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

BIOLOGICKY INSPIROVANÍ ROBOTI – BROUK

BIO-INSPIRED ROBOTS – HEXAPOD

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — Bc. Aleš Vymazal

AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. RADOMIL MATOUŠEK Ph.D.

SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav automatizace a informatiky
Student:	**Bc. Aleš Vymazal**
Studijní program:	Strojní inženýrství
Studijní obor:	Aplikovaná informatika a řízení
Vedoucí práce:	**doc. Ing. Radomil Matoušek, Ph.D.**
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 diplomové práce:

Biologicky inspirovaní roboti – brouk

Stručná charakteristika problematiky úkolu:

V práci půjde o rešerši několika typů robotů s biologickou inspirací v kontextu kinematiky brouka a vlastní realizaci zvoleného robota s využitím sytému Robotis–Bioloid, ROS a možných senzorů pro interakci robotu s okolím.

Cíle diplomové práce:

– Rešerše problematiky robotů typu hexapod.
– Využití ROS v kontextu robotu hexapod.
– Fyzická realizace robotu hexapod s využitím sytému Robotis–Bioloid.
– Návrh autonomního a distribuovaného systému řízení robotu.
– Diskuze k výsledkům a multimediální prezentace výsledků.

Seznam doporučené literatury:

ASAMA, Hajime, et al. (ed.). Distributed autonomous robotic systems 2. Springer Science & Business Media, 2013.

QUINN, Roger D.; RITZMANN, Roy E. Construction of a hexapod robot with cockroach kinematics benefits both robotics and biology. Connection Science, 1998, 10.3-4: 239-254.

HAWCOCK, Claire. Postav si svého robota: seznam se s roboty. Přeložil Kateřina BROUK. Praha: Svojtka & Co., 2017. ISBN 978-80-256-1972-8.

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

V Brně, dne

L. S.

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

Fakulta strojního inženýrství, Vysoké učení technické v Brně / Technická 2896/2 / 616 69 / Brno

ABSTRAKT

Cílem této diplomové práce bude objasnit problematiku biologicky inspirovaného robota hexapod. Teoretická část této práce stručně pojednává o fyzikální podstatě a možném využití v dnešní době. Praktická část této práce se bude zabývat řízením skutečného modelu a simulací robota hexapod. Jako fyzický model bude využit King Spider od firmy ROBOTIS a k simulaci tohoto robota poslouží framework ROS a několik dalších programovacích jazyků.

ABSTRACT

The objectice of this work is to discuss the problém of constructing a bio-inspired hexapod robot. The theoretical part of the thesis briefly summarizes the underlying physical principles and current applications of bio-inspired robots. The practical section discusses the control of a real model and simulation of an existing hexapod. The concrete robot of choice is King Spider by ROBOTIS and the simulation was carried out in the ROS software framework using multiple programming languages.

KLÍČOVÁ SLOVA

Hexapod, ROS, Gazebo, Simulace, C++, Python, Node, Service, Publisher, Subscriber, Topic, URDF, SDF, AX-12+, USB2Dynamixel, CM-530, King Spider, Robotis, Navigace

KEYWORDS

Hexapod, ROS, Gazebo, Simulation, C++, Python, Node, Service, Publisher, Subscriber, Topic, URDF, SDF, AX-12+, USB2Dynamixel, CM-530, King Spider, Robotis, Navigation

BIBLIOGRAFICKÁ CITACE

VYMAZAL, Aleš. *Biologicky inspirovaní roboti - brouk,* Brno, 2020. Diplomová práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky.

PODĚKOVÁNÍ

Děkuji rodině za podporu při zpracování a diplomové práce a také vedoucímu mé diplomové práce doc. Ing. Radomilu Matouškovi Ph.D, za návrh zajímavého podnětu pro zpracování diplomové práce

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením doc. Ing. Radomila Matouška Ph.D. a s použitím literatury uvedené v seznamu literatury.

V Brně dne 25. 6. 2020 ……

Aleš Vymazal

OBSAH

1 ÚVOD 17

2 HEXAPOD 19
2.1 Složení 19
2.2 Historie 19
2.2.1 Současné modely 20
2.3 Výhody a nevýhody hexapodu 21
2.4 Tvar končetin 22
2.5 Využití Hexapodu 22
2.6 Přímá a inverzní kinematika robotické ramene 23
2.6.1 Rozdíl mezi přímou a inverzní kinematikou 23
2.6.2 Denavit-Hartenbergova konvence a transformace souřadnic 24

3 REALIZACE FYZICKÉHO MODELU 29
3.1 Rámy FP04 29
3.2 Servomotor AX-12+ 29
3.3 USB2Dynamixel 31
3.4 CM-530 31
3.5 Knihovna Pyax12 32
3.6 Praktické problémy spojené s řízením robota King Spider 33

4 MODELOVÁNÍ ROBOTA 35
4.1 ROS 35
4.2 Základní pojmy k využití ROS 35
4.2.1 Stručný popis složek a příkazů 35
4.2.2 ROS Systém 36
4.3 ROS Topic, ROS Node a ROS Service. 37
4.3.1 ROS Topic 37
4.3.2 ROS Topic type 38
4.3.3 ROS Node 38
4.3.4 ROS Service 39
4.4 Gazebo 40
4.4.1 URDF a SDF 40
4.4.2 Mechanický prvek link 41
4.4.3 Mechanický prvek joint 41
4.5 RViz 42
4.6 Joint controller a C++ plugin 42
4.6.1 Využití existujících pluginů 43

5 UKÁZKA PRÁCE S ROS A GAZEBO 45
5.1 Vytvoření pracovního prostředí 45
5.2 Vytvoření modelu 46
5.3 Spuštění modelu v Gazebo 46
5.4 Spuštění modelu v RViz 47
5.5 Práce s ROS Topic 47
5.5.1 rostopic info/type 48
5.5.2 rostopic echo 48

5.5.3 rostopic pub 49
5.6 Vytvoření ROS Subscriber a ROS Publisher 50
5.6.1 Změna parametrů v CMakeLists.txt 50
5.6.2 Sestavení ROS Subscriber v C++ a Python 50
5.6.3 Vytvoření ROS Publisher v C++ 52
5.7 Práce s ROS Node 52
5.7.1 rosnode info 52
5.8 Práce s ROS Service 52
5.9 Vytvoření Joint controller 54

6 VYUŽITÍ ROS V KONTEXTU ROBOTA HEXAPOD 55
6.1 dynamixel_controllers 55
6.2 Navigace robota 56
6.2.1 Prohledávací algoritmy 56
6.2.2 Gmapping 56
6.2.3 tf 57
6.2.4 Další technologie kompatibilní s ROS 57

7 DISKUZE K VÝSLEDKŮM A MULTIMEDIÁLNÍ PREZENTACE 59
7.1 Tvorba simulace a modelu robota 59
7.1.1 Model robota 59
7.1.2 Catkin_workspace 59
7.1.3 Problémy spojené se simulací robota 60
7.2 Ukázky ze simulace 61
7.3 Stručný popis kódu a funkcionality 64
7.4 Užitečné tipy pro práci s ROS a Gazebo 67

8 ZÁVĚR 69

9 SEZNAM POUŽITÉ LITERATURY 71

10 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK 75

1 ÚVOD

Příroda inspiruje již dlouhou dobu vědce a inženýry k tvorbě a zlepšování autonomních robotů. Biologické organismy se vyvinuly, aby fungovaly a přežily ve světě, který se vyznačuje rychlými změnami, vysokou nejistotou a omezenou dostupností informací. S pokrokem v oblastech informačních technologií, elektroniky, řízení ale i biologie se biologicky inspirovaní roboti dovedou uplatnit v reálném světě. Hlavní využití těchto robotů se nachází v prohledávání a zdolávání míst či terénů, které jsou běžným robotům nepřístupné. Inspirace přírodou přispívá také k vytváření robustnějších a energeticky efektivnějších robotů. K nejčastějším vzorům při návrhu robotů patří hadi, mořští živočichové, dvounozí a čtyřnozí obratlovci a zejména pak hmyz.

Tato práce se bude zabývat robotickým broukem, kterému se říká latinským názvem hexapod, což v překladu znamená "šest chodidel". Teoretická část této práce stručně shrne historii, využití, výhody, nevýhody, kinematiku, možnosti řízení a složení tohoto robota. Praktická část má za úkol vytvořit řízení pro skutečného robota typu hexapod. Pro tento účel bude využit robot „King Spider„ od firmy ROBOTIS a k řízení robota bude využit programovací jazyk Python. Nejdůležitější části této práce je vytvoření simulace robota typu hexapod. Významným nástrojem pro tvorbu robotických aplikací je framework ROS (*Robot Operating System*) a open-source simulátor pro robotiku Gazebo, které budou využity i pro tento případ. Nezbytnou součástí bude vytvoření pluginů a skriptů pro ROS a Gazebo v programovacích jazycích C++, Bash a Python, ve kterém bude vytvořen navigační algoritmus k navigaci robota. Poslední část bude prezentovat dosažených cílů a problému spojených s tímto řešením.

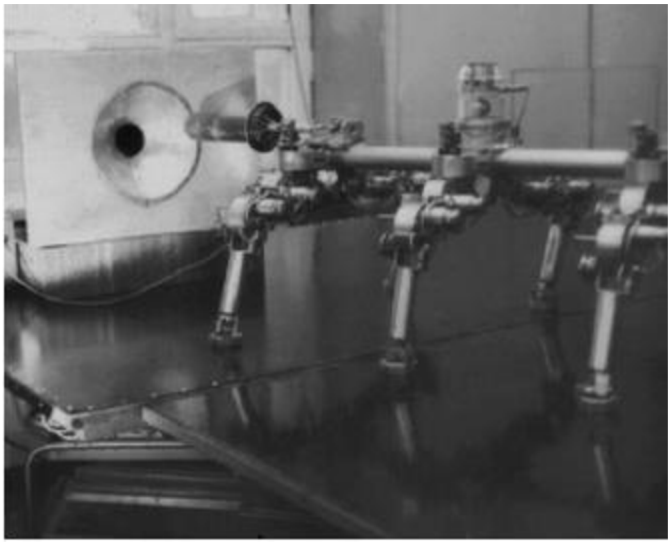
2 HEXAPOD

2.1 Složení

Každý hexapod má šest noh a tělo s řídící platformou. Nohy hexapodu se skladájí obvykle ze dvou, tří nebo čtyř servomotorů, ve výjimečných případech nohy hexapoda nemají žádný servomotor. Každý servomotor přidává hexapodu jeden stupeň volnosti. Tvar těla hexapodu je obvykle obdélníkový nebo šestiúhelníkový, méně často kruhový. Hexapod obdélníkového tvaru má nohy většinou rozložené symetricky po delších stranách, šestiúhelníkový nebo kruhový má úhel mezi každými dvěma sousedníma nohama přibližně 60 stupňů. Podle potřeby může pak hexapod být obohacen o kameru, sensory, LED diody apod. Nejčastějším možným způsobem napájení je akumulátor, některé hexapody nabízí možnost napájení přes kabel zapojený do zásuvky.[1]

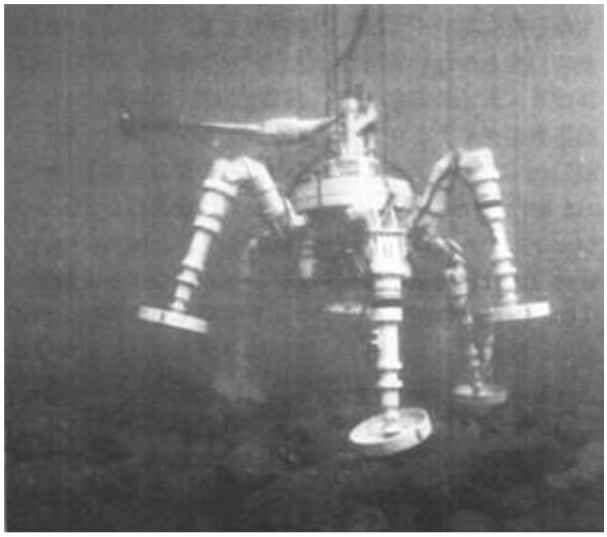
2.2 Historie

Robot hexapod je předmětem výzkumu již od padesátých let dvacátého století. Jeden z prvních hexapodů byl vytvořen v roce 1972 na římské univerzitě, jednalo se o chodící zařízení s elektrickými pohony. Pokročilejší hexapody byl vyvíjeny v druhé polovině 70. let 20. století v Moskvě. Zde se podařilo vyvinout šestinohé zařízení, podložené matematickým modelem s možností kontroly pohybu. Tento model byl pojmenován Masha a součástí jeho vybavení byl laserový dálkoměr, jenž byl spojen s řídícím systémem na vzdáleném počítači, pomocí čehož byl schopen detekovat překážky.[2]



Obrázek 1: Robot Masha [2]

Univerzitě v Ohiu se v roce 1977 podařilo zkonstruovat první hexapod schopný překonávat překážky. Na univeritě v Pittsburgu v roce 1983 byl vyvinut první hydraulicky poháněný hexapod s možností vést člověka a přitom navigovat terénem, vážil asi 800kg a byl poháněn 13kW benzinovým motorem. Od 90. let 20. století vývoj hexapodů značně zrychlil. Řídicí systémy výrazně pokročily ve složitosti a často byly instalovány k tělu hexapoda. Rozvoj v jiných oblastech inženýrství umožnil konstruovat hexapody menší, s lepšími pohybovými vlastnostmi, efektivnější, rychlejší a s možností větší kapacity nákladu. Za zmíňku stojí podvodní hexapod nazývaný AQUAROBOT vyvinutý v Japonsku. Tento hexapod sloužil k mapování podmořských profilů pro stavbu přístavů. AQUAROBOT se potápěl až do hloubek 50 m, byl plně automatizován, vybaven dálkoměrem a kamerou.[2][3]



Obrázek 2: AQUAROBOT [44]

2.2.1 Současné modely

Většina současných modelů má rozměry v řádu decimetrů, což znamená, že jejich velikost se s pokrokem doby spíše zmenšuje. Mezi firmy vyrábějící hexapody patří LynxMotion, EZ-Robot nebo Adeept. Konstrukce hexapodu je buď plastová nebo hliníková, pro řízení je využito Arduino, Raspberry pi nebo řídící jednotka jiného typu. V současně době se pohybuje cena komerčních hexapodů na trhu v rozmezí 150 Euro – 1100 Euro, v České republice se dá sehnat hexapod jen zřídka. Kromě komerčních hexapodů vznikají i hexapody od hobby skladatelů, které často překonávají hexapody komerční. K výrobě vlastního hexapoda se dá použít stavebnice Merkur nebo můžeme díly vytisknout na 3D tiskárně. Mezi dva nejpokročilejší hexapody dostupné na trhu patří tyto modely.[4]

PhantomX AX

od firmy trossenRobotics. Hlavním konstrukčním materiálem je hliník. Robot je vybaven baterií z litného polymeru, 18 servomotory a k řízení využívá Arduino. Jeho cena se pohybuje okolo $1200. V současné době patří mezi pohybově nejobratnější a nejrychlejší hexapody. Se servomotory typu AX-18 Bioloid je schopen kráčet rychlostí až 0.9m/s, umí se automaticky přizpůsobit terénu a obstojí pád z výšky několika decimetrů.[5]

Obrázek 3: PhantomX AX [5]

Hexa Hexapod Robot

od firmy Vincross. Z technologického hlediska se jedná o bezkonkurečně nejvyspělejší hexapod. Tento hexapod má v sobě zabudovanou kameru a uživatel má možnost sledovat to, co hexapod vidí. Nabízí možnost řízení přes Wi-Fi, internet nebo aplikaci v mobilu a uživatel si může naprogramovat zcela vlastní typ chůze. Obratnost tohoto hexapodu je velmi dobrá, celkem má 18 servomotorů ale rychlostně trochu zaostává zejména kvůli své menší velikosti. Cena tohoto robota se pohybuje okolo $1000.[6]

Obrázek 4: Hexa Hexapod Robot [6]

2.3 Výhody a nevýhody hexapodu

V současné době hexapod překonává svými vlastnostmi většinu biologických robotů. Zatímco co roboti s koly jsou ze všech robotů nejrychlejší, hexapod je ze všech robotů s nohami nejrychlejší a narozdíl od robotů s koly dokáže hexapod zdolávat mnohem náročnější terén s překážkami. Ukázalo se, že šest noh pro chůzi je ideální, zvýšení počtu nohou nevede ke zvýšení rychlosti.[7]

Výhody

- Většina hexapodu má schoponost se otáčet na místě.
- Hexapod může chodit i přesto, že některá z jeho noh se poškodí nebo se vyskytne drobná závada.
- Hexapod je velmi stabilní, pro udržení rovnováhy mu stačí tři nohy.

