

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV AUTOMATIZACE A INFORMATIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

MODELOVÁNÍ A SIMULACE ROBOTICKÝCH APLIKACÍ

MODELLING AND SIMULATION OF ROBOTIC APPLICATIONS

DIPLOMOVÁ PRÁCE
DIPLOMA THESIS

AUTOR PRÁCE **Bc. MARTIN ŠŤASTNÝ**
AUTHOR

VEDOUCÍ PRÁCE **prof. RNDr. Ing. JIŘÍ ŠŤASTNÝ, CSc.**
SUPERVISOR

BRNO 2015

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav automatizace a informatiky
Akademický rok: 2014/2015

ZADÁNÍ DIPLOMOVÉ PRÁCE

student(ka): Bc. Martin Šťastný

který/která studuje v **magisterském navazujícím studijním programu**

obor: **Aplikovaná informatika a řízení (3902T001)**

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

Modelování a simulace robotických aplikací

v anglickém jazyce:

Modelling and simulation of robotic applications

Stručná charakteristika problematiky úkolu:

V práci bude zpracována literární rešerše Open Source softwaru pro simulaci autonomních robotů. Student se seznámí se simulačním prostředím GAZEBO v rámci systému ROS (Robot Operating System). Po volbě vhodných senzorů pro vybranou robotickou úlohu bude v prostředí GAZEBO provedena simulace robotické úlohy (např: Bear Rescue Advanced, Ketchup House, RoboCarts).

Cíle diplomové práce:

- Zpracování rešerše v dané oblasti
- Zvládnutí prostředí GAZEBO a systému ROS
- Návrh robotických úloh s vybranými senzory
- Implementace a vyhodnocení simulačních experimentů

Seznam odborné literatury:

[1] DIRK, Thomas. Rosbag tool. WILLOW GARAGE. Robot Operating System [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/rosbag.
[2] FONTANA, Flavio. ROS on Windows. WILLOW GARAGE. Robot Operating System [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/win_ros/hydro/Msvc%20SDK.
[3] OPITZ, Tobias. Books. WILLOW GARAGE. Robot Operating System [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/Books.
[4] THRUN, S., BURGARD, W., FOX, D. Probabilistic robotics. The MIT Press, 2005, ISBN 0-262-20162-3.

Vedoucí diplomové práce: prof. RNDr. Ing. Jiří Šťastný, CSc.

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

V Brně, dne 21.11.2014

L.S.

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

ABSTRAKT

Cílem této diplomové práce je provést rešerši v oblasti Open Source softwaru pro simulaci autonomních robotů. V úvodu je provedena rešerše vybraných robotických simulátorů.

V první části této práce je seznámení se s robotickým simulátorem Gazebo a robotickým frameworkem ROS.

Druhá část této práce se zabývá simulováním a následnou implementací vybranné robotické úlohy prostřednictvím simulátoru Gazeba a frameworku ROS.

ABSTRACT

The aim of this master thesis is to make research of Open Source software, which are used for simulation autonomous robots. At the begining is performed research of selected robotic simulators.

In the first part of this work is to get familiar with robotic simulator Gazebo and robotic framework ROS.

The second part of this work deals with simulating and subsequent implementation of choosen robotic tasks through the simulator Gazebo and the ROS framework.

KLÍČOVÁ SLOVA

Simulace, Robotický simulátor, SW pro simulaci, V-rep, Webots, Gazebo, Open source, ROS, Sledování čáry, Catkin, Turtlebot, Lego NXT

KEY WORDS

Simulation, Robotic simulator, Software for the simulation, V-rep, Webots, Gazebo, Open source, ROS, Line following, Catkin, Turtlebot, Lego NXT

PROHLÁŠENÍ O ORIGINALITĚ

Tímto prohlašuji, že tato předkládaná diplomová práce je vypracována mnou samostatně a všechny zdroje, prameny a literaturu, které jsem při vypracování použil, nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

.....

Bc. Martin Šťastný, Brno, 2015

BIBLIOGRAFICKÁ CITACE

ŠŤASTNÝ, M. *Modelování a simulace robotických aplikací.* Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2015. 67s. Vedoucí diplomové práce prof. RNDr. Ing. Jiří Šťastný, CSc.

PODĚKOVÁNÍ

Tímto bych chtěl poděkovat svému vedoucímu diplomové práce prof. RNDr. Ing. Jiřímu Šťastnému, CSc. za jeho cenné rady a za celkovou pomoc při realizaci této práce.

Také bych chtěl poděkovat panu Ing. Tomáši Kubínovi, za cennou pomoc při realizaci této práce.

OBSAH

ABSTRAKT 5
KLÍČOVÁ SLOVA 5
PROHLÁŠENÍ O ORIGINALITĚ 7
BIBLIOGRAFICKÁ CITACE 7
PODĚKOVÁNÍ 9
1 ÚVOD 5
2 SW PRO SIMULACI 7
2.1 Open Source SW pro simulaci autonomních robotů 7
3 ROBOTICKÉ SIMULÁTORY 9
3.1 V-REP 9
3.1.1 V-REP – popis simulátoru 9
3.1.2 V-REP – simulace 13
3.2 Webots 14
3.2.1 Webots – popis simulátoru 15
3.2.2 Webots – simulace 15
3.3 Gazebo 16
3.3.1 Gazebo – popis simulátoru 17
3.3.2 Gazebo – simulace 19
4 ROS 23
4.1 Historie ROS 24
4.2 Cíle ROS 25
4.3 ROS koncept 25
4.3.1 Úroveň souborového systému 26
4.3.2 Úroveň výpočetního grafu 26
4.3.3 Úroveň ROS komunity 27
4.4 Build system 27
4.4.1 Build system Catkin 28
5 GAZEBO S VYUŽITÍM ROS 29
5.1 Instalace Gazeba 30
5.2 Instalace ROS 31
5.3 Propojení Gazeba s ROS 33
5.3.1 Otestování Gazeba s integrací ROS 33
5.3.2 Jiné cesty ke spuštění Gazeba s ROS 35

5.3.3 Roslaunch 35
6 Robotická úloha 39
6.1 Line Following 39
6.2 Simulace robotické úlohy 39
6.2.1 Model robota 40
6.2.2 Prostředí – svět 41
6.2.3 Vytvoření ROS balíčků 41
6.2.4 Simulace 43
6.3 Implementace řídícího algoritmu 45
6.3.1 Robot LEGO Mindstorms NXT 46
6.3.2 Propojení robotu NXT s frameworkem ROS 48
6.3.3 Vytvoření ROS balíčků 48
6.3.4 Provedení robotické úlohy 50
7 ZÁVĚR 53
SEZNAM CITOVANÝCH ZDROJŮ 55
SEZNAM ODBORNÉ LITERATURY 59
SEZNAM PŘÍLOH NA CD 59

1 ÚVOD

Robotika je věda o robotech, jejich designu a vývoji aplikací. V dnešní době se robotika významně rozšiřuje a rozšiřuje se tak i množství používaných robotických aplikací, které usnadňují lidem život, nebo se jimi lidé baví či vzdělávají. Poptávka po těchto aplikacích stále stoupá, a proto se hledají nové jednodušší způsoby, jak tyto robotické aplikace vytvářet.

Nejjednodušším způsobem jak vytvořit některou robotickou aplikaci, by bylo její sestavení, z již vytvořených částí. Například pro vytvoření robotické aplikace, kdy robot má za úkol najít nejkratší či nejrychlejší cestu z bludiště, by stačilo vybrat příslušné senzory, ke kterým by již byly vytvořeny knihovny a řídící algoritmy komunikující přes všechny robotické platformy.

Jednoduché vytváření robotických aplikací podporuje robotický operační systém ROS. ROS – Robotic Operation System, není plnohodnotným operačním systémem a tak bývá označován spíše jako framework. Jeho cílem je spojit co největší komunitu lidí, zabývající se oblastí robotiky. Jeho myšlenkou je poskytování a využívání vytvořené práce navzájem. Podporuje opětovné použití kódů pro výzkum a vývoj robotiky. Tato práce se bude zabývat použitelností, zmiňovaného robotického operačního systému ROS, ve vývoji robotických aplikací.

Všechny robotické aplikace, které se mají použít v reálném světě, se musejí před prvním reálným spuštěním otestovat. K tomu se využívají robotické simulátory, které umožňují simulaci robotů ve vymodelovaném prostředí. Tyto simulace jsou velmi přesné a jsou vhodné k vyhodnocování použitelnosti robotických aplikací v reálném světě. První část této práce bude zaměřena právě na tyto robotické simulátory, které jsou v oblasti robotiky velmi důležité. Vzhledem k tématu práce budou vybrány pouze Open Source robotické simulátory.

Druhá část této práce bude zaměřena na samotné simulace v robotickém simulátoru Gazebo. V této části bude ukázáno propojení simulátoru Gazebo s frameworkem ROS a využívání tzv. „ROS balíčků“ k usnadnění vytváření robotických aplikací.

Jednou z nejdůležitějších věcí pro robotické simulace, je možnost jejich implementace na skutečné roboty ve skutečném světě. Proto se třetí část práce bude zabývat vytvořením vybrané robotické úlohy a to jak z hlediska simulace, tak z hlediska implementace na reálného robota. Obě dvě hlediska budou probíhat v rámci frameworku ROS.

2 SW PRO SIMULACI

„Simulační programy převádějí matematický model (nebo transformovaný matematický model) do tvaru modelu simulačního, který je možno popsat výrazovými prostředky daného simulačního programu nebo pomocí programovacího jazyka.“[1]

V dnešní době existuje mnoho rozmanitých softwarových balíčků. Tyto balíčky jsou určené pro modelování počítačově založených simulací, které umožňují jeho relativně snadnou realizaci.[2]

2.1 Open Source SW pro simulaci autonomních robotů

Tato práce se bude zabývat Open Source softwarem pro robotické simulace. Open Source software je volně šiřitelný a má tzv. „otevřený“ zdrojový kód. „Otevřenost“ v tomto případě znamená, že se tento kód může volně využívat a upravovat.

