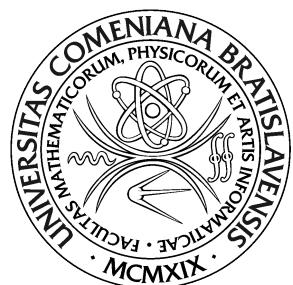


UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY



RIADENIE MODELU KRÁČAJÚCEHO HMYZU
DIPLOMOVÁ PRÁCA

2019
Bc. ADRIÁN PAVČO

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

RIADENIE MODELU KRÁČAJÚCEHO HMYZU
DIPLOMOVÁ PRÁCA

Študijný program: Aplikovaná informatika
Študijný odbor: 2511 Aplikovaná informatika
Školiace pracovisko: Katedra aplikovej informatiky
Školiteľ: RNDr. Andrej Lúčny, PhD.

Bratislava, 2019
Bc. Adrián Pavčo



Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Adrián Pavčo

Študijný program: aplikovaná informatika (Jednoodborové štúdium,
magisterský II. st., denná forma)

Študijný odbor: aplikovaná informatika

Typ záverečnej práce: diplomová

Jazyk záverečnej práce: slovenský

Sekundárny jazyk: anglický

Názov: Riadenie modelu kráčajúceho hmyzu
Controlling Model of Walking Insect

Ciel: Cieľom práce je rozvíjať sčasti vyvinutý model kráčajúceho hmyzu. Model je vyvinutý v C# pod MRDS, ale je možné zvážiť jeho reimplementáciu v iných prostriedkoch, napr. v C++ a ODE. Každopádne treba ho najprv správne vyvážiť, aby ho príliš ťažký abdomen neprekacoval dozadu a doplniť o nejakú riadiacu architektúru (yarp alebo Agent-Space). A vytvoriť v tom riadiaci systém, ktorý bude s modelom kráčať aspoň po rovine s možnosťou demonštrácie prechodu cez prekážku a krátku priepasť.

Literatúra: Cruse, H: Neural Networks as Cybernetic Systems (časti o kráčajúcim hmyze)
dokumentácia k MRDS prípadne ODE
články od Cruze-Schultz

Klúčové slová: riadenie, kráčajúci hmyz, 3D modelovanie

Vedúci: RNDr. Andrej Lúčny, PhD.

Katedra: FMFI.KAI - Katedra aplikovej informatiky

Vedúci katedry: prof. Ing. Igor Farkaš, Dr.

Dátum zadania: 30.09.2016

Dátum schválenia: 14.10.2016

prof. RNDr. Roman Ďuríkovič, PhD.

garant študijného programu

.....
študent

.....
vedúci práce

Čestne prehlasujem, že som túto diplomovú prácu vypracoval samostatne len s použitím uvedenej literatúry a s pomocou konzultácií u môjho školiteľa.

Bratislava, máj 2019

Bc. Adrián Pavčo

Poděkovanie

Ďakujem môjmu školiteľovi RNDr. Andrejovi Lúčnemu, PhD. za odborné vedenie, ochotu a trpezlivosť. Tiež dakuju mojej rodine a priateľom za podporu počas tvorby tejto práce, ale aj počas celého štúdia.

Abstrakt

Práca sa venuje problematike simulovania a riadenia modelu mravca a jeho chôdze. V zdedenom riešení sa nepokračovalo, bola zvolená reimplementácia v prostriedkoch Gazebo a ROS. V týchto nástrojoch sa vytvoril virtuálny simulátor, ktorý je v dosta- točnej miere verný biologickej predlohe. Od predchodcu boli prevzaté súbory modelu mravca, ktoré boli podrobene potrebnej úprave. Následne sa pomocou nich a biolo- gických dát vytvoril model na simuláciu, ktorý sa doplnil komunikačným rozhraním a riadením. Riešenie poskytuje rozhranie pre používateľský program a jednoduché ovlá- danie klávesnicou. V prostredí simulácie sa okrem roviny nachádzajú rôzne variácie prostredia. Pri kráčaní po rovine je chôdza mravca verná prírodnému modelu, robot prekoná aj väčšinu prekážok prostredia. Výsledné riešenie ako jedno z mála spája reálne anatomické a kinematické údaje mravca s jeho verným vizuálnym modelom a prináša dostupnú interaktívnu simuláciu. Tento dokument môže slúžiť aj ako návod pre čitateľa majúceho záujem o zoznámenie sa s používanými nástrojmi a vytváraním riadenia a simulácií v nich.

Kľúčové slová: riadenie, kráčajúci hmyz, 3D modelovanie

Abstract

This thesis deals with the problem of simulation and control of an ant model and its walking. A new solution is implemented instead of continuing in the works of our predecessor from whom only model files are inherited. Tools chosen for our solution are Gazebo and ROS. A sufficient level of faithfulness to a biological model is maintained. The legacy model files are converted and repaired. With the knowledge of ant anatomy and kinematics, a file describing model used in the simulation is created. Communication interface and control program is then added. The solution provides user interface program and simple keyboard teleoperation. In addition to the leveled plane, the simulated environment offers various changes in terrain profile. While walking on a plane, the model behaves true to the biological one and is able to cope with most of the demonstrative environment. Our solution is one of few which combine anatomy and kinematics data with a real visual representation of the model, all while offering accessible interactive simulation. The document may serve as a learning material for those interested in used tools and in the prospect of developing control and simulation using them.

Keywords: simulation, walking insect, 3D modeling

Obsah

1	Úvod	1
1.1	Cieľ práce	1
1.2	Členenie práce	2
1.3	Motivácia	2
2	Východiská	3
2.1	Predošlé riešenia	3
2.1.1	Zdedené riešenie	3
2.1.2	Historické riešenia	4
2.2	Iné východiská	4
3	Požiadavky na model	5
3.1	Anatómia mravca	5
3.2	Kinematika chôdze mravca	8
4	Implementačné nástroje	12
4.1	Úvod	12
4.2	Simulátor Gazebo	12
4.2.1	Inštalácia	13
4.2.2	Prostredie	14
Grafické rozhranie	14	
Súbory	16	
Štruktúra SDF modelov	16	
4.3	Pridanie riadiacej architektúry	18
4.4	Riadiaca architektúra ROS	18
4.4.1	Inštalácia	18
4.4.2	Architektúra a prostredie	19
5	Implementovaný model	24
5.1	Formáty modelov	24
5.2	Konvertovanie modelu	24

5.3	Tvorba a popis súborov modelu	25
6	Simulácia a výsledky	31
6.1	Tvorba a popis súborov simulácie	31
6.1.1	Balík ant_control	31
6.1.2	Balík simulácie ant_gazebo	34
	Konfiguračné súbory	35
	Launch súbor	36
	World súbor a prostredie simulácie	37
	Modul rozhrania	39
	Nodes súbory	41
6.2	Výsledky simulácie	47
7	Záver	51
	Prílohy	57

Zoznam obrázkov

2.1	Riškov model [Ris11]	3
3.1	Základné časti mrvavca [Hol09]	6
3.2	Stavba hlavy mrvavca [Hol09]	6
3.3	Stavba nožičky mrvavca [anaa]	7
3.4	Otáčanie kľbov končatín hmyzu [Sch07]	8
3.5	Chôdza striedaním trojnožiek [Zol94]	9
3.6	Vplyv rýchlosťi chôdze na trojnožku [Zol94]	9
3.7	Krajné body fáz chôdze [RWB09]	10
3.8	Detail stance fázy [RB14]	11
4.1	Architektúra Gazebo [KH04]	13
4.2	Gazebo GUI	15
4.3	Databáza modelov Gazebo	16
4.4	Vizuálny a kolízny element v SDF	17
4.5	Koncept ROS grafu [QGS15]	20
4.6	Štruktúra pracovného prostredia [MSFM16]	21
4.7	ROS Master diagram [MSFM16]	22
4.8	ROS graf talker-listener	23
5.1	Ant VRML súbor	25
5.2	Poskladaný Ant blend	25
5.3	Vizualizácia kľbov modelu	28
5.4	Nástroj Mesh Cleaner [mesb]	29
5.5	Model zobrazený v nástroji Rviz	30
6.1	Hierarchia balíka ant_gazebo	34
6.2	Príklad konfigurácie Gazebo a terminálov	48
6.3	Vizualizácia kontrolerov	49
6.4	Náčrt uhlov trojnožky v stance fáze	49
6.5	Podporný trojuholník a ťažisko v stance fáze	49

Kapitola 1

Úvod

Aj záležitosť ako je stretnutie a pozorovanie mrvaca sa pre nás môže zmeniť na zaujímavý zážitok. Naši predkovia na rozdiel od nás trávili v prírode viac času a mohli tento druh hmyzu častejšie sledovať v jeho prirodzenom prostredí. Asi sa nepýtali otázky, ktoré sa pýtajú dnešní vedci, no aj tak si museli všimnúť ich žitie v kolóniach, typické obydlia, disciplinované presúvanie a pracovanie v húfoch, pomerne rýchlu chôdzu a vôbec sálajúcemu energiu prejavujúcemu sa snahou stále niekam kráčať. Práve tieto znaky, ktoré v minulosti boli predmetom záujmu skôr detí, sa dnes tešia záujmu vedcov. Nielenže ich skúmame, no snažíme sa ich aj napodobniť. Vytvárame modely kopírujúce prírodu, prichádzame na rôzne koncepty a formy ich riadenia. V poslednom desaťročí sa do popredia dostáva na prvý pohľad svojská forma robotiky, a to simulovaná. Virtuálne robotické simulátory sú momentálne oblúbenou a dostupnou alternatívou na výučbu, vývoj a testovanie robotických systémov.

1.1 Ciel práce

Cieľom tejto práce je vytvorenie virtuálneho modelu mrvaca a simulovanie jeho chôdze. V zadaní práce sa hovorí o rozvíjaní predošlého riešenia. Na začiatku sme čeliли dileme, či pokračovať v zdedenom riešení, alebo reimplementovať riešenie v modernejších nástrojoch. Časom sme zistili, že za šest rokov sa dostupnosť a výkon prostriedkov citelne zlepšili. Okrem toho nástroje používané predchodom už niekoľko rokov nie sú podporované. Zvolili sme si teda vytvorenie systému v simulátore Gazebo, ktorý sme spojili s riadiacou architektúrou ROS (Robot Operating System). Existujú riešenia simulácie šestnohého hmyzu, no naša práca ako jedna z mála dopĺňa biologické dáta kinematiky o vernú vizuálnu vonkajšiu anatómiu. Práca obsahuje popis biologického modelu mrvaca slúžiaceho ako predloha pri tvorbe modelu pre simuláciu. Popisujeme tiež vybrané nástroje a nakoniec tvorbu, priebeh a zhodnotenie simulácie.

1.2 Členenie práce

Dokument práce sme sa snažili rozdeliť do logických a chronologicky nadväzujúcich celkov tak, aby bol ucelený, prehľadný a čitateľovi mohol slúžiť okrem iného aj ako návod k spusteniu, testovaniu a prípadne rozširovaniu vytvoreného riešenia.

Na začiatku popisujeme východiská práce, hlavne zdedené riešenie, odkazujeme na predchádzajúce riešenia a sumarizujeme všetky nami využívané zdroje. Nasleduje kapitola o biologickom modeli mrvaca, kde približujeme jeho anatomickú stavbu a kinematiku jeho chôdze. V časti venovanej nástrojom vysvetľujeme najskôr význam a výber zvolených nástrojov a postupne prechádzame k popisu ich inštalácie, architektúry, prostredia a práce s nimi. Riešenie naberá praktické črty v kapitole simulovaného modelu, kde objasňujeme kroky potrebné k vytvoreniu spomínaného modelu, od získania zdedených súborov, ich konvertovanie a úpravu, až po tvorbu súborov popisujúcich model. V záverečnej kapitole je vysvetlený proces tvorby a obsah súborov vytvorenia a riadenia kompletnej simulácie, ktorej výsledky sú nakoniec zhodnotené. Na konci dokumentu sa v prílohe nachádzajú rozmernejšie obrázky, ktoré kvôli ich veľkosti nebolo vhodné umiestniť do hlavného textu práce.

1.3 Motivácia

Veda je fascinujúca, je potrebné ju skúmať a porozumieť jej, priniest niečo nové a posunúť tak ľudstvo ďalej. Tak začína nejedna študentská práca. Či už to ich tvorcovia myslia úprimne, ukáže až ich životná cesta. Pravdou je, že väčšina z nás sa nezobúdza a nezaspáva s pocitom vedca. Trápia nás prízemnejšie problémy - niečo nás zvykne boliet, obávame sa výziev zajtrajška, utápame sa v hmle stereotypu. A keď si už povieme, že nás niečo fascinuje, je to skôr niečo ako zvuky bûrky, vôňa kvitnúceho orgovánu alebo tvár milovanej osoby. Keď som vyhľadával podklady k tejto práci, objasnila sa mi klúčová vec. Kým mnohí pri pohľade na mrvaca hľadajú insekticídnu kriedu, prípadne sa pousemejú nad jeho nemohúcnosťou a krehkosťou, existujú aj ľudia, ktorí ich skúmaniu obetovali celý život. Áno, existujú ľudia žijúci a neraz zomierajúci pre vedu, pre našu budúcnosť. Je to tvrdý oriešok a vyžaduje si veľké obety, zvlášť v tejto dobe poloprávd a širokých laktov. Sú to práve oni, kto vytvára skutočnú hodnotu pre ľudstvo. Je smutné, že ich poslanie a úsilie býva spoločnosťou nedocenené. Pre mňa osobne bola táto práca výzva a šanca na nahliadnutie do sveta biológie, robotiky a aj sveta vedcov. V tomto smere som si rozšíril znalosti a zručnosti, a hoci neviem, či to bude aj smer mojej kariéry, pokladám túto skúsenosť za cennú.

Kapitola 2

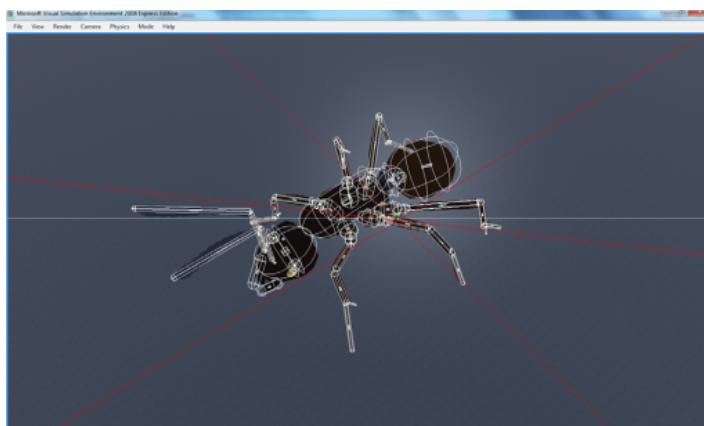
Východiská

2.1 Predošlé riešenia

2.1.1 Zdedené riešenie

V roku 2011 vytvoril Andrej Riška virtuálny simulátor na modelovanie chôdze mrvaca. Jeho simulátor ako jeden z mála spájal model mrvaca s jeho skutočnými biologickými dátami. Ako nástroj riešenia si vybral MRDS (Microsoft Robotics Developer Studio) [mrd], čo v tom čase bolo jedno z mála podporovaných voľne distribuovaných vývojových prostredí vhodných na prácu s reálnymi a simulovanými robotmi. MRDS používal programy napísané v C# a na zreálnenie simulácií fyzikálny engine AGEIA PhysX.

Začal úpravami získaného 3D modelu mrvaca a priradil mu fyzikálny model (obr. 2.1), ktorý bolo potrebné vypočítať podľa jednotlivých anatomických častí.



Obr. 2.1: Spojenie fyzikálneho modelu s vizuálnym v zdedenom riešení. [Ris11].

Následne implementoval rozhranie, ktoré dovoľovalo riadenie modelu. Na overenie správnosti do tohto riešenia zakomponoval multiagentový systém a spravil dva experimenty. Prvý dokazoval správnu spoluprácu fyzikálneho enginu s vytvoreným modelom a druhý implementoval riadenie pomocou dvoch agentov komunikujúcich medzi sebou

a rozhraním pomocou nepriamej komunikácie s prvkami subsumpčnej architektúry [Ris11]. V tom čase bolo jeho riešenie pripravené na ďalšie rozvíjanie, napríklad doplnenie o chôdzu alebo pridávanie senzorických vstupov. Riškove riešenie je dobrým priblížením problematiky a slúži nám ako náčrt možného postupu. 3D model organizmu, s ktorým pracoval, sa po úpravách použil aj v našej práci.

2.1.2 Historické riešenia

Východiská zahŕňajúce historické implementácie robotov a simulovaných šestohých modelov hmyzu približuje Andrej Riška vo svojej práci v kapitole 1.2, menovite sú to robot GENGHIS, jeho nástupcovia Hannibal a Attila, simulácia švába a hexapodu, simulácia a riadenie modelu pakobylky pomocou neurónovej siete WalkNet a robot HECTOR [Ris11]. Osobitne zaujímavým sú pre nás modely a riadenie paličkového hmyzu [CS05], [SHSC13], kde je dosiahnutá vernosť kinematike a dynamike biologickému modelu, no pracuje sa s vizuálne strohými modelmi. Ako sme už spomínali, to je medzera, ktorú sa naša práca snaží vyplniť, a síce spojenie vernosti prírodnej predlohy kinematiky a reálneho vizuálneho obalu v atraktívnom a dostupnom simulátore.

2.2 Iné východiská

Okrem vychádzania z predchádzajúcich riešení si práca vyžadovala čerpať zo širokej množiny komplementárnych zdrojov. Vychádzali sme z publikácií o anatómii mravca a kinematiky jeho chôdze, ktorým sú venované osobitné sekcie.

Riešenie zahrňalo prácu s 3D súbormi, od získania zdedených súborov, ich úpravou, po pripravenie na použitie v simulácii, pričom sme využili rôzne nástroje a návody. Tento proces je opísaný v kapitole o implementácii modelu mravca.

V neposlednom rade sme čerpali z publikácií a príkladov použitia zvoleného simulátora a riadiacej architektúry, ktorým je tiež venovaná osobitná kapitola.

Kapitola 3

Požiadavky na model

V tejto kapitole opíšeme anatómiu mrvaca a kinematiku jeho chôdze. Čerpáme pritom z reálnych dát biologického modelu, či už z encyklopédií, publikácií a vedeckých výskumov. Získané dáta a informácie neskôr používame pri tvorbe modelu a simulácie, aby bola v dostatočnej miere dosiahnutá vernosť riešenia prírode.

3.1 Anatómia mrvaca

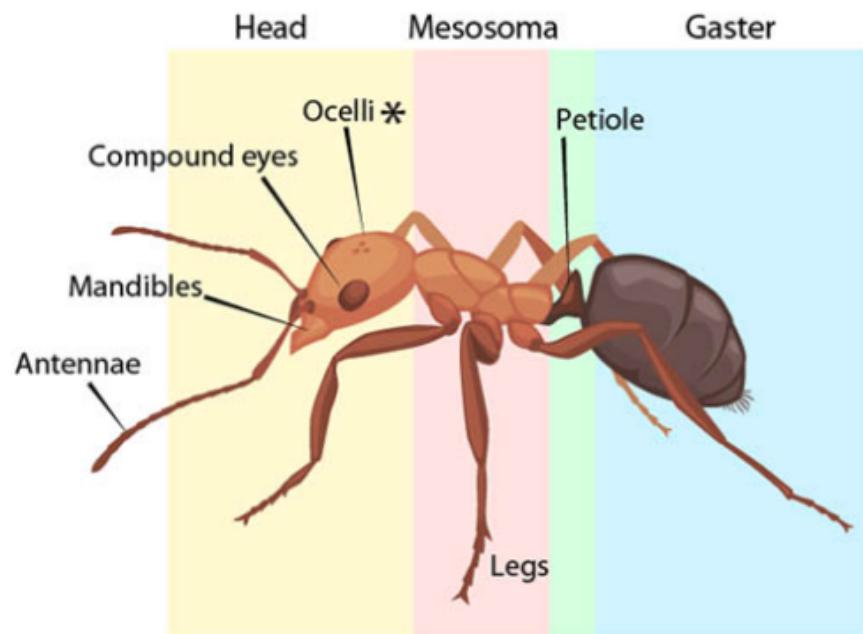
Priblížme si vonkajšiu anatomickú stavbu mrvaca. Jeho vnútornej stavbe sa budeme venovať len okrajovo a to z hľadiska funkcionality jednotlivých údov. Naopak, zaujímajú nás hlavne hmotnosti a tvary údov a spoje medzi nimi. Na popis mrvaca budeme používať aj anglické názvoslovie, keďže väčšina informácií je dostupná v tomto jazyku a je tomu tak aj v súboroch zdedeného riešenia.