Nevýhody

- Jejich kinematika je složitá a kontrolovat pohyb hexapodu je náročné. Rychlost oproti robotům s koly je podstatně nižší
- Poměr váha : náklad je o dost horší ve srovnání s roboty s koly.

2.4 Tvar končetin

Kinematika robota se odvíjí převážně od struktury jeho noh, která je zejména funkcí jeho zamýšleného využití. Hexapod může disponovat buď biologicky inspirovanými nohami nebo uměle vytvořenými (neinspirovanými přírodou). Nohy biologicky inspirované mají charakteristiky plazů, pavoukovců nebo savců. Hlavním rysem noh plazů jsou nohy umístěny na obou koncích vyčnívajícího těla a kolena na stranu od základny, což se hodí pro překonávání náročného terénu. Nohy savců jsou pod tělem, díky čemuž se stává pohyb energeticky méně náročný, ovšem stabilita je horší. Pavoukovci disponují dlouhými nohami, přičemž jejich kolena vyčnívají nad tělem. Orientace nohou vůči tělu hexapoda určuje směr jeho chůze. Uměle vytvořené nohy můžou být kombinované tvary nohou plazů, pavoukovců nebo savců. Různé typy a tvary servomotorů můžou být taktéž důvodem pro vytvoření umělého tvaru nohou.[2]

2.5 Využití Hexapodu

Využití hexapodu jako chodícího robota je v současné době stále poměrně úzké a málo časté. Může se uplatnit pro přenos nákladu, prozkoumávání terénu nebo při záchranářských akcích, a to obzvláště v místech, kde běžní roboti nejsou schopni fungovat. Avšak mnohem častější využití nachází nechodící hexapod, nazývaný také jako Stewartova platforma, který funguje na obdobném principu kinematiky jako chodící hexapod.

Stewartova platforma je druh paralelního manipulátoru, který má šest prizmatických pohonů, obvykle hydraulických zvedáků nebo elektrických lineárních ovladačů, připevněných ve dvojicích ke třem polohám k základní desce plošiny a překračujících tři montážní body na horní desce. Všech 12 spojení je provedeno pomocí univerzálních spojů. Zařízení umístěná na horní desce se mohou pohybovat v šesti stupních volnosti, kterými disponuje volně zavěšené těleso: tři lineární pohyby x, y, z (laterální, podélný a vertikální) a tři rotace (rozteč, svitek a vybočení).
Stewartova platforma najde časté využití v průmyslu, kdy je umístěna na konci robotických ramen. Stewartova platforma nabízí velmi dobrou flexibilitu při pohyblivosti

a možnost velmi přesných operací nebo v setrváni v určité poloze po delší dobu. Pro tyto vlastnosti se take využívá Stewartova platforma ve zdravotnictví při operacích nebo pro výrobu a opravu elektroniky. Rozlehlé Stewartovy platformy se uplatňují jako pohyblivé základny pro teleskopy nebo letecké simulátory.[8][9]

Obrázek 5: Stewartova Platforma [45]

2.6 Příma a inverzní kinematika robotické ramene

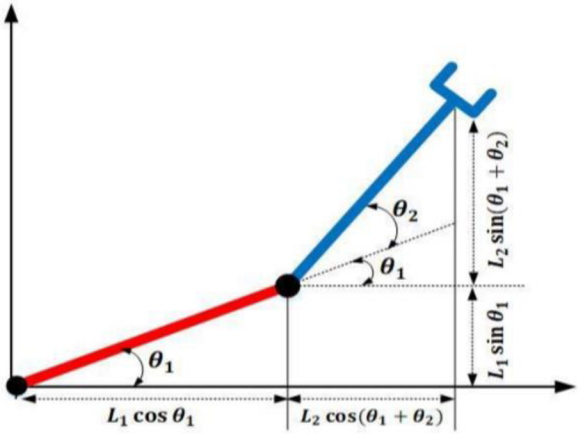
Pro určení požadované pozice robotického ramene, které v našem případě můžeme chápat jako jednu nohu hexapodu, slouží přímá a inverzní kinematika. Vzhledem k tomu, že hexapod je složen z mnoha servomotorů a má velký počet stupňů volností, tak se tato práce dále nebude zabývat konkrétním řešením pomocí inverzní nebo přímé kinematiky, ale jen poukáže na rozdíly a možnosti těchto dvou přístupů.[10]

2.6.1 Rozdíl mezi přímou a inverzní kinematikou

Přímá kinematika bere jako vstupní parametry úhly kloubů ramene, z nichž dopočítává pozici a natočení koncového bodu ramene. Přímá kinematika nabízí vždy jedinečné řešení pro určení požadované pozice robotického ramene.

Inverzní kinematika bere jako vstup souřadnice a natočení koncového bodu a podle toho dopočítává natočení úhlů robotického ramene. Na rozdíl od přímé kinematiky, postup pro získání rovnic je mnohem náročnější, protože záleží na tvaru a konfiguraci robotického ramene. Inverzní kinematika často nenabízí jedinečné řešení a pro její určení požadované pozice robotického ramene, které v našem případě můžeme chápat jako jednu nohu hexapodu, slouží přímá a inverzní kinematika.

V následujícím příkladu můžeme vidět rozdíl mezi přímou a nepřímou kinematikou. Obrázek znázorňuje robotické rameno, u kterého se snažíme dopočítat polohu koncových bodů px, py.



Obrázek 6: Příklad geometrie robotického ramene [10]

Jestliže známe vstupní hodnoty, můžeme snadno dopočítat pozici koncového bodu px, py pomocí přímé kinematiky.

$$p_x = l_1 \cos\theta + l_2 \cos(\theta_1 + \theta_2) \tag{1}$$

$$p_y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2) \tag{2}$$

Pokud známe pozici koncového bodu px, py, využijeme inverzní kinematiku pro dopočítání úhlů natočení.[10]

$$\cos\theta_2 = \frac{p_x^2 + p_y^2 - l_1^2 - l_2^2}{2 l_1 l_2} =: D \tag{3}$$

$$\sin\theta_2 = \pm\sqrt{1 - D^2} \tag{4}$$

$$\theta_2 = Atan2 \frac{\pm\sqrt{1 - D^2}}{D} \tag{5}$$

$$\theta_1 = Atan2\left(\frac{y}{x}\right) - tan^{-1} \frac{l_2 \sin\theta_2}{l_1 + l_2 \cos\theta_2} \tag{6}$$

2.6.2 Denavit-Hartenbergova konvence a transformace souřadnic

Pro popis geometrického uspořádání ramen a kloubů manipulátorů bylo zavedeno mnoho metod. Ty se pokouší jednoduchou a systematickou cestou rekurzivně definovat souřadné systémy reprezentující jednotlivá ramena manipulátoru a jejich vzájemnou polohovou

transformaci. Jedna z těchto úmluv se nazývá DH a používá se při výpočtu přímou kinematikou.

Robot je složen z mnoha kinematických dvojic, které vytváří kinematický řetězec robota. Počet těchto dvojic je určen počtem stupňů volnosti. Pro každý kinematický člen se volí souřadný systém. Ten se může obecně volit libovolně. Při libovolném zvolení souřadných systémů je ale komplikované sestavovat transformační matice pro přepočet souřadnic mezi systémy. Aby bylo nalezení těchto transformačních matic co možná nejjednodušší, je vhodné zavést pro zvolení souřadných systémů nějakou konvenci. Jednou z těchto dohod je Denavit-Hartenbergova konvence, pojmenovaná po svých zakladatelích (Denavit a Hartenberg, 1955). [11]

V následujícím příkladu můžeme vidět konkrétní využití Denavit – Hartenbergova principu na řešeném příkladu. Pro obě rotační pohybové jednotky na obrázku byly zvoleny lokální souřadné systémy, podle pravidel Denavit-Hartenbergovy konvence (pro stručnost zde tyto pravidla nejsou uvedena). Lokální souřadné systémy jsou vždy pevně spojeny s určitým tělesem a přechody mezi nimi jsou reprezentovány pomocí transformačních matic. Tyto transformační matice pro získání dalšího lokálního systému můžeme získáme řetězením pomocí operací jako rotace a translace vůči jinému lokálnímu souřadnému systému, nebo vůči celkovému globálnímu souřadnému systému. Symbolicky můžeme zapsat, že souřadnice bodu v jiném souřadném systému dostaneme pomocí vztahu (7), kde $\boldsymbol{T}$ značí transformační matici, $\boldsymbol{B}$ souřadnici bodu v existujícím souřadném systému a $\boldsymbol{A}$ souřadnici v novém souřadném systému.

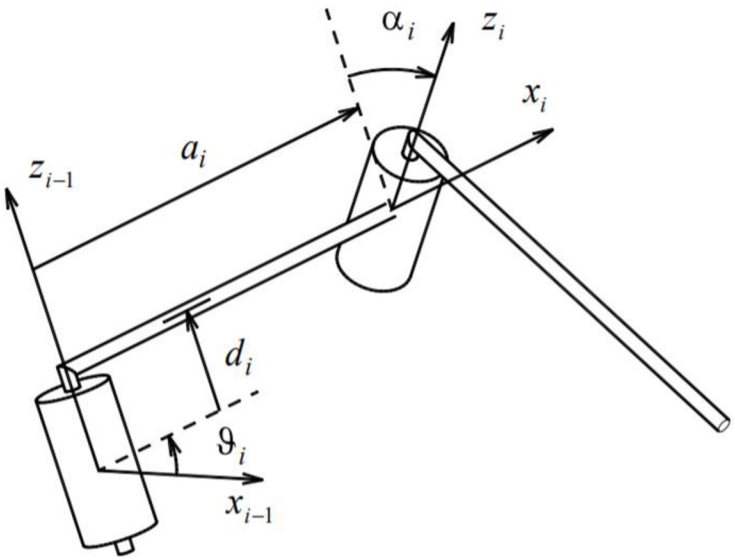
$$\boldsymbol{A} = \boldsymbol{T} * \boldsymbol{B} \tag{7}$$

Souřadnice bodu ve trojrozměrném prostoru jsou zde reprezentovány vektorem délky 4 jako

$$p = [x, y, z, 1]^T \tag{8}$$

Přidaná jednička na konci vektoru a použití čtvercových matic z $R^{4\times4}$ umožňuje jednotný zápis posuvů a rotací jako součinů matic a vektorů.

Tyto transformační vztahy také použijeme při aplikaci Denavit – Hartenbergova principu, pomocí kterého dostaneme transformační matici mezi souřadnými systémy LCS_i a LCS_{i-1}.



Obrázek 7: Transformace souřadnic podle DH principu [12]

Nejprve natočíme osu x_{i-1} kolem osy z_{i-1} o úhel ϑ_i tak, že osy x_{i-1} a x_i jsou rovnoběžné.

$$A_{z_{i-1},\vartheta_i} = \begin{bmatrix} \cos\vartheta_i & -\sin\vartheta_i & 0 & 0 \\ \sin\vartheta_i & \cos\vartheta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

Dále posuneme osu x_{i-1} ve směru osy z_{i-1} o vzdálenost d_i tak, že osy x_{i-1} a x_i jsou totožné.

$$A_{z_{i-1},d_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

V dalším kroku posuneme počátek souřadného systému LCS_{i-1} podél osy x_i o vzdálenost a_i tak, že počátky souřadných systému LCS_{i-1} a LCS_i jsou totožné.

$$A_{x_i,a_i} = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

Nyní zbývá natočit osu z_{i-1} kolem x_i o úhel α na osu z_i souřadné systémy LCS_{i-1} a LCS_i jsou totožné.

$$A_{x_i,\alpha_i} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Výsledná transformační matice mezi sousedními souřadnými systémy vznikne vynásobením jednotlivých dílčích transformačních matic v pořadí tak, jak byly prováděny pohyby

$$A_{i-1,i} = A_{z_{i-1,\vartheta_i}} * A_{z_{i-1,}d_i} * A_{x_{i1,}a_i} * A_{x_{i,}\alpha_i} \tag{13}$$

$$A_{i-1,i} = \begin{bmatrix} \cos\vartheta_i & -\sin\vartheta_i\cos\alpha_i & \sin\vartheta_i\sin\alpha_i & a_i\cos\vartheta_i \\ \sin\vartheta_i & \cos\vartheta_i\cos\alpha_i & -\cos\vartheta_i\sin\alpha_i & a_i\sin\vartheta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{14}$$

kde tzv. Denavit-Hartenbergovy parametry $\vartheta_i, d_i, a_i, \alpha_i$ plně charakterizují geometrické vztahy mezi sousedními články spojenými rotační nebo translační pohybovou jednotkou. Homogenní transformační matice dle vztahu (13) je univerzální transformační matice mezi dvěma sousedními souřadnými systémy a její výhodou je, že má stejný tvar pro všechny lokální souřadné systémy.[12]

3 REALIZACE FYZICKÉHO MODELU

Pro fyzickou realizaci robota byl využit model King Spider od firmy Robotis. Tento model se skládá z 18 servomotorů typu AX-12+, řídící jednotky CM-530, USB2Dynamixel, spojovacích kabelů a několika rámů typu FP04-F1, FP04-F2, FP04-F3, FP04-F4, FP04-F6 a BPF-ADAPTOR.

Obrázek 8: Robot King Spider [13]

3.1 Rámy FP04

Jedná se o lehké plastové rámy, které mají několik otvorů pro malé šrouby. Pomocí šroubů lze přimontovat tyto plastové rámy k sobě nebo k servomotorům. [13]

Obrázek 9: Jeden z rámů FP04 [13]

3.2 Servomotor AX-12+

Servomotor AX-12A je v současnosti nejvyužívanější model servomotoru pro příznivce robotiky. Předchůdcem servomotoru AX12-A byly typy AX-12 a AX-12+. AX-12 měl

o něco horší výkonnostní parametry a jiný převod, AX-12+ a AX-12A se liší pouze visuálním designem.

AX-12+ může sledovat aktuální polohu, rychlost otáček, teplotu servomotoru, zatížení nebo napětí. Tento servomotor by měl být napájen v rozmezí 9-12V, přičemž doporučená hodnota je 11.1V. K dispozici jsou režimy stálé rotace nebo posun v rozmezí 0°- 300°. Všechny servomotory mají své specifické ID v rozmezí 0-253 a váha jednoho servomotoru je do 60g.[14]

Obrázek 10: Dynamixel AX-12+ [47]

Komunikace mezi sběrnicí a servomotorem je zajištěna asynchronní sériovou komunikací s délkou jednoho rámce 8 bitů, 1 stop-bit a žádný paritní bit. Dále také jednotlivé typy používají half duplexní TTL nebo RS-485 signál. Servopohony Dynamixel obsahují řídící tabulku (control table), která slouží k jejich ovládání. Uchovávají se v ní veškeré informace o servopohonu, jako je například jeho ID, verze aktuálně nahraného firmwaru, aktuální poloha, cílová poloha, teplota a další.[15]

Samotné ovládání motoru je řešeno změnou hodnot v této řídící tabulce instrukčním paketem. Jeden instrukční paket obsahuje informace pro řízení jednoho nebo všech servomotorů. Jednotlivé byty pak specifikují typ operace, která se má provést.[16]

Header1	Header2	ID	Length	Instruction	Param 1	...	Param N	Checksum	
0xFF	0xFF	ID	Length	Instruction	Param 1	...	Param N	CHKSUM	

Obrázek 11: Instrukční paket

- Header1, Header2 – Pole označující začátek paketu.
- ID – Pole označující ID zařízení, které by mělo přijmout instrukční paket a zpracovat jej. Pokud se nachází hodnota tohoto bytu v rozmezí 0x00 ~ 0xFD (0-253), tak danou instrukci provede jeden servomotor. V případě, kdy je hodnota bytu 0xFE (254), tak se provede daná instrukce pro všechny servomotory.
- Length – Délka paketu (pole instrukcí, parametrů, kontrolního součtu). Délka = počet parametrů + 2
- Inctruction – Definuje požadovaný typ instrukce

- Parameters – Parametry se používají, když jsou pro instrukce požadována další data.
- Checksum – Slouží ke kontrole, zda během komunikace nebyl poškozen paket.