Simulace autonomních robotů je simulace, při které roboti dokážou pracovat samostatně (v reálném čase je neřídí člověk, ale program). Jedná se tedy o roboty, které jsou předem naprogramované k nějakému účelu.

Open Source softwarů (robotických simulátorů) pro simulaci autonomních robotů je v dnešní době mnoho. Některé mohou být pouze pro simulaci robotických ramen, jiné pouze pro simulaci specifických robotů. Dále se dají rozlišovat například podle 2D nebo 3D simulací, nebo podle reálnosti vytvořeného světa, ve kterém daná simulace probíhá. Některé simulátory obsahují fyzikální enginy, které zajišťují vlastnosti reálného světa (gravitace, tření, odpor vzduchu).

Tato práce je zaměřena na robotické simulátory, které mají širokou škálu použití. Poskytují 3D simulace reálného světa vhodné pro přenos na skutečné roboty. Mezi tyto simulátory patří robotické simulátory V-REP, Webots, Gazebo.



Obr. 1 Loga robotických simulátorů.[3][4][5]

3 ROBOTICKÉ SIMULÁTORY

Robotické simulátory jsou softwary pro simulaci autonomních robotů. Používají se k vytváření vestavěných robotických aplikací pro specifické, nebo jakékoli roboty, bez fyzické závislosti na skutečném robotovi. V některých případech se tyto aplikace dají přenést na skutečné roboty bez jakýchkoliv úprav.[6]

3.1 V-REP

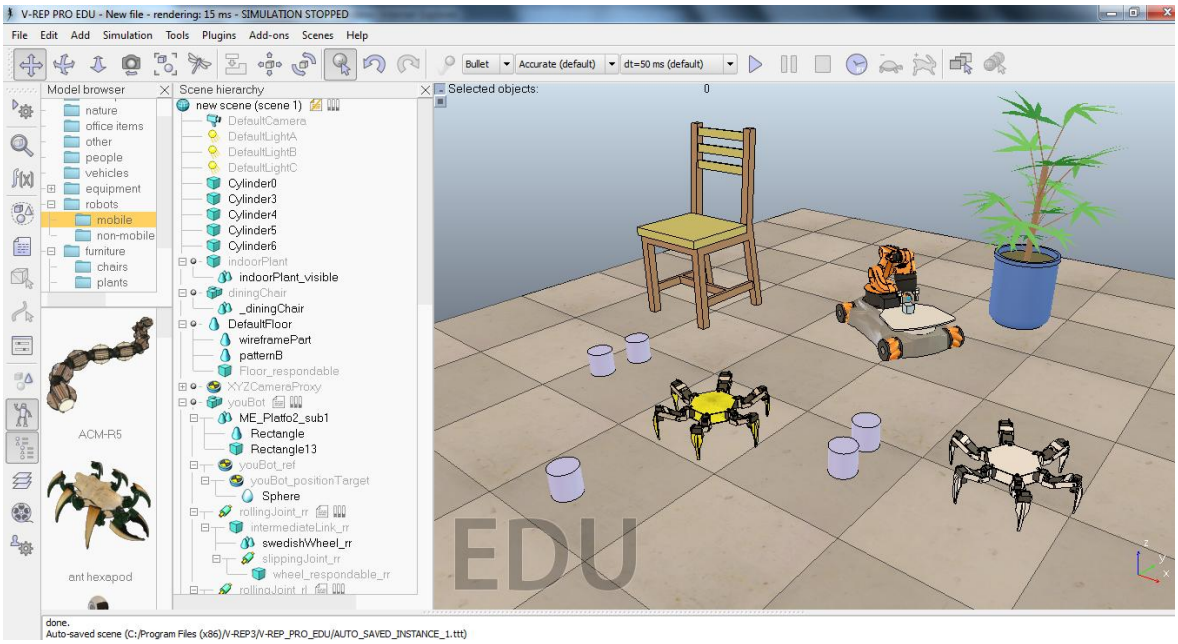
V-REP je švýcarský robotický simulátor. Tento simulátor je vybaven mnoha funkcemi a obsahuje jedno z nejpropracovanějších rozhraní API (Application Programming Interface).

Jeho integrované vývojové prostředí je založeno na distribuované řídící architektuře, tzn., že každý objekt/model může být individuálně ovládán pomocí vestavěného skriptu, pluginu, uzlu ROS nebo pomocí klienta vzdálené API. Tímto je V-REP velmi univerzální a ideální pro vytváření multi-robotických aplikací.[7]

Vestavěné skripty (pluginy, uzly ROS) mohou být psány v několika programovacích jazycích: C/C++, Python, Java, Lua, Matlab, Octave, Urbi 7.

3.1.1 V-REP – popis simulátoru

Robotický simulátor V-REP je dostupný pro operační systémy Windows, Mac OS, Linux.



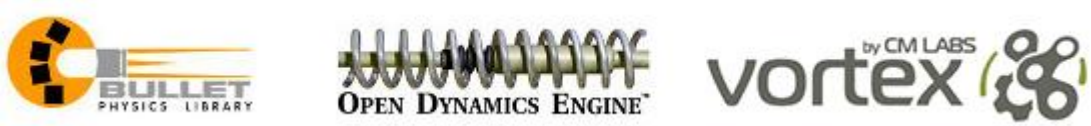
Obr. 2 Prostředí robotického simulátoru V-REP.[9]

V-REP je vhodný pro použití z hlediska:

- rychlého vývoje algoritmů,
- výrobní automatizační simulace,
- rychlého vytváření prototypů a jejich otestování,
- robotického vzdělání,
- vzdáleného monitorování,
- bezpečnostních simulací.[3]

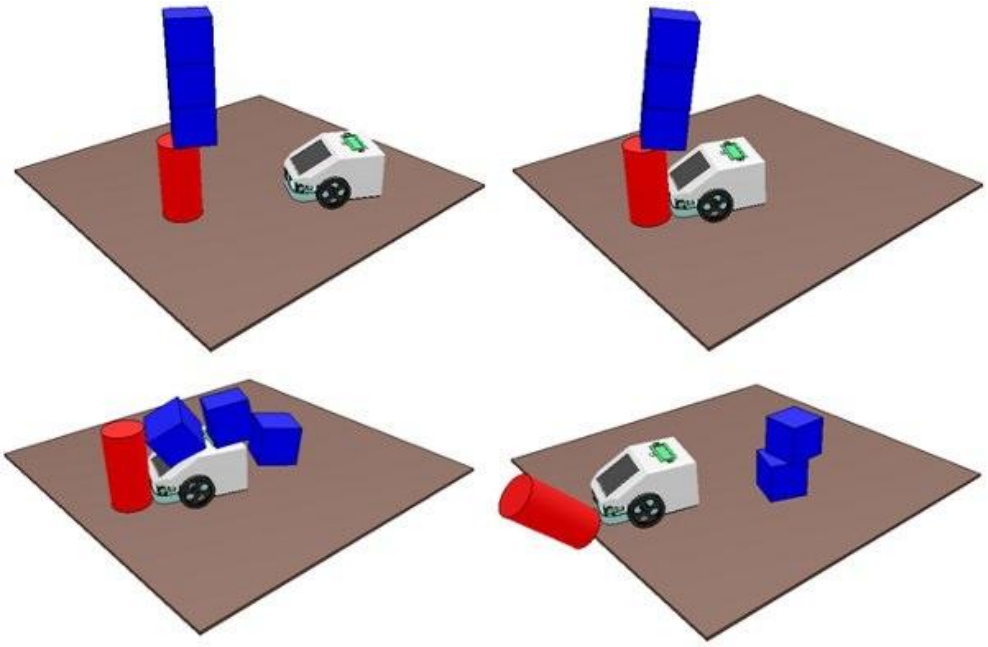
V-REP obsahuje tři fyzikální enginy, které se starají o reálnost simulace z dynamické stránky. Těmito enginy jsou: Bullet physics library, Open Dynamics Engine a Vortex Dynamics engine.

Důvodem pro tuto rozmanitost fyzikálních enginů je složitost provedení simulace s využitím fyzikálních zákonů. Díky těmto enginům může být v simulaci dosaženo vyššího stupně přesnosti, rychlosti a podpory mnoha dalších funkcí.[8]



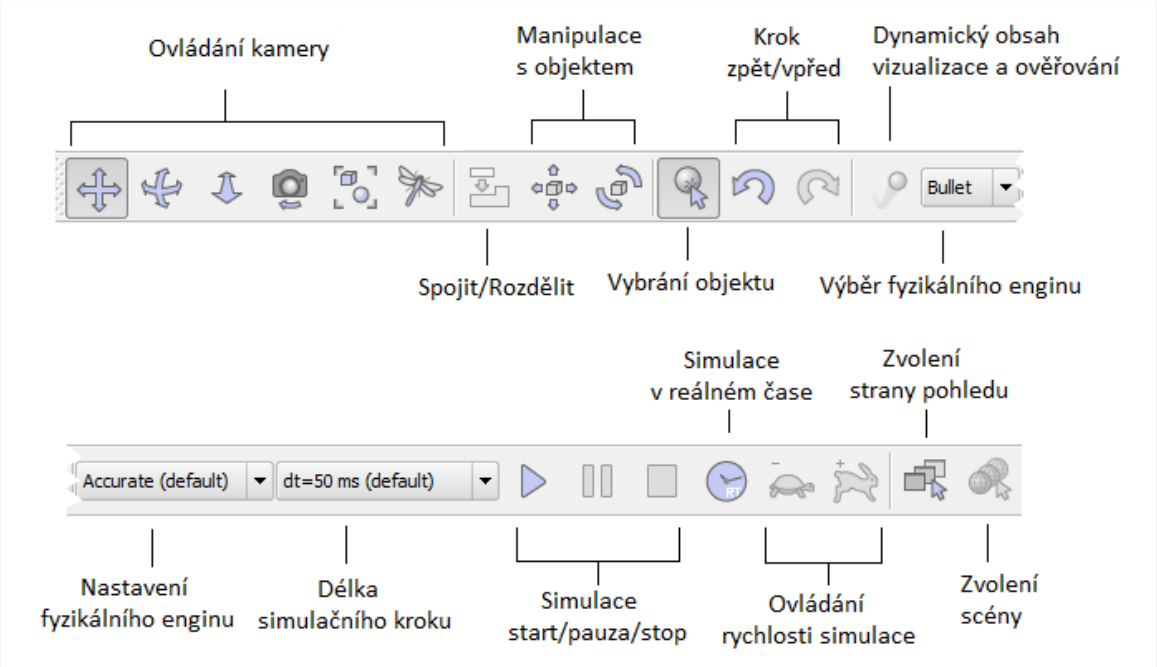
Obr. 3 Loga fyzikálních enginů.[8]