Najdôležitejšie časti mrvaca sú hlava, stredná časť, bruško a 6 nožičiek (obr. č. 3.1). Najdôležitejšou súčasťou strednej časti (mesosoma) je trup (thorax). Na ten je napojených všetkých 6 nožičiek. Nachádzajú sa v ňom svaly zabezpečujúce ich pohyb.

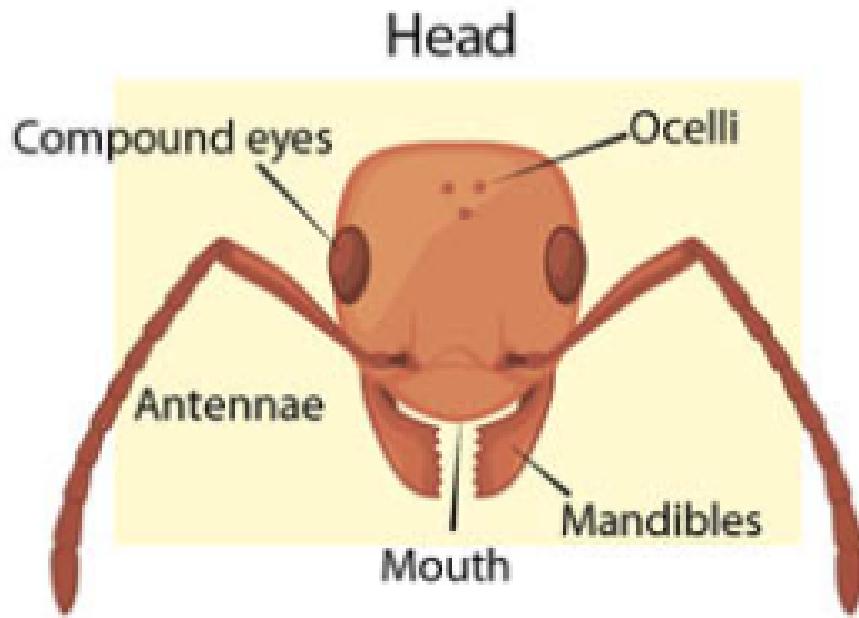
Bruško mrvaca (gaster) obsahuje srdce, tráviaci systém a rôzne žlazy. Môže byť zakončené žihadlom alebo otvorom na striekanie jedu. Táto časť predstavuje 50 až 60 percent hmotnosti mrvaca [RB14], [GLWL18]. Stred a bruško mrvaca spája orgán petiole, ktorý umožnuje flexibilný ohyb, napríklad pri bodaní.

Hlava mrvaca sa skladá z niekoľkých častí (obr č. 3.2). Jednou z vitálnych častí mrvaca sú hryzadlá (mandibles). Pomocou nich hryzie a bojuje. Na hlave sa nachádzajú aj zložené oči (compound eyes). Slepé druhy mravcov namiesto očí využívajú tri hlavové bodky (ocelli), ktoré sú citlivé na svetelné vnemy. Súčasťou hlavy sú dve tykadlá (antennae), nimi mravec skúma svoje okolie. Tykadlá sú umiestnené v jamkách (scape bed) a skladajú sa zo základu tykadla (scape) a článku (flagellum).

Nožičky mrvaca sú pre našu simuláciu kľúčové. Je ich šesť, po tri na oboch stranách. Skladajú sa z piatich častí v poradí od trupu k zemi: cox, trochanter, femur, tibia,



Obr. 3.1: Základné časti mravca – hlava (head), stred (mesosoma), bruško (gaster) a 6 nôh [Hol09].

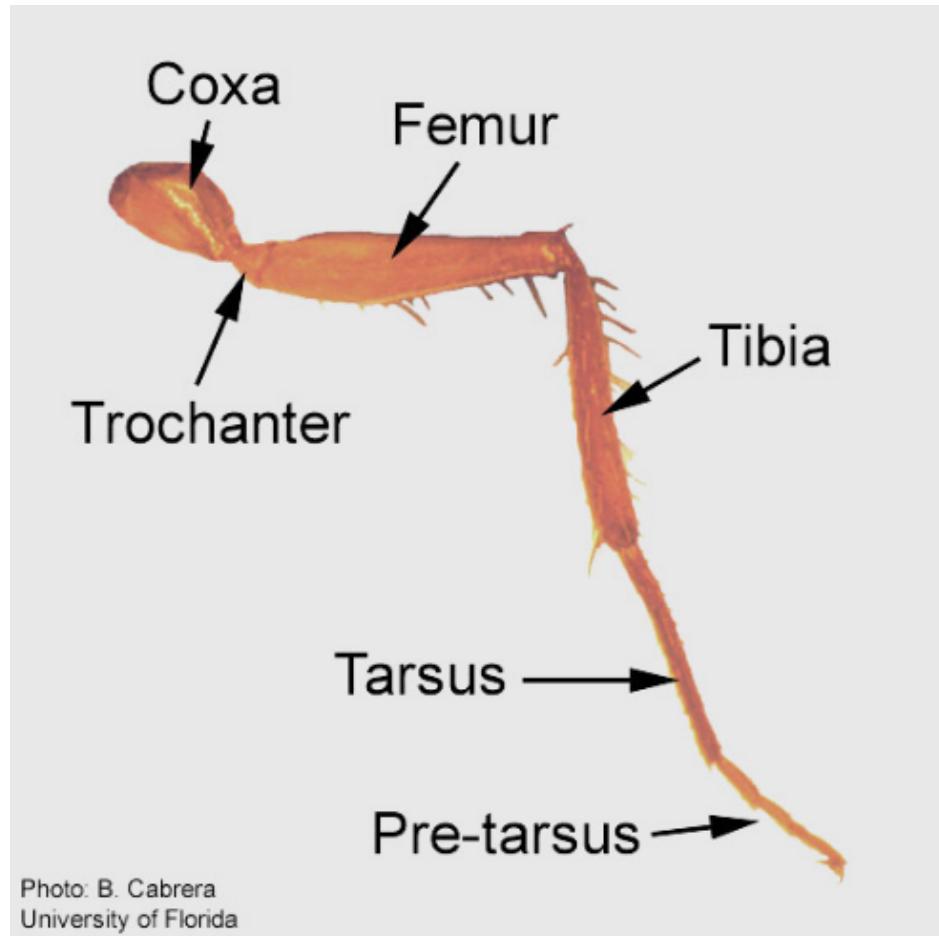


Obr. 3.2: Stavba hlavy mravca [Hol09].

tarsus. Tarsus sa môže rozdeliť na tarsus a pre-tarsus (obr č. 3.3).

Hmotnosti údov mravca sme získali zo zdrojov [RB14] a [GLWL18]. Porovnávame ich v tabuľke (3.1).

Vidíme, že najväčšiu hmotnosť má mravcovu bruško. Nožičky sú vzhľadom k telu ľahké. Hoci je celková hmotnosť mravca v meraniach takmer rovnaká, odlišujú sa v



Obr. 3.3: Stavba nožičky mrvaca, coxa sa pripája k trupu a pre-tarsus sa dotýka zeme [anaa].

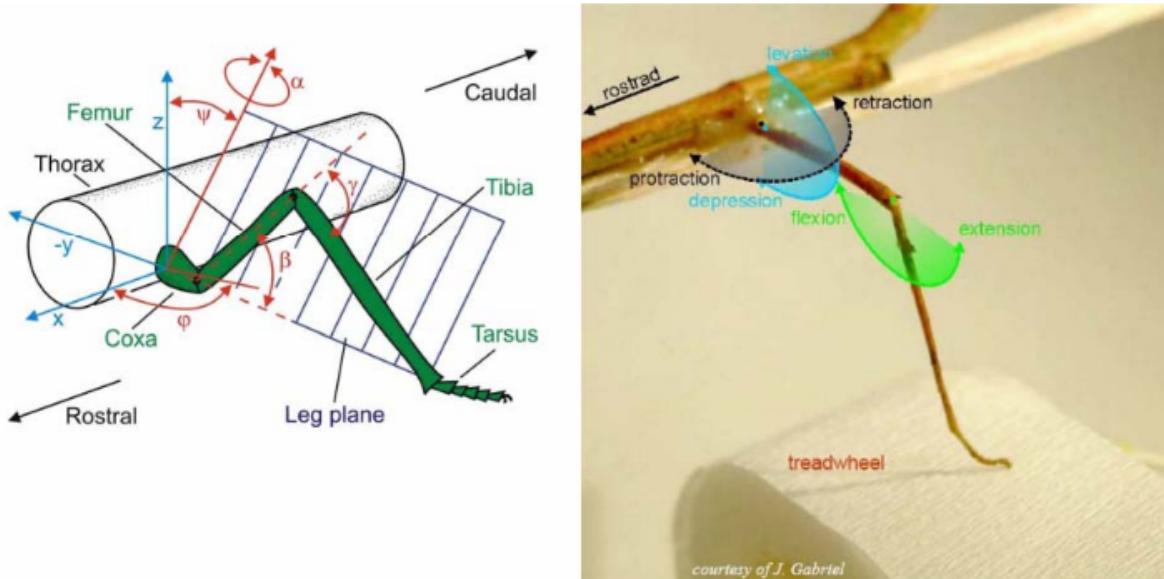
hmotnosti v mg	RB14	relat. hmotnosť	GLWL18	relat. hmotnosť
hlava	3.47	0.16	5.23	0.217
trup	3.38	0.16	3.87	0.161
bruško	13	0.58	11.35	0.472
predné nožičky			0.6	0.05
stredné nožičky			0.5	0.04
zadné nožičky			0.7	0.058
všetky nožičky	2.21	0.1		
spolu	22.06	0.997	22.25	0.998

Tabuľka 3.1: Hmotnosti mrvaca z dvoch rôznych zdrojov [RB14] a [GLWL18].

nameraných údajoch jednotlivých údov.

3.2 Kinematika chôdze mravca

Aby sme mohli simulovať kráčanie mravca, potrebujeme získať detailnejšie informácie o jeho nožičkách a samotnej chôdzi. Obrázok (obr. č. 3.4) popisuje možné roviny ohybu kľbov šestnohého hmyzu všeobecne (vľavo) a končatiny pakobylky (vpravo) [Sch07]. Týmto spôsobom sa budú hýbať aj končatiny nášho modelu. Každá končatina obsahuje

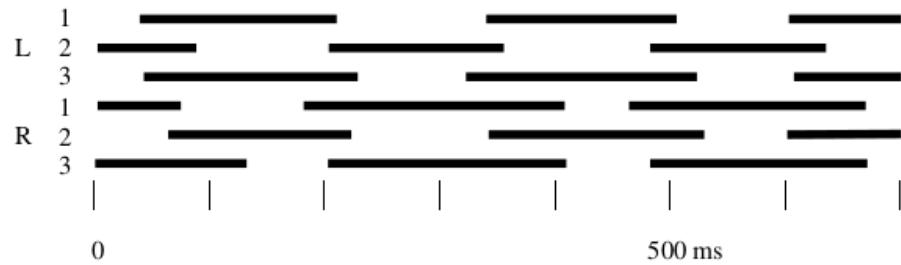


Obr. 3.4: Možnosti otáčania jednotlivých kľbov vo všeobecnom prípade (vľavo) a v prípade končatiny pakobylky (vpravo) [Sch07].

tri kľby, predstavuje teda systém s troma stupňami voľnosti. Prvý kľb spája thorax a coxu a rotuje okolo osi z a zabezbečuje presun končatiny pozdĺžne vzhľadom k telu mravca. Druhý kľb spája coxu a trochanter, rotuje okolo osi x a dvíha končatinu pri vykonávaní kroku vpred. Tretí kľb spája femur a tibiu, rotuje takisto ako druhý okolo osi x a svojim pohybom pomáha predĺžiť krok. Zostávajúce dva kľby spájajúce trochanter a femur a nižšie tiež tibiu a tarsus sú fixné.

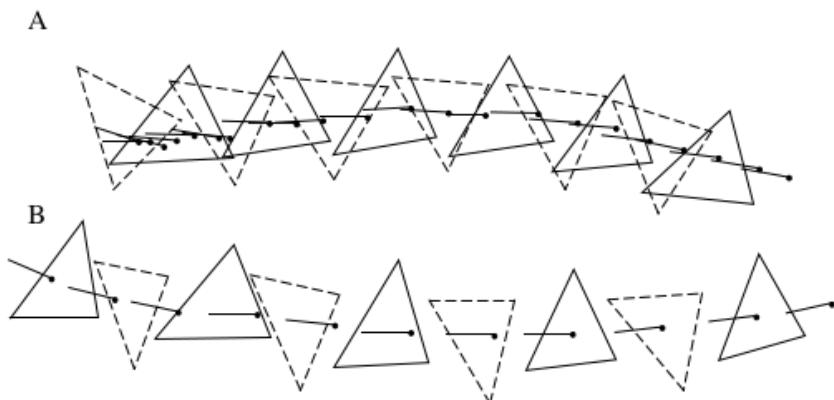
Koordinácia nožičiek pri chôdzi mravca, ale aj niektorých iných šestnohých druhov hmyzu je reprezentovaná takzvaným striedaním trojnožiek (alternating tripods pattern, [HUG52]). Mravec tak kráča pri rôznych rýchlosťach, od pomalej chôdze až po rýchly beh. Predná a zadná noha na jednej strane tvoria trojnožku so strednou nohou na strane opačnej. Zrkadlovo sa vytvorí doplňujúca opačná trojnožka. Trojnožky sa hýbu striedavo synchronizované v dvoch fázach: fáza swing (resp. phase) a fáza stance (antiphase)(obr. č. 3.5). Fáza swing predstavuje zdvihnutie a presun končatín smerom dopredu. Pri fáze stance končatiny zostávajú na zemi a otáčajú sa v prvom kľbe smerom dozadu, čím posúvajú samotné telo dopredu. Ak je teda jedna trojnožka vo fáze swing, druhá je práve vo fáze stance. Striedaním týchto fáz sa zabezpečí stály pohyb mravca smerom dopredu. Stabilita mravca je vždy zabezpečená trojnožkou, ktorá sa

práve dotýka zeme. Za zmienku stojí, že aj pri rýchлом behu je vždy mravec jednou trojnožkou na zemi, na rozdiel od mnohých dvoj a štvornohých živočíchov, ktorých beh obsahuje vzdušnú fázu, počas ktorej sa na chvíľu vôbec nedotýkajú zeme.



Obr. 3.5: Chôdza mravca reprezentovaná striedením trojnožiek. Čierne čiary predstavujú fázu stance, L ľavú a R pravú stranu mravca. Čísla 1, 2, 3 predstavujú predné, stredné a zadné nožičky [Zol94].

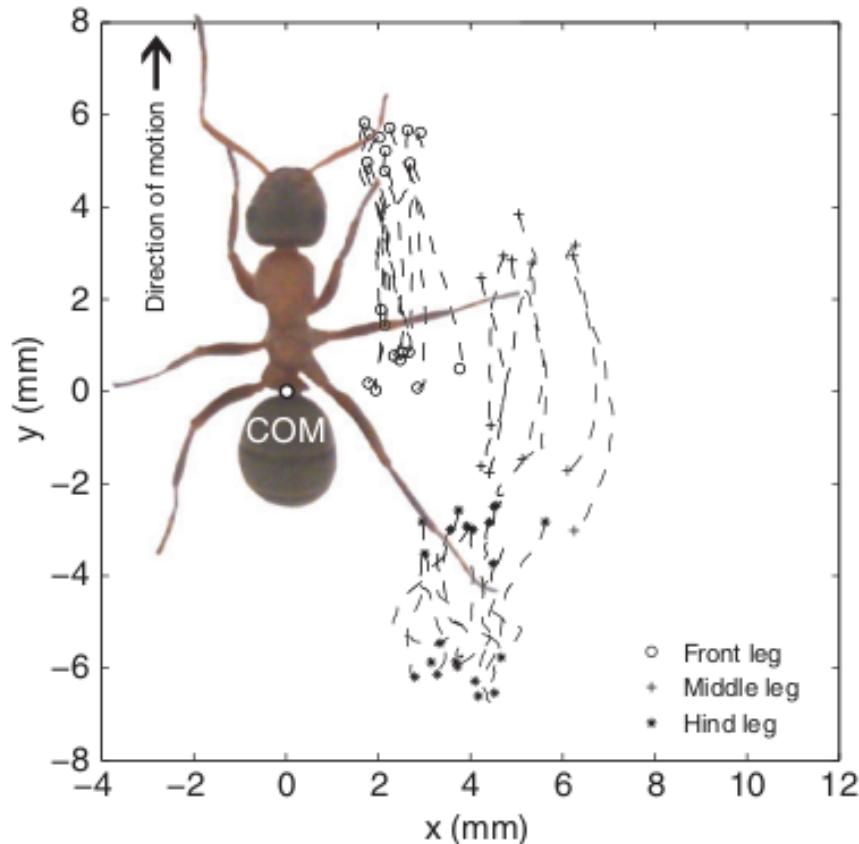
Zaujímavý je vplyv rýchlosťi chôdze na vzájomnú polohu bodov dotyku trojnožky so zemou. Vznikajúce trojuholníky sú pri vyššej rýchlosťi umiestnené ďalej od seba, no ich vrcholy nemenia s vyššou rýchlosťou vzájomnú polohu (obr. č. 3.6).



Obr. 3.6: Vplyv rýchlosťi mravca na trojnožku, rýchlosť 17 mm/s (A), 43 mm/s (B). Chôdza prebieha zľava doprava, celá čiara reprezentuje ľavú trojnožku, prerušovaná pravú [Zol94].

Nevyhnutnou súčasťou simulovalia chôdze mravca je informácia o predných a zadných krajiných bodoch fáz končatín. Tieto údaje boli získané sledovaním pohybu živého mravca na sklenenej podložke s odtlačkovou vrstvou. Proces je snímaný viacerými kamery a získané dátu sa softvérovo spracovávajú a vyhodnocujú. Tažisko mravca sa získalo zavesením mravca s odrezanými nohami do vzdachu pomocou nylonovej nite upevnenej na jeho trup a následným posunom spoja, až kým sa mravec nedostal do horizontálnej polohy [Man72].

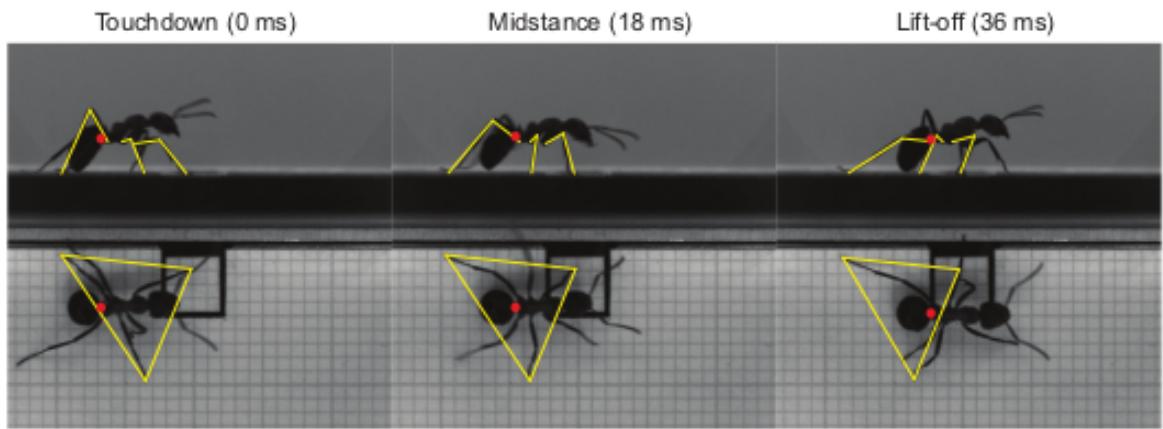
Tarsus, najspodnejšia časť končatiny ktorá sa dotýka zeme, sa počas chôdze pohybuje takmer rovnobežne s pozdĺžnou osou tela mrvaca. Predné nohy dopadajú na zem najbližšie k pozdĺžnej osi telu, najdalej od nej dopadajú stredné. Ku dotyku a zdvihu pri predných nohách dochádza pred tažiskom, pri zadných nohách za tažiskom. Stredné nohy dopadajú pred tažiskom a zdvívajú sa za tažiskom (obr. č. 3.7).



Obr. 3.7: Krajiné body fáz chôdze mrvaca. COM je Center of mass, čiže tažisko [RWB09].

Priblížme si pohyby a uhly končatín trojnožky v stance fáze. Pre zjednodušenie si rozdelme každú končatinu na dve funkčné časti: horná časť sa skladá z femuru a spodná z tibie a celého tarsusu (obr. č. 3.8). Pri pohlade z boku dopadá spodná časť prednej nohy na zem pod uhlom približne 50 stupňov, zatiaľ čo horná časť je pod uhlom okolo 10 stupňov vzhľadom k zemi. Pri dopade tak zvierajú uhol približne 120 stupňov. V priebehu fázy stance sa ich vzájomný uhol zmenší až na 55 stupňov. V momente zdvihu zviera spodná časť prednej nohy so zemou uhol 110 stupňov. Zadná noha sa naopak počas tejto fázy predlžuje. Kým na jej začiatku zviera horná a spodná časť uhol 55 stupňov, na jej konci je to 140 stupňov. Uhol spodnej časti so zemou sa zmenšuje zo 65 na 30 stupňov. Stredná noha sa v kľbe medzi femurom a tibiou počas tejto fázy skoro vôbec neohýba, čo zapríčinuje relatívne nemennú efektívnu dĺžku tejto končatiny. Jej

spodná časť zviera so zemou uhol od 25 do 30 stupňov.