3.3 USB2Dynamixel

Jedná se o zařízení, které slouží k přímému řízení různých modelů servomotorů přes počítač. Do počítače se připojí přes USB a je vybaven třípinovými a čtyřpinovými konektory, proto dovede řídit různé typy servomotorů. Může také sloužit jako redukce z USB na sériové rozhraní. Před použitím USB2Dynamixel je nutné provést v počítači drobnou konfiguraci pro správné fungování. USB2Dynamixel sám o sobě nezajišťuje napájení servomotorů, a proto je nutný externí zdroj napájení.[17]

Obrázek 12: USB2Dynamixel [17]

3.4 CM-530

Slouží k ovládání servomotorů, případně přídavných senzorů. Napájení této jednotky se může uskutečnit buď přes kabel zapojený do zásuvky nebo akumulátor, který je potřeba pravidelně dobíjet. Tuto jednotku je možné testovat přes počítač ale umožňuje i nahrání programu přímo do jednotky, takže CM-530 dovede ovládat servomotory zcela nezávisle. Standardně se jednotka CM-530 programuje v jazyce embedded C s využitím RoboPlus softwaru od firmy Robotis. V této práci bude jednotka CM-530 sloužit jako zdroj napájení přes kabel a program pro řízení se bude spouštět externě v počítači v jazyku Python. Software Roboplus funguje jen pod systémy Windows, což by později neumožňovalo kompatibilitu s ROS., který je dostupný v operačním systému Linux.[18][19]

Obrázek 13: Řídící jednotka CM-530 [18]

3.5 Knihovna Pyax12

PyAX-12 je jedna z dostupných open-source knihoven, pomocí které se dají řídit servomotory AX12+ v Pythonu a i v této práci bude využita pro řízení robota KingSpider. Práce s touto knihovnou vyžaduje pouze základní znalosti programování a je snadno využitelná pro kohokoliv. Všechny metody a požadované argumenty metod jasně vystihují jejich funkcionalitu. K této knihovně je dostupná i dokumentace.

Knihovna funguje pod systémy Windows, Linux a MacOS, ovšem dostupná je pouze pro Python 3.x. Práce s knihovnou PyAX12 se mezi systémy Windows, Linux a MacOS mírně liší jen v instalaci a připojení sériového portu.[20]

Pokud pracujeme pod systémem Linux, musíme nejdříve zadat příkaz pro povolení USB portu.

```
sudo chmod 777 /dev/ttyUSB0
```

Základní metody modulu PyAX-12.

```
from pyax12.connection import Connection
# Připojení k portu /dev/ttyUSB0
serial_connection = Connection(port="/dev/ttyUSB0", baudrate=57600)
# Natočení motoru do pozice -135, parametr dynamixel_id na id servomotoru
serial_connection.goto(dynamixel_id, -135, speed=512, degrees=True)
# Tiskne informace o daném servomotoru
serial_connection.pretty_print_control_table(dynamixel_id)
```

Část výčtu tabulky po zavolání metody pretty_print_control_table(*int*).

```
model_number..... AX-12+
firmware_version..... 24
id..... 1
baud_rate..... 1000000.0 bps
return_delay_time..... 0 µs
```

```
present_speed..... 1036
present_load..... -40
present_voltage..... 12.6V
```

3.6 Praktické problémy spojené s řízením robota King Spider

Ovládání robota King Spider je spojeno s několika obtížemi, které je potřeba odstranit, aby byl zajištěn hladký chod robota. Při řízení většího množství servomotorů se zvyšuje pravděpodobnost, že nastane některá z následujících situací.

Příliš velké zatížení některého z motorů vede k ukončení celého řídícího programu. Při řízení robota King Spider se tento problém projevoval obzvláště u servomotorů, které řídily části rámů dotýkající se země nebo v případě, kdy se dostal robot do takové pozice, že jeden servomotor byl zatížen větším momentem, než je dovolen. Tento problém lze řešit změnou typu chůze nebo vypnutím limitu pro přetížení, což by ale mohlo vést ke zničení servomotoru.

Originální konstrukce tohoto robota má velmi úzké rámy dotýkající se země. To znamená, že plocha dotyku robota se zemí je velmi malá a to způsobuje klouzání na kluzkém povrchu. Tento problém se dá řešit připevněním přídavného materiálu k rámu nebo omezením testů na povrchy s dostatečným třením, ovšem v obou těchto případech může toto řešení vést k předchozímu uvedenému problému o přílišném zatížení servomotorů. Dalším řešením může být výměna těchto rámů za jiné.

4 MODELOVÁNÍ ROBOTA

4.1 ROS

ROS je open-source, meta-operační systém pro vytváření robotického softwaru. Poskytuje služby, které se dají očekávat od operačního systému, včetně hardwarové abstrakce, nízkoúrovňového ovládání zařízení, implementace běžně používaných funkcí, předávání zpráv mezi procesy a správy balíčků. Poskytuje také nástroje a knihovny pro získávání, vytváření, psaní a spouštění kódu na více počítačích. Proces v ROS se nazývá Node (uzel), přičemž každý uzel je zodpovědný za určitý úkol. Jeden ROS program může mít několik Nodes, které spolu komunikují.[21]

ROS je možno používat v operačním prostředí Linux i Windows, ale vzhledem k tomu, že ROS byl vyvinut jako první pro Linux a velká většina uživatelů a dostupných materiálů jsou dostupné jen pro Linux, tak i v této práci bude využit operační systém Linux, konkrétně Ubuntu 16.04.[21]

4.2 Základní pojmy k využití ROS

Prvním krokem k práci s ROS je vytvoření pracovního prostředí, které vývojáři ROS označují jako *catkin_workspace*. Nejdříve vytvoříme prázdnou složku (vývojáři ROS pojmenovávají tuto složku často jako catkin_ws a taktéž v této části práce budeme označovat tuto složku jako *catkin_ws*, ale jméno této složky může být libovolné) a v ní pak vytvoříme další tři prázdné složky se jmény *build*, *devel*, *src*. Jména těchto tří složek nesmí být jiná, než je uvedeno.[22]

4.2.1 Stručný popis složek a příkazů

- *catkin_ws* – Naše pracovní prostředí obsahující balíky k práci s ROS.
- *build* – Tato složka obsahuje konfigurační a registrové informace, které jsou nezbytné pro fungování našeho pracovního prostředí
- *devel* – Do této složky se ukládají přeložené zdrojové kódy ze složky *src*.
- *src* – Pro nás nejdůležitější složka při práci s ROS. V této složce vytváříme kód pro tvar robota, simulační prostředí, ovládání robota apod.
- *catkin_make* – Jestliže jsme přesměrovaní do adresáře *catkin_ws* a existují zde složky *build, devel* a *src.* tak potřebné balíčky pro práci s ROS nainstalujeme pomocí příkazu *catkin_make.* V této chvíli máme funkční prostředí pro práci s ROS a můžeme si povšimnout, že složky *build* a *devel* nyní obsahují další soubory.
- *source devel/setup.bash* – Pomocí tohoto příkazu nastavíme proměnné prostředí tak, aby mohly využívat přeložené programy ze složky *devel*.

4.2.2 ROS Systém

Cílem této části je stručný výčet a popis hlavních součástí operačního systému ROS. Text se při popisu drží původní anglické terminologie a například uzel označuje jako Node a uzly jako Nodes.

Roscore
Roscore je souhrn všech uzlů a programů, které jsou nutné pro fungování systému ROS. Aby mohly Nodes v ROS komunikovat, musí být roscore spuštěn, což můžeme uskutečnit pomocí příkazu roscore.[23]

ROS Master
ROS Master poskytuje služby pojmenování a registrace zbývajících uzlů v systému ROS. Sleduje komunikaci mezi Publisher a Subcriber (tedy uzlem, který informace vysílá a přijímá) a fungování services. Hlavní rolí ROS Master je umožnit jednotlivým uzlům ROS najít jeden druhého. Jakmile se tyto uzly lokalizují, komunikují mezi sebou navzájem. ROS Master také poskytuje server parametrů. Master se nejčastěji spouští pomocí příkazu roscore, který načte ROS Master společně s dalšími důležitými součástmi.[24]

Roslaunch
Roslaunch je nástroj pro snadné spouštění více Nodes lokálně i vzdáleně přes SSH a pro nastavení parametrů na Parameter Serveru. roslaunch vezme jeden nebo více konfiguračních souborů XML (s příponou *.launch*), které specifikují parametry pro uzly, které mají být spuštěny. Zároveň určí specifikuje systém, na kterých tyto uzly mají běžet. Syntax pro příkaz roslaunch je následující.[25]

```
roslaunch package_name file.launch
```

Následující argumenty za roslaunch označují.

- packgane_name – jméno adresáře, který se nachází ve složce *src*
- file.launch – jméno launch souboru, který se nachází ve složce *launch*, která se nachází ve složce *package.name*

ROS Subscriber
Označuje se také jako listener. Pomocí Subscriber máme možnost přijímat zprávu od Publisher.[26]

ROS Publisher
Označuje se také jako talker. Vysílá zprávu (většinou v pravidelných intervalech) s informacemi k subscriber. Publisher musí posílat relevantní data v závislosti na struktuře Subscriberu (datové typy float, string apod.)[26]

CMakeLists.txt
Je hlavní soubor pro CMake, který se používá k překladu všech programů napsaných v C nebo C++ a k instalaci programů vytvořených v Pythonu.[22]

Package.xml
Obahuje jména balíků a nástrojů, které budou využity.[22]

file.launch
Obsahuje jména Nodes, která mají být spuštěny a specifikuje různé parametry pro naši simulaci. Tento soubor je nezbytný, jestliže budeme spouštět náš ROS program pomocí příkazu roslaunch.[22]

4.3 ROS Topic, ROS Node a ROS Service.

4.3.1 ROS Topic

Při práci s ROS se až na triviální případy neobejdeme bez Topic. Zjednodušeně se dá říct, že Topic je informace nebo zpráva, pomocí níž Nodes (uzly) v ROS komunikují. Topic může být přijmán pomocí Subscriber nebo vysílán pomocí Publisher. Jeden Topic může být vysílán nebo příjmán několika Publisher a několika Subscriber zároveň, přičemž Subscriber i Publisher fungují anonymně, tzn. žádný Subscriber a Publisher neví o jiných Subscriber a Publisher.[26][27]

Dříve než začneme pracovat s Topic, musíme spustit roscore. Roscore můžeme zapnout z kteréhokoliv adresáře. Jestliže spustíme další terminál a zadáme příkaz rostopic, uvidíme seznam všech dostupných příkazů, které můžeme v rámci Topic využít. S rostopic můžeme taktéž pracovat v kterémkoliv adresáři.[27]

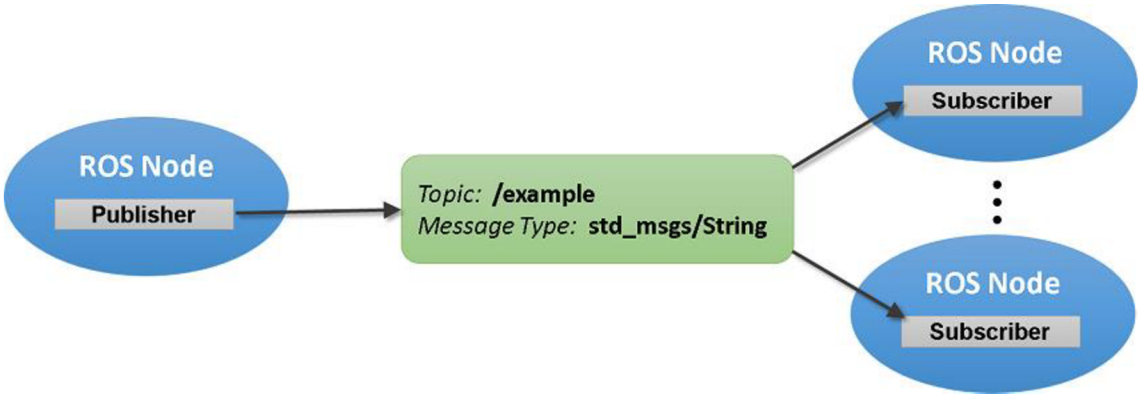
```
$ rostopic
rostopic is a command-line tool for printing information about ROS Topics.

        Commands:
          rostopic bw         display bandwidth used by topic
          rostopic delay      display delay of topic from timestamp in header
          rostopic echo       print messages to screen
          rostopic find       find topics by type
          rostopic hz         display publishing rate of topic
          rostopic info       print information about active topic
          rostopic list       list active topics
          rostopic pub        publish data to topic
          rostopic type       print topic or field type

Type rostopic <command> -h for more detailed usage, e.g. 'rostopic echo -h'
```

4.3.2 ROS Topic type

Topic má vždy určitý datový typ. Publisher a Subscriber musí znát datový typ, se kterým budou pracovat, a tento typ je odvozen z použitého datového typu Topic. Publisher a Subscriber mohou pracovat s Topic bez ohledu na programovací jazyk, který využívají (standardně C++ nebo Python), jedinou podmínkou je shoda datových typů.[27]



Obrázek 14: ROS Topic [50]

4.3.3 ROS Node

ROS Node je spustitelný program, který běží v naší ROS aplikaci. ROS aplikace může obsahovat několik Nodes. Každý z nich má obvykle na starosti specifickou funkci, např. jeden Node se stará o plánování cesty robota, další řídí pohyb robota a třetí Node kontroluje, zda robot dorazil do cíle. Node mezi sebou mohou komunikovat např. pomocí Topic. Node může být vytvořen v kterémkoliv programovacím jazyce podporovaném ROS (standardně C++ nebo Python) a může mít několik Subscriber a Publisher. Pro fungování aplikace nám teoreticky stačí jediný Node, ale z hlediska přehlednosti a složitosti je lepší jich využívat několik. Každý Node musí mít jedinečné jméno.[28][29]

Práci s Node využijeme podstatně méně než práci s Topic. Pro přehled dostupných příkazů použijeme příkaz rosnode. Obdobně jako u příkladu s Topic musíme nejdříve v jiném terminálu zapnout roscore.[28]

```
$ rosnode
rosnode is a command-line tool for printing information about ROS Nodes.

Commands:
        rosnode ping        test connectivity to node
        rosnode list        list active nodes
        rosnode info        print information about node
        rosnode machine     list nodes running on a particular machine or list machines
        rosnode kill        kill a running node
        rosnode cleanup     purge registration information of unreachable nodes

Type rosnode <command> -h for more detailed usage, e.g. 'rosnode ping -h'
```

4.3.4 ROS Service

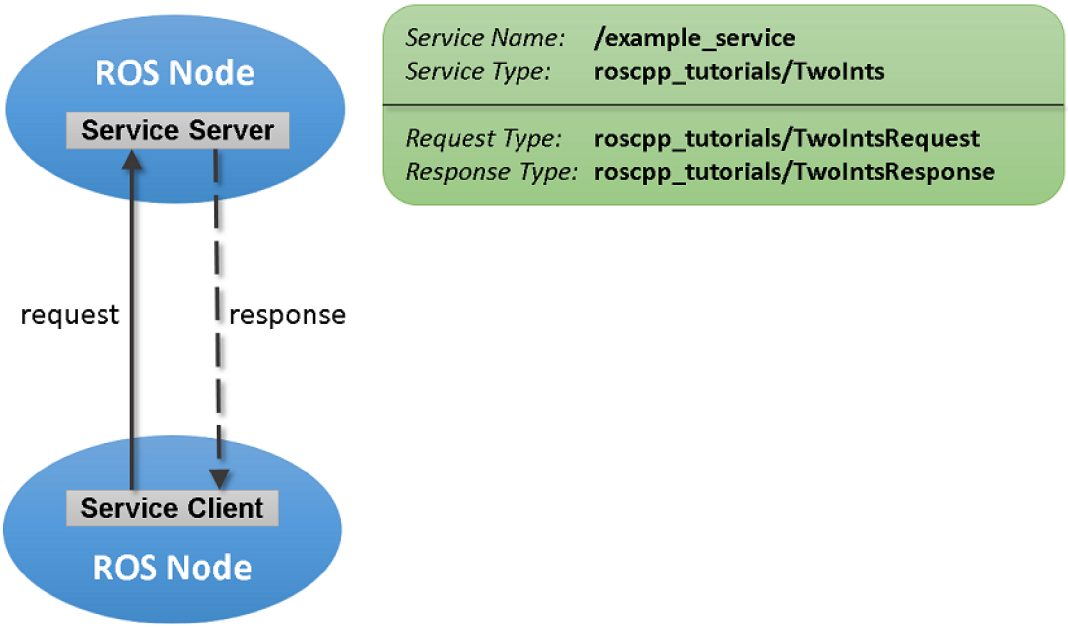
Service funguje na principu klient/server. Uživatel pošle Service data a vzápětí dostane odpověď. Druh odpovědi se řídí vždy podle dat, které uživatel odeslal. Service může být stejně jako Topic využít pro komunikaci mezi Nodes. Service fungují anonymně a může být vytvořen v kterémkoliv programovacím jazyce. Service je vhodné využít pro rychlé akce, protože v momentě, kdy uživatel pošle data, tak se Service zablokuje a čeká až uživatel dostane odpověď.[30]

Práci s Service využijeme méně než práci s Topic, přesto je dobré vědět, jak Service fungují, protože nabízí hodně již předem vytvořených Service k využití. Pro přehled příkazů použijeme příkaz *rosservice*. Postupujeme stejně jako u Topic a Node.[30]

```
$ rosservice

Commands:
        rosservice args      print service arguments
        rosservice call      call the service with the provided args
        rosservice find      find services by service type
        rosservice info      print information about service
        rosservice list      list active services
        rosservice type      print service type
        rosservice uri       print service ROSRPC uri

Type rosservice <command> -h for more detailed usage, e.g. 'rosservice call -h'
```



Obrázek 15: ROS Service [49]

4.4 Gazebo

Gazebo je robotické simulační prostředí, které slouží zejména k vytváření a simulaci jednoho nebo více robotů a simulačních prostředí.