Dynamické moduly umožňují simulovat interakce mezi objekty, které vypadají jako interakce objektů v reálném světě. Tzn., že objekty v simulaci mohou padat, srážet se nebo odrážet. Také umožňují manipulátoru uchopit předměty, dopravnímu pásu přepravovat objekty a vozidlu přejet nerovný terén realistickým způsobem. Dynamika simulace je znázorněna na *Obr. 4*.[8]

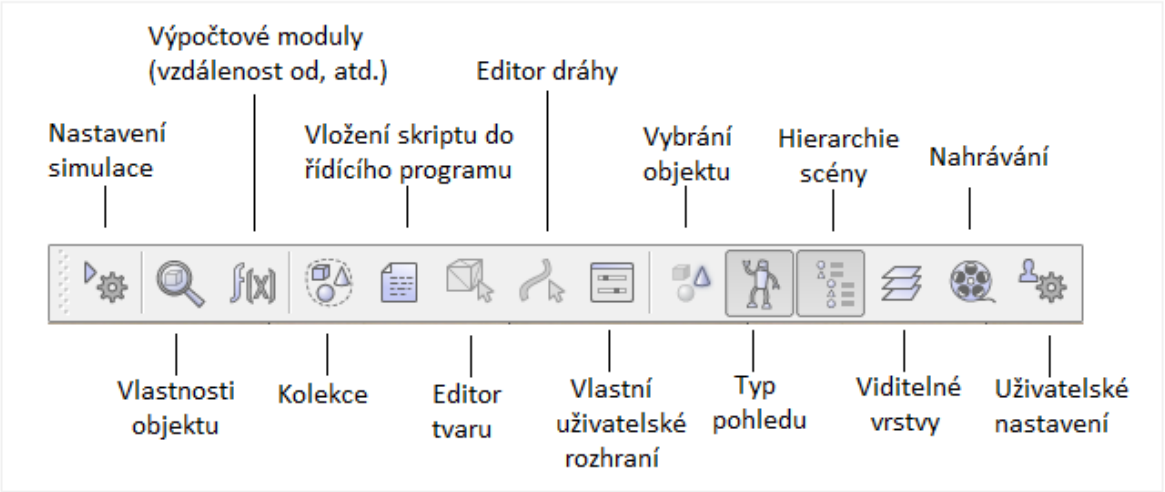


Obr. 4 Ukázka dynamické simulace.[8]

V simulovaném prostředí slouží k ovládání simulátoru V-REP dva hlavní panely. První panel má na starost ovládání samotné simulace (*Obr. 5)*, druhý pak ovládání scény (*Obr. 6)*.

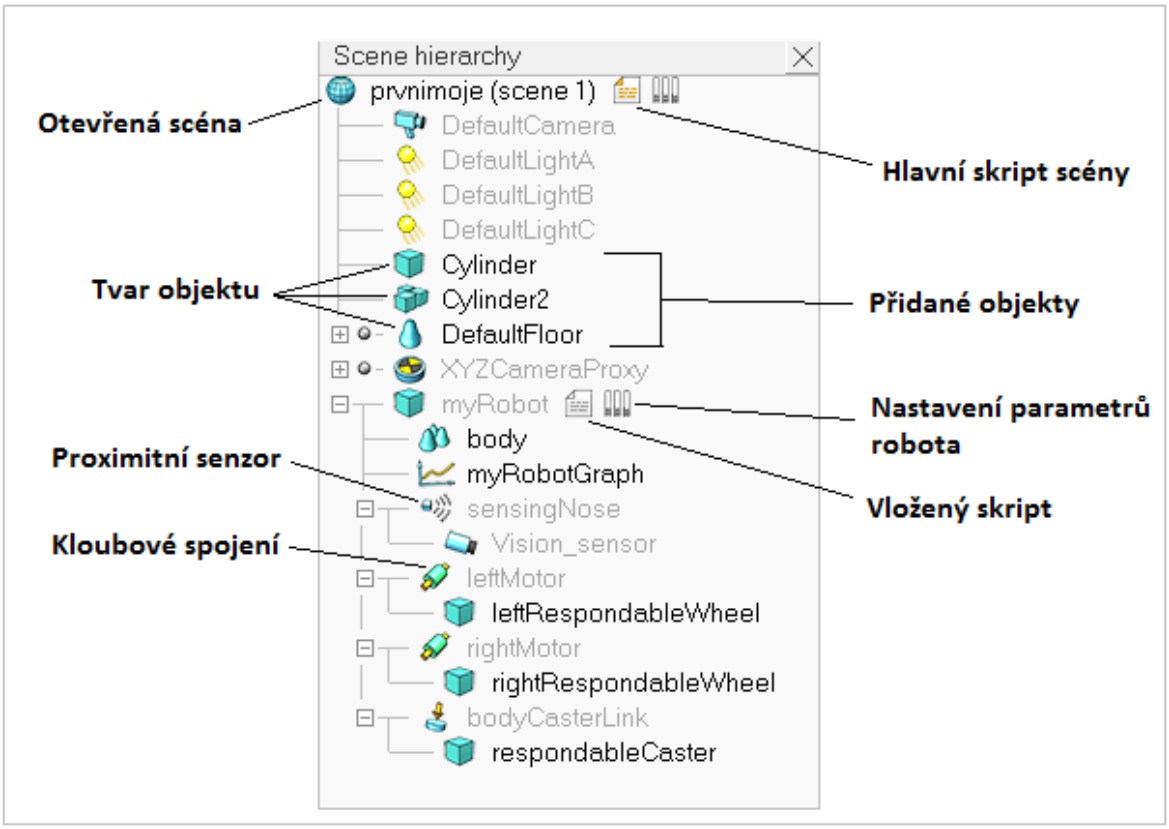


Obr. 5 Ovládací panel 1 (Toolbar 1).[9]



Obr. 6 Ovládací panel 2 (Toolbar 2).[9]

Velmi důležitým prvkem je hierarchie scény *(Obr. 7)*, která zobrazuje všechny objekty ve vytvořeném světě. Uvádí jejich vlastnosti, vzájemné závislosti a návaznosti. V této hierarchii se dají, snadným přetažením objektů/skriptů, vytvořit vztahy mezi objekty/skripty typu nadřazený nebo podřazený.



Obr. 7 Hierarchie scény V-REP.[9]

Robotický simulátor V-REP nabízí mnoho modelů robotů nebo objektů, které lze libovolně přidávat do scény jednoduchým přetažením. Z nabídky modelů *(Obr. 8)*, je možné tyto modely přidávat i do právě probíhající simulace.

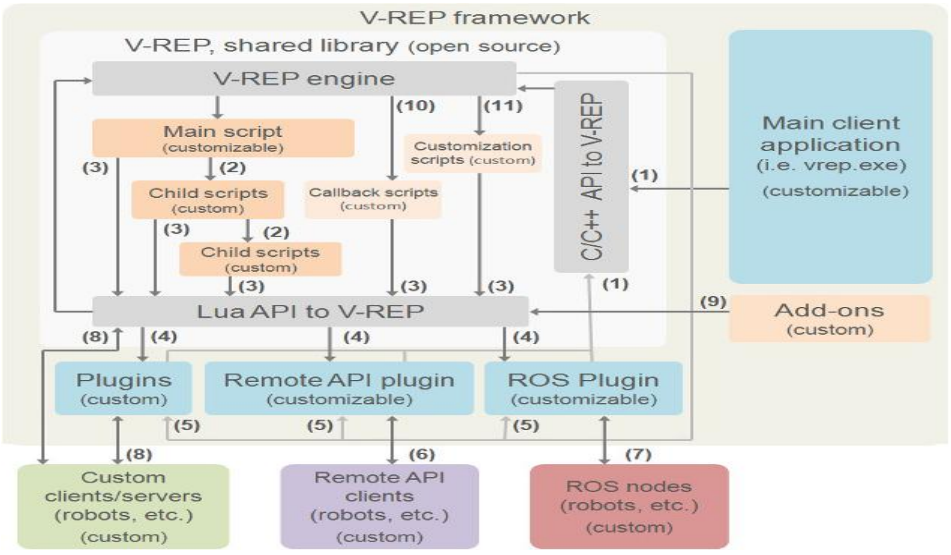


Obr. 8 Nabídka modelů V-REP.[9]

3.1.2 V-REP – simulace

V-REP je vysoce přizpůsobitelný simulátor, každý aspekt simulace lze přizpůsobit. To zajišťuje velmi propracované API (Application Programming Interface), které podporuje šest různých programovacích nebo kódovacích přístupů, z nichž každý má určité výhody (a samozřejmě i nevýhody) nad ostatními, ale všechny přístupy jsou vzájemně kompatibilní (tj. mohou být použity ve stejnou dobu, nebo dokonce „ruku v ruce“). Řídící entita modelů, scén, nebo samotné simulace může být umístěna v jedné z těchto variant:

- Vložený skript – přizpůsobení simulace (scény, modelů) se děje pomocí skriptů. Tyto skripty jsou psány pomocí skriptovacího jazyka Lua. Tato metoda umožňuje přizpůsobení konkrétní simulace, simulační scény a do určité míry přizpůsobení samotného simulátoru. Je to nejjednodušší a nejpoužívanější programovací přístup.
- Add-on (doplněk) – tato metoda využívá Lua skripty a umožňuje rychlé přizpůsobení simulátoru. Doplněk se spouští automaticky a běží na pozadí. Doplňky nejsou specifické pro určité simulace nebo modely, nabízejí spíše obecnější funkce simulátoru.
- Plugin (zásuvný modul) – tj. přizpůsobení simulátoru nebo simulace pomocí psaného pluginu pro V-REP. Pluginy se vyvolávají pomocí Lua skriptů, proto jsou pluginy používány ve spojení s vloženými skripty.
- Vzdálené API – tato metoda umožňuje přizpůsobení simulátoru nebo simulace pomocí externí aplikace, která je umístěna na robotu nebo jiném stroji. Tato externí aplikace se připojuje k simulátoru V-REP pomocí příkazů vzdáleného rozhraní API. Díky tomu je možné, aby stejný kód obsažený v robotu, ovládal i jeho virtuální simulaci ve V-REPU.
- Uzel ROS – tato metoda přizpůsobuje simulaci pomocí externí aplikace (umístěné na robotu nebo na jiném zařízení), která se připojuje k simulátoru V-REP pomocí ROS (Robot Operating System) *(viz kapitola 4)*.
- Vlastní klient/server – simulace je přizpůsobena pomocí klient/server aplikace, která využívá buď skripty nebo pluginy pomocí vlastních komunikačních prostředků (sockets, pipes, atd.).[10]

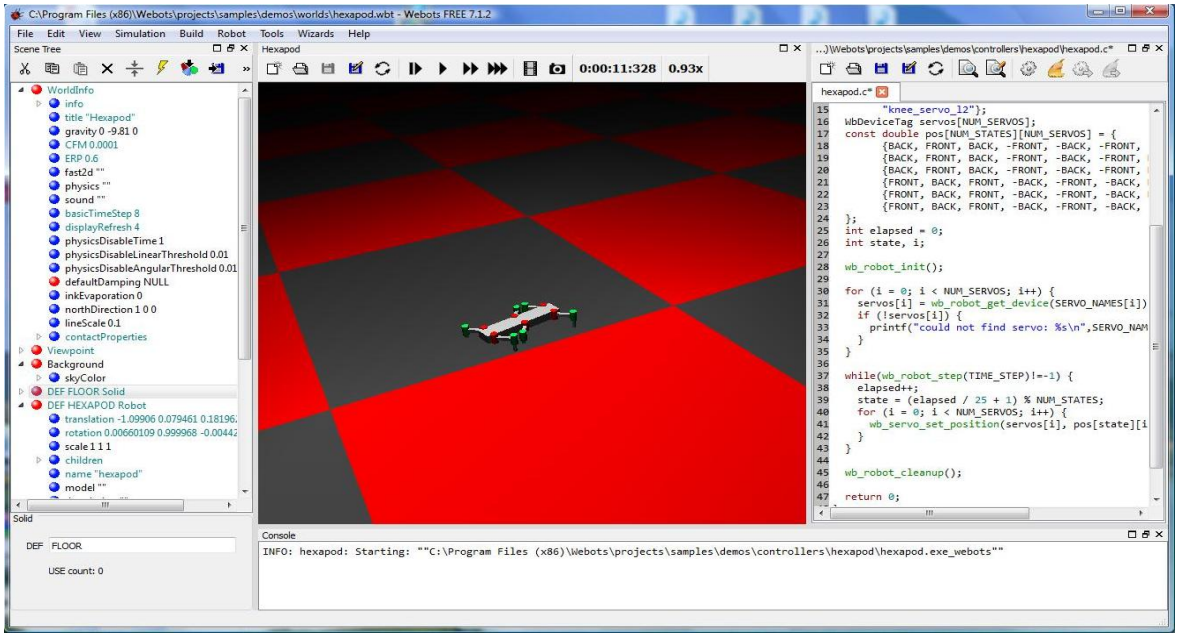


Obr. 10 V-REP framework.[10]

3.2 Webots

Webots je vývojové prostředí sloužící k modelování, programování a simulaci mobilních robotů. Tento robotický simulační software umožňuje uživateli vytvořit 3D virtuální světy s fyzikálními vlastnostmi jako jsou hmotnost nebo součinitel tření. Uživatel může navrhnout komplexní robotizované sestavy s jedním nebo více robotů ve sdíleném prostředí. Řídící programy mohou být naprogramovány pomocí integrovaného vývojového prostředí (v programovacích jazycích C, C++, Java, Python a další), nebo mohou být vytvořeny přes rozhraní ROS *(viz kapitola 4)*.[11]

Robotický simulátor Webots je používán více než 1.193 univerzitami a výzkumnými centry po celém světě. Je spustitelný na operačních systémech Windows, Mac OS, Linux.



Obr. 11 Prostředí robotického simulátoru Webots.[9]

Webots také obsahuje řadu rozhraní pro reálné roboty, takže když je simulace hotová mohou se řídící programy (bez jakýchkoliv modifikací) přenášet na skutečné roboty, jako jsou Darwin-OP nebo Nao.



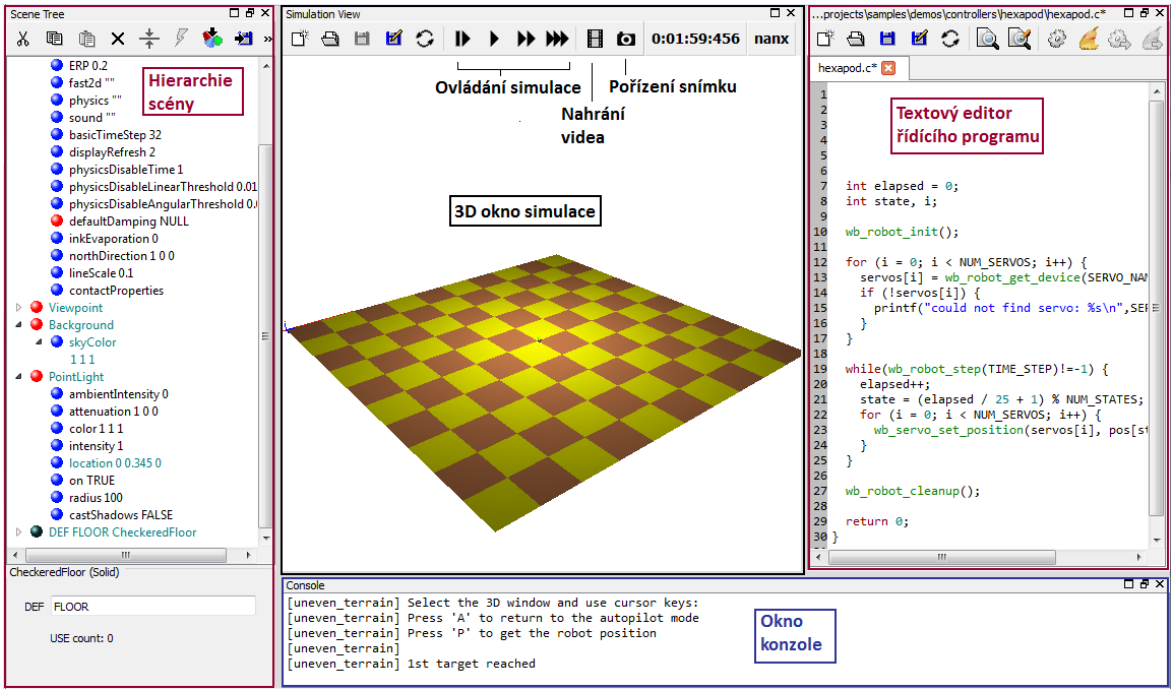
Obr. 12 Robot Nao a Darwin-OP.[12] [13]

3.2.1 Webots – popis simulátoru

Webots se používá pro výzkumné a vzdělávací projekty mobilních robotů. Mnoho těchto projektů spoléhá na Webots v těchto oblastech:

- vývoj prototypů mobilních robotů (akademický výzkum, automobilový průmysl, letectví, atd.),
- výzkum pohybu robotů (humanoidní, čtyřnozí),
- výzkum multi-agentního prostředí (inteligence roje, spolupráce skupiny mobilních robotů, atd.),
- výzkum adaptivního chování (genetický algoritmus, neuronové sítě, atd.),
- výuka robotiky (robotické přednášky),
- robotické soutěže.[14]

Webots GUI *(Obr. 13)* se skládá ze čtyř hlavních oken: *3D okno*, které zobrazuje a umožňuje interakci s 3D simulací, *hierarchie scény*, která zobrazuje objekty ve vytvořeném světě, *textový editor*, který umožňuje upravovat zdrojové kódy a *konzole*, která zobrazuje kompilační a řídící výstupy.