Obr. 3.8: Detail pohybu a uhlov trojnožky na začiatku, v strede a konci stance fázy pri rýchlosťi mravca 121 mm/s. Červená bodka predstavuje tažisko tela, žlté čiary pri bočnom pohľade znázorňujú horné a spodné funkčné časti nôh, pri pohľade zhora spájajú kontaktné body trojnožky so zemou. [RB14].

Kapitola 4

Implementačné nástroje

4.1 Úvod

Simulátory v dnešnej dobe zohrávajú v robotike vitálnu úlohu ako nástroj na modelovanie reálneho sveta vo virtuálnom prostredí. Prinášajú hned niekoľko výhod, práca s nimi je všeobecne rýchlejšia, lacnejšia, bezpečnejšia a menej frustrujúca. Nepotrebueme hardvér robota, zmeny jeho parametrov sú jednoduchšie, simulátor, naše telo a peňaženka zvláda experimentálne a nepredvídateľné správanie lepšie. Vďaka tomu sú ideálne na výučbu, tvorbu a testovanie nových konceptov a algoritmov.

V zadaní práce sa predpokladalo rozvíjanie riešenia z roku 2011 spraveného v MRDS, prípadne jeho reimplementácia v modernejších nástrojoch. MRDS však od roku 2012 nebolo aktualizované a v roku 2014 Microsoft pozastavil činnosť svojej robotickej skupiny [lay]. Projekt tak stratil podporu a následne aj komunitu. Preto sme si vybrali reimplementáciu v nástrojoch Gazebo [Gaza] a ROS [ROSA], ktoré tu popisujeme.

4.2 Simulátor Gazebo

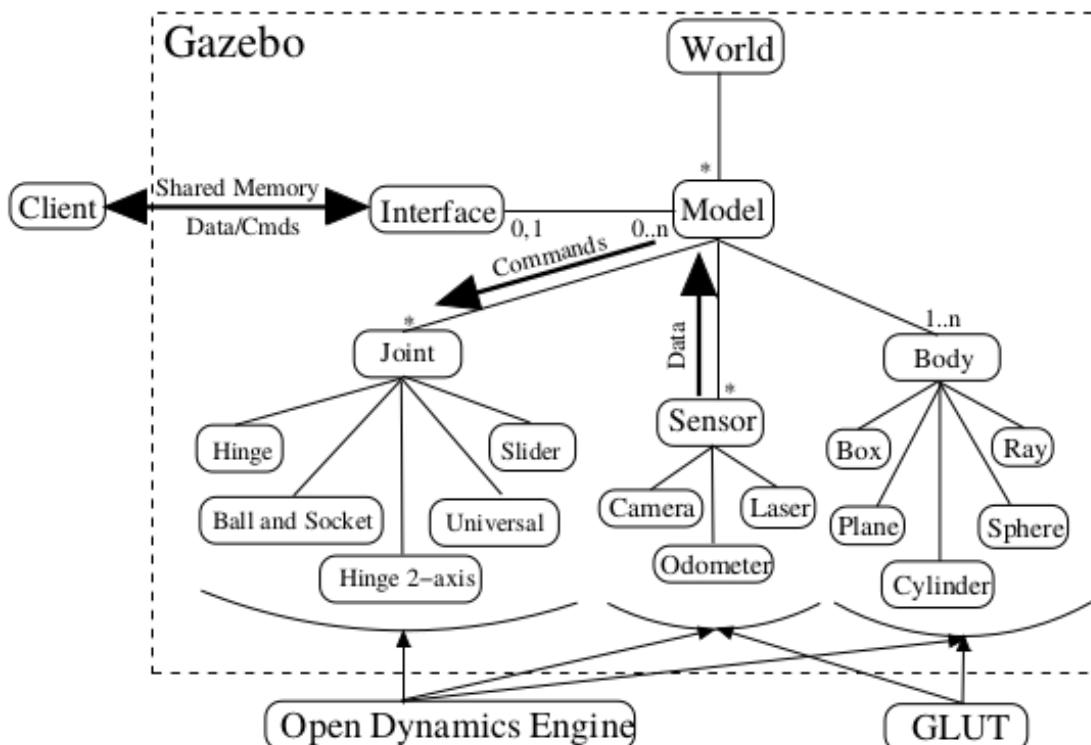
Pri našom výbere simulátora sme hľadali moderný, rozšírený a voľne šíriteľný nástroj. Práve takýmto je Gazebo, realistický 3D simulátor robotov, ktorého začiatky siahajú do roku 2003. Bol vytvorený kvôli potrebe simulovalia mobilných robotov v 3D prostredí, ktorí s prostredím môžu fyzikálne uspokojivo interagovať. Jeho 2D predchodcovia, napríklad Stage [Sta], sa zaoberali hlavne navigáciou kinematicky a geometricky jednoduchých objektov v statickom planárnom prostredí. Takéto simulátory boli rýchle a efektívne, no nedokázali simulať robotov vo vzduchu či vo vode alebo manipulovanie robota s objektami prostredia.

Gazebo túto medzeru zaplnil, môže simulať viacerých robotov naraz vo vnútornom aj vonkajšom 3D prostredí, interakciu robotov s prostredím, podporuje senzory a senzorickú odozvu.

Náčrt pôvodnej architektúry Gazebo (obr č. 4.1). World, čiže svet, reprezentuje množinu všetkých modelov a faktorov prostredia, ako sú napríklad gravitácia alebo svetlo. Každý model sa skladá aspoň z jedného tela (body, neskôr ho nahradil pojem link) a ľubovoľného počtu kľbov a senzorov. Knižnice fyzikálnych enginov a grafiky komunikujú s Gazebom na najnižšej úrovni. Klient komunikuje s rozhraním cez zdieľanú pamäť [KH04].

Na modelovanie dynamiky tuhých telies využíval pôvodne len knižnicu ODE (Open Dynamics Engine)[ODE], neskôr bola pridaná podpora pre Bullet [Bul], Simbody [Sima] a DART [DAR]. Pri spustení Gazebo alebo vo world súbore môžeme simulátoru poviedať, aký fyzikálny engine má byť použitý. V prípade záujmu môže čitateľ získať ich podrobnejší opis a porovnanie na videu z príspevku počas konferencie ROSCon 2014 [eng].

Na vizualizáciu simulácie a GUI boli v prvých verziách využívané knižnice OpenGL [Ope] a GLUT (OpenGL Utility Toolkit)[GLU]. V nových verziach simulátora sa používa na účely renderovania engine OGRE [OGR] a na GUI framework Qt [Qt].



Obr. 4.1: Pôvodná architektúra Gazebo [KH04].

4.2.1 Inštalácia

Simulátor Gazebo je otvorený voľne šíriteľný softvér. Má viac verzií, je dobré pracovať s najnovšou, aktuálne je to verzia 9. Vo väčšine prípadov sa inštaluje a používa na sys-

témoch Linux distribúcie Ubuntu. My používame v tomto čase najaktuálnejšiu verziu 18, čitateľom odporúčame aby bola aspoň verzie 16. Je možné inštalovať a používať ho aj na iných distribúciach Linuxu a tiež aj na Windows alebo Macu, no nie je to vždy priamočiare a vyžaduje si to niekedy viac práce. Na webstránke tohto nástroja sú tutoriály aj na tieto prípady [Gazc], my budeme popisovať a pracovať s Ubuntu.

Inštalácia sa skladá z niekoľko príkazov. Prvým sa pridá adresa Gazebo repozitárov do súboru sources.list, druhým sa zabezpečí správnosť zdroja. Tretím aktualizujeme všetky naše repozitáre, vrátane novopridaného. Až posledný príkaz vykoná samotnú inštaláciu:

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-stable.list'
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt-get install gazebo9
```

Systém sa nás spýta na používateľské heslo a potvrdenie inštalácie. Správnosť inštalácie overíme príkazom, ktorým spustíme simulátor:

```
gazebo
```

Týmto príkazom sa v skutočnosti spustia dva navzájom komunikujúce programy:

- gzserver - server, hlavná časť simulátora, používa fyzikálny engine na simulovanie, generuje senzorické dátá
- gzclient - klient, pripája sa na server, zobrazuje GUI, vizualizuje prostredie simulácie, môžeme cez neho interagovať so simuláciou, pracovať s modelmi

4.2.2 Prostredie

Grafické rozhranie

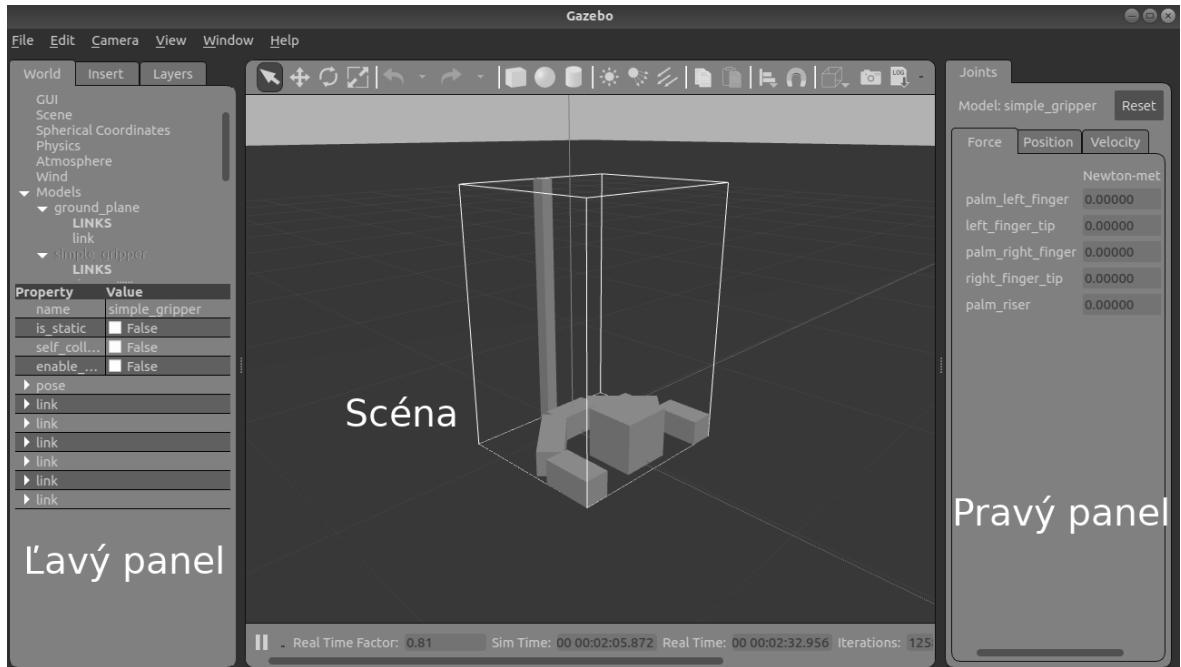
Spustime Gazebo so zabudovaným ukážkovým súborom gripper.world:

```
gazebo worlds/gripper.world
```

Na tomto jednoduchom modeli sa zoznámime s grafickým rozhraním simulátora, prípadne môžeme spustiť a testovať iný predpripripravený model zo zoznamu:

```
ls /usr/share/gazebo-9/worlds
```

Hlavnou časťou rozhrania (obr č. 4.2) je scéna. Vľavo od scény sa nachádza lavý panel, vpravo nájdeme pravý, ten je však po spustení simulátora skrytý, musíme ho najskôr rozbalíť. Nad a pod scénou sa nachádzajú horný, respektíve dolný panel nástrojov. Na úplnom vrchu rozhrania sa tradične nachádza menu aplikácie.



Obr. 4.2: Gazebo GUI.

Na scéne sa vykresluje svet simulácie, myšou s ňou môžeme interaktívne pracovať. Lavý panel obsahuje tri karty:

- World - tu sú zobrazené entity a parametre nášho sveta, modely a ich štruktúra, svetlá, rôzne nastavenia zobrazenia scény, fyziky
- Insert - do scény môžeme pridať modely z rôznych databáz, či už vlastné alebo z komunity
- Layers - modely môžeme zaradzovať do rozdielnych skupín viditeľnosti

Po nakliknutí modelu sa zobrazia jeho klíby v pravom paneli. S jeho pohyblivými časťami môžeme manipulovať tak, že im zadávame rôzne fyzikálne parametre a pozorujeme zmenu v simulácii.

Cez spodný panel nástrojov pozastavujeme alebo spúšťame simuláciu. Tiež tu môžeme sledovať informácie o čase a dĺžke simulácie a FPS (framerate per second) kvalite vykreslovania.

Horný panel nástrojov obsahuje najviac používané nástroje simulácie. Transláciu, rotáciu, škálovanie modelov, pridávanie jednoduchých objektov či svetiel. Nechýba tlačidlo dozadu alebo snímka obrazovky.

V aplikačnom menu sa nachádzajú nástroje na prácu so súbormi, úpravu modelov, nastavenia simulácie a zobrazenia a používateľská pomoc. Grafické rozhranie je podrobnejšie popísané na webstránke nástroja [Gazb].

Súbory

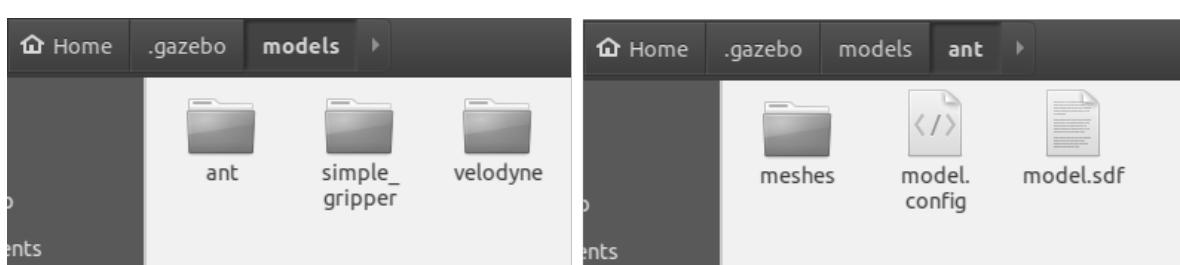
Pri práci s Gazebo sa stretneme s dvoma typmi XML súborov popisujúcich simuláciu alebo jej časti:

- .sdf súbory - popisujú štruktúru a vlastnosti konkrétneho modelu robota
- .world súbory - popisujú celý svet simulácie vrátane nastavení fyziky, svetiel, scény, kamier, pluginov, modely sa do nich pridávajú buď referenciou na SDF súbor cez include tag, alebo sú v nich popísané podobne ako v osobitnom SDF súbore

Gazebo je cez terminál spustiteľné len s world súbormi. V spustenej aplikácii môžeme importovať SDF modely do scény cez Insert, vytvárať, upravovať a exportovať ich cez Model Editor. Celú scénu je možné uložiť len ako súbor typu world. Do novovzniknutého súboru sa zapíšu nastavenia scény, ktoré vidíme vo World tabe ľavého panelu.

Databáza modelov obsahuje priečinky jednotlivých modelov (obr č. 4.3 vľavo), každý z nich obsahuje tieto položky (obr č. 4.3 vpravo) [Gazd]:

- model.sdf - povinný súbor popisujúci štruktúru a vlastnosti modelu
- model.config - povinný súbor obsahujúci meta dátá modelu - verzia SDF, autor a slovný popis modelu
- meshes, nepovinný priečinok s 3D súbormi modelu
- materials, nepovinný priečinok obsahujúci textúry, resp. materiály
- plugins, nepovinný priečinok obsahujúci súbory na riadenie modelu



Obr. 4.3: Štruktúra priečinkov databázy modelov Gazebo.

Štruktúra SDF modelov

Model definovaný v SDF je kolekciou týchto častí [sdfb]:

- linky (články) - fyzické časti modelu, obsahujú vizuálne, kolízne a inerciálne dátá

- jointy (klíby) - rôzne spoje medzi článkami modelu
- pluginy - knižnice slúžiace na riadenie modelu

Všeobecný postup pri tvorbe SDF modelu je nasledovný [sdfb]:

1. pridanie článku
2. pridanie kolízneho elementu danému článku
3. pridanie vizuálneho elementu danému článku
4. pridanie inerciálnych vlastností danému článku
5. opakovanie tohto postupu na všetky články modelu
6. pridanie klíbov, ak existujú
7. pridanie pluginov, ak existujú

Vizuálne a kolízne elementy sú definované bud jednoduchými geometrickými tvarmi alebo sa odvolávajú na 3D súbory (obr č. 4.4) v meshes podpriečinku modelu. Podporované sú súbory OBJ (.obj) [obj], COLLADA (.dae) - COLLABorative Design Activity [dae], STL (.stl) - skratka stereolithography [stl].

```
<link name="link_with_geom">
  <collision name="collision_geom">
    <geometry>
      <box>
        <size>0.05 0.05 1.0</size>
      </box>
    </geometry>
  </collision>
  <visual name="visual_geom">
    <geometry>
      <box>
        <size>0.05 0.05 1.0</size>
      </box>
    </geometry>
  </visual>
</link>
```

```
<link name="link_with_mesh">
  <collision name="collision_mesh">
    <geometry>
      <mesh>
        <uri>model://example/meshes/example.stl</uri>
      </mesh>
    </geometry>
  </collision>
  <visual name="visual_mesh">
    <geometry>
      <mesh>
        <uri>model://ant/meshes/example.dae</uri>
      </mesh>
    </geometry>
  </visual>
</link>
```

Obr. 4.4: Vľavo použitie geometrického tvaru, vpravo referencia na 3D súbor pri vizuálnom a kolíznom elemente SDF.

V SDF možeme spájať články týmito typmi klíbov [sdfd]:

- fixed - pevný spoj, spája dva linky do nemennej vzájomnej polohy
- revolute - klíb otáčajúci sa na jednej osi
- revolute2 - dva revolute klíby v sérii
- prismatic - posuvný klíb na jednej osi

- ball - guľový kĺb
- universal - guľový kĺb obmedzujúci pohyb na jednej osi
- screw - závitový kĺb posúvajúci sa a otáčajúci sa na jednej osi
- gearbox - sprevodované revolute jointy

4.3 Pridanie riadiacej architektúry

V začiatkoch našej práce sme sa zoznámili a pracovali len s Gazebo. Po čase sme zistili, že samotný simulátor nám stačiť nebude. Náš model je verne poskladaný a cez grafické rozhranie simulátora s ním môžeme aj pohybovať, no predmetom práce je jeho riadenie, konkrétnie schopnosť chôdze. Aby to náš model mravca dokázal, potrebujeme simulátor doplniť riadiacou architektúrou. Vtedy sa nám objasnil jav, prečo sa často v zdrojoch a na fórách objavuje spoločne s Gazebo nástroj ROS, znamenajúci Robot Operating System.

4.4 Riadiaca architektúra ROS

ROS je otvorený framework na prácu s robotmi. Je to kolekcia nástrojov a knižníc, umožňúca tvorbu komplexných a robustných robotických systémov naprieč rozličnými platformami. Jeho hlavným charakteristickým znakom je distribuovanosť, kde sú komplexné systémy tvorené mnohými úzko zameranými navzájom komunikujúcimi nezávislými programami. Plusom je aj široká množina podporovaných programovacích jazykov [rosb].

Podobne ako pri našom simulátore, jeho začiatky sa datujú do polky 2000's [QGS15]. V tej dobe sa veľa jednotlivcov a inštitúcií venujúcich sa robotike obrazne povedané hralo chtiac či nechtiac na svojom pieskovisku. Hlavným dôvodom jeho vzniku bola potreba vytvoriť nástroj na riadenie robotov, ktorý by bol pre komunitu a z komunity. Časom sa ukázalo, že zvolený postoj a postup robotike nesmierne pomohol a posunul ju správnym smerom. Dôkazom sú tisíce používateľov, od gárážových nadšencov, vzdelávacie a priemyselné inštitúcie, až po NASA [BGE⁺16].

4.4.1 Inštalácia

ROS, podobne ako samotné Ubuntu, vydáva nové verzie softvéru ako distribúcie. V čase našej práce je najnovšia a zároveň odporúčaná verzia Melodic Morenia. Popisujeme a pracujeme s touto distribúciou. Postup inštalácie je podobný ako pri Gazebo:

```

sudo sh -c \
'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > \ /etc/
 apt/sources.list.d/ros-latest.list'
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
sudo apt-get update
sudo apt-get install ros-melodic-desktop-full python-rosinstall

```

Aby ROS na v našom systéme fungoval správne, musíme ešte inicializovať rosdep a takzvané enviroment variables:

```

sudo rosdep init
rosdep update
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc

```

V tomto momente je dobré nainštalovať aj ROS balík, ktorý budeme používať na spojenie ROS a Gazebo:

```

sudo apt-get install ros-melodic-gazebo-ros-pkgs ros-melodic-gazebo-
ros-control

```

4.4.2 Architektúra a prostredie

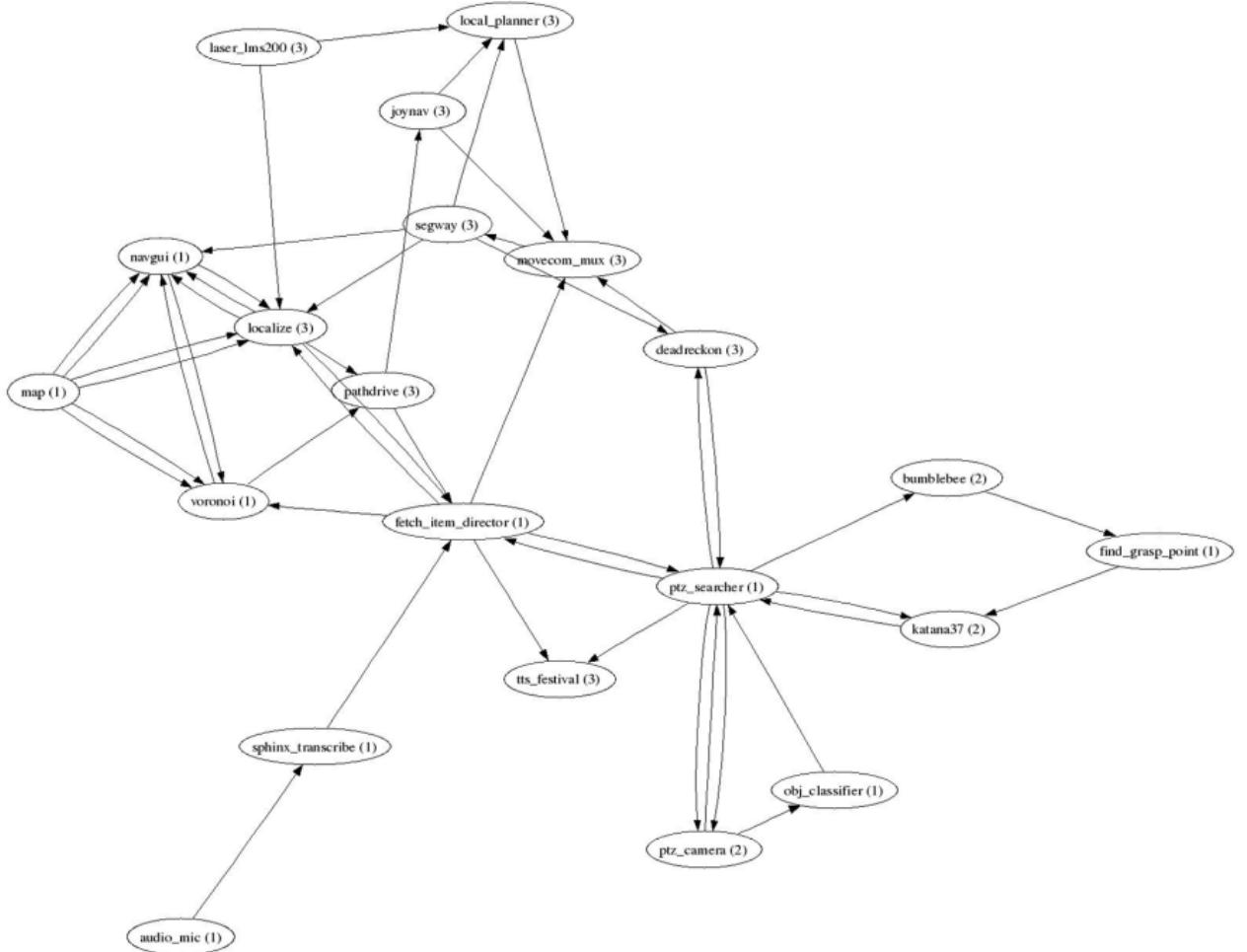
Systém ROS sa v zmysle decentralizovanosti skladá z množstva nezávislých častí a nástrojov. Priblížme si tie dôležité z nich, s ktorými sa stretneme a získajme predstavu o jeho koncepte a fungovaní. Akýkolvek ROS systém je možné vykresliť ako matematický graf. Jeho vrcholy nazývame nodes a sú to nezávislé programy, hrany reprezentujú komunikáciu medzi vrcholmi vo forme správ. Vo všeobecnosti sú vrcholy POSIX procesy a hrany sú TCP konekcie. Tento koncept realizuje dizajnové ciele ROS [QGS15]:

- riešenie sa dá rozdeliť na nezávislé podsystémy
- oddelené podsystémy sa môžu použiť na iné riešenia
- pri správnej hardvérovej a geometrickej abstrakcii sa časti softvéru dajú použiť s akýmkolvek robotom

Ilustrujme tento koncept na grafe "nájdi a prines" robota (obr č. 4.5). Horná polovica grafu je dedikovaná navigácii, pravý spodný roh videniu a manipuláciu s predmetmi.

Ako buildovací nástroj je aktuálne používaný catkin [cat], v starších verziach sa môžeme stretnúť s rosbuid. Je to rozšírenie nástroja CMake [cma] na potreby ROS. Používame ho na tvorbu priečinkov a súborov našich riešení. Z hľadiska komplilácie a linkovania sú pre catkin dôležité súbory CMakeLists.txt a package.xml, ktoré je potrebné upraviť podľa potrieb konkrétneho riešenia.

Pracovné prostredie, takzvané workspace, je priečinok v ktorom sa nachádzajú jednotlivé balíky, pričom catkin zabezpečuje ich kompliláciu a prácu s nimi. Môže existovať



Obr. 4.5: ROS graf "nájdi a prines" robota [QGS15].

viacero prostredí, no nachádzat sa môžeme v danom čase práve v jednom, pričom pracujeme iba so súbormi v ňom. Prostredie s názvom catkin_ws vytvoríme a inicializujeme nasledovými príkazmi:

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

Do súboru .bashrc je potrebné pridať devel nastavovací súbor, aby shell vedel nájsť nás kód a aby sme nemuseli tento krok opakovať po každom otvorení nového terminálu. Tento postup funguje iba pri práci s jedným pracovným prostredím.

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

Štruktúra pracovného prostredia je nasledovná (obr č. 4.6) [MSFM16]:

- priečinok build obsahuje dočasné súbory používané catkin a cmake

- priečinok devel obsahuje skompilované súbory
- priečinok src obsahuje balíky a ich súbory



Obr. 4.6: Štruktúra novovytvoreného pracovného prostredia catkin_ws [MSFM16].

Všetok ROS softvér je organizovaný do balíkov, takzvaných packages. Balík je ucelená kolekcia súborov, či už spustiteľných alebo komplementárnych, z ktorých každý plní svoju špecifickú úlohu [O'K13]. Nachádzajú sa v priečinku src pracovného prostredia. Pri práci s balíkmi sa najčastejšie používajú tieto príkazy:

- príkaz catkin_create_pkg sa používa na vytváranie nových balíkov
- príkaz catkin_make sa používa na kompliaciu pracovného prostredia
- príkaz rospack sa používa na získavanie informácií o balíkoch alebo ich hľadaní
- príkaz rosdep sa používa na zobrazenie obsahu balíkov

V priečinku každého balíka sa musia nachádzať súbory CMakeLists.txt a package.xml popisujúce balík a obsahujúce dôležité informácie pre kompliaciu. Balíky môžu obsahovať rôzne podpriečinky, v našom riešení sa stretнемe s:

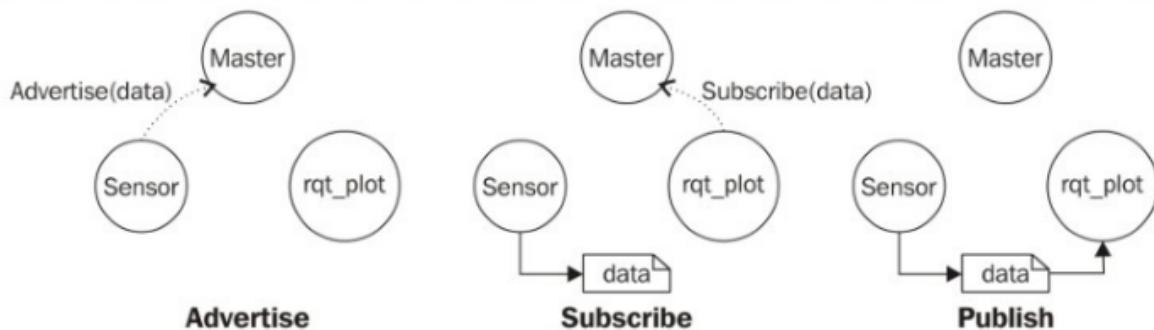
- priečinok config obsahuje konfiguračné súbory simulácie
- priečinok launch obsahuje spúšťacie súbory simulácie
- priečinok meshes obsahuje 3D súbory modelov
- priečinok scripts obsahuje programy v skriptovacom jazyku
- priečinok src obsahuje programy, typicky moduly vyžadujúce inštaláciu
- priečinok urdf obsahuje súbory popisujúce modely používané simuláciou
- priečinok worlds obsahuje world súbory využívané Gazebo

Nato, aby programy vedeli medzi sebou nadviazať spojenie a následne komunikovať na báze rovný s rovným, je potrebný takzvaný ROS master. Ten sa spúšta príkazom:

```
roscore
```

Bez spusteného roscore nemôže fungovať žiadny ROS systém. Každý program sa pri svojom spustení spája s roscore. Ak chce program posielat dátu, ako prvé musí cez roscore zverejniť (advertise) názov témy a typ správ, aké v nej bude zdieľať. Po tomto zverejnení program odosiela dátu do témy a je nazývaný odosielateľ (publisher). Program, ktorý chce získať dátu z témy pri spustení oznámi roscore, z akej témy chce prijímať. Tento program je nazývaný prijímateľ (subscriber). Následne programy komunikujú cez publisher-subscriber vzťah, v ktorom už roscore nezohráva úlohu. Komunikácia medzi programami v ROS je teda hybridom medzi klient-server systémom a plne distribuovaným. Tento proces komunikácie je znázornený na diagrame (obr č.4.7).

Roscore tiež poskytuje ROS systému takzvaný parameter server, implementovaný cez protokol XMLRPC [xmlb]. Programy cez neho zdieľajú dátu a informácie, napríklad popis robotov alebo parametre pre algoritmy.



Obr. 4.7: Diagram komunikácie medzi dvoma ROS programami [MSFM16].

Programy, čiže vrcholy ROS grafu sú nazývané nodes. Sú to nezávislé spustiteľné súbory. Aby sa vyhlo nejednoznačnosti, každý program má v systéme svoje unikátné meno. ROS poskytuje viacero knižníc na tvorbu programov, najčastejšie sa používajú roscpp na C++ a rospy na Python. Na ukončenie programov používame klávesovú skratku Ctrl+C. Základné príkazy, s ktorými pracujeme sú:

- príkaz rosnode list vypíše všetky spustené programy
- príkaz rosnode info vypíše informácie o konkrétnom programe
- príkaz rosrun spustí konkrétny program
- príkaz roslaunch spustí launch súbor, cez ktorý môžeme spustiť niekoľko programov naraz

Komunikácia medzi programami je realizovaná výmenou správ cez témy, takzvané topics, procesom, ktorý sme popísali vyššie. Typ správy v téme je jednoznačne daný, odosielá a prijíma sa jeden typ správy. Môže to byť napríklad string, float64, Image, Twist a veľa ďalších. Je možné si definovať aj vlastné typy správ. Do jednej témy môže odosielať dátu viacero odosielateľov naraz a tiež získavať z nej dátu viacero prijímateľov naraz. S témami pracujeme pomocov príkazov:

- príkaz rostopic list vypíše všetky aktívne témy
- príkaz rostopic info vypíše informácie o téme
- príkaz rostopic echo pošle dátu do témy

Spustime ilustračný príklad:

```
roslaunch rospy_tutorials talker_listener.launch
```

ROS najskôr zistí, či je spustené roscore a ak nie je, spustí ho. Následne sa spustia programy z daného launch súboru, v tomto prípade je to talker a listener. Talker posielá s určitou frekvenciou informácie o aktuálnom čase typu string do témy chatter. Program listener prijíma dátu z tejto témy a vždy pri prijatí ich vypíše na konzolu. ROS graf tohto systému je jednoduchý, je znázornený na diagrame (obr č.4.8). ROS graf akéhokoľvek spusteného ROS systému sa zobrazí zadaním príkazu v osobitnom termináli:

```
rqt_graph
```



Obr. 4.8: ROS graf systému talker-listener.

Nechajme systém spustený a pozrime sa, čo o ňom zistíme príkazmi rosnode list, rostopic list, rosnode info talker, rosnode info listener, rostopic info chatter.

Kapitola 5

Implementovaný model

V tejto kapitole priblížime proces dedenia modelu, jeho úprav a tvorbu súboru modelu na našu simuláciu.

5.1 Formáty modelov

Model mrvaca sme získali vo dvoch formátoch, OBJ a VRML2 [vrma]. Riška vo svojej práci opisuje ako bol tento model získaný pomocou stereoskopie [Ris11]. Pri obidvoch formátoch šlo spolu o päťdesiatdva súborov jednotlivých častí mrvaca, tak ako sú prezentované v časti venovanej anatómii.

Aby sme súbory mohli použiť na simuláciu v našom Gazebo+ROS systéme, je potrebné, aby súbor popisujúci model dostal odkazy na 3D súbory v niektorom z formátov OBJ, COLLADA alebo STL.

Po otvorení OBJ súborov sme zistili, že sú všetky vycentrované na začiatok globálnej súradnej sústavy, kvôli čomu z nich nevieme zistiť pozíciu údov vzhľadom k telu mrvaca. Skladanie modelu bez týchto informácií by bolo takmer nemožné. Pri modeli vo formáte VRML2 sme sa najskôr presvedčili pomocou nástroja FreeWRL [fre], že zdedený model je korektný. Pri importovaní hlavného súboru ant.wrl sa skutočne zobrazil celý model mrvaca. Začali sme sa tak pokúšať nájsť spôsob konverzie z formátu VRML2 do nami použiteľných formátov spomínaných vyššie.

5.2 Konvertovanie modelu

Model mrvaca vo formáte VRML2 sa skladal z 52 wrl súborov popisujúcich jednotlivé údy a hlavného súboru ant.wrl, ktorý na tieto súbory odkazoval a spájal ich do hierarchie (obr č. 5.1). Tieto súbory sme sa pokúšali konvertovať cez rôzne online nástroje, meshconv [mesc], simtrans [simb]. Tiež sme sa pokúsili o import do Blenderu [ble], Meshlabu [mesd], CAD nástrojov. Spomínané nástroje nefungovali, respektíve priniesli

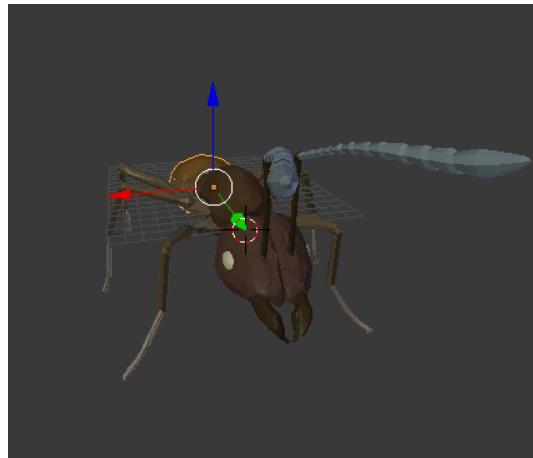
```

20 EXTERNPROTO ant-head[] "ant-head.wrl"
21 EXTERNPROTO ant-head-dots[] "ant-head-dots.wrl"
22 EXTERNPROTO ant-eye-left[] "ant-eye-left.wrl"
23 EXTERNPROTO ant-eye-right[] "ant-eye-right.wrl"
24 EXTERNPROTO ant-scape-left[] "ant-scape-left.wrl"
25 EXTERNPROTO ant-scape-right[] "ant-scape-right.wrl"
26 EXTERNPROTO ant-flagellum-left[] "ant-flagellum-left.wrl"
27 EXTERNPROTO ant-flagellum-right[] "ant-flagellum-right.wrl"
28 EXTERNPROTO ant-scape-bed-left[] "ant-scape-bed-left.wrl"
29 EXTERNPROTO ant-scape-bed-right[] "ant-scape-bed-right.wrl"
30 EXTERNPROTO ant-mandible-left[] "ant-mandible-left.wrl"
31 EXTERNPROTO ant-mandible-right[] "ant-mandible-right.wrl"
32 EXTERNPROTO ant-thorax[] "ant-thorax.wrl"
33 EXTERNPROTO ant-gaster[] "ant-gaster.wrl"
34 EXTERNPROTO ant-petiole[] "ant-petiole.wrl"
35 EXTERNPROTO ant-node[] "ant-node.wrl"
36 EXTERNPROTO ant-cox-front-left[] "ant-cox-front-left.wrl"
37 EXTERNPROTO ant-cox-front-right[] "ant-cox-front-right.wrl"
38 EXTERNPROTO ant-femur-front-left[] "ant-femur-front-left.wrl"
39 EXTERNPROTO ant-femur-front-right[] "ant-femur-front-right.wrl"
40 EXTERNPROTO ant-femur-trochanter-front-left[] "ant-femur-trochanter-front-left.wrl"
41 EXTERNPROTO ant-femur-trochanter-front-right[] "ant-femur-trochanter-front-right.wrl"
42 EXTERNPROTO ant-tibia-front-left[] "ant-tibia-front-left.wrl"
43 EXTERNPROTO ant-tibia-front-right[] "ant-tibia-front-right.wrl"
44 EXTERNPROTO ant-tibia-strip-front-left[] "ant-tibia-strip-front-left.wrl"
45 EXTERNPROTO ant-tibia-strip-front-right[] "ant-tibia-strip-front-right.wrl"
46 EXTERNPROTO ant-tarsus-front-left[] "ant-tarsus-front-left.wrl"
47 EXTERNPROTO ant-tarsus-front-right[] "ant-tarsus-front-right.wrl"
48 EXTERNPROTO ant-cox-middle-left[] "ant-cox-middle-left.wrl"
49 EXTERNPROTO ant-cox-middle-right[] "ant-cox-middle-right.wrl"
50 EXTERNPROTO ant-femur-middle-left[] "ant-femur-middle-left.wrl"
51 EXTERNPROTO ant-femur-middle-right[] "ant-femur-middle-right.wrl"
52 EXTERNPROTO ant-femur-trochanter-middle-left[] "ant-femur-trochanter-middle-left.wrl"
53 EXTERNPROTO ant-femur-trochanter-middle-right[] "ant-femur-trochanter-middle-right.wrl"
54 EXTERNPROTO ant-tibia-middle-left[] "ant-tibia-middle-left.wrl"
55 EXTERNPROTO ant-tibia-middle-right[] "ant-tibia-middle-right.wrl"
56 EXTERNPROTO ant-tibia-strip-middle-left[] "ant-tibia-strip-middle-left.wrl"
57 EXTERNPROTO ant-tibia-strip-middle-right[] "ant-tibia-strip-middle-right.wrl"
58 EXTERNPROTO ant-antenna-left Transform {
59   translation 0 0 0.145
60   rotation 0 1 0 0.45
61   scale 0.2 0.2 0.2
62   children [
63     DEF body Transform {
64       rotation 1 0 0 0.15
65     }
66     children [
67       DEF head Transform {
68         children [
69           ant-head {}
70           ant-head-dots {}
71           ant-eye-left {}
72           ant-eye-right {}
73           ant-scape-left {}
74           ant-scape-bed-left {}
75           ant-scape-bed-right {}
76           DEF antenna-left Transform {
77             children [
78               ant-scape-left {}
79               DEF flagellum-left Transform {
80                 children ant-flagellum-left {}
81               }
82             ]
83           }
84           DEF antenna-right Transform {
85             children [
86               ant-scape-right {}
87               DEF flagellum-right Transform {
88                 children ant-flagellum-right {}
89               }
90             ]
91           }
92         ]
93       }
94     ]
95   }
96 }

```

Obr. 5.1: Odvolávanie sa na wrl súbory údov (vľavo) a hierarchia údov (vpravo) v hlavnom ant.wrl súbore. [Ris11]

len čiastočné výsledky. Neskôr sme našli program VrmlMerge [vrmb], za pomocí ktorého sa dali jednotlivé údy skonvertovať do formátu X3D [x3d]. Tieto súbory už bolo možné importovať do Blenderu a následne ich exportovať vo formáte COLLADA alebo STL. Dôležité pri tom bolo, že zostali zachované údaje o ich globálnej pozícii. Údaje o hierarchii údov z hlavného wrl súboru boli ale stratené. Keď sme všetky výsledné súbory importovali do Blenderu, vznikol nám korektný model mrvaca (obr č. 5.2).



Obr. 5.2: Poskladaný ant.blend z COLLADA súborov

5.3 Tvorba a popis súborov modelu

Model sme najskôr prvoplánovo skladali vo formáte SDF, ktorý je súčasťou Gazebo. V snahe pridať k riešeniu riadiacu architektúru ROS bolo ale potrebné vytvoriť model vo

formáte URDF[urd], keďže on je v tom prípade oficiálne podporovaný a rozšírený. Je možné použiť formát SDF aj v systéme Gazebo+ROS, no táto voľba poskytuje značne menšie možnosti a ukázala sa ako nepostačujúca. Obidva formáty sú XML[xml] typu a robotov popisujú podobným spôsobom. Popisovať preto budeme len prácu s URDF modelom.

Vytvorime si v súbore riešení pracovného prostredia priečinok ant_sim, kde budeme pridávať balíky nášho riešenia:

```
cd catkin_ws/src
mkdir ant_sim
```

Zatiaľ tu vytvoríme jediný balík, ktorý bude mať za úlohu iba popisovať model mrvaca:

```
cd ant_sim
catkin_create_pkg ant_description
```

Nástroj catkin v ňom sám vytvorí súbory CMakeLists.txt a package.xml. Sú to doplnkové súbory popisujúce samotný balík a jeho kompliaciu. Upravme ich na naše riešenie:

```
<package>
  <name>ant_description</name>
  <version>0.0.1</version>
  <description>The ant_description package</description>
  <license>MIT</license>
  <author email="ado.pavco@gmail.com">Adrian Pavco</author>
  <maintainer email="ado.pavco@gmail.com">Adrian Pavco</maintainer>

  <buildtool_depend>catkin</buildtool_depend>
  <run_depend>joint_state_publisher</run_depend>
  <run_depend>robot_state_publisher</run_depend>
  <run_depend>rviz</run_depend>
</package>
```

Listing 5.1: Obsah súboru package.xml

```
cmake_minimum_required(VERSION 2.8.3)
project(ant_description)
find_package(catkin REQUIRED)
catkin_package()
```

Listing 5.2: Obsah súboru CMakeLists.txt.

Následne vytvoríme v priečinku tohto balíku tri podpriečinky:

```
cd ant_description
mkdir launch meshes urdf
```

Do priečinku meshes skopírujeme všetky COLLADA súbory údov mrvaca. Do priečinku urdf neskôr pridáme súbor popisujúci mrvaca a do priečinku launch spúšťací súbor vizualizácie modelu.

Pri tvorbe súboru popisujúceho model použijeme formát xacro [xac]. Je to XML formát veľmi podobný URDF doplnený o makrá zjednodušujúce a sprehľadňujúce častokrát rozsiahle a opakujúce sa časti kódu modelov. Využijeme ich napríklad pri dátach kíbov nožičiek. ROS systém vie ako vstup opisujúci robota prijať aj tento formát, no existuje aj jednoduchý príkaz na vytvorenie URDF z xacro:

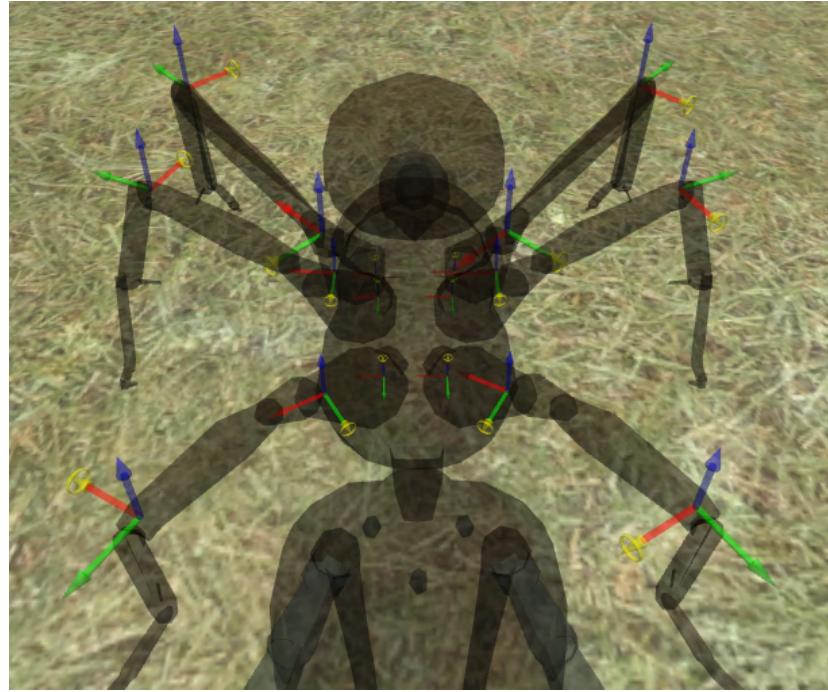
```
rosrun xacro xacro example.xacro > example.urdf
```

Vytvoríme ant.xacro súbor v podpriečinku urdf. Tento súbor tak ako pri SDF formáte obsahuje články (link) jednotlivých údov a spoje (joint) medzi nimi. Každému článku pridávame vizuálny a kolízny element referenciou na jeho 3D súbor.

Na spojenie článkov použijeme fixný kĺb, okrem troch pohyblivých kíbov každej nožičky (obr. č. 5.3, tab. č. 5.1). V prvom stĺpci tabuľky je názov kĺbu. Číslo znamená jeho poradie smerom od trupu k zemi, písmená popisujú jednu zo šiestich nožičiek. Stĺpce rodič a dieta popisujú, ktoré články kĺb spája a v akej hierarchii. Tabuľka približuje aj kinematické informácie o rozsahoch ohybov a osiach otáčainia. Príkazom check_urdf verifikujeme zvniknutý model (príloha č. 2.3) a jeho hierarchiu vizualizujeme príkazom urdf_to_graphviz (príloha č. 2.4).

kĺb	rodič	dieta	spodný limit	horný limit	os rot.
j_1_f_l	thorax	cox_f_l	-0.4	0.6	z
j_2_f_l	cox_f_l	trochanter_f_l	-0.3	0.5	y
j_3_f_l	femur_f_l	tibia_f_l	0	0.72	x
j_1_f_r	thorax	cox_f_r	-0.6	0.4	z
j_2_f_r	cox_f_r	trochanter_f_r	-0.5	0.3	y
j_3_f_r	femur_f_r	tibia_f_r	0	0.72	x
j_1_m_l	thorax	cox_m_l	-0.65	0.15	z
j_2_m_l	cox_m_l	trochanter_m_l	-0.13	0.5	y
j_3_m_l	femur_m_l	tibia_m_l	0	0.4	x
j_1_m_r	thorax	cox_m_r	-0.15	0.65	z
j_2_m_r	cox_m_r	trochanter_m_r	-0.5	0.13	y
j_3_m_r	femur_m_r	tibia_m_r	0	0.4	x
j_1_r_l	thorax	cox_r_l	-0.4	0.25	z
j_2_r_l	cox_r_l	trochanter_r_l	-0.15	0.5	y
j_3_r_l	femur_r_l	tibia_r_l	0	0.4	x
j_1_r_r	thorax	cox_r_r	-0.25	0.4	z
j_2_r_r	cox_r_r	trochanter_r_r	-0.5	0.15	y
j_3_r_r	femur_r_r	tibia_r_r	0	0.4	x

Tabuľka 5.1: Informácie o pohyblivých kíbov modelu.



Obr. 5.3: Vizualizácia osemnástich pohyblivých kŕbov modelu mravca v Gazebo.

Články máme pospájané, ostáva priradiť im inerciálne dátu. Sú to tieto: hmotnosť, pozícia tažiska a matica zotrvačnosti. Pozíciu tažiska a maticu zotrvačnosti získame cez online nástroj Mesh Cleaner [mesb] (obr č. 5.4), ktorému odovzdáme súbor článku a nami zvolenú hmotnosť. Takto postupujeme pri všetkých súboroch modelu. Pri výbere hmotností článkov mravca sme si za zdroj vybrali [GLWL18], kedže popisuje aj hmotnosť jednotlivých párov nôh. V tabuľke 5.2 sú popísané hmotnosti hlavy a jej článkov, v 5.3 hmotnosti trupu a bruška a v tabuľke 5.4 hmotnosti jednotlivých nožičiek. Celková zvolená hmotnosť mravca je 22.22 miligramov. Tu je dôležité dodať, že v simulácii pracujeme s modelom, ktorý je zväčšený. Bolo tomu tak pri zdedených 3D súboroch a hoci by ich bolo jednoduché škálovaním zmenšiť, usúdili sme, že taký model nie je nevyhnutný a bol by na vizuálne potreby našej simulácie príliš malý. Veľkosť a hmotnosť nášho modelu sú tak tisíc násobkom reálneho mravca.

článok	head	eye l, r	mandible l, r	scape l, r	flagellum l, r	head spolu
hmotnosť v mg	4.9	0.01	0.05	0.05	0.05	5.22

Tabuľka 5.2: Zvolené hmotnosti článkov hlavy modelu na simuláciu.

článok	thorax	petiole	node	gaster	gaster spolu
hmotnosť v mg	3.9	0.4	0.4	10.5	11.3

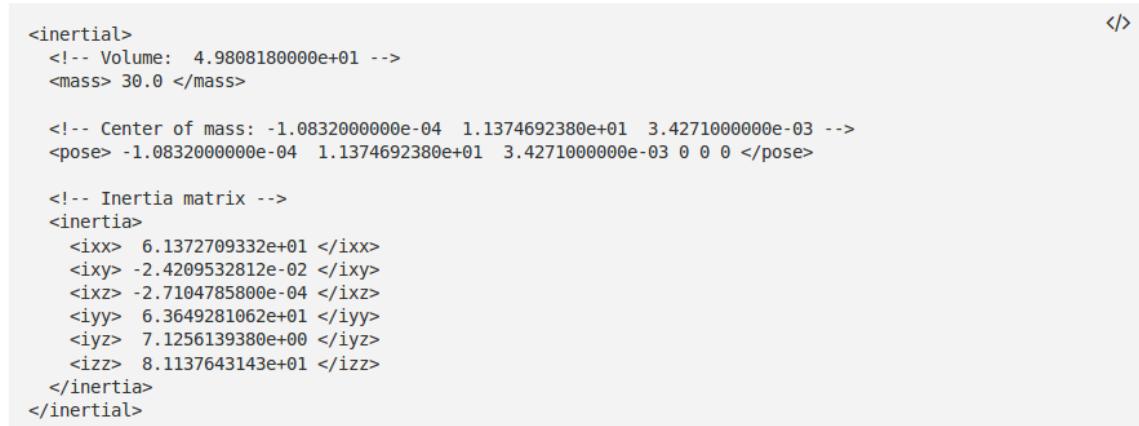
Tabuľka 5.3: Zvolené hmotnosti článkov trupu a bruška modelu na simuláciu.

článok	cox	trochanter	femur	tibia	tarsus	predná nož. spolu
hmotnosť v mg	0.04	0.02	0.1	0.1	0.04	0.3
článok	cox	trochanter	femur	tibia	tarsus	stredná nož. spolu
hmotnosť v mg	0.04	0.02	0.075	0.075	0.04	0.25
článok	cox	trochanter	femur	tibia	tarsus	zadná nož. spolu
hmotnosť v mg	0.04	0.02	0.125	0.125	0.04	0.35

Tabuľka 5.4: Zvolené hmotnosti nožičiek modelu na simuláciu.

Mesh Cleaner

In case everything went well, you should see the volume of the mesh, its center of mass and inertia matrix.



The screenshot shows the XML code generated by the Mesh Cleaner tool. It includes information about the volume (4.9808180000e+01), mass (30.0), center of mass (-1.0832000000e-04 1.1374692380e+01 3.4271000000e-03), pose (-1.0832000000e-04 1.1374692380e+01 3.4271000000e-03 0 0 0), and the inertia matrix components (ixx, ixy, ixz, iyy, iyz, izz) in a 6x6 matrix format.

```

<inertial>
    <!-- Volume: 4.9808180000e+01 -->
    <mass> 30.0 </mass>

    <!-- Center of mass: -1.0832000000e-04 1.1374692380e+01 3.4271000000e-03 -->
    <pose> -1.0832000000e-04 1.1374692380e+01 3.4271000000e-03 0 0 0 </pose>

    <!-- Inertia matrix -->
    <inertia>
        <ixx> 6.1372709332e+01 </ixx>
        <ixy> -2.4209532812e-02 </ixy>
        <ixz> -2.7104785800e-04 </ixz>
        <iyy> 6.3649281062e+01 </iyy>
        <iyz> 7.1256139380e+00 </iyz>
        <izz> 8.1137643143e+01 </izz>
    </inertia>
</inertial>

```

Mass: Update

Obr. 5.4: Nástroj Mesh Cleaner počíta inerciálne dátá zvoleného 3D súboru [mesb].

Mesh Cleaner akceptuje len vodotesné 3D súbory. Náš model obsahoval niekoľko súborov, ktoré také neboli. Museli sme ich preto opraviť za pomoci Blenderu a Meshlabu [mesa]. Takým bol napríklad súbor tykadla (príloha č. 2.1). Na zaplátanie diery v 3D súbore sme použili voľne dostupné rozšírenie Blenderu nazvané 3D Print Toolbox [3dp], konkrétnie jeho funkciu make manifold. Niektoré zo súborov obsahovali vnútorné plochy (interior faces). Tie sme museli manuálne odstrániť v Meshlabe (príloha č. 2.2).

Aby sme mohli poskladaný model vizualizovať a verifikovať, vytvoríme v priečinku launch spúšťiaci súbor ant_rviz.launch:

```

<launch>
    <param name="robot_description"
    command="$(find xacro)/xacro --inorder '$(find ant_description)/urdf/ant.xacro'" />

```

```

<!-- send fake joint values -->
<node name="joint_state_publisher" pkg="joint_state_publisher" type="joint_state_publisher">
<param name="use_gui" value="TRUE"/>
</node>

<!-- combine joint values -->
<node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"/>

<!-- show in Rviz -->
<node name="rviz" pkg="rviz" type="rviz" args="-d $(find ant_description)/launch/ant.rviz"/>
</launch>

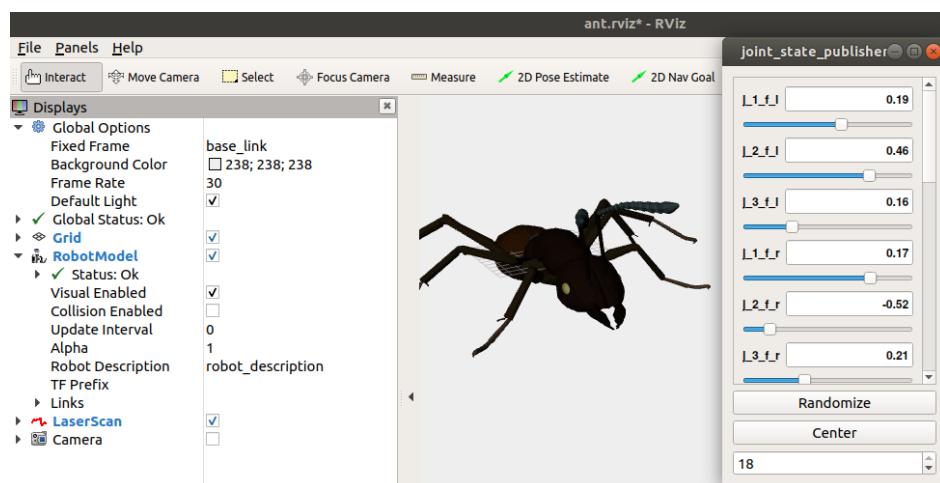
```

Listing 5.3: Spúšťací súbor ant_rviz.launch.

Model zobrazíme v nástroji Rviz príkazom:

```
roslaunch ant_description ant_rviz.launch
```

Pri prvom spustení však ešte neexistuje konfiguračný súbor ant.rviz, v ktorom sú uložené informácie popisujúce vizualizáciu. Musíme tak najskôr model pridať kliknutím na tlačidlo Add a vybrať RobotModel. Zobrazí sa model, ktorý ale nemá správnu pozíciu článkov. Je tomu tak, pretože Fixed frame je nastavený na map, zmeníme ho na base_link. Následne by sa mal zobraziť korektný model (obr. č. 5.5). Toto nastavenie uložíme ako ant.rviz a v budúcnosti sa nám hned po spustení model zobrazí správne. Všimnime si osobitný panel (joint_state_publisher), cez ktorý môžeme posielat systému fungované dátá o stave klíbov a testovať tak trajektórie a hranice pohybov.



Obr. 5.5: Model zobrazený v nástroji Rviz.

Kapitola 6

Simulácia a výsledky

Balík, ktorý sme v predošej kapitole vytvorili, má iba jedinú funkciu, a to popis stavby modelu mrvaca. Model sa nenachádza vo fyzikálnom prostredí, ani s ním nedokážeme programovo pohybovať. V tejto kapitole vytvoríme a popíšeme systém, v ktorom vložíme mrvaca do nami vytvoreného simulovaného prostredia, doplníme ho o riadenie, schopnosť chôdze a ovládanie používateľom.

6.1 Tvorba a popis súborov simulácie

Na potreby spojenia simulátora Gazebo a riadiacej architektúry ROS a následnej simulácie v tomto systéme budeme potrebovať dva ďalšie balíky. Nazvýme ich ant_control a ant_gazebo, vytvoríme ich tým istým spôsobom ako už existujúci balík popisujúci model mrvaca.

6.1.1 Balík ant_control

Jedinou úlohou balíka ant_control je vytvorenie a nastavenie kontrolerov pozície pohyblivých klíbov mrvaca. Ako prvé však doplníme popisujúci súbor ant.xacro o prvky umožňujúce riadenie. Každému pohyblivému klíbu priradíme transmission element spájajúci klíb s aktuátorom. Aby sme sa k tomuto súboru nemuseli už neskôr vracať, pridáme hneď aj gazebo elementy trenia pre časti, ktorými sa model dotýka zeme. Do súboru nakoniec pridáme gazebo_ros_control plugin, ktorý prečíta vytvorené transmission elementy a načíta príslušné hardvérové rozhranie a kontroler manager.

```
<robot>
  ...
  <transmission name="transm_f_l_1">
    <type>transmission_interface/SimpleTransmission</type>
    <joint name="j_1_f_1">
      <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
    </joint>
```

```

<actuator name="m_f_l_1">
  <hardwareInterface>hardware_interface/EffortJointInterface</hardwareInterface>
  <mechanicalReduction>1</mechanicalReduction>
</actuator>
</transmission>
...
<gazebo reference="tarsus_f_l">
  <mu1>100</mu1>
  <mu2>50</mu2>
</gazebo>
...
<gazebo>
  <plugin filename="libgazebo_ros_control.so" name="gazebo_ros_control">
    <robotNamespace>/ant</robotNamespace>
  </plugin>
</gazebo>
</robot>

```

Listing 6.1: Doplnenie súboru ant.xacro o prvky riadenia a simulácie.

V balíku ant_control upravíme automaticky vytvorené konfiguračné súbory package.xml a CMakeLists.txt. Druhý spomínaný je až na názov referencie identický ako v predošлом prípade. Najdôležitejšou časťou súboru package.xml sú informácie o externých knižniciach používaných počas behu programov:

```

<package>
  ...
  <buildtool_depend>catkin</buildtool_depend>

  <run_depend>controller_manager</run_depend>
  <run_depend>joint_state_controller</run_depend>
  <run_depend>robot_state_publisher</run_depend>
  <run_depend>rqt_gui</run_depend>
  <run_depend>effort_controllers</run_depend>
</package>

```

Listing 6.2: Hlavná časť súboru package.xml

Nachádzajú sa tu podpriečinky config a launch. V priečinku config je iba jeden súbor vo formáte YAML [yaml] ant_control.yaml. Je to konfiguračný súbor kontrolerov klíbov modelu mravca:

```

ant:
# Publish all joint states -----
joint_state_controller:
  type: joint_state_controller/JointStateController
  publish_rate: 50

# Position Controllers -----

```

```
j_1_f_l_position_controller:
  type: effort_controllers/JointPositionController
  joint: j_1_f_l
  pid: {p: 6, i: 0.01, d: 0.2}

j_2_f_l_position_controller:
  type: effort_controllers/JointPositionController
  joint: j_2_f_l
  pid: {p: 6, i: 0.01, d: 0.2}

j_3_f_l_position_controller:
  type: effort_controllers/JointPositionController
  joint: j_3_f_l
  pid: {p: 6, i: 0.01, d: 0.2}
```

Listing 6.3: Časť súboru ant_control.yaml

Na začiatku súboru je definovaný kontroler, ktorý s danou frekvenciou posiela do ROS systému informácie o momentálnom stave kľbov. Všetkým kľbom sú priradené pozičné kontroly, s konkrétnymi PID argumentami [pid]. Tie zabezpečujú aplikovanie sily na aktuátor kľbu a tým jeho pohyb z pôvodnej do požadovanej pozície podľa vstupných dát.

V priečinku launch sa nachádza spúšťací súbor ant_control.launch. Ten odkazuje na konfiguračný súbor kontrolerov a načíta ich pri spustení do systému v získanej konfigurácii. Na tento súbor budeme neskôr odkazovať a spúštať ho v hlavnom spúšťacom súbore simulácie.