Gazebo samo o sobě nabízí možnosti k simulaci robotů, ale často se jedná pouze o základní nebo často se vyskytující modely (robotické rameno, podvozek s dvěma koly apod.) Důležitou součástí Gazeba je možnost propojení s ROS, které nám nabízí široké možnosti pro simulaci v Gazebo. Pomocí ROS jsme schopni simulovat prakticky jakýkoliv pohyb robota. V této práci bylo využito Gazebo 7.[31]

4.4.1 URDF a SDF

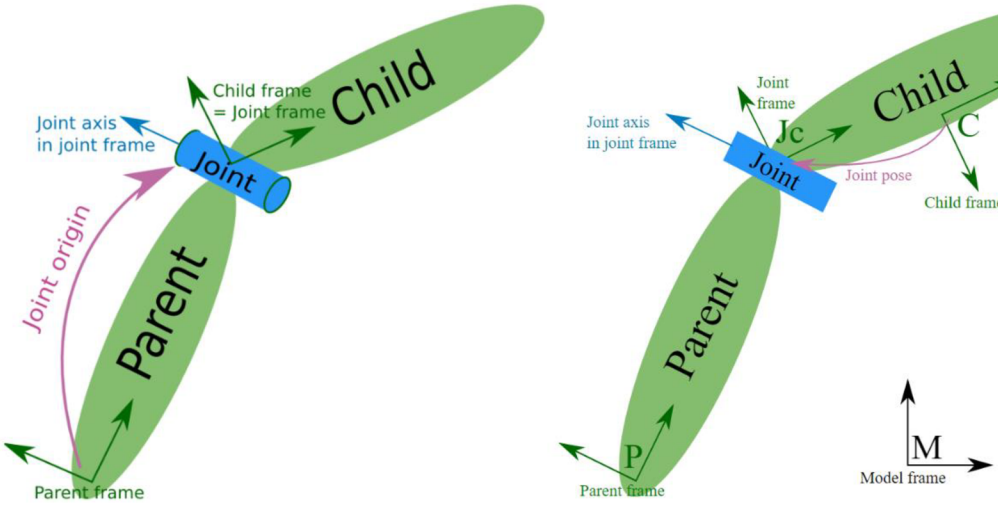
Pro modelování robotů a světů se využívají formáty URDF a SDF, které svou syntaxí připomínají jazyk XML. Principem obou těchto modelovacích formátu je sestavování modelu robota pomocí geometrických tvarů kvádr, koule a válec. Složitější tvary lze vytvořit pomocí importování souborů typu *.dae*., což nám dovoluje využívat různé 3D mechanické softwary typu Inventor, AutoCAD nebo Solidworks.[32]

URDF

Z praktického hlediska je snažší využívat formát URDF, protože pro vytvoření modelu nevyžaduje specifikaci většiny atributů a počátek každého následujícího tělesa se nachází ve výchozím nastavení v kloubu spojujícím jej s předchozím tělesem. Pro simulaci robota v Gazebo je tento typ občas nedostačující.[32]

SDF

Vyžaduje specifikaci více atributů, počátek každého tělesa se odměřuje od počátku souřadného systému tj. <0 0 0 0 0 0>. Pokud je model robota ve formátu URDF, tak je při spuštění simulace vždy automaticky převeden do formátu SDF.[32]



Obrázek 16: Formáty URDF a SDF [32]

xacro
Tento typ formátu slouží pro vytváření formátu URDF pomocí parametrů a maker. Makro může být např. definice kvádru s několika vlastnostmi a parametry můžou být jeho rozměry. Po zavolání makra se nám vytvoří daný požadavek, což nám výrazně zkrátí a zpřehlední kód.[33]

Všechny zde uvedené atributy lze specifikovat ve formátu SDF, v URDF lze specifikovat jen některé z uvedených, nejsou zde uvedeny všechny možné atributy. Atributy vztahující se k poloze neplatí pro URDF formát.

Následující dvě sekce představí základní stavební prvky robotů ve formátech SDF a URDF – link a joint.

4.4.2 Mechanický prvek link

Takto se nazývá těleso s atributy tvar, hmota, momenty setrvačnosti apod. V závorce je požadovaný datový pro tyto atributy.[34]

link atributy
<*name*> Jméno link. (string)
<*gravity*> Specifikuje, zda na těleso bude působit gravitační síla. (double)
<*pose*> Specifikuje polohu vůči bodu 0 0 0 0 0 0 (počátek souřadného systému). První tři body specifikují vzdálenost na osách x, y, z a poslední tři body specifikují natočení link. (6x double)
<*inertial*> Specifikuje hmotností vlastnosti link.
  <*mass*> Hmota link (double)
  <*inertia*> Specifikuje momenty setrvačnosti (6x double)
<*collision*>Specifikuje vlastnosti při dotyku s jiným tělesem, standardně se shoduje s vlastnostmi uvedenými výše.
<*visual*> Specifikuje visuální vlastnosti při simulaci. Standardně se shoduje s vlastnostmi definovanými dříve.

4.4.3 Mechanický prvek joint

Spojí dva links dohromady, určitým typem joint (kloub, bod otáčení). V závorce je požadovaný datový pro tyto atributy.[35]

Druhy joints
fixed – nepohyblivý joint, vytvoří jednu součást s parent link.
revolute – pohyblivý kloub, otáčí se okolo z jedné os x,y, z v určitém rozsahu.
continuous - pohyblivý kloub, otáčí se okolo jedné z os x,y, z.
prismatic – posuvný kloub, klouže po jedné z os x,y,z.

joint atributy
<*parent*> – rodičovský link, který se vůči podřazenému prvku (child link) nepohybuje. První link má joint
v bodě <0 0 0 0 0 0> (double)
<*child*> – pohybuje se okolo parent link, přičemž joint je by default ve stejném místě jako joint parent link, ale lze ho posunout kterýmkoliv směrem. (string)
<*axis*> - Specifikuje osu/osy, okolo které se bude child link vůči parent link otáčet. (6x double)
<*pose*> - Specifikuje polohu, vůči bodu <0 0 0 0 0 0>.
<*dynamics*> Specifikuje fyzické vlastnosti kloubu
<*damping*> Koeficient tlumení, na child link musí být vyvinuta větší síla, aby se link pohnul. (double)
<*friction*>Tření (double)
<*limit*> Specifikuje limity kloubu.
<*lower*> (double)
<*upper*> Tvoří dolní a horní limit úhlu, ve kterém se může child link otáčet. (double)
<*effort*> Limituje maximální sílu, která může působit na child link (double)
<*velocity*> Limituje maximální rychlost, kterou se pohybuje child link. (double)
<*stiffness*> Tuhost kloubu. (double)
<*dissipation*> Rozptýlení (double)

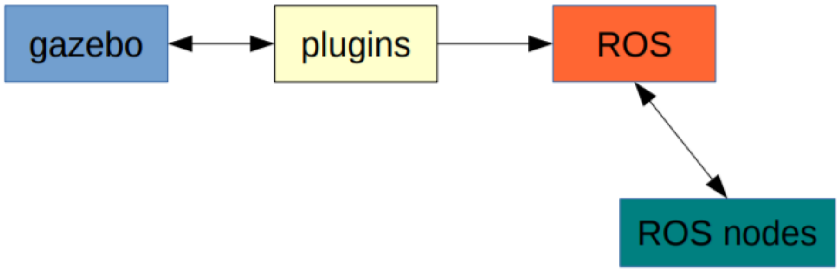
4.5 RViz

RViz je vizualizační nástroj, ve kterém můžeme sestavovat robotické modely. Poskytuje celkový náhled na robota, vlastností kloubů nebo může zachycovat data ze sensorů.[36]

4.6 Joint controller a C++ plugin

Pro simulaci pohybu robota v Gazebo si můžeme zvolit jeden ze dvou způsobů. V obou případech, tj. buď při Joint controlleru nebo vlastního C++ plugini se propojí ROS a Gazebo a to nám nabízí možnost ovládat joints a links v reálném čase. Joint controller funguje na principu využití již existujícího pluginu a vytvoření konfiguračního souboru .yaml, který specifikuje pro zvolené prvky joints nebo links způsob, kterým se mají pohybovat. Pomocí vlastního C++ pluginu docílíme totéž, avšak C++ plugin nabízí větší možnosti a flexibilitu pro ovládání robota. Oba tyto způsoby využívají pro specifikaci pohybu princip PID kontroléru. Parametry při specifikaci PID určují, jaká síla musí být vyvinuta pro uvedení do pohybu, plynulost, rychlost a překmit pohybu. Pokud si zvolíme pro ovládání robota Joint controller, tak musíme použít formát typu URDF. Do našeho modelu robota v souboru URDF pak nakonec zahrneme značky <*gazebo*>, v nichž pak specifikujeme konkrétní pohyb robota. Spuštění Joint controlleru pak provedeme pomocí *.launch* souboru. Pokud využijeme C++ plugin, tak musíme pracovat s formátem typu

SDF. Po vytvoření C++ pluginu musíme provést kompilaci tohoto souboru (podobně jako u C++ Publisher, Subscriber) a název tohoto zkompilovaného souboru potom vložíme do našeho SDF souboru.[37]



Obrázek 17: Plugin pro Gazebo [37]

4.6.1 Využití existujících pluginů

ROS nabízí k využití již předem několik vytvořených pluginů, které lze použít. Jedním z nejvíce využívaných je *differential drive* plugin, který slouží k ovládání robotů s koly. Pro zprovoznění tohoto pluginu potřebujeme určit parametry např. rozměry robota, požadované zrychlení apod. Ve výsledků jsme schopni dosáhnout prakticky jakéhokoliv námi požadovaného pohybu. V praktické části této práce bude pak ukázka využití pluginu pro kameru a scanner.[37]

5 UKÁZKA PRÁCE S ROS A GAZEBO

Model vytvořený v Gazebo lze testovat ve virtuální laboratoři s pomocí ROS a Gazebo. Cílem této kapitoly je praktická ukázka tohoto pracovního postupu. Zdrojový kód k tomuto příkladu lze nalézt v příloze.

5.1 Vytvoření pracovního prostředí

Kdekoliv si vytvoříme prázdný adresář a v něm vytvoříme tři prázdné složky *src, devel, build*.

```
$ catkin_make -p catkin_ws/{src,devel,build}
```

Dále přejdeme do složky *src*, kde pomocí příkazu *catkin_create_pkg example rospy roscpp* vytvoříme balík, pomocí kterého budeme moct spustit model v Gazebo a následně jej ovládat.

```
$ cd catkin_ws/src
~/catkin_ws/src$ catkin_create_pkg example rospy roscpp
```

Můžeme si povšimnout, že ve složce src se vytvořil balík s názvem example, ve kterém se nachází Package.xml, CMakeLists.txt a složky include a src. Pro vytvoření balíku by nám postačil pouze příkaz *catkin_create_pkg example*, ovšem argumenty rospy a roscpp vytvoří CMakeLists.txt nastavené tak, abychom mohli přidávat programy v jazycích Python a C++. V dalším kroku se přepneme do složky example, kde vytvoříme složky launch, world a sdf, do kterých později umístíme soubory nutné pro vytvoření a spuštění modelu. Do těchto adresářů budeme umisťovat soubory typu *.launch*, *.world* a *.sdf*.

```
~/catkin_ws/src/example$ mkdir -p launch worlds sdf
```

Složka launch smí mít pouze tento název, protože ROS vyhledává pomocí příkazu roslaunch spouštěcí soubory ve složce launch. Ostatní složky mohou mít jiná jména, ale výše uvedené pojmenování je standardní.

Nakonec se přepneme do složky catkin_ws a pomocí příkazu catkin_make vytvoříme kompletní pracovní prostředí.

```
~/catkin_ws$ catkin_make
```

5.2 Vytvoření modelu

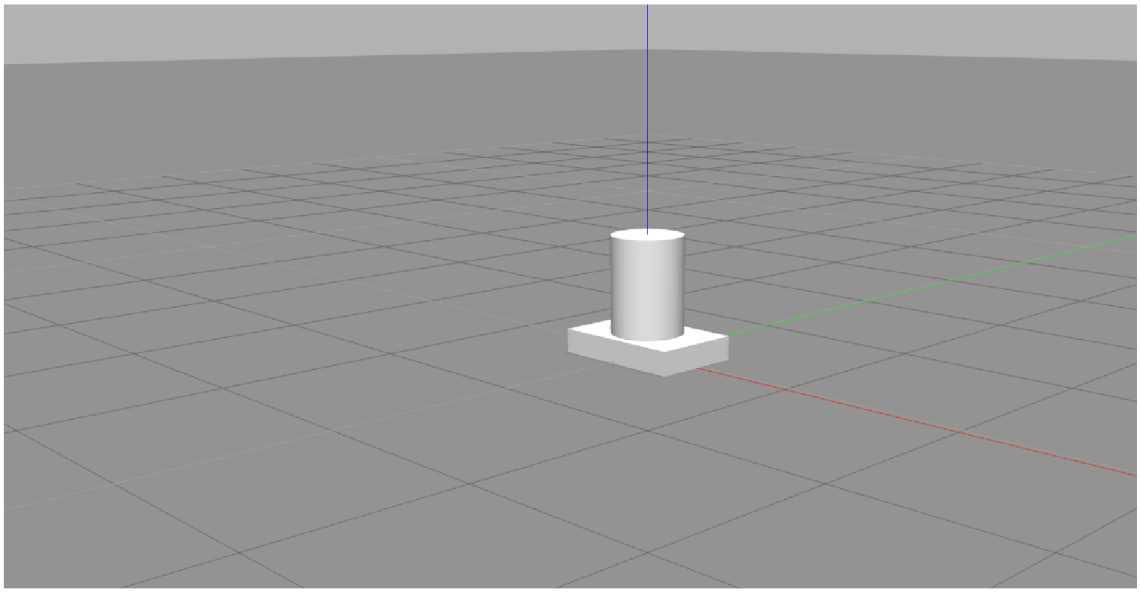
Pro vytvoření modelu můžeme použít formát SDF nebo URDF. Vzhledem k tomu, že se jedná o velmi jednoduchý model, využijeme formát URDF a pro ukázku ho následně převedeme do formátu SDF. Tento příklad znázorňuje model kvádru, na kterém stojí válec. Význam jednotlivých značek (tagů) je vysvětlen v kapitole 4. Model můžeme vytvářet v kterémkoliv textovém editoru, koncovka tohoto souboru musí být ovšem .urdf. Soubor example.urdf vložíme do složky sdf a do formátu .sdf ho převedeme příkazem.

```
~/catkin_ws/src/example/sdf$ gz sdf --print example.urdf > example.sdf
```

