. Dostupné z: http://wiki.ros.org/xacro

[15] *Gazebo plugins in ROS [online].* [cit. 2018-04-25]. Dostupné z: http://gazebosim.org/tutorials?tut=ros_gzplugins

[16] *diff_drive_controller documentation [online].* [cit. 2018-04-25]. Dostupné z: http://wiki.ros.org/diff_drive_controller

[17] VYBÍRAL, B.: Kinematika a dynamika tuhého tělesa [online]. *Knihovnička FO č, 31.* Dostupné z: http://black-hole.cz/euso/down/dynamika.pdf

Příloha **A**

Seznam použitých zkratek

HAL Hardware Abstraction Layer

I^{2}C Inter-Integrated Circuit

IMU Inertial Motion Unit

IR Infrared Radiation

PWM Pulse Width Modulation

ROS Robot Operating System

SDF Simulator Description Format

URDF Unified Robot Description Format

USB Universal Serial Bus

XML Extensible Markup Language

YAML YAML Ain't Markup Language

Příloha B

Návod pro spuštění

Pro spuštění simulace je třeba mít nainstalované tyto programy:

- `ros-kinetic`
- `gazebo7`
- `catkin`

B.1 Instalace

Prvním krokem je instalace balíčku s modelem. Toho je dosaženo pomocí programu catkin, kterému je třeba vytvořit workspace pomocí těchto příkazů:

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

Poté je třeba přesunou přiloženou složku `fit_robot_description` do složky `/catkin_ws/src` a znovu spustit příkaz `catkin_make`. Tím se nainstaluje balíček s modelem.

Před spuštěním programu `robot-server` je třeba nainstalovat knihovnu `libstereo` pomocí těchto příkazů:

```
cd robotserver/libstereo
make
make install
```

Také je třeba nakopírovat složku `robot` z CD do vašeho adresáře `/etc/`. Ta je zde potřeba kvůli kalibraci kamer.

B.2 Spuštění

Pro plnohodnotné spuštění instalace je třeba několik instancí konzole, běžících najednou. Doporučuji spouštět pod uživatelem `root`. V první konzoli je třeba spustit jádro frameworku ROS pomocí:

```
roscore
```

V další instanci konzole je třeba spustit simulátor s modelem pomocí těchto příkazů:

```
source ~/catkin_ws/devel/setup.bash
roslaunch fit_robot_description spawn_fitrobot.launch
```

V další instanci je třeba vytvořit tunel pro propojení `firmware-board` a `robot-server` pomocí příkazu:

```
socat pty,raw,echo=0,link=/dev/ttyS20 pty,raw,echo=0,link=/dev/ttyS21
```

V další instanci konzole se spustí program `firmware-board` pomocí příkazů:

```
cd firmware-board/build
make robot-fit-sim
./robot-fit-sim
```

Posledním krokem je spuštění programu `robot-server` těmito příkazy:

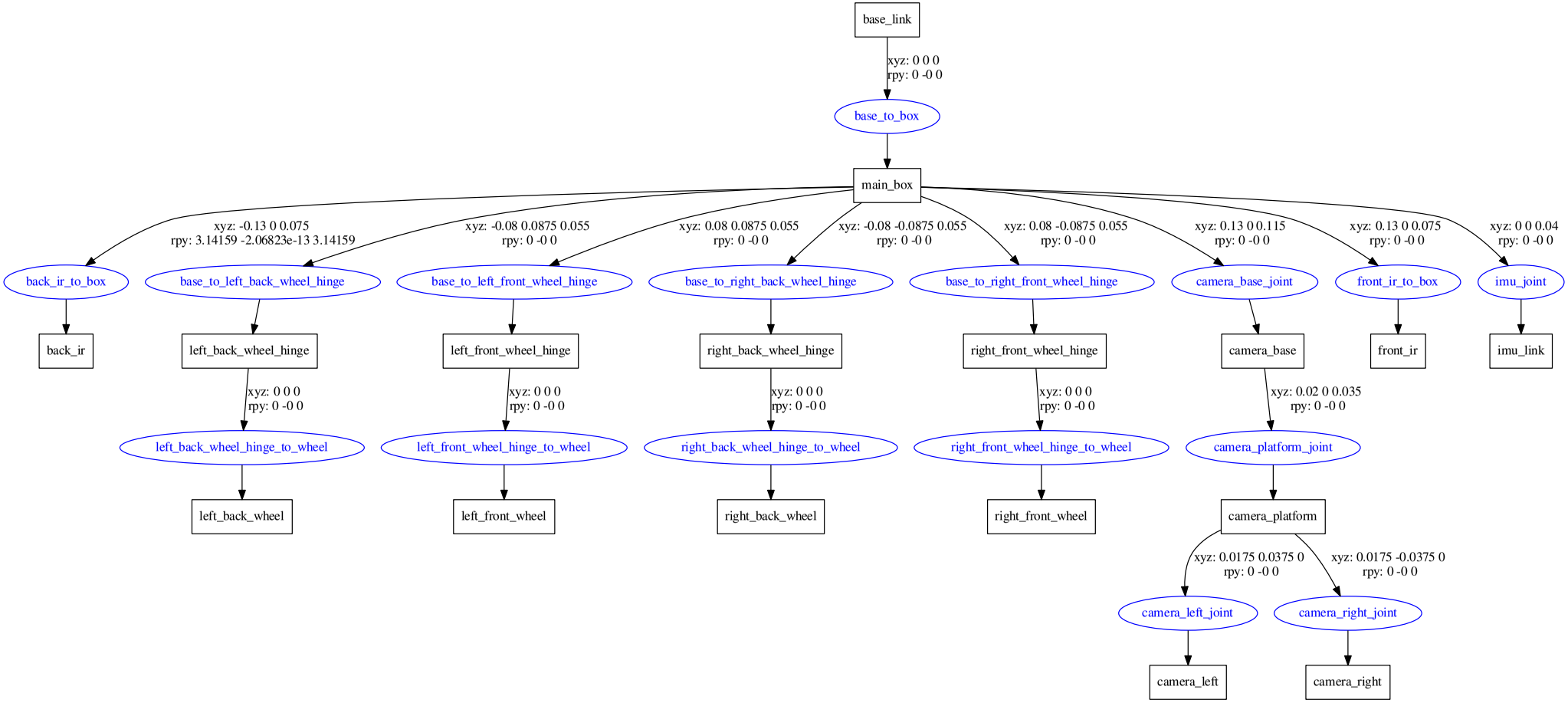
```
cd robotserver/robot-server
make robot-server
./robot-server --tty /dev/ttyS21
```

Tímto je vše spuštěno a server čeká na připojení klienta.

Příloha C

URDF vizualizace

Na další straně můžete vidět vizualizaci URDF souboru s modelem robota, vygenerovanou programem `urdf_to_graphiz`. Každý obdélník reprezentuje část robota popsanou tagem `<link>` a každá elipsa označuje spoj popsaný tagem `<joint>`. Číselné popisy mezi nimi udávají posun středů mezi dvěma částmi.


base_link
xyz: 0 0 0
rpy: 0 -0 0
base_to_box
main_box
xyz: -0.13 0 0.075
rpy: 3.14159 -2.06823e-13 3.14159
xyz: -0.08 0.0875 0.055
rpy: 0 -0 0
xyz: 0.08 0.0875 0.055
rpy: 0 -0 0
xyz: -0.08 -0.0875 0.055
rpy: 0 -0 0
xyz: 0.08 -0.0875 0.055
rpy: 0 -0 0
xyz: 0.13 0 0.115
rpy: 0 -0 0
xyz: 0.13 0 0.075
rpy: 0 -0 0
xyz: 0 0 0.04
rpy: 0 -0 0
back_ir_to_box
base_to_left_back_wheel_hinge
base_to_left_front_wheel_hinge
base_to_right_back_wheel_hinge
base_to_right_front_wheel_hinge
camera_base_joint
front_ir_to_box
imu_joint
back_ir
left_back_wheel_hinge
left_front_wheel_hinge
right_back_wheel_hinge
right_front_wheel_hinge
camera_base
front_ir
imu_link
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0 0 0
rpy: 0 -0 0
xyz: 0.02 0 0.035
rpy: 0 -0 0
left_back_wheel_hinge_to_wheel
left_front_wheel_hinge_to_wheel
right_back_wheel_hinge_to_wheel
right_front_wheel_hinge_to_wheel
camera_platform_joint
left_back_wheel
left_front_wheel
right_back_wheel
right_front_wheel
camera_platform
xyz: 0.0175 0.0375 0
rpy: 0 -0 0
xyz: 0.0175 -0.0375 0
rpy: 0 -0 0
camera_left_joint
camera_right_joint
camera_left
camera_right

Příloha **D**

Obsah přiloženého CD

```
readme.txt ..................................stručný popis obsahu CD
src
  sim ........................... zdrojové kódy implementační části
    robotserver ..............upravená verze řídícího serveru robota
    firmware-board ...........upravená verze ovládací desky robota
    fit_robot_description ......model robota pro simulátor Gazebo
    robot ............................kalibrační data pro kamery
  tex ........................zdrojová forma práce ve formátu LaTeX
    BP_BALIHAR.tex
text ..................................................... text práce
  BP_BALIHAR.pdf ......................text práce ve formátu PDF
```