```
<launch>
  <!-- load joint controller configurations from YAML file to parameter server -->
  <rosparam file="$(find ant_control)/config/ant_control.yaml" command="load"/>

  <!-- load the controllers -->
  <node name="controller_spawner" pkg="controller_manager" type="spawner" respawn="false"
        ns="/ant" args="joint_state_controller
        j_1_f_l_position_controller
        ...
        j_3_r_r_position_controller
      "/>

  <!-- convert joint states to TF transforms for rviz, etc -->
  <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_publisher" respawn="false" output="screen">
    <remap from="/joint_states" to="/ant/joint_states" />
  </node>
```

</`launch`>

Listing 6.4: Spúšťací súbor ant_control.launch

6.1.2 Balík simulácie ant_gazebo

Následne je potrebné pridať do riešenia simulované prostredie, vloženie modelu do tohto prostredia, vytvoriť rozhranie na komunikáciu a programové riadenie modelu. To všetko bude zabezpečovať balík ant_gazebo. Je to posledný balík, ktorý v našom riešení vytvárame, je tiež obsahovo a funkčne najširší. Na ozrejmenie ho môžeme prirovnáta k centrálnej nervovej sústave simulácie. V zmysle tohto prirovnania by potom balík ant_description boli kosti a klíby a ant_control šľachy. Je zjavné, že každá časť tohto systému v ňom má nevyhnutné miesto.

Vytvoríme ho podobne ako predošlé balíky. Opíšme pred bližším popisom jeho jednotlivých častí finálnu hierarchiu (obr č. 6.1). .

Name	Size
launch	1 item
ant_sim.launch	1,5 kB
scripts	2 items
keys.py	615 bytes
walk.py	10,4 kB
src	1 item
ant_gazebo	3 items
ant.py	2,0 kB
ant.pyc	2,6 kB
__init__.py	0 bytes
worlds	1 item
ant.world	2,3 kB
CMakeLists.txt	343 bytes
package.xml	773 bytes
setup.py	222 bytes

Obr. 6.1: Kompletná hierarchia balíka ant_gazebo.

Priečinok launch obsahuje len jeden súbor ant_sim.launch. Ako argument ho posúvame príkazu roslaunch do terminálu zakaždým, keď spúšťame simuláciu. V priečinku scripts sa nachádzajú dva programy, takzvané nodes nášho systému, písané v jazyku Python. Prvý program keys.py vykonáva časť ovládania mravca pomocou klávesnice. Druhý, walk.py zabezpečuje riadenie modelu. V priečinku src nájdeme súbory modulu komunikačného rozhrania modelu mravca. Priečinok worlds obsahuje súbor ant.world popisujúci prostredie simulácie. Balík tradične obsahuje konfiguračné súbory, súbor setup.py je komplementárny súbor pre catkin. Funkciu a obsah týchto súborov si postupne popíšeme.

Konfiguračné súbory

Priblížme si obsah konfiguračných súborov:

```
<package>
  ...
  <buildtool_depend>catkin</buildtool_depend>

  <build_depend>gazebo_ros</build_depend>
  <build_depend>rospy</build_depend>
  <build_depend>gazebo_ros_control</build_depend>
  <build_depend>sensor_msgs</build_depend>

  <run_depend>gazebo_ros</run_depend>
  <run_depend>rospy</run_depend>
  <run_depend>gazebo_ros_control</run_depend>
  <run_depend>sensor_msgs</run_depend>
  <run_depend>xacro</run_depend>
  <run_depend>ant_control</run_depend>
  <run_depend>ant_description</run_depend>
</package>
```

Listing 6.5: Hlavná časť súboru package.xml

```
cmake_minimum_required(VERSION 2.8.3)
project(ant_gazebo)

find_package(catkin REQUIRED COMPONENTS
  gazebo_ros
  rospy
  std_msgs
  sensor_msgs
)
find_package(Boost REQUIRED COMPONENTS system)
find_package(gazebo REQUIRED)

catkin_python_setup()

catkin_package(
  CATKIN_DEPENDS
    sensor_msgs
    gazebo_ros
  DEPENDS
    gazebo
)
```

Listing 6.6: Obsah súboru CMakeLists.txt.

Vidíme, že tieto súbory odkazujú na potreby kompilátora catkin na niekoľko externých balíkov a knižníc. Na podporu jazyka Python, na získanie modelu z formátu xacro, na spojenie simulátora a riadiacej architektúry, knižnice podporujúce rôzne typy správ. Tiež sú tu referencie na naše predošlé balíky, aby systém vedel, že s nimi bude počas behu pracovať.

Launch súbor

V spúšťacom súbore zabezpečujeme niekoľko dôležitých procesov. Simulátoru vravíme, ktorý world súbor má načítať a volíme testovacie parametre simulácie, napríklad či je pri spustení pozastavená, alebo či sa majú vypisovať údaje o procesoch v pozadí. Systému ďalej posúvame náš model, ktorý sa načíta na parameter server a nato sa aj vloží do simulácie. Potom zavoláme vedľajší spúšťací súbor ant_control.launch a nakoniec spustíme programy walk.py a keys.py.

```
<launch>
    <!-- launch our world file, change arguments for testing -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="world_name" value="$(find ant_gazebo)/worlds/ant.world"/>
        <arg name="debug" value="false" />
        <arg name="gui" value="true" />
        <arg name="paused" value="false"/>
        <arg name="use_sim_time" value="true"/>
        <arg name="headless" value="false"/>
    </include>

    <!-- load the URDF into the ROS Parameter Server -->
    <param name="robot_description"
        command="$(find xacro)/xacro --inorder '$(find ant_description)/urdf/ant.xacro'"
        />

    <!-- script to the send a service call to gazebo_ros to spawn an URDF model -->
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model" respawn="false"
        args="-urdf -model ant -z 13 -param robot_description"/>

    <!-- ros_control ant launch file -->
    <include file="$(find ant_control)/launch/ant_control.launch"/>

    <!-- start our nodes -->
    <node name="ant_walk" pkg="ant_gazebo" type="walk.py" output="screen"/>
    <node name="ant_keys" pkg="ant_gazebo" type="keys.py" output="screen"/>
</launch>
```

Listing 6.7: Hlavný spúšťací súbor ant_sim.launch.

World súbor a prostredie simulácie

Prostredie simulácie v Gazebo je definované world súbormi. Vytvoríme si taký súbor podľa našich požiadaviek. Element gravity definuje existenciu gravitácie. Pri finálnej simulácii je samozrejomou súčasťou, no na účely testovania ju niekedy vypíname. Naše prvé riešenia používali ako podklad celkom rovnú štvorcovú podložku. Neskôr sme vytvorili SDF model prostredia z výškovej mapy reprezentovanej čiernobielym obrázkom. Tento model poskytuje nielen rovinu, ale aj vlnitý terén, jamy a kopce. Simulátor mu zvláda priradiť aj textúry podľa výšky jednotlivých bodov. Modely pridávané vo world súbore cez include tag musia byť uložené v Gazebo model path `/.gazebo/models` a musia mať stavbu opísanú v časti o Gazebo súboroch. Cez element population prídáme do prostredia niekoľko náhodne umiestnených rozličných modelov kamienkov. Ďalej pridáme zdroj svetla imitujúci slnko, pohyblivé oblaky, vypneme zobrazenie osí a mriežky, nastavíme pozíciu a natočenie kamery pri spustení. Kedže tu nešpecifikujeme inak, simulátor bude používať predvolený fyzikálny engine ODE.

```

<sdf version="1.4">
<world name="default">
    <!-- uncomment for testing -->
    <!-- <gravity>0</gravity> -->

    <include>
        <uri>model://heightmap</uri>
    </include>

    <population name="stone1_population">
        <model name="stone1">
            <include>
                <static>true</static>
                <uri>model://stone1</uri>
            </include>
        </model>
        <pose>140 140 5 0 0 0</pose>
        <box>
            <size>200 200 4</size>
        </box>
        <model_count>10</model_count>
        <distribution>
            <type>random</type>
        </distribution>
    </population>

    <population name="stone2_population">
        ...
        <uri>model://stone2</uri>
    </population>

```

```

...
</population>

<population name="stone3_population">
...
<uri>model://stone3</uri>
...
</population>

<!-- sun placed above the enviroment -->
<light type="directional" name="sun">
  <cast_shadows>true</cast_shadows>
  <pose>0 0 50 0 0 0</pose>
  <diffuse>0.8 0.8 0.8 1</diffuse>
  <specular>0.2 0.2 0.2 1</specular>
  <attenuation>
    <range>1000</range>
    <constant>0.9</constant>
    <linear>0.01</linear>
    <quadratic>0.001</quadratic>
  </attenuation>
  <direction>0 0 -1</direction>
</light>

<!-- scene settings -->
<scene>
  <shadows>0</shadows>
  <grid>0</grid>
  <origin_visual>0</origin_visual>
  <sky>
    <clouds>
      <speed>5</speed>
    </clouds>
  </sky>
</scene>

<!-- camera looking at ant from its front -->
<gui fullscreen='0'>
  <camera name='user_camera'>
    <pose>-25.603 45.9495 22.1185 -0 0.355643 -1.07499</pose>
    <view_controller>orbit</view_controller>
  </camera>
</gui>
</world>
</sdf>
```

Listing 6.8: Obsah súboru ant.world.

Modul rozhrania

Vytvoríme rozhranie, ktoré budeme využívať na komunikáciu s modelom v simulácii. Takýto program je zaužívané nazývať modulom. Kvôli jeho importu a použítiu v hlavnom riadiacom programe je potrebné ho nielen skompilovať, ale aj nainštalovať. To spravíme cez vytvorenie a úpravu súboru setup.py. Všetko potrebné vykoná caktin príkazom catkin_python_setup() v súbore CMakeLists.txt. Popíšme obsah súbor modulu ant.py:

```

import rospy
import time
from sensor_msgs.msg import JointState
from std_msgs.msg import Float64

class Ant:
    def __init__(self, ns='/ant/'):
        self.ns = ns
        self.joints = None
        self.angles = None
        self._sub_joints = rospy.Subscriber(
            ns + 'joint_states', JointState, self.get_joint_states,
            queue_size=1)

        while not rospy.is_shutdown():
            if self.joints is not None:
                break
            rospy.sleep(0.1)

        self._pub_joints = {}
        self.centered_angles = {}
        for j in self.joints:
            p = rospy.Publisher(
                ns + j + '_position_controller/command', Float64, queue_size=1)
            self._pub_joints[j] = p
            self.centered_angles[j] = 0.0
        rospy.sleep(1)

    def get_joint_states(self, msg):
        if self.joints is None:
            self.joints = msg.name
            self.angles = msg.position

```

Listing 6.9: Trieda Ant a jej konštruktor v súbore modulu.

Na začiatku súboru importujeme všetky potrebné knižnice na fungovanie programu. Rospy na podporu jazyku Python v ROS, time na programovanie s informáciami o čase, JointState ako dátová štruktúra stavu klíbov a Float64 ako dátová štruktúra po-

zície klíbov. Súbor definuje jedinú triedu Ant. Konštruktor nastaví premennú ns na retazec /ant/, aby sme vedeli nájsť adresy tém nášho systému. Vytvoríme prijímateľa stavov klíbov modelu. Vieme, že do témy joint_states balík ant_control pri spustenom systéme neustále posiela informácie o aktuálnych stavoch klíbov v simulácii. Nastavíme tohto prijímateľa tak, že pri prijatí novej správy volá procedúru get_joint_states, ktorá napĺňa zoznam uhlov. Pri jej prvom vykonaní sa naplní aj zoznam klíbov, ktorý je už počas behu nemenný. Naopak zoznam uhlov sa počas spusteného systému neustále aktualizuje novými údajmi zo simulácie. V konštruktore čakáme v nekonečnom cykle, až kým sa zoznam uhlov nenaplní, to znamená kým kontroler manager spustený súborom ant_control.launch nenačíta do systému potrebné kontrolery. Na konci konštruktora vytvoríme slovník, kde klíčom sú klíby a hodnotou sú odosielatelia, ktorí do témy ns + názov klíbu + '_position_controller/command posielajú dátu o požadovanej cielovej pozícii klíbov, ktorú príslušné pozičné kontrolery z balíka ant_control vyhodnocujú. Popis ostatných procedúr modulu:

```

def get_angles(self):
    if self.joints is None:
        return None
    if self.angles is None:
        return None
    return dict(zip(self.joints, self.angles))

def set_angles(self, angles):
    for j, v in angles.items():
        self._pub_joints[j].publish(v)

def center_angles(self):
    start_angles = self.get_angles()
    start_time = time.time()
    stop_time = start_time + 1
    r = rospy.Rate(100)
    while not rospy.is_shutdown():
        t = time.time()
        if t > stop_time:
            break
        ratio = (t - start_time) / 1
        interp = {}
        joints = self.centered_angles.keys()
        for j in joints:
            interp[j] = self.centered_angles[j] * ratio + start_angles[j]
            * (1 - ratio)
        self.set_angles(interp)
        r.sleep()

```

Listing 6.10: Procedúry triedy Ant v súbore modulu.

Funkcia get_angles vráti slovník kľbov a ich momentálnych uhlov, procedúra set_angles dostáva ako parameter slovník kľbov a uhlov. Pre každý kľb zo slovníka povie odosielateľovi s kľúčom ako je daný kľb, aby poslal do systému nový požadovaný cielový uhol. Procedúra center_angles interpoláciou vycentruje v priebehu jednej sekundy hodnoty uhlov všetkých kľbov z akýchkoľvek hodnôt na nulu. Toto je využívané na začiatku simulácie a vždy pred začatím mravcovej chôdze, kvôli zabezpečeniu vždy rovnakého počiatočného stavu uhlov. Odôvodnenie tohto riešenia sa nachádza pri popise súboru chôdze.

Nodes súbory

V spúšťacom súbore ant_sim.launch sme do systému pridali dva nami vytvorené programy, keys.py a walk.py. Prvý spracováva a zdieľa vstupy z klávesnice, druhý riadi a vykonáva cyklus chôdze. Objasníme obsah prvého programu:

```
import getch
import rospy
from std_msgs.msg import Int8

def keys():
    while not rospy.is_shutdown():
        k = ord(getch.getch())
        if ((k >= 65) & (k <= 68) | (k == 114)
            | (k == 122) | (k == 120) | (k == 99)):
            pub.publish(k)
        rate.sleep()

if __name__ == '__main__':
    rospy.init_node('keys_node', anonymous = True)
    pub = rospy.Publisher('/ant/keys', Int8, queue_size = 1)
    rate = rospy.Rate(100)
    try:
        keys()
    except rospy.ROSInterruptException:
        pass
```

Listing 6.11: Node program keys.py.

Program obsahuje jednu procedúru, ktorá sa vykonáva až do vypnutia simulácie. Vytvoríme odosielateľa do témy /ant/key typu Int8, filtrujeme stlačené klávesy reprezentované číselným formátom, ak získame jeden z očakávaných klávesov, pošleme ho do vytvorenej témy. Je možné, že čitateľ podobne ako my nemal nainštalovaný modul getch. Získame ho napríklad príkazom pip install getch.

Popíšme postupne najdôležitejšie časti hlavného programu simulácie walk.py. Definuje spolu štyri triedy: WalkControl, MotionFunc, Joint a JointAmplit. Najskôr pri-

blížme importovanie nášho modulu rozhrania:

```
from ant_gazebo.ant import Ant
...
if __name__ == '__main__':
    rospy.init_node('walk_node')
    robot = Ant()
    walk_control = WalkControl(robot)
    walk_control.start()
```

Listing 6.12: Importovanie modulu a spustenie simulácie.

Importujeme triedu Ant z nainštalovaného modulu a vytvoríme jeho inštanciu. Pri vytvorení inštancie triedy WalkControl ju jej posúvame ako argument. Simuláciu spusťme volaním procedúry start vytvorenej inštancie riadiacej triedy.

```
class WalkControl:
    def __init__(self, robot):
        self.robot = robot
        self.func = MotionFunc()
        self.thread = None
        self.motion = 'S'
        self.phase = True
        self.first_step = True
        self.i = 0
        self.n = speeds[1]
        self.rate = rospy.Rate(50)

        self._sub_keys = rospy.Subscriber(
            '/ant/key', Int8, self.got_key, queue_size = 1)

        rospy.wait_for_service('/gazebo/reset_world')
        self.reset_world = rospy.ServiceProxy('/gazebo/reset_world',
            Empty)
```

Listing 6.13: Konštruktor triedy WalkControl.

V konštruktore priradíme rozhranie získané ako vstupný argument, vytvoríme a priradíme funkciu kráčania MotionFunc a nastavíme stavové a pomocné premenné používané simuláciou. Pridáme prijímateľa informácie o číselnej reprezentácii stlačeňného klávesu s volaním procedúry got_key. Na konci konštruktora je definovaná Gazebo služba, ktorá vykonáva vrátenie modelov simulácie do ich počiatočných pozícii.

```
def start(self):
    self.thread = Thread(target = self.walk)
    self.thread.start()

def walk(self):
    rospy.loginfo('Starting simulation.')
```

```

    self.robot.center_angles()
    while (not rospy.is_shutdown()):
        if (self.motion != 'S'):
            x = float(self.i) / self.n
            if (self.motion == 'F'):
                angles = self.func.get_forward(self.phase, self.first_step,
                                                x)
            elif (self.motion == 'L'):
                angles = self.func.get_turn(True, self.phase, x)
            else:
                angles = self.func.get_turn(False, self.phase, x)
            self.robot.set_angles(angles)
            self.i += 1
            if self.i > self.n:
                self.i = 0
                self.first_step = False
                self.phase = not self.phase
            self.rate.sleep()
        self.thread = None

def reset_walk(self):
    self.robot.center_angles()
    self.phase = True
    self.first_step = True
    self.i = 0

def got_key(self, msg):
    if msg.data == 65:
        if self.motion == 'S':
            self.motion = 'F'
            rospy.loginfo('Walking forward.')
    elif msg.data == 66:
        ...
    elif msg.data == 114:
        rospy.loginfo('Reseting model position, standing.')
        self.motion = 'S'
        self.reset_world()
        self.reset_walk()
    ...

```

Listing 6.14: Procedúry triedy WalkControl.

Cyklus chôdze sa dej vo vlákne. Procedúrou start ho vytvoríme, priradíme mu vykonávanie procedúry walk a spustíme ho. Procedúra reset_walk centruje pozície kľbov a resetuje stavové a pomocné premenné. Procedúra got_key obsahuje podmienky na stlačené klávesy a podľa toho upravuje premenné a volá procedúry. Na začiatku procedúra walk vynuluje uhly kľbov a potom v cykle až do skončenia simulácie vykonáva

nasledovné. Zistí, aký pohyb má momentálne model vykonávať, môže to byť státie na mieste, chôdza vpred a otáčanie na mieste vľavo alebo vpravo. Ak je momentálny stav odlišný od státia, získa sa z pomocných premenných i (na začiatku nula) a n (horná hranica cyklu fázy podľa aktuálne zvolenej rýchlosťi simulácie) hodnota x , kde $x = i/n$ typu float v rozmedzí nula až jeden. Táto hodnota sa spolu s informáciou o aktuálnej fáze posunie volaniu funkcie inštancie triedy MotionFunc. Pri chôdzi dopredu je to `get_forward`, pri otáčaní `get_turn`. Volania vrátia na základe vstupných argumentov cieľové uhly, ktoré sa zdieľajú cez modul rozhrania s pozičnými kontrolermi, ktoré zabezpečia ich pohyb v simulácii. Pomocná premenná i sa každým cyklom inkrementuje, až kým sa nevyrovná hornej hranici n , v tom momente sa i vynuluje a zmení sa fáza chôdze. Na konci cyklu sa vlákno na veľmi krátku chvíľu pozastaví príkazom `sleep`. Pri chôdzi dopredu je prvý krok fázy len polovičný, pretože začína z nulových uhlov klbov a nie z hraničných bodov fáz, preto vtedy voláme funkciu `get_forward` s argumentom `first_step True`. Po vykonaní tohto prvého polkroku je argument `first_step False`, až kým znova nezačíname chôdzu dopredu z vycentrovaných uhlov. Treba dodať, že pod pojmom rýchlosť simulácie neimplementujeme predĺženie kroku mravca, ale len meníme parameter hornej hranice cyklu fázy.

Popíšme triedu MotionFunc.