Obr. 13 GUI robotického simulátoru Webots.[9]

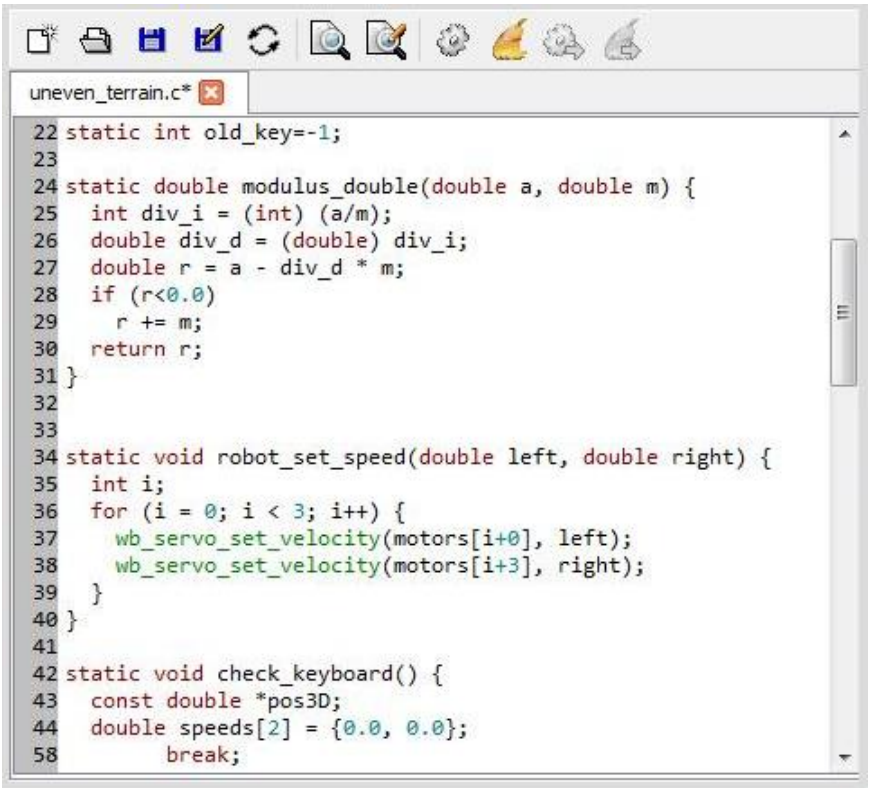
3.2.2 Webots – simulace

Webots simulace se skládá z následujících položek:

1. Webots World file (s příponou .wbt), tento soubor definuje jednoho nebo více robotů v jejich prostředí.
2. Jeden nebo více řídících programů (psané v C, C++, Java, Python, Matlab).
3. Fyzikální plugin, který upravuje fyzikální zákony ve Webots světě.[14]

Webots World file je 3D popis vlastností robotů a jejich prostředí. Obsahuje popis každého objektu, jako polohu, orientaci, geometrii, vzhled, fyzikální vlastnosti, atd. Tyto soubory jsou organizovány jako hierarchické struktury, kde objekty mohou obsahovat jiné objekty, např. robot obsahuje dvě kola, senzor vzdálenosti a tělo, které obsahuje kameru.[14]

Řídící program ovládá robota ve vymodelovaném světě. Může být napsán v některém z programovacích jazyků podporovaných simulátorem Webots. Po spuštění simulace, Webots spustí řídící program jako samostatný proces. Tento řídící program lze použít pro více robotů.[14]



Obr. 14 Náhled řídícího programu.[9]

3.3 Gazebo

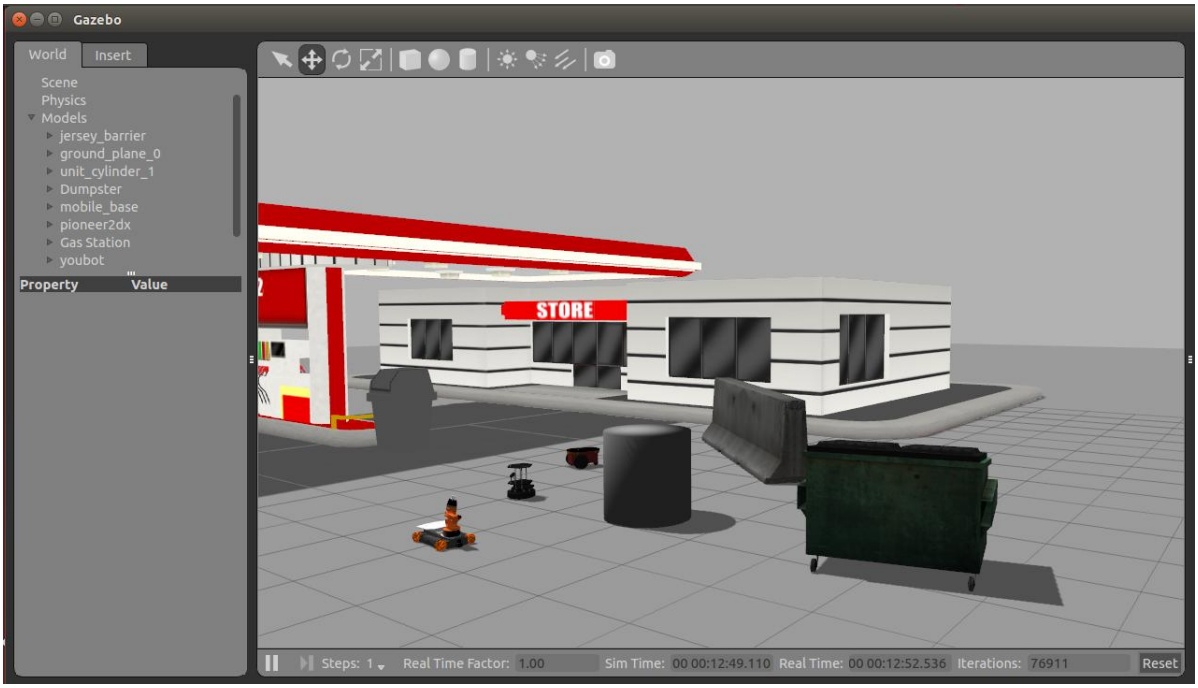
Vývoj robotického simulátoru Gazebo začal v roce 2002 na University of Southern California. Původními tvůrci byli Dr. Andrew Howard a jeho student Nate Koenig. Původně byl koncipován pro simulaci robotů ve venkovním prostředí za různých klimatických podmínek. Název „Gazebo“ (v překladu altán), byl vybrán, kvůli nejčastěji používané konstrukci ve venkovní simulaci. Tento název se zachoval i přesto, že v současné době většina uživatelů provádí své simulace ve vnitřním prostředí. Od roku 2009 se robotický simulátor Gazebo stal primárním nástrojem používaným v ROS komunitě *(viz kapitola 5.)*[15]

3.3.1 Gazebo – popis simulátoru

Robotický simulátor Gazebo umožňuje:

- rychlé testování algoritmů,
- vytvářet nové designy robotů,
- provádět regresivní testování pomocí realistického scénáře,
- simulovat populace robotů ve složitých vnitřních i venkovních prostředích,
- řešit nápady, návrhy nebo problémy s Gazabo „živou“ komunitou.[15]

Gazebo lze spustit na operačních systémech Windows, Mac OS, avšak hlavní podporu poskytuje operační systém Linux.



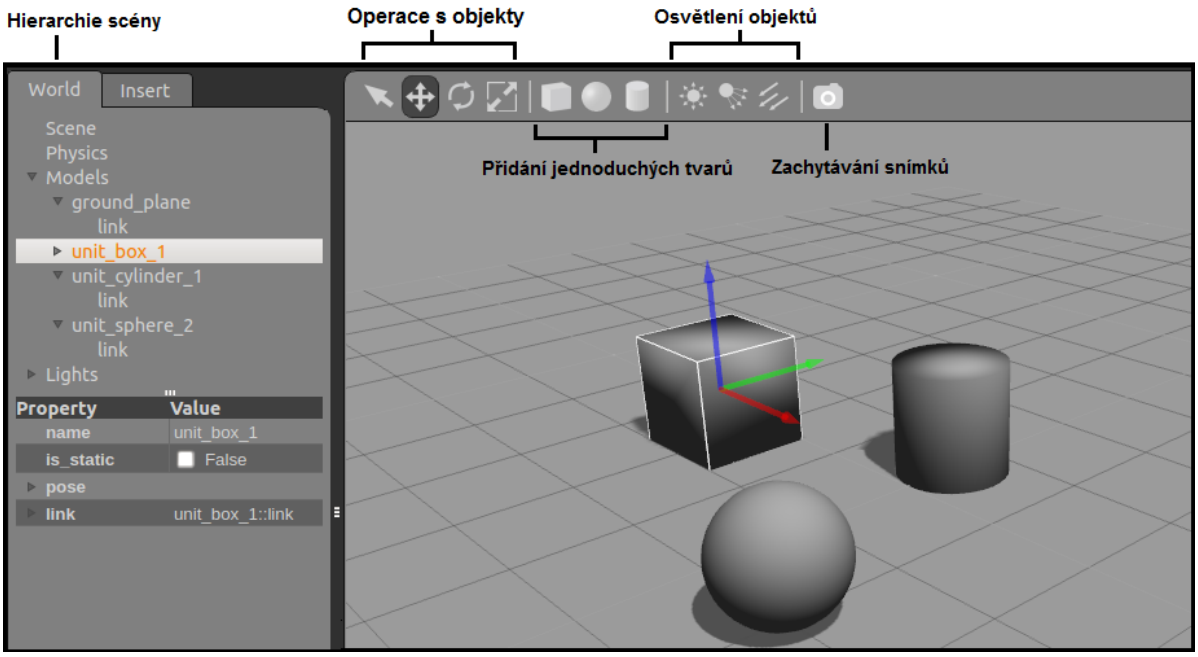
Obr. 15 Prostředí robotického simulátoru Gazebo.

Fyzikální zákony v Gazebu zajišťují čtyři vysoce výkonné enginy: ODE *(viz Obr. 3)*, Bullet *(viz Obr. 3)*, Simbody a DART.