5.3 Spuštění modelu v Gazebo

Abychom mohli model spustit, potřebujeme .launch soubor a ‘svět‘ – prostředí, ve kterém se bude robot pohybovat. Můžeme buď využít připravený svět z Gazebo serveru nebo si vytvořit vlastní. Pro využití vlastního světa vytvoříme jednoduchý .world soubor a vložíme ho do složky world. Dále je pak nutné vytvořit .launch soubor, který zahrnuje cestu ke světu, modelu a balíku který chceme využít. Standardním způsobem se využívají dva .launch soubory, přičemž jeden slouží jako šablona a druhý poskytuje skutečné parametry pro spuštění. Oba tyto .launch soubory umístíme do složky launch. Nakonec se přepneme do složky catkin_ws a proměné prostředí přesměrujeme pomocí příkazu *source devel/setup.bash*. Celý model pak nastavíme pomocí příkazu *roslaunch example spawn.launch*.

```
~/catkin_ws$ source devel/setup.bash
~/catkin_ws$ roslaunch example spawn.launch
```

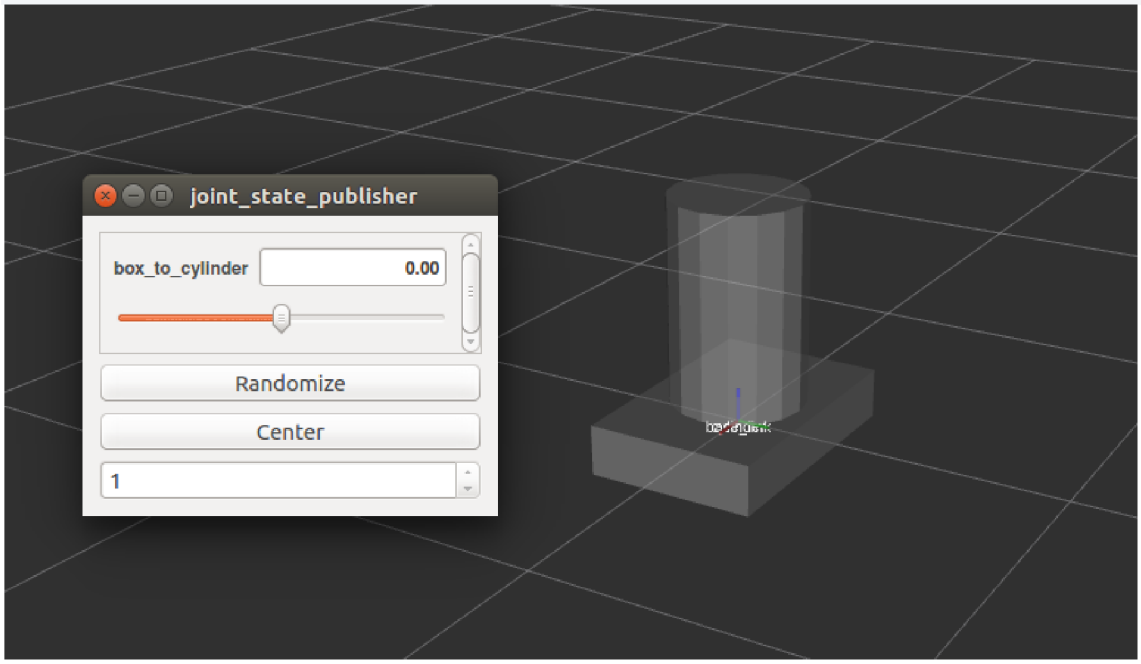


Obrázek 18: Model Example v Gazebo

5.4 Spuštění modelu v RViz

Jestliže potřebujeme pro náš model využít práci s RViz, budeme potřebovat další .launch soubor. Tento .launch soubor funguje na stejném principu, jako předchozí .launch soubory, jen specifikace parametrů je rozdílná. Model v RViz spustíme pomocí příkazu.

```
roslaunch example rviz.launch
```



Obrázek 19: Model Example v RViz

5.5 Práce s ROS Topic

Jestliže se nám podařilo spustit daný model, můžeme na něm demonstrovat práci s Topic. Pokud otevřeme další terminál a zadáme příkaz rostopic list, uvidíme seznam všech dostupných Topic, se kterými můžeme pracovat. V následujících příkladech budeme pracovat s Topics ModelStates, LinkState, které slouží pro zjištění polohy modelu, a SetModelState, které slouží pro přemístění modelu do jiného místa.[27]

```
~/catkin_ws$ rostopic list
/clock
/gazebo/link_states
/gazebo/model_states
/gazebo/parameter_descriptions
/gazebo/parameter_updates
/gazebo/set_link_state
/gazebo/set_model_state
/gazebo_gui/parameter_descriptions
/gazebo_gui/parameter_updates
/rosout
```

```
/rosout_agg
```

Následující sekce vysvětlí nejvíce využívané argumenty příkazu rostopic, které usnasňuj í práci při pohybu a testování modelu robota.

5.5.1 rostopic info/type

Příkaz *rostopic info* se hodí pro získání datového typu daného Topic a zjištění, zda je Topic využíván některým Subscriber nebo Publisher. Příkaz *rostopic type* zobrazí informaci jen o datovém typu daného Topic.[27]

```
~/catkin_ws$ rostopic info /gazebo/model_states
Type: gazebo_msgs/ModelStates

Publishers:
 * /gazebo (http://alda-Inspiron-5537:44219/)

Subscribers: None
```

5.5.2 rostopic echo

Využijeme pro Topic, který je využíván některým Publisher. Periodicky v řádu setin sekund zobrazuje Topic, který daný Publisher vysílá. Pokud tento příkaz využijeme u Subscriber, nedostaneme žádnou odezvu.[27]

```
~/catkin_ws$ rostopic echo /gazebo/link_states
name: ['ground_plane::link', 'example::base_link', 'example::cylinder']
pose:
 -
  position:
    x: 0.0
    y: 0.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 1.0
 -
  position:
    x: -5.10284918448e-12
  ..........
```

5.5.3 rostopic pub

Vytvoří dočasný Publisher s daným Topic pro určitý Subscriber. *Rostopic pub* můžeme využít i pro jiný Pubslisher, ale nebude to mít žádný význam. Pokud zadáme příkaz *~/catkin_ws$ rostopic pub /gazebo/set_model* a budeme mačkat tabulátor, tak se nám automaticky doplní předvyplněné argumenty, které daný Subscriber očekává.[27]

```
~/catkin_ws$ rostopic pub /gazebo/set_model_state gazebo_msgs/ModelState "model_name:
'example'
pose:
  position:
    x: 2.0
    y: 1.0
    z: 0.0
  orientation:
    x: 0.0
    y: 0.0
    z: 0.0
    w: 0.0
twist:
  linear:
    x: 0.0
    y: 0.0
    z: 0.0
  angular:
    x: 0.0
    y: 0.0
    z: 0.0
reference_frame: ""
publishing and latching message. Press ctrl-C to terminate
```

Pokud teď zadáme příkaz *rostopic info /gazebo/set_model_state*, můžeme vidět, že daný Topic je využíván Subscriber a Publisher.

```
 ~$ rostopic info /gazebo/set_model_stat
Type: gazebo_msgs/ModelState

Publishers:
 * /rostopic_5530_1591870444418 (http://alda-Inspiron-5537:43363/)

Subscribers:
 * /gazebo (http://alda-Inspiron-5537:44219/)
```

5.6 Vytvoření ROS Subscriber a ROS Publisher

V dalších podkapitolách se budeme zabývat vytvořením vlastních ROS programů, které implementují Subscriber a Publisher. V předchozích podkapitolách jsme mohli vidět, že práce s Topic a Service pomocí terminálu se může hodit pro jednoduché operace. Motivace pro tvorbu vlastních ROS programů může být taková, že v těchto programech můžeme konkrétněji specifikovat operace v závislosti na datech. Příkladem může být naprogramované chování robota, který změní směr pohybu, když se ocitne na určité souřadnici.

5.6.1 Změna parametrů v CMakeLists.txt

Jestliže chceme použít kterýkoliv program v rámci ROS, musíme nejdříve provést změny parametrů v CMakeLists.txt, aby se mohla provést správně kompilace souborů. Jedinou výjimkou jsou Subscriber v Pythonu, pro které není nutné provádět žádné změny CMakeLists.txt. V CMakeLists.txt odstraníme komentáře u příkazů `add_executable()`, `add_dependencies()` a `target_link_libraries()` a doplníme jednotlivé argumenty těchto příkazů.[38]

- `add_executable(pose include/pose.cpp)` – První argument značí název spustitelného souboru, druhý argument představuje cestu k souboru, který má být zkompilovaný.
- `add_dependencies(pose ${pose_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})` – Označuje pořadí kompilace souborů závislých pro zkompilování našeho požadovaného souboru (v našem případě tento řádek nemá zásadní význam.)
- `target_link_libraries(pose ${catkin_LIBRARIES})` - Vytvoří knihovnu použitelnou ROS z dříve přeloženého `.cpp` souboru

5.6.2 Sestavení ROS Subscriber v C++ a Python

V následujícím příkladu využijeme již existující Publisher. Při práci s ROS je vždy nutné importovat knihovnu ros/ros.h pro C++ a modul rospy pro Python a další knihovny podle potřeby naší práce.

Pro sestavení Subscriber, který přijímá Topic od Publisher, potřebujeme znát datový typ daného Publisheru, což můžeme zjistit pomocí příkazu *rostopic type*. Podle tohoto datového typu naimportujeme knihovnu, bez které by program nemohl používat Topic od daného Publisher. Dalšími nezbytnými náležitostmi jsou funkce pro inicializaci Node a Subscriber, který se volá periodicky každých několik setin sekund. Jako parametry bere tato funkce název Publisher, od kterého přijímá Topic, a referenci na callback funkci, která bude vykonávat námi požadovanou funkcionalitu. Funkce spin uvede celý program do smyčky, který můžeme vypnout pomocí ctrl+c. Program v C++ zkompilujeme pomocí příkazů.[38]

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

Abychom mohli spustit program v Pythonu, použijeme příkaz.

```
~/catkin_ws/src/example/include$ sudo chmod +x pose.py
```

Následující dvě ukázky kódu mají stejnou funkcionalitu, do terminálu tisknou informaci o aktuální poloze modelu. První příklad je v C++ a druhý v Pythonu.

Ukázka v C++

```
#include "ros/ros.h"
#include "gazebo_msgs/ModelStates.h"   // → rostopic type /gazebo/model_state
#include <sstream>
#include <iostream>

void chatterCallback(const gazebo_msgs::ModelStates::ConstPtr& msg) // callback funkce
{
 for(auto it = msg->name.begin(); it != msg->name.end(); ++it)
 {
  if((strcmp((*it).c_str(), "example") == 0))
  {
   int index = int((std::distance(msg->name.begin(), it)));
   ROS_INFO("Coordinates of model\n x: [%f]\n y: [%f]\n z: [%f]\n",
    msg->pose[index].position.x,msg->pose[index].position.g->pose[index].position.z);
  }
 }
}
int main(int argc, char **argv)
{
 ros::init(argc, argv, "listener");    // inicializace Node
 ros::NodeHandle n;
 ros::Subscriber sub = n.subscribe("/gazebo/model_states", 1000, chatterCallback);
 ros::spin(); //Nekonečná smyčka
 return 0;
}
```

Stejný algoritmus v jazyce Python:

```
#!/usr/bin/env python
import rospy
 from gazebo_msgs.msg import ModelStates   # → rostopic type /gazebo/model_state

def chatterCallback(msg):            # callback funkce
{
  rospy.loginfo(msg.pose[msg.name.index('example')].position)

if __name__ == '__main__':
 rospy.init_node('listener', anonymous=True)            # inicializace Node
 rospy.Subscriber("/gazebo/model_states", ModelStates, chatterCallback)
 rospy.spin()            #Nekonečná smyčka
```

Již na první pohled lze vidět, že při práci s ROS je mnohem snadnější využívat Python než C++. Pro kompilaci ROS souborů využívající C++ musíme vždy dělat změny v CMakeLists.txt. Další výzvou při práci s C++ v ROS jsou datové typy, nejen že musíme znát práci se standardními datovými typy C++, ale musíme také vědět, jak uchopit datové typy z knihovny ros/ros.h. Hlavní výhodou C++ oproti Pythonu je rychlost.

5.6.3 Vytvoření ROS Publisher v C++

Postup pro vytvoření vlastního Publisheru je stejný jako u vytváření vlastního Subscriberu. Nejdříve musíme provést potřebné změny v CMakeLists.txt tzn. přidáme příkazy, které budou sloužit jako vstup pro CMake.[38]

- `add_executable(pub include/pub.cpp)`
- `add_dependencies(pub ${pub_EXPORTED_TARGETS} ${catkin_EXPORTED_TARGETS})`
- `target_link_libraries(pub ${catkin_LIBRARIES})`

Princip funkcí, které jsou zde využity je podobný jako u námi vytvořeného Subcriberu v předchozí podkapitole. Tento Publisher bude vysílat námi požadovanou zprávu. Význam jednotlivých funkcí je popsán v komentářích programu.

5.7 Práce s ROS Node

Práce s Node nám může posloužit k získání informací o funkcionalitě programu. Příkaz *rosnode list* zobrazí všechny aktivní Nodes, které lze využít.[28]

```
~/catkin_ws$ rosnode list
/gazebo
/gazebo_gui
/rosout
```

5.7.1 rosnode info

Příkaz *rosnode info* nám vypíše všechny Publisher, Subscriber a Service a jejich datové typy, které jsou v daném Node použity.

```
~$ rosnode info /gazebo
Publications:
 * /clock [rosgraph_msgs/Clock]
 …
Subscriptions:
 * /clock [rosgraph_msgs/Clock]
 …
Services:
 * /gazebo/apply_body_wrench
  …
```

5.8 Práce s ROS Service

Service nabízí spoustu možností při práci s ROS a Gazebo. Příkaz *rosservice list* zobrazí všechny aktivní service, které lze využít.[30]

```
~$ rosservice list
/gazebo/apply_body_wrench
/gazebo/apply_joint_effort
/gazebo/clear_body_wrenches
/gazebo/clear_joint_forces
/gazebo/delete_light
/gazebo/delete_model
/gazebo/get_joint_properties
/gazebo/get_light_properties
/gazebo/get_link_properties
/gazebo/get_link_state
/gazebo/get_loggers
....
```

Jedním z mnoha příkladů využití Service může být změna souřadnic modelu. Zde si můžeme povšimnout, že jsme docílili stejného efektu, jako při práci s *rostopic pub* v podkapitole 5.4.3, nicméně pro přesunutí modelu jsme potřebovali aktivní Subscriber i Publisher. V tomto příkladě nám postačil jeden objekt typu Service. Pro získání předvyplněných argumentů můžeme opět použít tabulátor.

```
~/catkin_ws$ rosservice call /gazebo/set_model_state "model_state:
 model_name: 'example'
 pose:
   position:
     x: -2.0
     y: -1.0
     z: 0.0
   orientation:
     x: 0.0
     y: 0.0
     z: 0.0
     w: 0.0
 twist:
   linear:
     x: 0.0
     y: 0.0
     z: 0.0
   angular:
     x: 0.0
     y: 0.0
     z: 0.0
 reference_frame: ""
success: True
```

```
status_message: "SetModelState: set model state done"
```

Dalším příkladem může být service, který informuje o vlastnostech určitého joint.

```
~/catkin_ws$ rosservice call /gazebo/get_joint_properties "joint_name: 'box_to_cylinder'"
type: 0
damping: []
position: [2.23132534671322e-08]
rate: [5.324767931153024e-12]
success: True
status_message: "GetJointProperties: got properties"
```

5.9 Vytvoření Joint controller

Pro vytvoření reálného pohybu v simulaci potřebujeme zahrnout do URDF souboru značky *<gazebo>* a do nich uvést námi požadovaný typ pohybu. Do souboru .launch přidáme parametry pro spuštění Joint Controlleru.[37]

```
example:
joint_state_controller:
type: joint_state_controller/JointStateController
publish_rate: 20

base_to_second_joint_position_controller:
type: effort_controllers/JointPositionController
joint: box_to_cylinder_joint
pid: {p: 1.0, i: 1.0, d: 0.0}
```

Pozn. Joint controller často vyžaduje instalaci dalších balíčků a někdy způsobuje problémy při spouštění, proto veškeré náležitosti tykající se Joint controller jsou zakomentované, takže simulace se pouští bez Joint controller.