```
class MotionFunc:
    def __init__(self):
        self.generate_forward()
        self.generate_first_step()
        self.generate_left_turn()
        self.generate_right_turn()

    def get_forward(self, phase, first_step, x):
        angles = []
        for j in self.forward_ph.keys():
            if first_step:
                angles[j] = self.first_stp[j].get(x)
            elif phase:
                angles[j] = self.forward_ph[j].get(x)
            else:
                angles[j] = self.forward_ap[j].get(x)
        return angles

    def get_turn(self, left, phase, x):
        angles = []
        for j in self.forward_ph.keys():
            if 'j_2' in j or 'j_3' in j:
                if phase:
                    angles[j] = self.forward_ph[j].get(x)
                else:
```

```

        angles[j] = self.forward_ap[j].get(x)
    else:
        if left:
            if phase:
                angles[j] = self.left_ph[j].get(x)
            else:
                angles[j] = self.left_ap[j].get(x)
        else:
            if phase:
                angles[j] = self.right_ph[j].get(x)
            else:
                angles[j] = self.right_ap[j].get(x)
    return angles

```

Listing 6.15: Časť triedy MotionFunc.

V konštruktore zavoláme vytvorenie slovníkov kľbov a funkcií všetkých na všetky potrebné pohyby a fázy, tieto procedúry sú popísané nižšie. Definujeme funkcie get_forward a get_turn, ktoré podľa vstupných argumentov vracajú cielové uhly pre všetky kľby.

Priblížme procedúry generovania slovníkov pre typy našej chôdze:

```

def generate_forward(self):
    self.forward_ph = {} # forward phase functions
    self.forward_ap = {} # forward anti phase functions
    self.forward_ph['j_1_f_1'] = Joint(-f1_upp, -f1_low, -1)
    self.forward_ap['j_1_f_1'] = Joint(f1_low, f1_upp, 1)
    ...
    self.forward_ph['j_2_f_1'] = JointAmplit(f2_low, f2_upp, 1, 1,
                                              f2_low_front)
    self.forward_ap['j_2_f_1'] = Joint(f2_low_front, f2_low, 1)
    ...
    self.forward_ph['j_3_f_1'] = Joint(f3_low, f3_upp, 1)
    self.forward_ap['j_3_f_1'] = Joint(-f3_upp, f3_low, -1)
    ...

```

Listing 6.16: Procedúra generovania chôdze dopredu.

Procedúra generate_forward vytvorí slovníky forward_ph a forward_ap, jeden pre fázu a druhý pre antifázu. Pre každý z kľbov modelu mravca priradíme inštanciu triedy Joint, respektíve JointAmplit s parametrami spodnej a hornej hranice uhlov pohybu kľbu, parameterom invert a v prípade JointAmplit aj phase a voliteľným parametrom change_low.

```

def generate_first_step(self):
    self.first_stp = {} # first step functions
    self.first_stp['j_1_f_1'] = Joint(0, -f1_low, -1)
    self.first_stp['j_1_f_r'] = Joint(0, f1_upp, -1)
    ...

```

```

def generate_left_turn(self):
    self.left_ph = {} # left turn phase functions
    self.left_ap = {} # left turn anti phase functions
    self.left_ph['j_1_f_1'] = Joint(0, 0, -1)
    self.left_ap['j_1_f_1'] = Joint(0, 0, 1)
    ...

def generate_right_turn(self):
    self.right_ph = {} # right turn phase functions
    self.right_ap = {} # right turn anti phase functions
    self.right_ph['j_1_f_1'] = Joint(-f1_upp, -f1_low, -1)
    self.right_ap['j_1_f_1'] = Joint(f1_low, f1_upp, 1)
    ...

```

Listing 6.17: Procedúry generovania prvého polkroku a otáčania.

Procedúry na prvý polkrok a otáčanie sa od chôdze mierne odlišujú, pri otáčaní vykonáva pozdĺžny pohyb len trojica nožičiek, pri polkroku sa nezačína od krajnej hraničnej pozície ale od nuly.

Priblížme triedu Joint. Jediná funkcia get premapuje a vráti vstupný údaj z rozmezia nula až jeden do rozmedzia definovaného spodnej a hornou hranicou pohybu klíbu. Argument invert používame v prípade obráteného klíbu, vtedy vstupným argumentom otočíme znamienko, vymeníme ich pozície a funkciu povieme, že znamienko výsledku chceme otočiť.

```

class Joint:
    def __init__(self, low, upp, invert):
        self.low = low
        self.upp = upp
        self.invert = invert

    def get(self, x):
        return self.invert * interp(x, [0, 1], [self.low, self.upp]))

```

Listing 6.18: Trieda Joint.

Trieda JointAmplit je používaná spravidla na klíb medzi coxou a trochanterom. Keďže je tento klíb je pri antifáze nehybný, posunieme túto informáciu parametrom phase. Naopak pri fáze musí vykonať pohyb smerom nahor, dosiahnuť hornú hranicu a naspäť klesať na spodnú hranicu. To dosiahneme rozdelením intervalu na dve polovice. Pri fáze predných a antifáze zadných nožičiek využívame voliteľný parameter change_low. Tieto končatiny sa vtedy v klíbe medzi femurom a tibiu charakteristicky otvárajú a tak predlžujú krok. Na zachovanie správneho vyváženia modelu sa preto ich spodná hranica uhlia posúva nižšie k zemi.

```

class JointAmplit:

```

```

def __init__(self, low, upp, invert, phase, change_low = 0):
    self.low = low
    self.upp = upp
    self.invert = invert
    self.phase = phase
    self.change_low = change_low

def get(self, x):
    if x < 0.5:
        return self.phase * self.invert * interp(x, [0, 0.5], [self.low,
            self.upp])
    else:
        if (self.change_low == 0):
            return -1 * self.phase * self.invert * interp(x, [0.5, 1], [
                -self.upp, -self.low])
        else:
            return -1 * self.phase * self.invert * interp(x, [0.5, 1], [
                -self.upp, -self.change_low])

```

Listing 6.19: Trieda JointAmplit.

6.2 Výsledky simulácie

Simuláciu spustíme hlavným spúšťacím súborom ant_sim.launch:

```
roslaunch ant_sim.launch
```

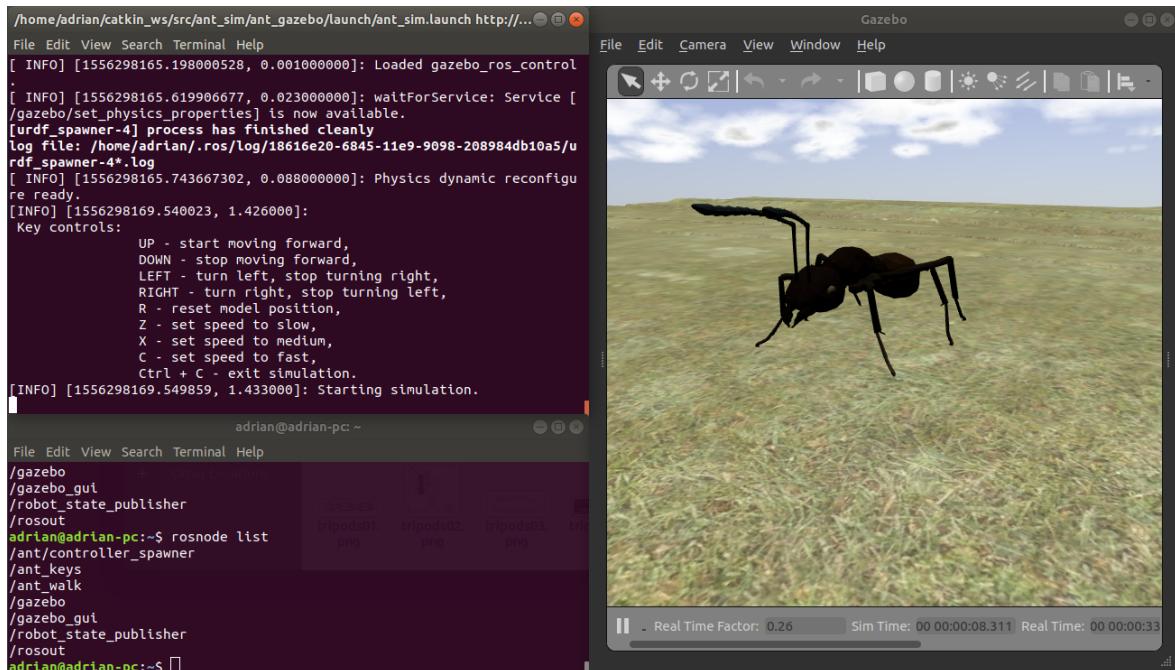
Otvorí sa používateľské rozhranie simulátora Gazebo s naším modelom vloženým do prostredia. Parameter paused v launch súbore sme nastavili na false, takže simulácia je hned od začiatku spustená. Máme možnosť ju kedykoľvek pozastaviť a spustiť cez tlačidlo na spodnej lište. Pre ovládanie simulácie klávesnicou (tab. č. 6.1) musí byť okno terminálu spúštajúceho program aktívne.

Odporúčame čitateľovi presunúť terminál, resp. terminály na jednu polovicu displeja a aplikáciu na druhú (obr. č. 6.2), prípadne na rozličné displeje. Následne nemusíme minimalizovať a maximalizovať používané okná pri striedení ovládania klávesnicou a práce s grafickým rozhraním. Gazebo umožnuje uložiť konfiguráciu pozície a veľkosti svojho okna cez File a Save Configuration, ktorá sa automaticky načíta pri dalších spusteniach. Ak nás nezaujíma obsah ľavého panelu, možeme ho skryť, vďaka čomu zostane viac priestoru oknu vykreslovania.

Všimnime si spodný panel simulátora s informáciami o čase a kvalite. Číslo Real Time Factor informuje, v akom pomere k reálnemu času je čas v simulácii. Závisí od výkonnosti nášho systému, veľkosti vykresľovacieho okna a od momentálnej zložitosti výpočtov fyzikálneho enginu. Pri chôdzi po rovine je simulácia rýchlejšia ako pri prechode prekážky alebo pohybe vo zvlnenom teréne.

kláves	funkcia
šípka hore	spustenie chôdze dopredu
šípka dole	zastavenie chôdze dopredu
šípka vľavo	otáčanie na mieste vľavo, ukončenie otáčania vpravo
šípka vpravo	otáčanie na mieste vpravo, ukončenie otáčania vľavo
R	resetovanie pozície modelu mrvaca
Z	nastavenie nízkej rýchlosťi simulácie
X	nastavenie strednej rýchlosťi simulácie
C	nastavenie vysokej rýchlosťi simulácie
Ctrl + C	ukončenie programu simulácie

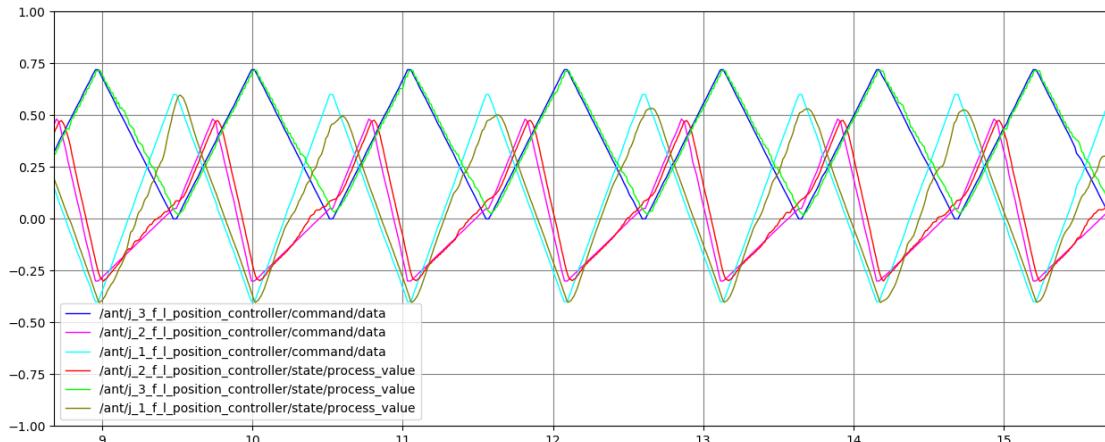
Tabuľka 6.1: Ovládanie systému klávesnicou.



Obr. 6.2: Príklad konfigurácie veľkostí a pozícii okien Gazebo a terminálov.

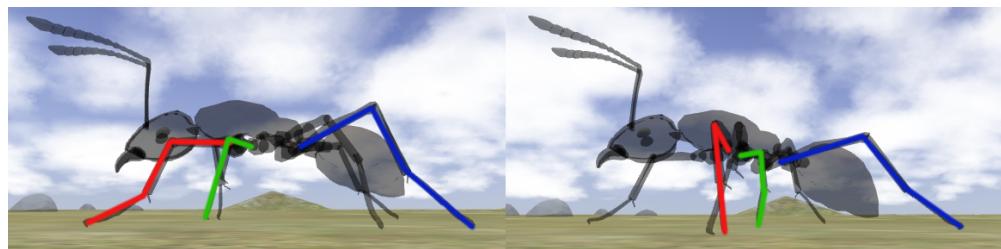
Správnosť nastavenia pozičných kontrolerov môžeme overovať rôznymi spôsobmi. Nechceme, aby boli nastavené ako príliš slabé ale ani ako príliš silné. To môžeme odpozorovať aj voľným okom, keď vidíme, že nám mravec pod svojou váhou padá a nedokáže sa zdvihnuť alebo naopak pohyb v klboch je príliš rýchly, neplynulý a pri náraze na svoju hranicu sa kľb odrazí naspäť. Program rqt dokáže okrem množstva iných činností v ROS systéme vizualizovať prácu našich kontrolerov (obr. č. 6.3). Vidíme informácie o kontroleroch kľbov prednej ľavej nožičky počas spustenej simulácie a chôdzi dopredu. Command/data sú cieľové uhly posielané kontroleru, state/process_value sú reálne údaje modelu v simulácii. V korektnom nastavení by táto dvojica mala mať podobnú až identickú trajektóriu. V grafe môžeme zreteľne vidieť striedanie fáz chôdze

končatiny.

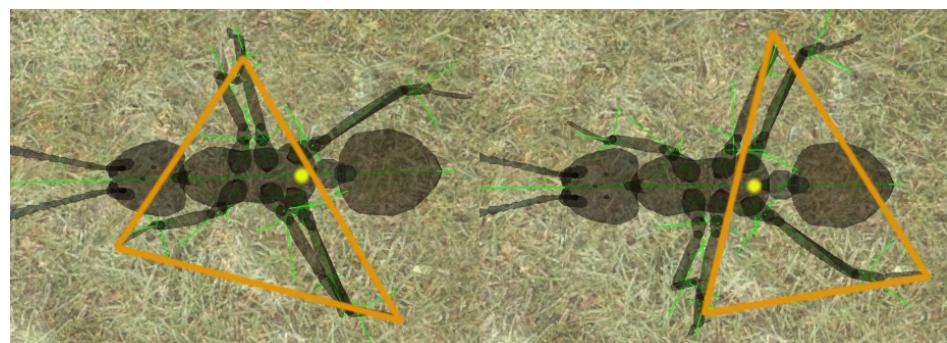


Obr. 6.3: Vizualizácia činnosti pozičných kontrolerov prednej nožičky.

Z hľadiska kinematiky modelu sme pracovali s informáciami popísanými v kapitole venovanej problematike chôdze. Pri chôdzi po rovine uhly kíbov na začiatku a konci stance fázy (obr. č. 6.4) a tažisko nachádzajúce sa vnútri podporného trojuholníka (obr. č. 6.5) korešpondujú s biologickým modelom.



Obr. 6.4: Bočný pohľad na uhly nožičiek ľavej trojnožky na začiatku (vľavo) a konci (vpravo) jej stance fázy.



Obr. 6.5: Pohľad zhora na dotykové body ľavej trojnožky na začiatku (vľavo) a konci (vpravo) stance fázy vytvárajúce podporný trojuholník. Tažisko je znázornené žltou bodkou.

V prostredí simulácie sa nachádza rovina, rovina s tridsiatimi náhodne umiestnenými modelmi troch rôznych kamienkov, mierny vrch a jama, priečne priepasti rôznej hĺbky a šírky a zvlnený terén. Ovládaním model nasmerujeme na tieto elementy a sledujeme jeho správanie. Môžeme konštatovať, že hoci je chôdza modelu naprogramovaná a vyladená iba na rovinu, mravec si dokáže s väčšinou prostredia poradiť. Je tomu tak, kvôli jeho stabilnej anatomickej stavbe a robustnosti chôdze realizovanej spolu šiestimi končatinami procesom striedajúcich sa trojnožiek. Výsledný ROS graf systému získame príkazom `rqt_graph` (príloha č. 2.5). Všetky súbory tu popisované potrebné k simulácii sa nachadzajú vo voľne prístupnom github repozitári github.com/a-pavco/ant_simulation. Videozáznamy vytvoreného riešenia sú dostupné na youtube.com/playlist?list=PLGknvKuL0I3BR1XRI57Kiox47zz4MR9cN.

Kapitola 7

Záver

Vytvorili sme virtuálny simulátor mrvaca a jeho chôdze, ktorý je v dostatočnej miere verný biologickej predlohe. Na jeho implementáciu sme použili moderné otvorené prostriedky Gazebo a ROS. Podľa našich znalostí vedci zaoberajúci sa pohybom hmyzu pracujú s kinematicky vernými, no vizuálne strohými modelmi. Naše riešenie ako jedno z mála spája anatomické a kinematické dátá s verným vizuálnym modelom simulovaného živočícha a ponúka čitateľovi atraktívny a dostupný demonštračný program simulácie.

Od predchodcu sme prevzali súbory modelu mrvaca získané stereoskopiou, ktoré sme najskôr museli konvertovať do nami použiteľných formátov. Vytvorili sme model mrvaca v natívnom Gazebo formáte SDF, ktorý sme sa následne pokúšali doplniť systémom ROS a vytvoriť v ňom jeho riadenie. Tento prístup sa neosvedčil, neposkytoval totiž dostatočnú podporu medzi SDF a riadiacou architektúrou. Preto sme vytvorili model mrvaca vo formáte URDF. Počas vývoja riešenia sme ho neustále upravovali a dopĺňali podľa nových informácií, ladenia a potrieb simulácie. Následne sme k nemu vytvorili a pridali komunikačné rozhranie a riadenie charakteristické pre Gazebo+ROS systémy. Demonštračný program poskytuje používateľovi možnosť interaktívneho ovládania klávesnicou. V simulovanom prostredí sa okrem roviny nachádzajú rôzne variácie terénu. Pri kráčaní po rovine je chôdza mrvaca verná prírodnému modelu. Simulovaný robot prekoná aj väčšinu prekážok prostredia, pričom sú vtedy zreteľné výhody chôdze realizovanej vzorom striedajúcich sa trojnožiek a anatomickej stavby mrvaca.

Prácu sme písali spôsobom, aby zároveň s jej čítaním mohol čitateľ získavať praktické informácie a skúsenosti o používaných konceptoch a nástrojoch, ich prostredí a použití. Na konci by tak čitateľ mal mať prehľad a základ zručností k testovaniu, ladeniu a rozširovaniu vytvoreného systému. Riešenie poskytuje východiská aj pre prípadnú nadväzujúcu svojstojnú prácu. Použité nástroje sú široko rozšírené a majú ešte niekoľko rokov podporu (Gazebo 9 aj ROS Melodic do 2023), pričom migrácia na novšie by nemala predstavovať problém. Aspekty ako ladenie chôdze, vytvorenie sofistikovanejšieho riadenia, práca so senzormi alebo orientácia v prostredí sú všetko zaujímavé témy na ďalšiu prácu. Okrem toho môže dokument pomôcť každému, kto sa chce zoznámiť, alebo vytvárať akékolvek iné modely, riadenie a simulácie v daných nástrojoch.

Literatúra

- [3dp] 3D Print Toolbox nástroj. <https://github.com/caretdashcaret/MeshRepairFor3DPrinting>. verz. z 2018/12.
- [anaa] Ant anatomy. <https://flrec.ifas.ufl.edu/media/flrecifasufledu/pdfs/pestants/AntAnatomy.pdf>. verz. z 2018/12.
- [anab] Ant anatomy, ask a biologist. <https://askabiologist.asu.edu/explore/ant-anatomy>. verz. z 2018/12.
- [BGE⁺16] Julia Badger, Dustin Gooding, Kody Ensley, Kimberly Hambuchen, and Allison Thackston. *ROS in space: A case study on robonaut 2*, volume 625, pages 343–373. 02 2016.
- [ble] Blender nástroj. <https://www.blender.org/>. verz. z 2018/12.
- [Bul] Bullet. <https://github.com/bulletphysics/bullet3>. verz. z 2019/1.
- [cat] catkin. <http://wiki.ros.org/catkin>. verz. z 2019/3.
- [cma] cmake. <https://cmake.org/overview/>. verz. z 2019/3.
- [CS05] Holk Cruse and Malte Schilling. *First order and second order embodiment – robots with the ability to plan ahead*. Bielefeld University, 2005.
- [dae] COLLADA formát. <https://www.khronos.org/collada/>. verz. z 2018/12.
- [DAR] DART. <https://dartsim.github.io/>. verz. z 2019/1.
- [enc] Formica cinerea Mayr, 1853 - Encyclopedia of Life. <http://www.eol.org/pages/400890>. verz. z 2018/12.
- [eng] Comparison of Rigid Body Dynamic Simulators for Robotic Simulation in Gazebo, ROSCon 2014. <https://vimeo.com/107517366>. verz. z 2019/4.
- [fre] FreeWRL nástroj. <http://freewrl.sourceforge.net/>. verz. z 2018/12.

- [Gaza] Gazebo. <http://gazebosim.org/>. verz. z 2019/1.
- [Gazb] GazGUI. <http://gazebosim.org/tutorials?tut=guided%5Fb2>. verz. z 2019/1.
- [Gazc] GazInstall. <http://gazebosim.org/tutorials?cat=install>. verz. z 2019/1.
- [Gazd] GazStruct. http://gazebosim.org/tutorials?tut=model_structure. verz. z 2019/1.
- [GLU] GLUT. <https://www.opengl.org/resources/libraries/glut/>. verz. z 2019/1.
- [GLWL18] Shihui Guo, Juncong Lin, Toni Wöhrl, and Minghong Liao. A neuro-musculo-skeletal model for insects with data-driven optimization. *Scientific Reports*, 8, 12 2018.
- [Hol09] Tate Holbrook. Face to face with ants. 2009.
- [HUG52] G. M. HUGHES. The co-ordination of insect movements. *Journal of Experimental Biology*, 29(2):267–285, 1952.
- [KH04] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sep. 2004.
- [lay] Microsoft lays off Robotics team. https://www.roboticsbusinessreview.com/research/microsoft_lays_off_robots_team/. verz. z 2018/12.
- [Man72] S. M. Manton. The evolution of arthropodan locomotory mechanisms. *Zoological Journal of the Linnean Society*, 51(3-4):203–400, 1972.
- [mesa] Let's talk mesh repair. <https://caredashcaret.com/2014/12/04/lets-talk-mesh-repair/>. verz. z 2018/12.
- [mesb] Mesh Cleaner nástroj. https://www.hamzamerzic.info/mesh_cleaner/. verz. z 2018/12.
- [mesc] Meshconv nástroj. <https://www.patrickmin.com/meshconv/>. verz. z 2018/12.
- [mesd] Meshlab nástroj. <http://www.meshlab.net/>. verz. z 2018/12.

- [mrd] MRDS. [https://docs.microsoft.com/en-us/previous-versions/microsoft-robotics/bb483024\(v=msdn.10\)](https://docs.microsoft.com/en-us/previous-versions/microsoft-robotics/bb483024(v=msdn.10)). verz. z 2018/12.
- [MSFM16] Anil Mahtani, Luis Sanchez, Enrique Fernandez, and Aaron Martinez. *Effective Robotics Programming with ROS - Third Edition*. Packt Publishing, 3rd edition, 2016.
- [obj] Wavefront obj formát. http://www.cs.utah.edu/~boulos/cs3505/obj_spec.pdf. verz. z 2018/12.
- [ODE] ODE. <https://www.ode.org/>. verz. z 2019/1.
- [OGR] OGRE. <https://www.ogre3d.org/>. verz. z 2019/1.
- [O'K13] Jason M. O'Kane. *A Gentle Introduction to ROS*. Independently published, October 2013. Available at <http://www.cse.sc.edu/~jokane/agitr/>.
- [Ope] OpenGL. <https://www.opengl.org/>. verz. z 2019/1.
- [pid] PID Control. <https://apmonitor.com/pdc/index.php/Main/ProportionalIntegralDerivative>. verz. z 2019/4.
- [QCG⁺⁰⁹] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [QGS15] Morgan Quigley, Brian Gerkey, and William D. Smart. *Programming Robots with ROS: A Practical Introduction to the Robot Operating System*. O'Reilly Media, Inc., 1st edition, 2015.
- [Qt] Qt. <https://www.qt.io/>. verz. z 2019/1.
- [RB14] Lars Reinhardt and Reinhard Blickhan. Level locomotion in wood ants: evidence for grounded running. *Journal of Experimental Biology*, 217(13):2358–2370, 2014.
- [Ris11] Andrej Riska. Modelovanie správania živých systémov, FMFI UK, diplomová práca, 2011.
- [ROSa] ROS. <http://wiki.ros.org/>. verz. z 2019/1.
- [rosb] Rosmultiling. <http://wiki.ros.org/Client%20Libraries>. verz. z 2019/1.

- [RWB09] Lars Reinhardt, Tom Weihmann, and Reinhard Blickhan. Dynamics and kinematics of ant locomotion: do wood ants climb on level surfaces? *Journal of Experimental Biology*, 212(15):2426–2435, 2009.
- [Sch07] Josef Schmitz. *Neurobiological foundations of hexapod locomotion in insects and robots*. Bielefeld University, 2007.
- [sdfa] SDF inertial tutoriál. <http://gazebosim.org/tutorials?tut=inertia>. verz. z 2018/12.
- [sdfb] SDF model tutoriál. http://gazebosim.org/tutorials?tut=build_model. verz. z 2018/12.
- [sdfc] SDFormat. <http://sdformat.org/>. verz. z 2018/12.
- [sdfd] SDFormat Joints. <http://sdformat.org/spec?elem=joint>. verz. z 2018/12.
- [SHSC13] Malte Schilling, Thierry Hoinville, Josef Schmitz, and Holk Cruse. Walknet, a bio-inspired controller for hexapod walking. *Biological Cybernetics*, 107(4):397–419, 2013.
- [Sima] Simbody. <https://github.com/simbody/simbody>. verz. z 2019/1.
- [simb] Simtrans nástroj. <http://fkanehiro.github.io/simtrans/html/simtrans.html>. verz. z 2018/12.
- [SPSS14] Axel Schneider, Jan Paskarbeit, Malte Schilling, and Josef Schmitz. Hector, a bio-inspired and compliant hexapod robot. In Armin Duff, Nathan F. Lepora, Anna Mura, Tony J. Prescott, and Paul F. M. J. Verschure, editors, *Living Machines*, volume 8608 of *Lecture Notes in Computer Science*, pages 427–429. Springer, 2014.
- [Sta] Stage. <http://wiki.ros.org/stage>. verz. z 2019/1.
- [stl] STL formát. http://www.fabbers.com/tech/STL_Format. verz. z 2018/12.
- [urd] URDF formát. <http://wiki.ros.org/urdf>. verz. z 2018/12.
- [vrma] VRML formát. <http://xml.coverpages.org/related.html#vrml>. verz. z 2018/12.
- [vrmb] VrmlMerge nástroj. <http://www.deem7.com/vrmlmerge.php>. verz. z 2018/12.

- [x3d] X3D formát. <http://www.web3d.org/x3d/what-x3d>. verz. z 2018/12.
- [xac] xacro. <http://wiki.ros.org/xacro>. verz. z 2019/4.
- [xmla] xml. <https://www.w3.org/XML/>. verz. z 2019/3.
- [xmlb] xmlrpc. <https://docs.python.org/3/library/xmlrpc.html>. verz. z 2019/3.
- [yam] yaml. <https://yaml.org/>. verz. z 2019/4.
- [Zol94] Christoph Zollikofer. Stepping patterns in ants. *J Exp Biol*, 192(1):95–106, July 1994.

Prílohy

Zoznam príloh práce:

1. súborové prílohy:

1.1. github repozitár všetkých súborov práce:

https://github.com/a-pavco/ant_simulation

2. obrázkové prílohy:

2.1. ilustrácia zaplátania 3D súboru v Meshlabe

2.2. ilustrácia odstránenia vnútornej plochy 3D súboru v Meshlabe

2.3. verifikácia a hierarchia URDF modelu

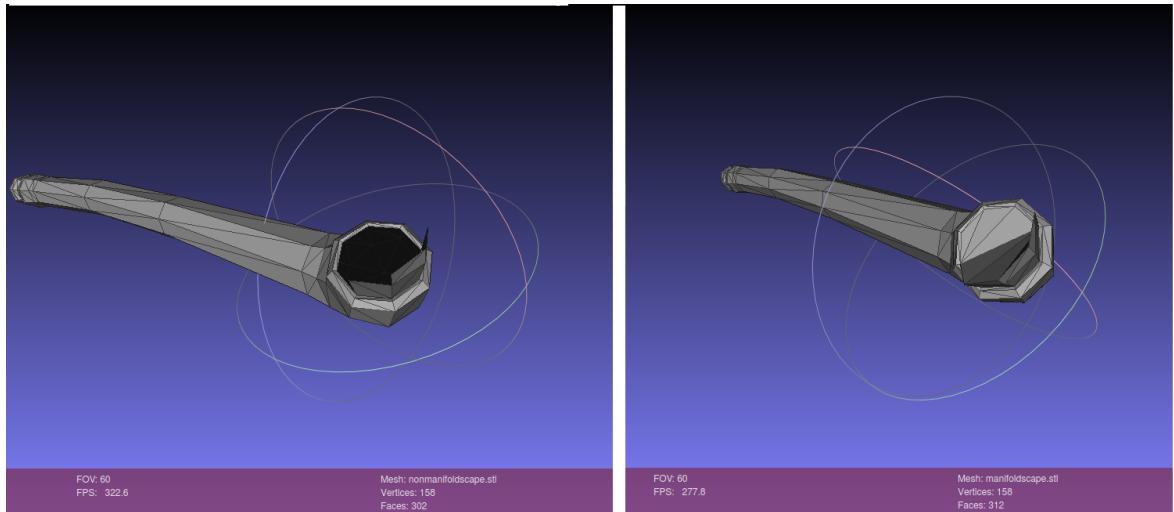
2.4. Graphviz diagram URDF modelu

2.5. ROS graf kompletného riešenia

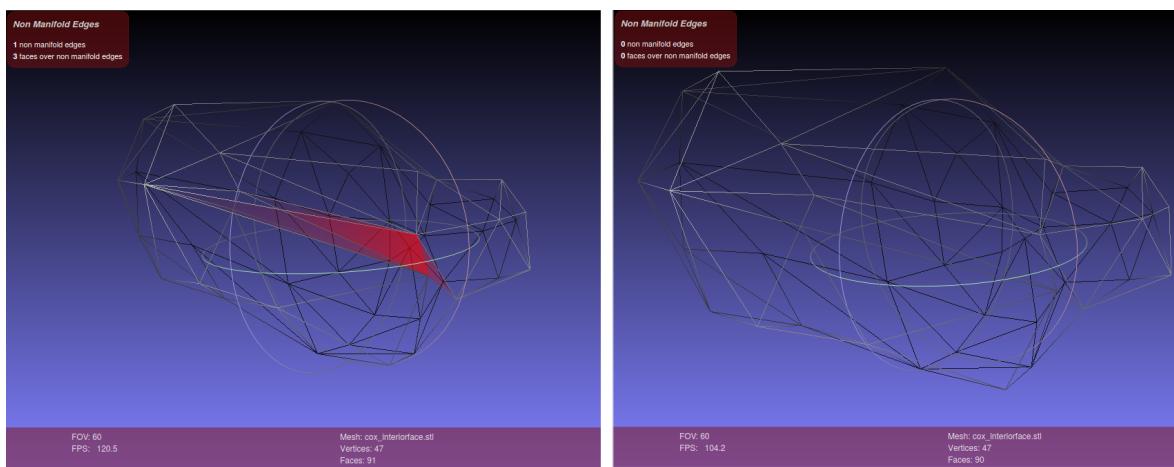
3. video prílohy:

3.1. videozáznamy simulácie:

youtube.com/playlist?list=PLGknvKuL0I3BR1XRI57Kiox47zz4MR9cN

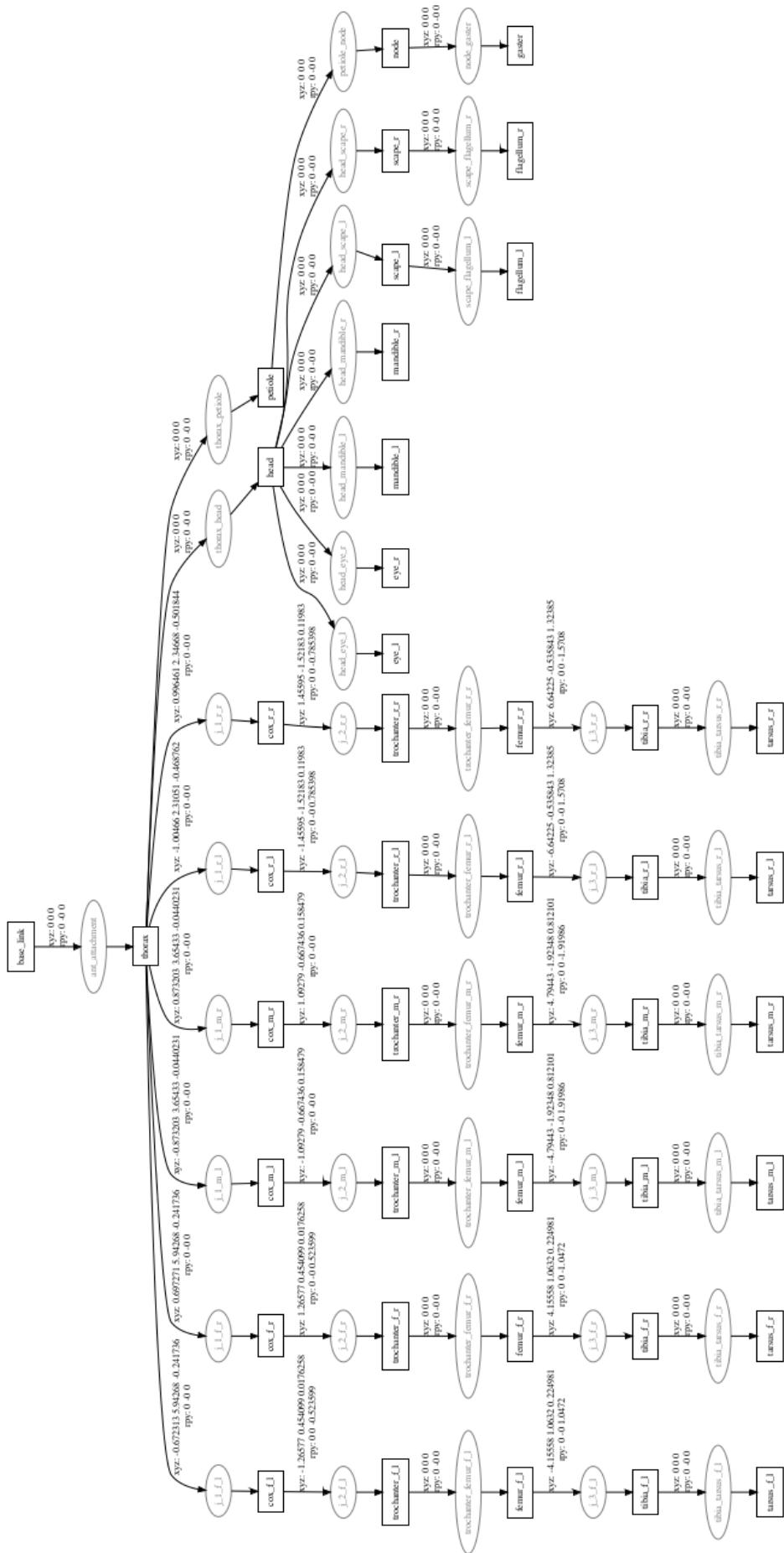


Príloha č. 2.1: Nevodotesný 3D súbor tykadla pred (vľavo) a po zaplátaní (vpravo) v Meshlabe.

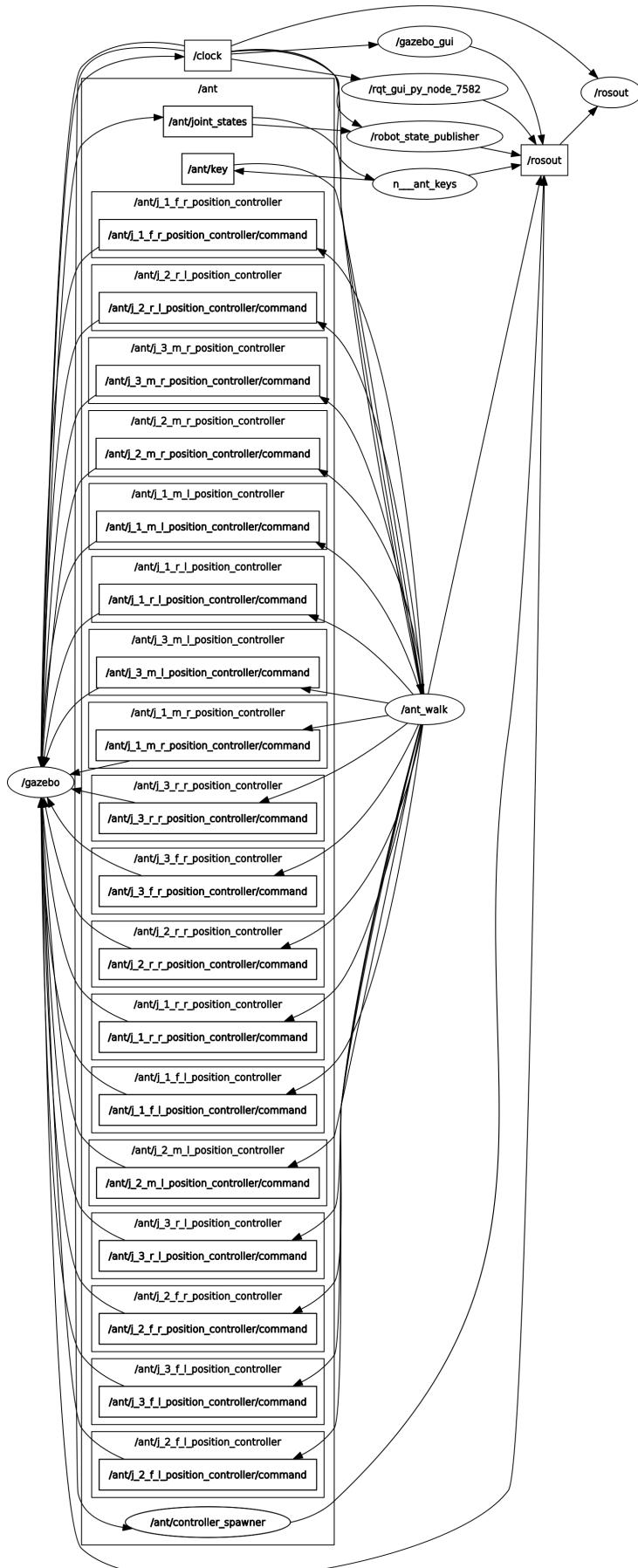


Príloha č. 2.2: Odstránenie vnútornej plochy v 3D súbore coxy v Meshlabe, pred (vľavo) a po (vpravo).

```
robot name is: ant
----- Successfully Parsed XML -----
root Link: base_link has 1 child(ren)
    child(1): thorax
        child(1): cox_f_l
            child(1): trochanter_f_l
                child(1): femur_f_l
                    child(1): tibia_f_l
                        child(1): tarsus_f_l
        child(2): cox_f_r
            child(1): trochanter_f_r
                child(1): femur_f_r
                    child(1): tibia_f_r
                        child(1): tarsus_f_r
        child(3): cox_m_l
            child(1): trochanter_m_l
                child(1): femur_m_l
                    child(1): tibia_m_l
                        child(1): tarsus_m_l
        child(4): cox_m_r
            child(1): trochanter_m_r
                child(1): femur_m_r
                    child(1): tibia_m_r
                        child(1): tarsus_m_r
        child(5): cox_r_l
            child(1): trochanter_r_l
                child(1): femur_r_l
                    child(1): tibia_r_l
                        child(1): tarsus_r_l
        child(6): cox_r_r
            child(1): trochanter_r_r
                child(1): femur_r_r
                    child(1): tibia_r_r
                        child(1): tarsus_r_r
        child(7): head
            child(1): eye_l
            child(2): eye_r
            child(3): mandible_l
            child(4): mandible_r
            child(5): scape_l
                child(1): flagellum_l
            child(6): scape_r
                child(1): flagellum_r
        child(8): petiole
            child(1): node
                child(1): gaster
```



Príloha č. 2.4: Graphviz diagram URDF modelu mravca.



Príloha č. 2.5: ROS graf kompletného riešenia.