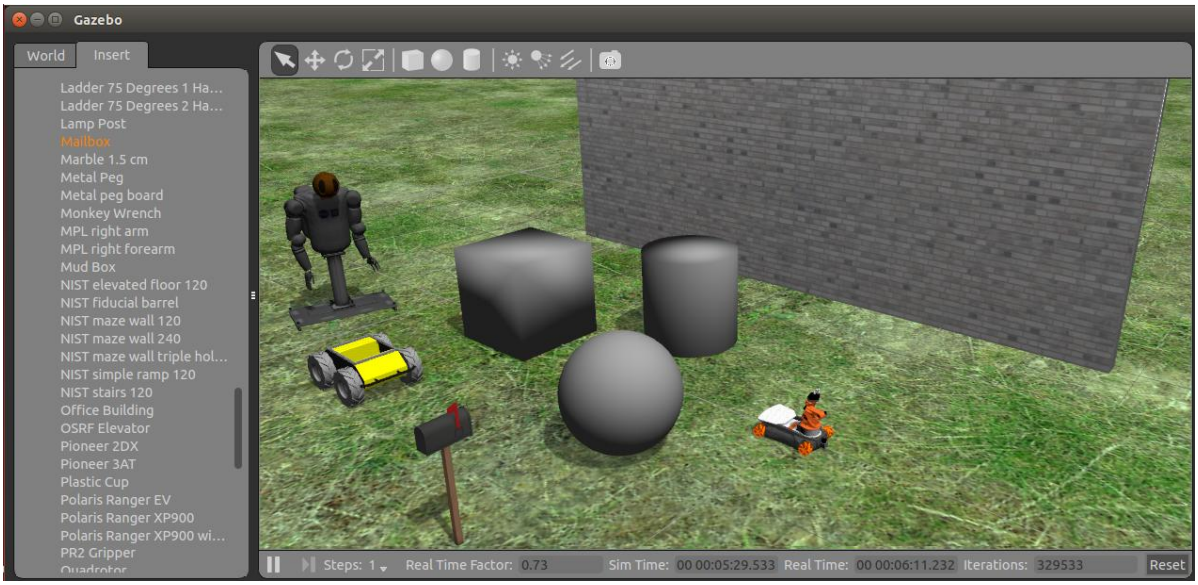
Obr. 16 Loga enginů Simbody a DART.[16][17]

Ovládání prostředí Gazeba, „zevnitř“ simulátoru, je velmi jednoduché. Do prostředí Gazeba se, pomocí nástrojové lišty, dají přidávat tři jednoduché tvary: krychle, válec, koule. Tato lišta zajišťuje i osvětlení objektů a jednoduché operace s objekty (posouvání, otáčení, přiblížení). V této liště se nachází i ikona pro zachycování snímků ze simulací.



Obr. 17 Gazebo – popis prostředí.

V levé části prostředí se nachází dvě záložky: *World* a *Insert*. Záložka *World* představuje hierarchii scény a obsahuje všechny objekty, které byly do vytvořeného světa vloženy. Záložka *Insert* nabízí databázi modelů a objektů, které lze do prostředí libovolně přidávat.



Obr. 18 Gazebo – přidání objektů/modelů z databáze.

3.3.2 Gazebo – simulace

V Gazebo simulaci jsou veškeré objekty nebo modely popsány pomocí skriptů formátovaných v SDF (Simulation Description Format).

SDF je XML formát, který popisuje objekty a prostředí robotických simulátorů. Popisuje jak stránku vizuální, tak i stránku řízení. Původně byl vyvinut jako součást robotického simulátoru Gazebo. V průběhu let se formát SDF stal stabilním, robustním a velmi rozšířeným formátem. Tento formát je schopný popsat všechny aspekty robotů a statických i dynamických objektů, je v něm popsáno i osvětlení, terén a fyzikální zákony.[18]



Obr. 19 Logo SDF (Simulation Description Format).[18]

Gazebo používá distribuovanou architekturu s oddělenými knihovnami pro simulaci fyziky, vykreslování uživatelského rozhraní, komunikaci a generování senzorů. Poskytuje dva spustitelné programy pro běh simulace:

- server `gzserver` pro simulaci fyzikálních zákonů, vykreslování a senzory
- klient `gzclient` , který poskytuje grafické rozhraní pro vizualizaci a interakci se simulací.

Klient a server komunikují pomocí komunikační knihovny Gazeba.[19]

Simulace se skládá z několika komponentů. Zde jsou jednotlivé položky podílející se na chodu simulace:

- World File (soubor světa) – obsahuje všechny prvky simulace, včetně robotů, světla, senzorů a statických objektů. Tento soubor je formátován pomocí SDF a typicky má příponu *.world* . Gazebo server (`gzserver`) tento soubor přečte a podle něho vygeneruje virtuální svět. Gazebo obsahuje množství příkladů vytvořených světů. Tyto světy jsou umístěny v `<install_path>/share/gazebo-<version>/worlds`.
- Model File (soubor modelu) – používá stejný formát SDF jako *World File*. Obsahuje pouze jeden skript typu `<model> ... </model>`. Účelem těchto souborů je usnadnit opakované používání modelu a zjednodušit *World File*. Když je *Model File* vytvořen, může být vložen do souboru světa pomocí následující SDF syntaxe:

```
<include>
<uri>model://model_file_name</uri>
</include>
```

- Environment Variables (proměnné prostředí) – Gazebo využívá řadu proměnných pro lokalizaci souborů a nastavení komunikace mezi serverem a klienty. Počátkem Gazeba 1.9.0, se nemusí nastavovat žádné proměnné. V gazebu jsou sestavené výchozí hodnoty, které fungují ve většině případů. Tyto výchozí hodnoty jsou zahrnuty v shell skriptu:

```
source <install_path>/share/gazebo/setup.sh
```

- Gazebo Server – tento server je hnací silou Gazeba. Analyzuje popis světa (*World File*) uvedený v příkazovém řádku a pak nasimuluje svět pomocí fyziky a senzorů motoru. Server neobsahuje žádnou grafiku a lze spustit pomocí následujícího příkazu:

```
gzserver <world_filename>
```

  Gazebo také nabízí již vymodelované světy, které se nachází v `<install_path>/share/gazebo-<version_number>/worlds`. Pro použití, například prázdného světa (empty.world), se použije následující příkaz:

```
gzserver worlds/empty.world
```

- Grafický Klient – grafický klient se připojí k běžícímu `gzserveru` a zajistí vizualizaci všech prvků. Tento nástroj umožňuje upravit běžící simulaci. Grafický klient lze spustit pomocí příkazu:

```
gzclient
```

- Server a Grafický Klient v jednom – příkaz `gazebo` spojí server a klienta do jednoho spustitelného souboru. Místo příkazu `gzserver worlds/empty.world` a následně `gzclient`, lze spustit jeden příkaz:

```
gazebo worlds/empty.world
```

- Plugins – poskytují jednoduchý a výhodný mechanismus pro spolupráci s Gazebem. Plugin má přímý přístup ke všem funkcím Gazeba prostřednictvím standartních tříd C++. Může být spuštěn buď z příkazového řádku, nebo může být vložen do souboru světa nebo modelu. První je načten plugin spuštěný z příkazového řádku a poté jsou načteny pluginy ze souborů světů/modelů.[20]

  Načtení pluginu z příkazového řádku serverem:

```
gzserver -s <plugin_filename> <world_file>
```

  Načtení pluginu z příkazového řádku klientem:

```
gzclient -g <plugin_filename>
```

Načtení pluginu ze souboru světa/modelu:

```
<?xml version="1.0"?>
<sdf version="1.4">
  <world name="default">
     <plugin name="hello_world" filename="libhello_world.so"/>
  </world>
</sdf>
```

4 ROS

ROS (The Robot Operating System) je flexibilní framework pro vytváření robotických aplikací. Je to sbírka nástrojů, knihoven a konvekcí, které mají za cíl zjednodušit úkol vytváření komplexních a robustních chování robotů přes širokou škálu robotických platforem.[21]

ROS byl od základu stavěn na podporu rozvoje spolupráce v robotických softwarech. Například, jedno robotické středisko může mít odborníky na mapování vnitřního prostředí, a tak může přispět k prvotřídnímu systému pro výrobu map. Další robotické středisko může mít odborníky na využití map pro navigaci a jiné by mohlo zajistit přístup počítačového vidění. ROS byl navržen speciálně pro tyto skupiny, aby spolupracovali a využívali své práce navzájem.[21]



Obr. 20 Logo projektu ROS.[21]

ROS je Open Source software poskytující služby, které se dají očekávat od operačního systému, včetně hardware abstrakce, low-level ovládání zařízení, realizace běžně používaných funkcí, předávání zpráv mezi procesy a správu balíků. Poskytuje nástroje a knihovny pro získání, vytváření, psaní a spouštění kódů na více počítačích. ROS je v některých ohledech podobný robotickým frameworkům jako jsou Player, YARP, Orocos, Carmen, Orca, Moos a Microsoft Robotics studio.[22]



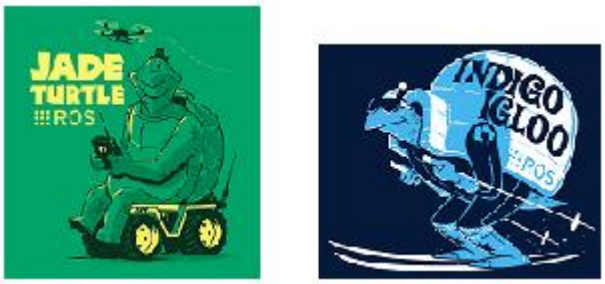
Obr. 21 Loga robotických frameworků.[23][24][25][26][27][28][29]

ROS v současné době běží pouze na platformách typu Unix. Primárně je testován na Ubuntu a Mac OS X. Použití ROS pro Microsoft Windows je možné, ale tato varianta je ve fázi výzkumu.