6 VYUŽITÍ ROS V KONTEXTU ROBOTA HEXAPOD

V kontextu řízení robota typu hexapod můžeme ROS využít kromě simulování robota také pro ovládání servomotorů, navigaci robota a dalších technologií kompatibilních s ROS. Níže zmíníme některé aplikace ROS v těchto oblastech. Některé nástroje jsou dostupné pouze pro určité distribuce ROS.

6.1 dynamixel_controllers

Tento balíček obsahuje nízkoúrovňové řízení pro servomotory od firmy Robotis Dynamixel. Plně podporuje modely typu AX-12, AX-12+, AX-18, RX-24, RX-28, MX-28, RX-64, EX-106. Hardwarově specifické konstanty jsou definovány pro čtení a zápis informací z Dynamixel servomotorů. Tento balíček nízké úrovně většina uživatelů nepoužívá, protože spousta alternativ pro řízení Dynamixel servomotorů je znatelně jednodušší, viz kapitola 3. Dynamixel_controllers má smysl využít jen ve specifických případech.

Balíček *dynamixel_controllers* obsahuje konfigurovatelný uzel, služby a skript pro vytvoření, spuštění, zastavení a restartování jednoho nebo více pluginů řadiče. Typy opakovaně použitelných ovladačů jsou definovány pro běžné klouby servomotorů Dynamixel. Pro každý kloub lze nastavit rychlost i točivý moment.[39]

Pro práci a řízení servomotorů Dynamixel vytvoříme pracovní prostředí dynamixel_ws.

```
$ mkdir -p dynamixel_ws/{src,devel,build}
```

Do složky src naklonujeme potřebné soubory.

```
~/dynamixel_ws/src$ clone https://github.com/arebgun/dynamixel_motor.git
```

Pomocí příkazu catkin_make vytvoříme kompletní pracovní prostředí.

```
~/dynamixel_ws/$ catkin_make
```

Nastavíme proměnné prostředí.

```
~/dynamixel_ws/$ source devel/setup.bash
```

Pro ovládání servomotorů využijeme zejména práci s Topic, Service a *.launch* soubory, viz kapitola 4 a 5.

6.2 Navigace robota

Pro navigování robota máme hned několik možností, které se dají využít. Nezbytností pro navigaci je využití jednoho nebo více sensorů.

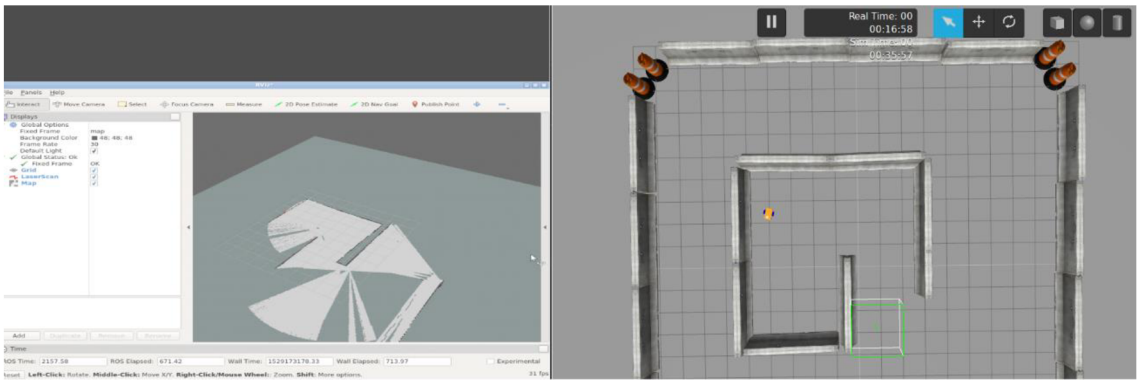
6.2.1 Prohledávací algoritmy

Prvním způsobem, který se pro navigaci nabízí je naprogramování vlastního prohledávacích algoritmů typu A*, BFS, DFS nebo Dijkstrova algoritmus. Tyto algoritmy jsou schopné efektivně najít průchod bludištěm, ovšem hlavním nedostatkem při využití simulátoru Gazebo je neschopnost získání bodové reprezentace mapy.

6.2.2 Gmapping

Gmapping je balík, který obsahuje několik Nodes, pomocí kterých jsme schopni vytvořit bodovou reprezentaci mapy. Podstatou tohoto balíku je sbírání dat ze sensoru LaserScan, tzn., že existující robot musí mít na své konstrukci připevněn funkční scanner. Prvním krokem k vytvoření takové mapy je spuštění robota v simulátorech Gazebo a RViz. Dále se pak snažíme pohybovat s naším robotem danou mapu tak, abychom pomocí skeneru získali digitální model co největší části naší mapy, přičemž si můžeme všimnout, že z prázdného prostředí v RViz se nám vytvořily obrysy hranic naší skutečné mapy. Z této vytvořené mapy pak můžeme generovat obrázek ve formátu *.png* a data o struktuře mapy ve formátu *.yaml*. Tyto dva soubory jsou nezbytné pro vytvoření navigace robota z jednoho bodu do druhého. K navigaci pak potřebujeme ještě *map_server*, který je automaticky dostupný po instalaci ROS. *Map_server* bere jako jeden ze svých parametrů zmíněné soubory *.yaml* a *.png*, pomcí nichž vytvoří bodovou reprezentaci mapy.

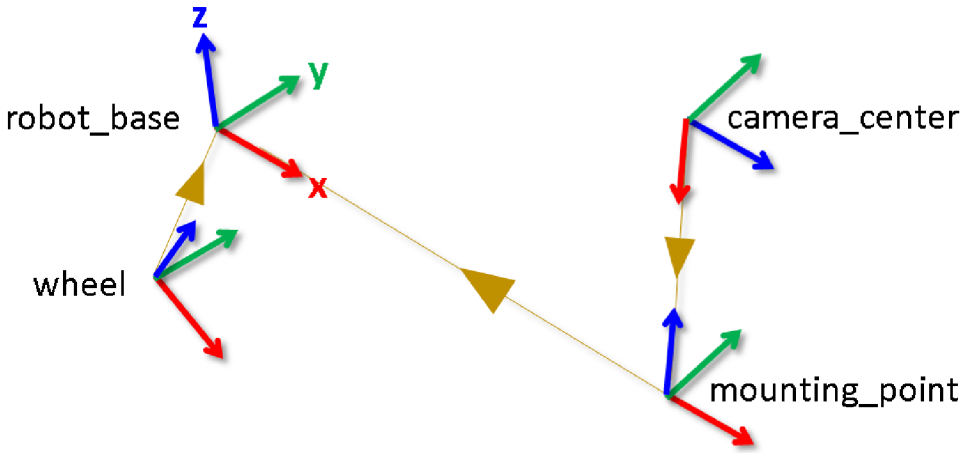
Jestliže používáme robota, který má kola, můžeme v dalším kroku využít *ROS Navigation Stack,* což je nástroj, který implementuje navigaci. V případě robota typu Hexapod nám nezbývá nic jiného, než si vytvořit vlastní způsob pro navigaci. V tomto případě můžeme využít prohledávacích algoritmů viz. 6.2,1, ovšem implementovat takovýto algoritmus pro data vytvořená pomocí *map_server* představuje skutečnou výzvu.[40]



Obrázek 20: Ukázka Gmapping [40]

6.2.3 tf

Balík tf udržuje vztah mezi souřadnicovými rámci ve stromové struktuře vyrovnávací paměti v čase a umožňuje uživateli transformovat body, vektory atd. mezi libovolnými dvěma souřadnicovými rámci v libovolném časovém bodě. Využití tf můžeme vhodně demonstrovat na obrázku níže. Jestliže máme kameru umístěnou v bodě *camera_center,* která nám udržuje informace o poloze a natočení jednotlivých těles, můžeme pomocí nástroje tf jednoduše získat polohu a natočení mezi jednotlivými tělesy navzájem. Pokud bychom chtěli získat informace o poloze a natočení např. mezi tělesy *robot_base* a *wheel* bez tf, tak bychom museli nejdříve zjistit vzájemný vztah mezi tělesy *camera_center* a *mounting_point*, *mounint_point* a *robot_base* a nakonec *robot_base* a *wheel,* viz podkapitola 2.6.[41]



Obrázek 21: Princip tf [48]

6.2.4 Další technologie kompatibilní s ROS

Tensorflow

Jestliže stáhneme Open-source Tensorflow, máme možnost využít tento software v rámci ROS. Tensorflow je open-source knihovna v programovacím jazyce Python se zaměřením na neuronové sítě. Ty lze využít k tomu, aby se robot naučil orientovat v novém prostředí. Řada výzkumných týmů se dnes například zabývá tím, jak využít zpětnovazebné učení (reinforcement learning) jako součást klasického algoritmu SLAM (Simultaneous Localization and Mapping), jehož cílem je zároveň mapovat neznámé prostředí, ve kterém se robot ocitnul, a zároveň si vytvořit informaci o jeho vlastní poloze v tomto prostředí. V angličtině se pro tento přístup vžil termín Neural SLAM.[42]

OpenAI Gym

Jestliže stáhneme Open-source OpenAI Gym, máme možnost využít tento software v rámci ROS. OpenAI Gym je sada nástrojů pro vývoj a porovnávání algoritmů zpětnovazebného strojového učení (reinforcment learning). OpenAI Gym poskytuje prostředí a je na vývojáři, aby implementoval konkrétní algoritmy učení. Vývojáři ROS

vytvořili nástroje, které jsou schopny využít funkcí OpenAI Gym. V rámci ROS můžeme tento software využít pro učení plánování cesty robota nebo pro učení pohybů robota, viz předchozí poznámka k Tnesorflow.[43]

7 DISKUZE K VÝSLEDKŮM A MULTIMEDIÁLNÍ PREZENTACE VÝSLEDKŮ

Hlavním cílem praktické části diplomové práce bylo vytvořit řízení pro robota King Spider od firmy Robotis a dále pak simulovat pohyb tohoto robota pomocí ROS a Gazebo. Při řízení skutečného modelu se až na problémy zmíněné v kapitole 3 podařilo splnit požadované zadání, avšak kvůli těmto problémům bylo možné robota testovat jen na krátký časový úsek a dané pohyby nesměly být příliš náročné.

Převážnou část praktické části zabrala simulace robota. V první řadě bylo potřeba se seznámit s ROS a Gazebo. Tyto frameworky nabízí poměrně široké možnosti při simulaci, a proto je důležité vědět, které nástroje využít. Samotná práce s ROS a Gazebo se dá z programátorského hlediska hodnotit za středně obtížnou. Při práci se složitějšími modely je také nutné znát alespoň průměrně programovací jazyky C++ a Python. Za největší problém při sestavování simulace považuji dostupnost informací pro práci s ROS a Gazebo. Stránky ros-wiki nabízí kurzy zdarma a některé informace jsou taktéž dostupné v online knihách nebo na Youtube, ale většina těchto informací se neustále opakuje a pokrývají převážně znalosti jen pro začátečníky. Podrobnější informace o ROS a Gazebo se dají získat v placených kurzech. Z tohoto důvodu jsem se při tvorbě inspiroval také u jiných projektů dostupných v online repozitářích jako např. GitHub.

7.1 Tvorba simulace a modelu robota

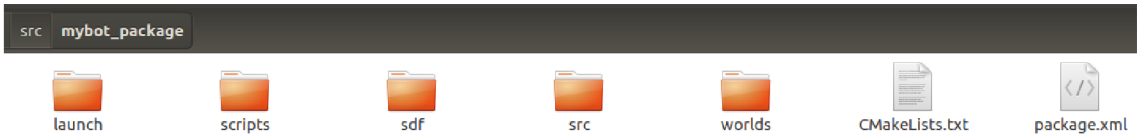
7.1.1 Model robota

Pro uskutečnění simulace bylo potřeba jako první vytvořit model robota King Spider ve formátu URDF nebo SDF (viz kapitola 4). Robotis nabízí veškeré své produkty také ve formátech CAD ke stažení, což se dá využít k získání potřebných dílů a následném sestavení do podoby robota King Spider. Přestože importování z formátu CAD na potřebný formát k zobrazení v Gazebo bylo bezproblémové, tak jen zobrazení jednoho dílu bylo pro mojí grafickou kartu počítače tak náročně, že se simulátor Gazebo ani nespustil. Z tohoto důvodu tato varianta nebyla dále prozkoumávána a byl vytvořen vlastní model robota. Tento model vypadá samozřejmě zjednodušeně oproti skutečnému modelu, ale robota King Spider stále připomíná. Největší překážkou při tvorbě modelu bylo to, že není možné odměřovat vzdálenosti při tvorbě robota jako v CADu nebo Inventoru. Z tohoto důvodu byl nejdříve vytvořen model v Inventoru a dané vzdálenosti byly si odměřeny a zapsány. Poté byl vytvořen model ve formátu URDF, který lze vizualizovat pomocí RViz. V posledním kroku byl model vytvořený ve formátu URDF převeden na formát SDF a upravil některé detaily.

7.1.2 Catkin_workspace

V dalším kroku bylo potřeba vytvořit pracovní prostředí pro práci s ROS, neboli catkin_workspace. Obecný postup pro vytvoření tohoto balíku je popsán v kapitole 4.

Catkin_workspace obsahuje soubory potřebné pro spuštění simulace. Ve složkách src a scripts se nachází C++ plugin a několik skriptů v Pythonu pro uskutečnění simulace. Složka worlds, obsahuje dva světy, ve kterém je možné robota simulovat. Ve složce launch se nachází soubory, pomocí nichž se program spouští a ve složce sdf se nachází model robota.



Obrázek 22: Složka mybot_package

Nejtěžším úkolem celé práce bylo vytvořit plugin v C++ pro ovládání robota. Tento plugin obsahuje funkce, jenž ovládají pohyb kloubů robota a subscribers, které umožňují komunikovat s těmito funkcemi v reálném čase. Funkce vždy seskupují klouby podle toho, ve který daný časový okamžik se mají pohnout. Pro komunikaci s pluginem byly vytvořeny Publishers, které zasílají s určitou frekvencí parametry pro funkce v pluginu. Všechny tyto Publishers byly vytvořeny v Pythonu. Rychlost pohybu robota se dá měnit.

Na těle robota se nachází i kamera a scanner. Obě tyto zařízení mají různé využití, v mém příkladě jsem využil scanner ve skriptu hexapod.py pro navigaci okolo překážek a kameru ve skriptu camera.py pro rozeznávání objektů různé barvy.

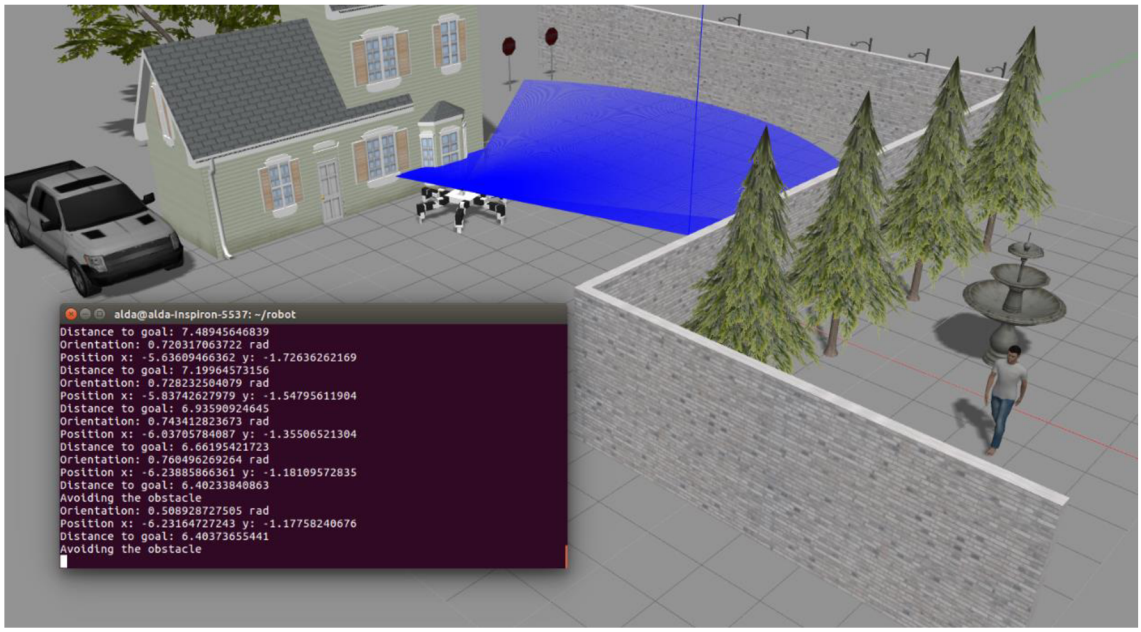
7.1.3 Problémy spojené se simulací robota

Simulaci robota se podařilo úspěšně zprovoznit a robot se při spuštění simulace pohybuje plynule a bez problémů, avšak simulace má několik nedostatků. Největší problém nastává, když se robot srazí s cizím tělesem. Fyzický model by s největší pravděpodobností přestal fungovat a zůstal by stát na místě, ovšem model v simulaci odletí a jeho chování je zcela nepředvídatelné. Podobný problém také nastane, když se části dotýkající se země mají posunout do určité polohy, ale ostatní části robota jsou v takové poloze, že mu to nedovolí. V tomto případě robot začne skákat různými směry. Obzvláště druhý zmíněný problém je poměrně velkou překážkou. Pohyb robota King Spider je specifický v tom, že svou základnou pohybuje při chůzi směrem nahoru a dolů a části dotýkající se země se posouvají po podložce. Z tohoto důvodů musí být simulace poměrně opatrná. Dalším problém nastává při nastavování parametrů typu váha, momenty setrvačnosti, PID pohybu robota, tlumení, tuhost kloubů apod. Tyto parametry je prakticky nemožné nastavit přesně tak, aby odpovídaly skutečnému modelu, avšak pokud se nastaví alespoň přibližně správně, tak se simulace liší od skutečného modelu minimálně. Posledním problém simulace je, že se robot neustále pohybuje a nikdy není zcela v klidu, i když na něj působí jen gravitační síla. Tento pohyb je velmi malý, ale po delší době simulace je viditelný a může působit problémy při simulování chůze robota.