4.1 Historie ROS

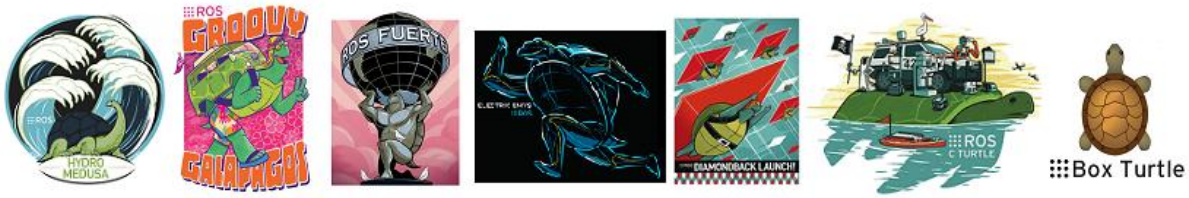
ROS začal vznikat v roce 2007 jako univerzitní projekt na prestižní americké univerzitě Stanford. V roce 2008, tento projekt, převzal robotický výzkumný inkubátor Willow Garage. Od počátku vývoje ROS vzniklo už celkem devět verzí:

- současné distribuované verze – ROS Jade Turtle (nejnovější, zveřejněno 23.5.2015), ROS Indigo Igloo



Obr. 22 ROS JADE a ROS Indigo.[30]

- ukončené distribuované verze – ROS Hydro Medusa, ROS Groovy Galapagos, ROS Fuerte Turtle, ROS Electric Emys, ROS Diamondback, ROS C Turtle, ROS Box Turtle (nejstarší, zveřejněno 2.3.2010)



Obr. 23 Ukončené distribuované verze ROS.[30]

ROS byl vyvinut na více institucích pro více robotů. I když by bylo pro všechny přispěvatele mnohem jednodušší ukládat svůj kód na stejných serverech, v průběhu let se „federovaný model“ ukázal jako jeden z nejsilnějších aspektů ROS ekosystému. Jakákoliv skupina může vytvořit své vlastní ROS úložiště na vlastních serverech a nad tímto úložištěm mít plnou kontrolu. Když dají jejich ROS úložiště veřejně k dispozici, mohou těžit ze specifické zpětné vazby a vylepšení jejich kódů. ROS ekosystém se nyní skládá z desítky tisíc uživatelů po celém světě, kteří pracují v různých oblastech, od stolních hobby projektů po velké průmyslové automatizační projekty.[31]



Obr. 24 ROS ekosystém.[32]

4.2 Cíle ROS

Cílem ROS není být frameworkem s nejvíce funkcemi. Primárním cílem ROS je podpořit opětovné použití kódů ve výzkumu a vývoji robotiky. ROS je distribuovaný framework procesů (uzlů), který umožňuje, z individuálně navržených souborů, udělat spustitelné soubory a různě je spojovat za běhu. Tyto procesy mohou být seskupeny do „balíčků“ a „stohů“, které lze snadno sdílet a distribuovat. ROS také podporuje federovaný systém kódových úložišť, které umožňují spolupráci s ostatními uživateli.[22]

Na podporu primárního cíle sdílení a spolupráce, existuje několik dalších cílů frameworku ROS:

- „Tenkost“ – ROS je navržen tak, aby byl co „nejtenčí“, nezabaluje se *main()*, tzn., že kód psaný pro ROS lze použít i pro ostatní robotické frameworky.
- ROS-agnostik (zpochybňované) knihovny – výhodným vývojovým modelem je napsat ROS-agnostik knihovnu s čistým funkčním rozhraním.
- Jazyková nezávislost – framework ROS je snadno implementovatelný v jakémkoliv moderním programovacím jazyce. ROS je implementován v jazyce Python, C++, Lisp a obsahuje experimentální knihovny v Javě a Lua.
- Snadné testování – ROS má vestavěný test framework zvaný Rostest, ten usnadňuje vyvolání a ukončení příslušného testu.
- Měřítko – ROS je vhodný pro velké runtime systémy a pro velké vývojové procesy.[22]

4.3 ROS koncept

ROS obsahuje tři úrovně konceptů: Úroveň souborového systému, Úroveň výpočetního grafu, Úroveň ROS komunity.

4.3.1 Úroveň souborového systému

Koncepty na této úrovni pokrývají především ROS prostředky, na které narazíte na disku, jako například:

- Balíčky – Balíčky jsou hlavní jednotkou pro organizaci softwaru v ROS. Balíček může obsahovat ROS runtime procesy (uzly), ROS závislé knihovny, datové soubory, konfigurační soubory, nebo cokoliv jiného, co je smysluplně organizováno. Balíčky jsou základní položky pro sestavení a uvolnění položek v ROS.
- Metabalíčky – Metabalíčky jsou specializované balíčky, které pouze reprezentují skupinu propojených balíčků.
- Balíčky Manifesty – Manifesty (Package.xml) poskytují metadata o balíčku, včetně jeho názvu, verzi, popisu, licenčních informací, závislostí a dalších meta informací.
- Typy zpráv – popisy zpráv, uložené `my_package/msg/MyMessageType.msg`, definují datové struktury odeslaných zpráv v ROS.
- Typy služeb (srv) – popisy služeb, které jsou uložené v `my_package/srv/MyServiceType.srv`, definují žádosti a odezvy datových struktur v ROS.[33]

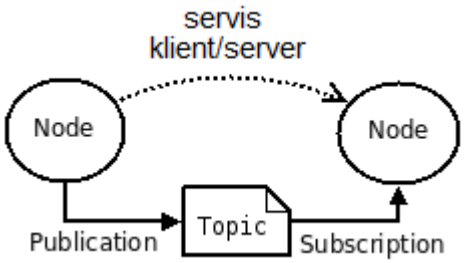
4.3.2 Úroveň výpočetního grafu

Výpočetní graf je síť typu peer-to-peer procesů, které zpracovávají data společně. Základní koncept grafu jsou ROS uzly, master, parametry serveru, zprávy, témata, služby a bagy. Všechny tyto položky poskytují různými způsoby data do grafu.

- Uzly – Uzly jsou procesy, které provádějí výpočty. ROS je navržen tak, že má stavebnicovou strukturu. Řídící systém obvykle obsahuje mnoho uzlů. Například, jeden uzel kontroluje laserový senzor vzdálenosti, jeden uzel ovládá motory kola, další uzel provádí lokalizaci, jiný uzel provádí plánování cesty a jeden uzel poskytuje grafické zobrazení systému. Uzel ROS je psán s použitím knihovny klienta ROS, jako je *roscpp* nebo *rospy*.
- Master – ROS Master poskytuje registraci názvů a vyhledávání ve výpočetním grafu. Bez Master by se uzly nebyly schopny navzájem najít, vyměňovat si zprávy nebo vyvolávat služby.
- Parametr Serveru – V současné době je součástí Master. Umožňuje, aby data byla uložena pod klíčem v centrálním umístění.
- Zprávy – Uzly spolu vzájemně komunikují pomocí předáváním zpráv. Zpráva je jednoduchá datová struktura obsahující zadané pole. Obsahují standardní primitivní typy (integer s pohyblivou desetinou čárkou, boolean, atd.).
- Témata – Zprávy jsou směrovány přes transportní systém se sémantikou publikace/odběr. Uzel vysílá zprávu publikovanou na dané téma. Téma je název, který se používá k identifikaci obsahu zprávy. Uzel, který má zájem o určitý druh dat, popíše příslušné téma. Může být více současných vydavatelů/odběratelů na jedno téma a jeden uzel může zveřejnit, nebo se přihlásit k odběru, více témat.
- Služby – Model publikování/odběr je velmi flexibilní komunikační paradigma, ale jeho struktura many-to-many (mnohé-mnohým) a jednosměrný transport nejsou vhodné pro interakci požadavek/odpověď, která je v distribuovaném systému často potřebná. Požadavek/odpověď se provádí prostřednictvím služby, která je definována dvojicí zpráv, první se strukturou požadavku a druhá se strukturou odpovědi.

Poskytující uzel nabízí služby pod jménem a klient používá službu odesláním zprávy požadavku a čeká na odpověď.

- Bag – Bag je formát pro ukládání a přehrávání dat ROS zpráv. Bagy jsou důležitým mechanismem pro ukládání dat, například ze snímačů, které může být obtížné získat, ale jsou nezbytné pro vývoj a testování algoritmů.[33]



Obr. 25 Komunikační model ROS.[33]

4.3.3 Úroveň ROS komunity

Koncepty na této úrovni zahrnují zdroje, které umožňují odděleným komunitám výměnu softwaru a znalostí. Tyto zdroje zahrnují:

- Distribuce – ROS distribuce jsou kolekce různých verzí „stohů“ (stacks), připravené k instalaci.
- Repozitáře – ROS se opírá o federalizované sítě kódových úložišť, kde mohou různé instituce rozvíjet a uvolňovat své vlastní robotické softwarové komponenty.
- ROS Wiki – ROS komunita Wiki, dostupná na *http://wiki.ros.org/*, je hlavním fórem pro dokumentování informací o ROS. Každý si může vytvořit účet, přes který lze přispět svou vlastní dokumentací, poskytnout opravy nebo aktualizace, psát výukové programy a další.
- Adresní seznamy – ROS seznam adresátů je primární komunikační kanál o nových změnách ROS, stejně jako fórum *ask questions* týkající se softwaru ROS.
- ROS odpovědi – ROS Answers, dostupné na *http://answers.ros.org/questions/*, je místo pro odpovědi na dotazy související s ROS.
- ROS Blog – dostupný na *http://ros.org*, poskytuje pravidelné aktuality, včetně fotografií a videí.[33]

4.4 Build system

Build systém (sestavení systému) je zodpovědný za generování „cíle“ ze surového zdrojového kódu, který může být používán koncovým uživatelem. Tyto cíle mohou být ve formě knihoven, spustitelných programů, generovaných skriptů, exportovaných rozhraní (např. C++ hlavičkové soubory), nebo cokoli jiného, co není statický kód. V ROS je zdrojový kód organizován do balíčků, kde každý balíček obvykle obsahuje jeden nebo více sestavených cílů.[34]

K vygenerování „cílů“, build systém potřebuje informace, například o umístění nástrojů komponent řetězce (např. C++ Compiler), umístění zdrojového kódu, závislosti

kódu, externí závislosti, dále informace, kde se tyto závislosti nacházejí, které cíle by měly být sestaveny, kde by měly být sestaveny, atd. V integrovaném vývojovém prostředí jsou tyto informace obvykle uloženy jako část pracovní plochy/projektu, jako meta-informace v souboru, který se nazývá *CMakeLists.txt.*[34]

4.4.1 Build system Catkin

Catkin je oficiální buil systém ROS a je nástupcem původního build systému *rosbuild*. Catkin kombinuje cmake makra a Python skripty. Catkin byl navržen tak, aby byl více konvenční než rosbuild, což umožňuje lepší distribuci balíčků, lepší cross-kompilační podporu a lepší přenositelnost. Catkinův pracovní postup je velmi podobný *cmake,* ale přidává podporu automatického „hledání balíčku".[34]