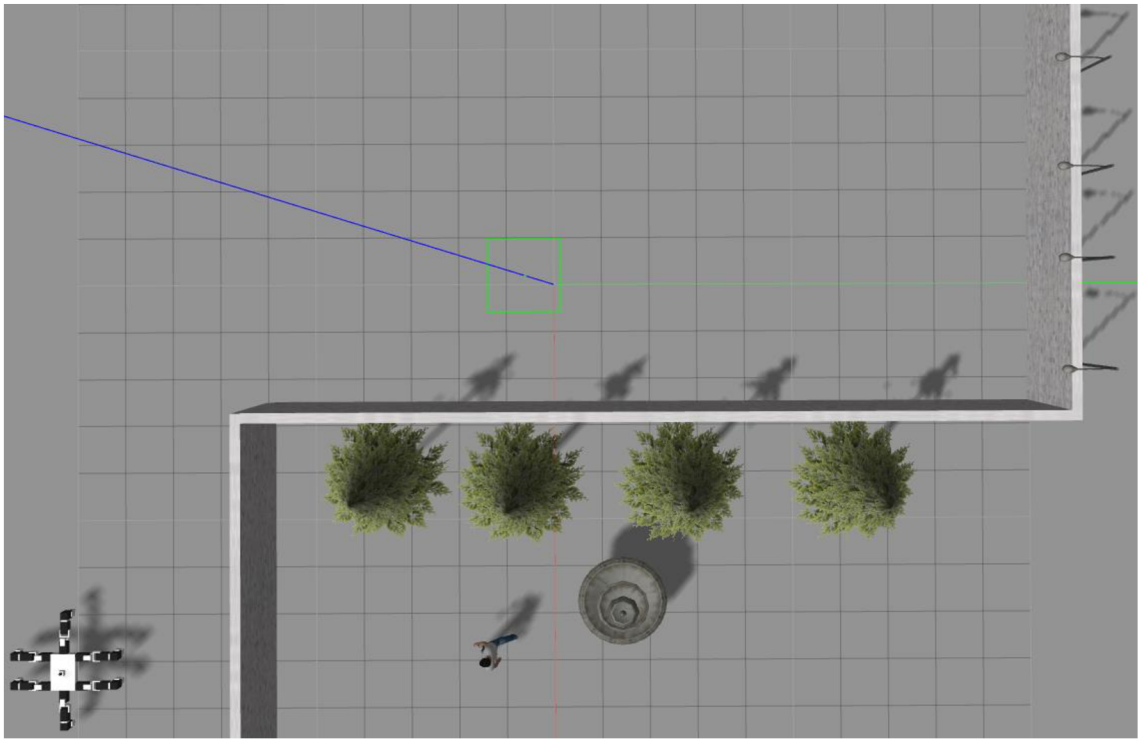
Výše zmíněné problémy by se pravděpodobně daly do určité míry odstranit, např. lepším nastavením parametrů nebo hlubší znalostí ROS a jeho nástrojů, ovšem se stejnými problémy jsem setkal i u jiných projektů, které jsem testoval.

Po seznámení s ROS a Gazebo se dá říct, že tyto frameworky mají velký potenciál v robotice a jsou v této oblasti nejlepší volbou. Na druhou stranu není vhodné použít ROS a Gazebo pro jednorázovou simulaci robota, protože nezkušenému uživateli potrvá nějakou dobu se seznámit s těmito frameworky. V případě robota King Spider bylo mnohem snadnější provádět změny u fyzického modelu než při simulaci. Z tohoto hlediska je vhodné mít určité zkušenosti s ROS a Gazebo, jestliže se rozhodneme tyto frameworky využít.

7.2 Ukázky ze simulace

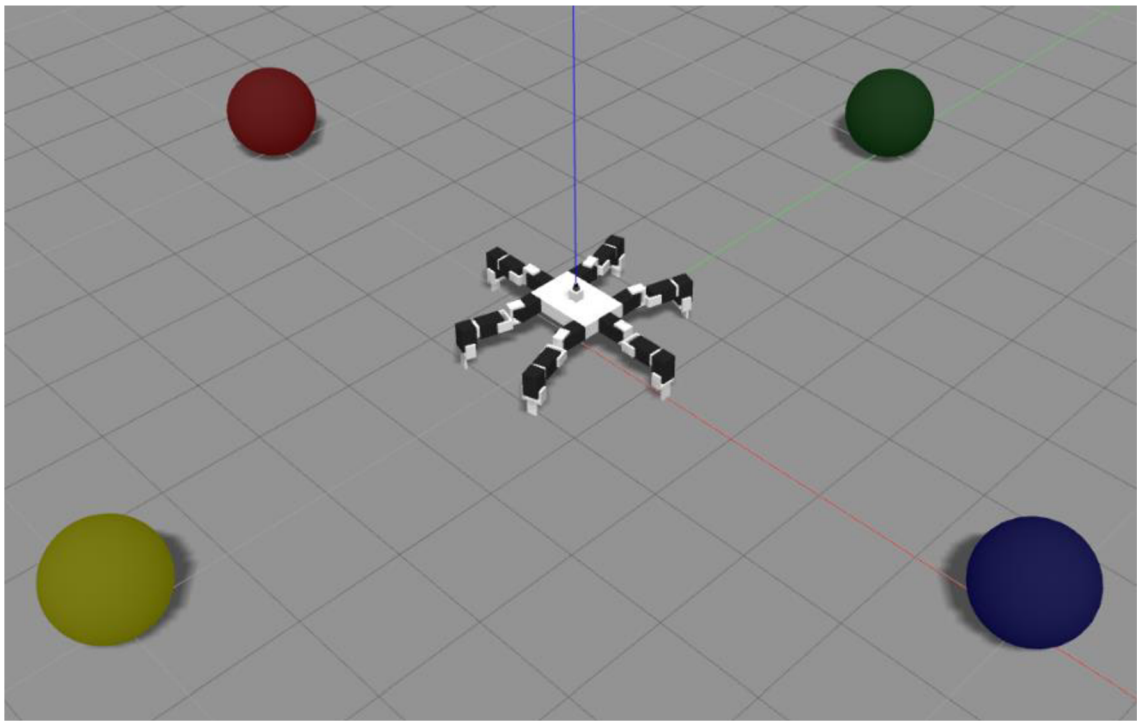


Obrázek 23: Navigace robota okolo překážek

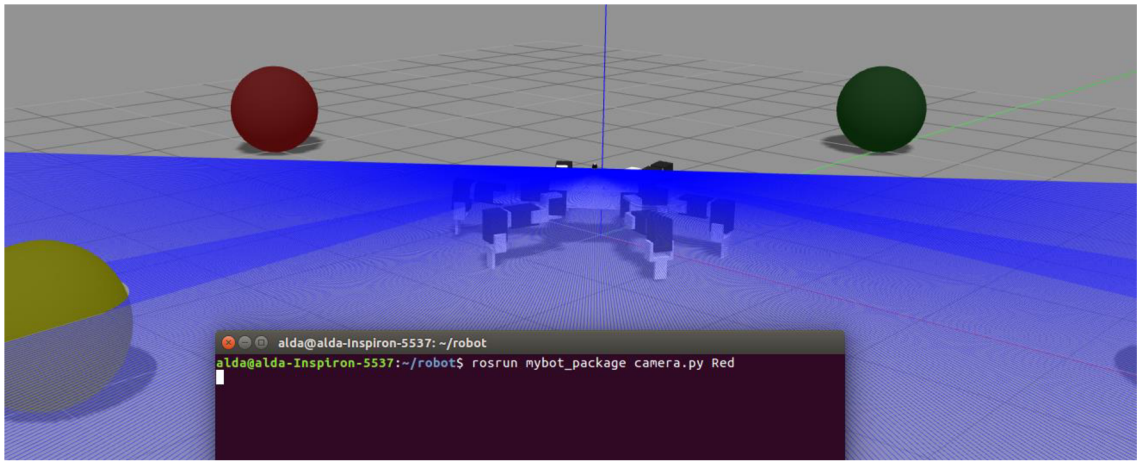


Obrázek 24: Robot před spuštění simulace

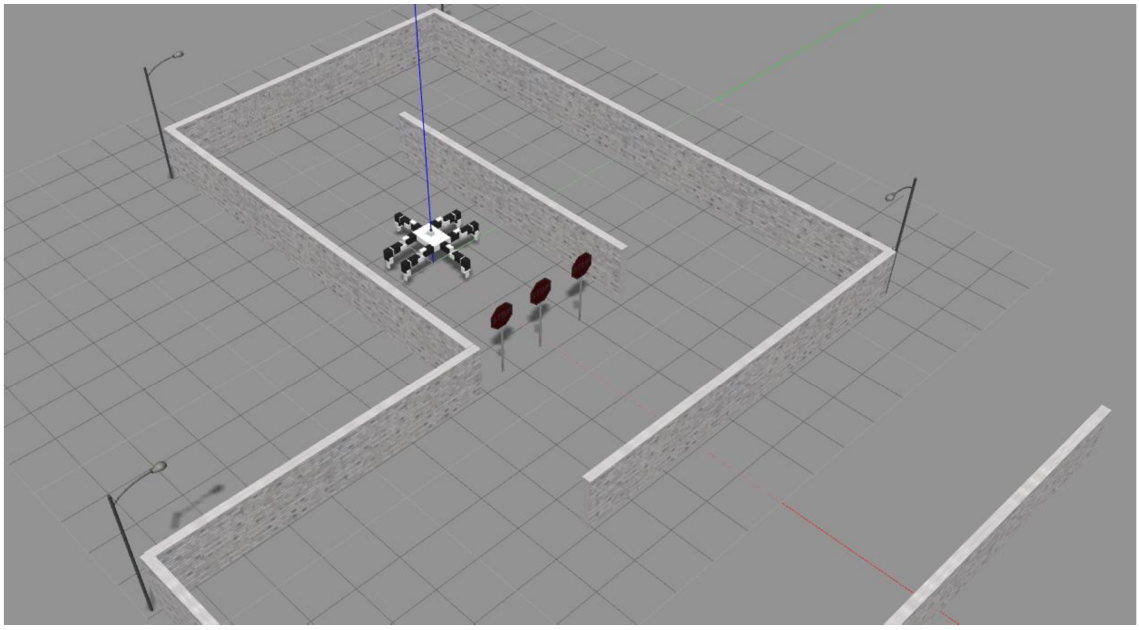
Obrázek 25: Výhled z kamery robota



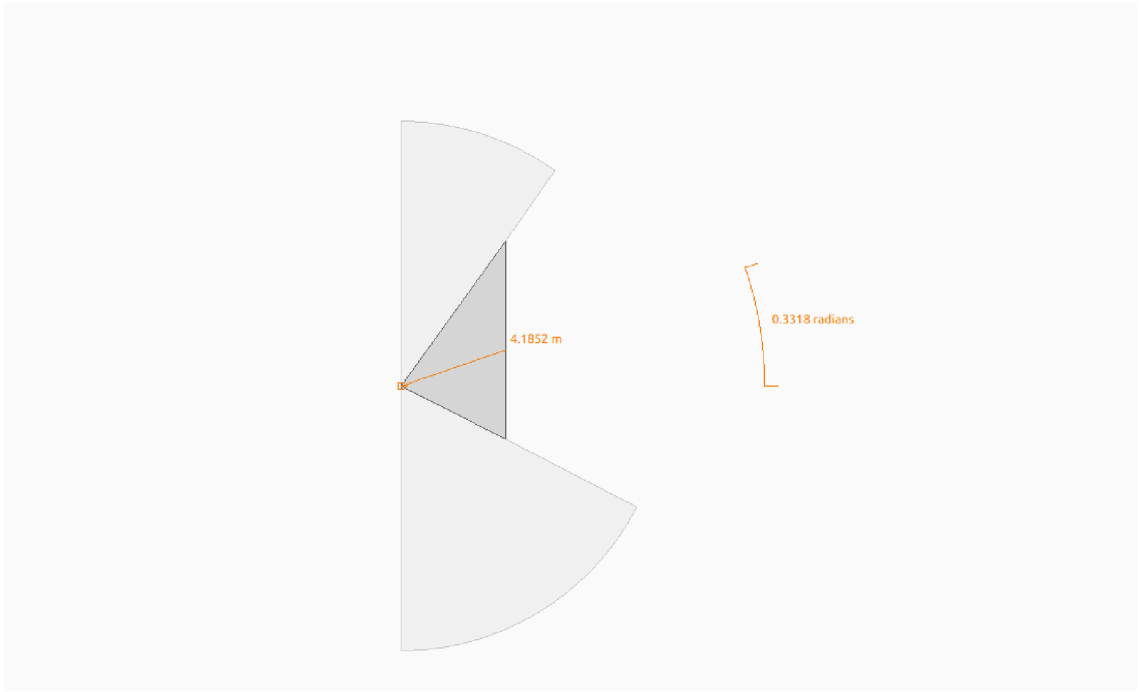
Obrázek 26: Vyhledávání tělesa podle barvy



Obrázek 27: Spuštění simulace pomocí terminálu



Obrázek 28: Svět vhodný pro Gmapping



Obrázek 29: Zobrazení překážek pomocí skeneru

7.3 Stručný popis kódu a funkcionality

Složka launch obsahuje soubory typu .launch, které jsou nezbytné pro spuštění různých světů a modelů.

Složka scripts obsahuje skripty pro ovládání chůze a funkcionality robota. Soubory back.py, front.py, left.py, right.py zajišťují pohyb směrem dozadu, dopředu, doleva a doprava. Po spuštění jednoho z těchto skriptů vykoná robot jeden krok daným

směrem. Prvním krokem nezbytným pro práci s ROS je importování modulů rospy a v závislosti na další práci případně další moduly. Všechny tyto skripty obsahují topics typu Publisher, které jsou nutné pro spojení se Subscriber v pluginu. Pro komunikaci mezi Publisher a Subscriber je nutné zvolit správnou metodu z modulu std_msgs.msg. Důležitým atributem této třídy je proměnná data, se kterou budeme dále pracovat v Subscriber. Metoda publish pak předá daný objekt Subscriber. *rospy.Rate(int)* určuje frekvenci odesílání dat. Složka pak dále obsahuje skript hexapod.py, který demonstruje využití scanneru. Skript pracuje již s vytvořenými moduly front, back, left, right. Tento skripty využívá dva Subscriber pro získávání dat o aktuální poloze a vzdálenosti překážek pomocí scanneru. Důležitými nástroji zhlediska ROS jsou zde callback funkce, které po předání vlastní reference metodě rospy.Subscriber vykonávají neustále námi požadovanou činnost. Skript pak dále obsahuje navigační algoritmus pro pohyb robota z bodu A do bodu B okolo překážek a nekolik dalších funkcí, které jsou k tomuto algoritmu nezbytné. Tento algoritmus není úplně ideální, protože potřebuje pomocné body ke správnému průchodu, ovšem cílem práce bylo spíše ukázat využití scanneru.

```
import rospy
import sys
import math
from gazebo_msgs.msg import ModelStates
from sensor_msgs.msg import LaserScan
from tf.transformations import euler_from_quaternion
from front import front
from back import back
from left import left
from right import right

Laser = None
Position = None
Orientation = None

def laser(msg):
        global Laser
        Laser = msg.ranges

def pose(msg):
        global Position
        global Orientation
        Position = msg.pose[msg.name.index('pexod')].position
        Orientation = msg.pose[msg.name.index('pexod')].orientation

rospy.Subscriber('/gazebo/model_states', ModelStates, pose)
rospy.Subscriber('/pexod/laser', LaserScan, laser)
```

Posledním skriptem této složky je camera.py, pomocí něhož je robot schopen najít objekt určité barvy. Skript využívá callback funkci pro získání obrázku, který je reprezentován maticí s číselnými údaji RGB.

```
import numpy
import rospy
import sys
from sensor_msgs.msg import Image
```

```
from rospy.numpy_msg import numpy_msg

def camera(msg):
        global Matrix
        Matrix = numpy.frombuffer(msg.data, dtype=numpy.uint8).reshape(msg.height, msg.width, -1)

rospy.Subscriber('/pexod/camera/image_raw', Image, camera)
```

Složka sdf obsahuje model robota ve formátu sdf. Tento model také zahrnuje tři pluginy, které jsou nezbytné k rozpohybování robota a pro fungování scaneru a kamery.

Složka src obsahuje námi vytvořený plugin, který slouží k pohybu robota. Základem pro práci s robotem je zahrnutí nezbytných knihoven a funkce Load, která je volaná vždy, když se spustí simulace v Gazebu. Parametry této funkce jsou ukazatel na model, který obsahuje sadu všech links, joints a pluginů, druhým parametrem je ukazatel na sdf prvek pro plugin v souboru .world. Pro uvedení robota do pohybu nám stačí pracovat pouze s ukazatelem na model.

```
#include <ros/ros.h>
#include <gazebo/gazebo.hh>
#include <gazebo/physics/physics.hh>
#include <gazebo/transport/transport.hh>
#include <gazebo/msgs/msgs.hh>
#include <thread>
#include "ros/callback_queue.h"
#include "ros/subscribe_options.h"
#include "std_msgs/Float32.h"

namespace gazebo
{
 /// /brief A plugin to control an hexapod.
 class PexodPlugin : public ModelPlugin
 {
   public: PexodPlugin() {}
   public: virtual void Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
```

Dále je zde využita metoda SetPositionPID, která bere jako parametry název kloubu a PID pohybu.

```
this->joint_leg0_0_compound0_0 = _model->GetJoint("leg0_0_compound0_0");
this->pid_leg0_0_compound0_0 = common::PID(600,100,50);
this->model->GetJointController()->SetPositionPID(this->joint_leg0_0_compound0_0->GetScopedName(),
this->pid_leg0_0_compound0_0);
```

Funkce create ze třídy *ros::SubscribeOptions* slouží pro vytvoření subscriber. Jako argumenty bere námi požadovaný název pro subscriber a název pro funkci, která je volána, když do subscriber zapisuje publisher. Template této funkce určuje datový typ, se kterým funkce bude pracovat.

```
ros::SubscribeOptions so01 = ros::SubscribeOptions::create<std_msgs::Float32>("/" + this->model->GetName() +
"/vel_cmd_Compounds0_025",1, boost::bind(&PexodPlugin::OnRosMsg01, this, _1),  ros::VoidPtr(), &this->rosQueue);
this->rosSub01 = this->rosNode->subscribe(so01);
```

Metoda SetPositionTarget bere jako argument název joint a pozici, do které se má daný joint posunout.

```
this->model->GetJointController()->SetPositionTarget(this->joint_leg0_0_compound0_0->GetScopedName(), _position-0.3);
```

GZ_REGISTER_MODEL_PLUGIN(*ClassName*) předá Gazebu informaci o existenci tohoto pluginu.

```
GZ_REGISTER_MODEL_PLUGIN(PexodPlugin)
```

Složka worlds obsahuje různé světy, které jsou spouštěny pomocí .launch souborů.

7.4 Užitečné tipy pro práci s ROS a Gazebo

Po určitých zkušenostech se můžeme hned od začátku vyvarovat těchto základních chyb.

- Zvolit správnou distribuci ROS pro požadovanou práci. Nejvíce možností a dostupných příkladu pro práci nabízí ROS Kinetic.
- Využití formátu xacro při tvorbě robota zpřehlední a výrazně zkrátí kód.
- Naučit se nejdříve používat ROS a poté až Gazebo.
- Upřednostnit existujících balíku a nástrojů, které ROS nabízí, před vlastní tvorbou.

8 ZÁVĚR

Cílem této diplomové práce bylo objasnit teoreticky a prakticky problematiku biologicky inspirovaného robota – hexapod.

Teoretická část se stručně zabývá tématy, které se běžně vyskytují při konstrukci a chůzi hexapoda. Jsou zde zmíněny historické a současné modely tohoto robota, jeho složení a využití. V poslední části úvodní kapitoly zde objasněna problematika přímé a nepřímé kinematiky, se kterou se lze často setkat při návrhu chůze pro tento typ robota.

První část praktické části této diplomové práce se zabývala fyzickým modelem robota hexapod, konkrétně typem King Spider od firmy Robotis Bioloid. Tato část popisuje části tohoto robota a knihovnu pro programovací jazyk Python PyAX12, pomocí které byl fyzický model řízen. Přes drobné problémy, které se u tohoto modelu objevily, se podařilo tohoto robota zprovoznit. Řídící software pro konkrétní model je v příloze.

Zbytek praktické části této diplomové práce se zabývá virtuální simulací robota King Spider pomocí frameworků ROS a Gazebo. Pro uskutečnění této simulace bylo nejdříve potřeba se s těmito frameworky seznámit, proto se text také věnuje základním principům ROS a Gazebo. Za tímto účelem také vznikl pomocný projekt, který vysvětluje základy práce s těmito softwarovými balíky. Nakonec byl vytvořen i řídící program pro simulaci robota King Spider.

Zadání diplomové práce bylo v rámci možností splněno v plném rozsahu. Přes určité nedostatky u řízení fyzického modelu i simulace tyto programy demonstrují, jak zprovoznit fyzický model robota a jak provést simulaci pomocí ROS a Gazebo.

9 SEZNAM POUŽITÉ LITERATURY

[1] PA, P. S., C. M. Wu. *Design and Application of High-Sensitivity Hexapod Robot*. 2009 [cit. 2020-06-23]. Dostupné z: https://link.springer.com/chapter/10.1007/978-3-642-10817-4_103

[2] TEDESCHI, Franco a Giuseppe CARBONE. 2014. Design Issues for Hexapod Walking Robots. *Robotics* [online]. 2014, roč. 3, č. 2 [cit. 2020-23-06]. ISSN 2218-6581. Dostupné z: https://www.mdpi.com/2218-6581/3/2/181

[3] UR REHMAN, Bilal, Claudio SEMINI a Darwin CALDWELL. *Centaur robots - a survey*. 2017 [cit. 2020-23-06]. Dostupné z: https://www.researchgate.net/publication/319399104_CENTAUR_ROBOTS_-_A_SURVEY

[4] *https://www.robotshop.com/* [online] 2020 [cit. 2020-06-24]. Dostupné z: https://www.robotshop.com/eu/en/hexapod-development-platforms.html

[5] TROSSENROBOTICS. PhantomX AX Metal Hexapod Mark III Kit. *trossenrobotics.com* [online]. 2018 [cit. 2020-06-24]. Dostupné z: https://www.trossenrobotics.com/phantomx-ax-hexapod.aspx

[6] VINCROSS. Hexa. *vincross.com* [online]. 2019 [cit. 2020-06-24]. Dostupné z: https://www.vincross.com/en/hexa

[7] HEXAPODROBOTS. Advantages of Hexapod Gait. *hexapodrobots.weebly.com* [online]. 2010 [cit. 2020-06-24]. Dostupné z: https://hexapodrobots.weebly.com/advantages-of-hexapod-gait.html

[8] LINEARMOTIONTIPS. End of Arm Tooling for Automotive Applications. *linearmotiontips.com* [online]. 2019 [cit. 2020-06-24]. Dostupné z: https://www.linearmotiontips.com/hexapod-robot-design-variations-applications/3/

[9] *wikipedia.org* [online] 2018 [cit. 2020-06-24]. Dostupné z: https://en.wikipedia.org/wiki/Stewart_platform

[10] RAHEEM, Firas, Ahmed SADIQ a Noor Alhuda ABBAS. *Robot Arm Free Cartesian Space Analysis for Heuristic Path Planning Enhancement*. 2019 [cit. 2020-06-23]. Dostupné z: https://www.researchgate.net/publication/331685972_Robot_Arm_Free_Cartesian_Space_Analysis_for_Heuristic_Path_Planning_Enhancement

[11] OPÁLKA, Jan. *Inverzní úloha robotiky a její řešení*. Liberec, 2008. Bakalářská práce. Technická univerzita v Liberci, Fakulta mechatroniky a mezioborových inženýrských studií, Ústav mechatroniky a technické informatiky

[12] MOSTÝN, Vladimír a Václav KRYS. *Mechatronika průmyslových robotů* [online]. Vyd. 1. Ostrava: Vysoká škola báňská – Technická univerzita, 2012, 1 DVD-ROM [cit. 2020-06-23]. ISBN 978-80-248-2610-3.

[13] ROBOTIS. Frames. *robotis.us* [online]. 2020 [cit. 2020-06-24]. Dostupné z: http://www.robotis.us/frames/
.

[14] EADY, Fred. *Unwinding The AX-12+Communication Protocol* [online] 2009 [cit. 2020-06-24]. Dostupné z: https://www.servomagazine.com/uploads/issue_downloads/Unwinding_the_AX12.pdf

[15] DVORSKÝ, Pavel. *Řídící jednotka pro servopohony Dynamixel AX-12A*. Liberec, 2014. Bakalářská práce. Technická univerzita v Liberci, Fakulta mechatroniky, informatiky a mezioborových studií, Ústav informačních technologií a elektroniky.

[16] ROBOTIS CO.,LTD. *Dynamixel Protocol 1.0: User's Manual.* 2019. vyd. 6 s.

[17] ROBOTIS CO.,LTD. *USB2Dynamixel: User's Manual*. 2012. vyd. 8 s.

[18] MEGAROBOT. CM-530. *megarobot.cz* [online]. 2020 [cit. 2020-06-24]. Dostupné z: http://www.megarobot.cz/index.php?route=product/product&product_id=86

[19] KOHOUTEK,Tomáš. *Robotický KIT Bioloid Robotis Premium.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky

[20] DECOCK, Jérémie. PyAX-12 Documentation. *readthedocs.org* [online]. 2015 [cit. 2020-06-24]. Dostupné z: https://readthedocs.org/projects/pyax-12/downloads/pdf/latest/

[21] *ros.org* [online] 2020 [cit. 2020-06-24]. Dostupné z: https://www.ros.org/about-ros/

[22] *wiki.ros.org* [online] 2017 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/catkin

[23] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes

[24] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/Master#:~:text=Overview,nodes%20to%20locate%20one%20another
.

[25] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/roslaunch

[26] THE ROBOTICS BACKEND. What is a ROS Topic. *roboticsbackend.com* [online]. 2014 [cit. 2020-06-24]. Dostupné z: https://roboticsbackend.com/what-is-a-ros-topic/

[27] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/Topics

[28] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/Nodes

[29] THE ROBOTICS BACKEND. What is a ROS Topic. *roboticsbackend.com* [online]. 2014 [cit. 2020-06-24]. Dostupné z: https://roboticsbackend.com/what-is-a-ros-node/

[30] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/Services

[31] gazebosim.org [online] 2018 [cit. 2020-06-25]. Dostupné z: http://gazebosim.org/

[32] *sdformat.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://sdformat.org/tutorials?tut=spec_model_kinematics#link

[33] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/xacro

[34] *sdformat.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://sdformat.org/spec?ver=1.7&elem=link

[35] *sdformat.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://sdformat.org/spec?ver=1.7&elem=joint

[36] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/rviz

[37] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/ros_control

[38] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29

[39] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/dynamixel_controllers/Tutorials

[40] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/gmapping

[41] *wiki.ros.org* [online] 2018 [cit. 2020-06-25]. Dostupné z: http://wiki.ros.org/tf

[42] *tensorflow.org* [online] 2019 [cit. 2020-06-24]. Dostupné z: https://www.tensorflow.org/

[43] *gym.openai.com* [online] 2019 [cit. 2020-06-24]. Dostupné z: https://gym.openai.com/

[44] cyberneticzoo.com [online] 2010 [cit. 2020-06-21]. Dostupné z: http://cyberneticzoo.com/underwater-robotics/1985-aquarobot-aquatic-walking-robot-japanese/attachment/aquarobot-1-x640/

[45] https://www.researchgate.net/ [online] 2014 [cit. 2020-06-21]. Dostupné z: https://www.researchgate.net/profile/Andreas_Pott/publication/317721191/figure/fig1/AS:507690012745729@1498054139671/Comparative-display-of-a-general-Gough-Stewart-platform-a-and-a-cable-robot-b.png

[46] https://www.trossenrobotics.com/ [online] 2010 [cit. 2020-06-21]. Dostupné z: https://www.trossenrobotics.com/shared/images/PImages/RO-902-0062-000-a.jpg

[47] https://www.servomagazine.com/ [online] 2009 [cit. 2020-06-21]. Dostupné z: https://www.servomagazine.com/uploads/issue_downloads/Unwinding_the_AX12.pdf

[48] es.mathworks.com [online] 2019 [cit. 2020-06-21]. Dostupné z: https://es.mathworks.com/help///examples/ros/win64/AccessTheTfTransformationTreeInROSExample_02.png

[49] es.mathworks.com [online] 2019 [cit. 2020-06-21]. Dostupné z: https://es.mathworks.com/help/examples/ros/win64/CallAndProvideROSServicesExample_01.png

[50] es.mathworks.com [online] 2019 [cit. 2020-06-21]. Dostupné z: https://es.mathworks.com/help/examples/ros/win64/ExchangeDataWithROSPublishersAndSubscribersExample_01.png

10 SEZNAM ZKRATEK, SYMBOLŮ, OBRÁZKŮ A TABULEK

Seznam obrázků

Obrázek 1: Robot Masha [2] 19
Obrázek 2: AQUAROBOT [44] 20
Obrázek 3: PhantomX AX [5] 21
Obrázek 4: Hexa Hexapod Robot [6] 21
Obrázek 5: Stewartova Platforma [45] 23
Obrázek 6: Příklad geometrie robotického ramene [10] 24
Obrázek 7: Transformace souřadnic podle DH principu [12] 26
Obrázek 8: Robot King Spider [13] 29
Obrázek 9: Jeden z rámů FP04 [13] 29
Obrázek 10: Dynamixel AX-12+ [47] 30
Obrázek 11: Instrukční paket 30
Obrázek 12: USB2Dynamixel [17] 31
Obrázek 13: Řídící jednotka CM-530 [18] 32
Obrázek 14: ROS Topic [50] 38
Obrázek 15: ROS Service [49] 39
Obrázek 16: Formáty URDF a SDF [32] 40
Obrázek 17: Plugin pro Gazebo [37] 43
Obrázek 18: Model Example v Gazebo 46
Obrázek 19: Model Example v RViz 47
Obrázek 20: Ukázka Gmapping [40] 56
Obrázek 21: Princip tf [48] 57
Obrázek 22: Složka mybot_package 60
Obrázek 23: Navigace robota okolo překážek 61
Obrázek 24: Robot před spuštění simulace 62
Obrázek 25: Výhled z kamery robota 62
Obrázek 26: Vyhledávání tělesa podle barvy 63
Obrázek 27: Spuštění simulace pomocí terminálu 63
Obrázek 28: Svět vhodný pro Gmapping 64
Obrázek 29: Zobrazení překážek pomocí skeneru 64

Seznam zkratek

ROS	Robotic Operation Systém

CAD	Computer aided design
USB	Universal Serial Bus
RGB	model červená-zelená-modrá
XML	Extensible Markup Language
URDF	Unified Robot Description Format
SDF	Simulation Description Format