Ačkoli *rosbuild* sloužil k sestavení systému ROS od počátku, rychlý růst ROS kódové základny a dlouholeté zkušenosti odhalily některé nevýhody jeho přístupu. Catkin se má pokusit o zmírnění těchto nevýhod.[34]

Jedním z největších problémů *rosbuild* je jeho složitost přenosu na všechny operační systémy, zejména na Microsoft Windows. Je to tím, že *rosbuild* využívá pro sestavení kódu směs Bash skriptů, GNU make a cmake.[34]

5 GAZEBO S VYUŽITÍM ROS

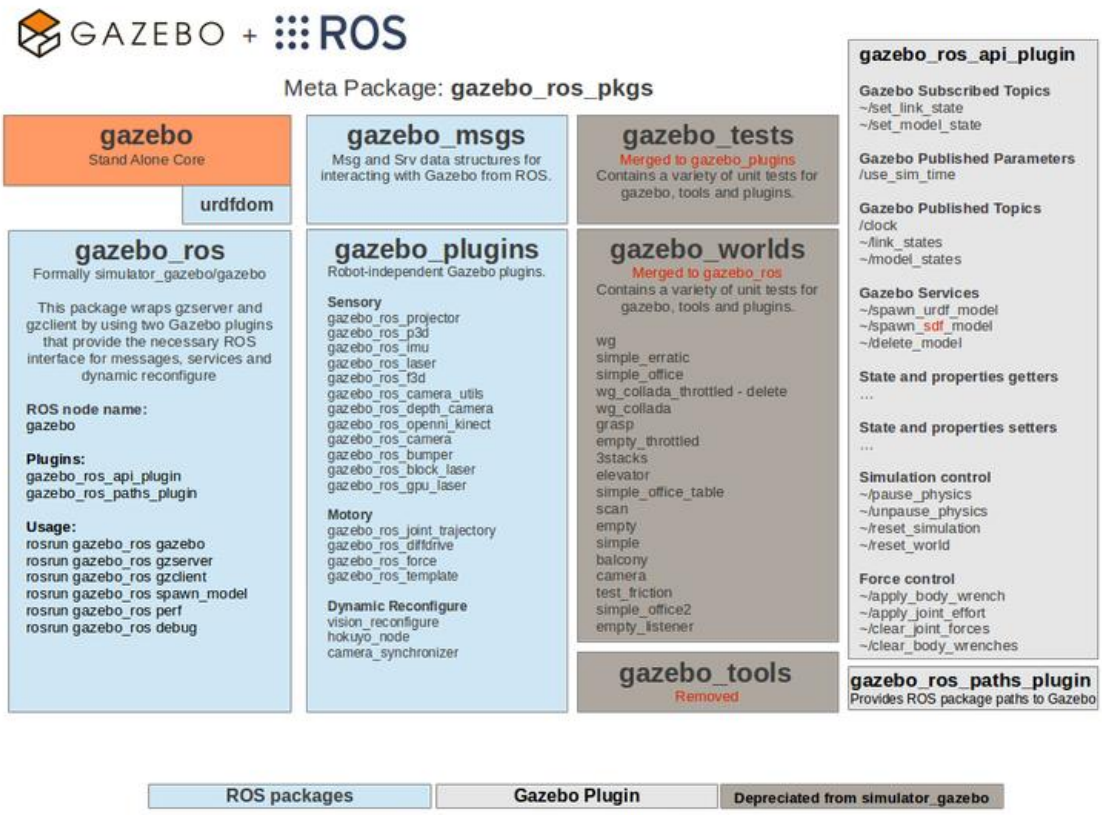
Tato kapitola se bude zabývat propojením robotického simulátoru Gazebo s frameworkem ROS.

Od verze Gazeba 1.9 a verze ROS Hydro, nemá Gazebo přímé závislosti na ROS a může tedy být nainstalován jako samostatný balík Ubuntu. V minulosti byla, pro použití Gazeba s ROS, nutná specifická verze Gazeba sestavená s odkazem na stoh (stack) „simulator_gazebo“.[35]

K dosažení ROS integrace se samostatným Gazebem byla vytvořena nová sada ROS s názvem `gazebo_ros_pkgs`, aby poskytla „obal“ samostatnému Gazebu. Tato sada poskytuje potřebné rozhraní pro simulaci robota v Gazebu pomocí ROS zpráv, služeb a dynamické rekonfigurace. Výhody „gazeboros_pkgs“ oproti „simulator_gazebo“ jsou:

- Podpora Gazeba jako samostatného systému. Gazebo nemá vlastní vazby na ROS.
- Sestavení systému pomocí *catkin.*
- Považuje URDF (Universal Robotic Description Format) a SDF jako rovnocenné.
- Snižuje duplicitu kódu s Gazebem.
- Zlepšení podpory boxu pro řízení s využití ros_control.
- Odstraní starý kód z předchozích verzí ROS a Gazebo.[35]

Přehled nového rozhraní je zobrazen na následujícím obrázku:



Obr. 26 Rozhraní Gazebo + ROS.[35]

5.1 Instalace Gazeba

Instalace Gazeba na operační systém Ubuntu 14.04 LTS se dá provést dvěma způsoby. První způsob je tzv. „One-line“ instalace a druhý způsob je instalace „Step-by-step“:

- One-line instalace – do terminálu se vloží následující příkaz pro instalaci:

```
wget -O /tmp/gazebo5_install.sh
http://osrf-distributions.s3.amazonaws.com/gazebo/gazebo5_install.sh; sudo sh
/tmp/gazebo5_install.sh
```

- Step-by-step instalace:
    1. Nastavení počítače pro akceptování softwaru z *packages.osrfoundation.org*:

```
sudo sh -c 'echo "deb http://packages.osrfoundation.org/gazebo/ubuntu
`lsb_release -cs` main" > /etc/apt/sources.list.d/gazebo-latest.list'
```

    2. Nastavení klíče:

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

    3. Instalace Gazeba:

```
sudo apt-get update
sudo apt-get install gazebo5
```

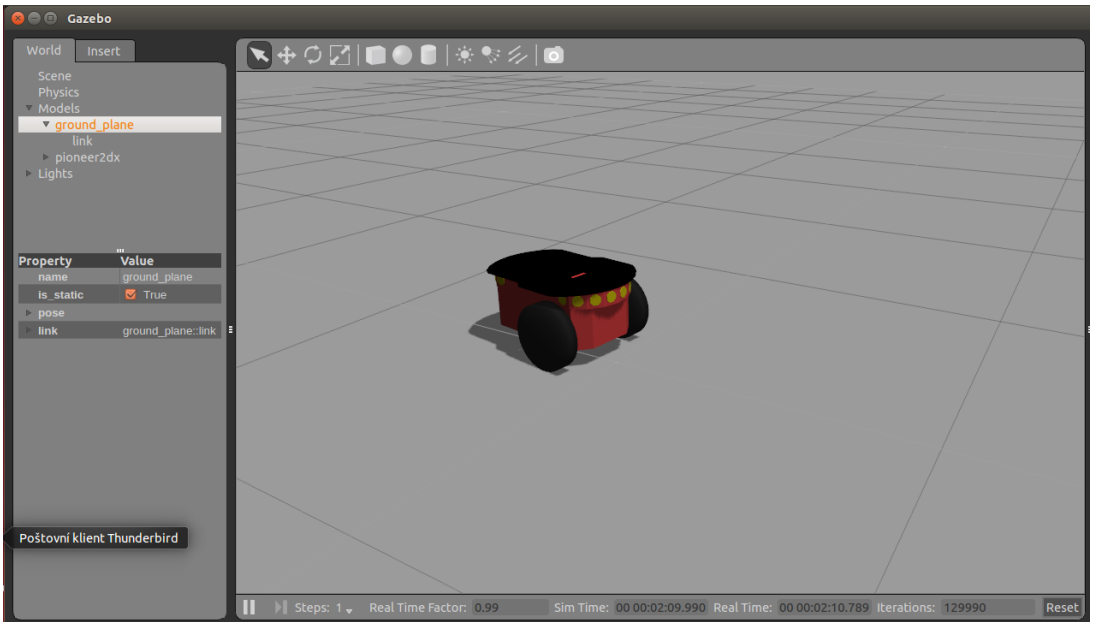
Úspěšnou instalaci Gazeba ověříme jeho spuštěním. Gazebo zkusíme spustit s předdefinovaným světem z databáze modelů (např. *pioneer2dx.world*). Do terminálu se vloží následující příkaz:

```
gazebo worlds/pioneer2dx.world
```



Obr. 27 Logo operačního systému Ubuntu 14.04 LTS.[36]

Pokud byla instalace úspěšná, měl by se spustit simulátor Gazebo s předdefinovaným světem, ve kterém je vložen model robota *pioneer2dx* jako na následujícím obrázku:



Obr. 28 Spuštěný simulátor Gazebo s modelem robota Pioneer2dx.

5.2 Instalace ROS

Instalace ROS na Ubuntu 14.04 LTS:

1. Nastavení *sources.list* – nastavení počítače pro přijetí softwaru z *package.ros.org.* se provede příkazem v terminálu:

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

2. Nastavení klíče:

```
sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net --
recv-key 0xB01FA116
```

3. Otestování aktuální verze Debian balíčků:

```
sudo apt-get update
```

4. Instalace – „Desktop-Full Install“, tato instalace obsahuje ROS, nástroje rqt a rviz, generické robotické knihovny, 2D/3D simulátory, navigace a 2D/3D vnímání:

```
sudo apt-get install ros-<verze_ros>-desktop-full
```

V tomto případě použijeme verzi ROS-Indigo:

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

5. Inicializace *rosdep* – před prvním použitím ROS, je nutné inicializovat *rosdep*, který umožňuje snadnou instalaci systémových závislostí ze zdroje, který chceme kompilovat, a spuštění některých základních komponent ROS:

```
sudo rosdep init
rosdep update
```

6. Nastavení prostředí – je vhodné, aby při každém novém spuštění shell příkazu, byly automaticky přidány proměnné prostředí ROS do bash relace.

```
echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

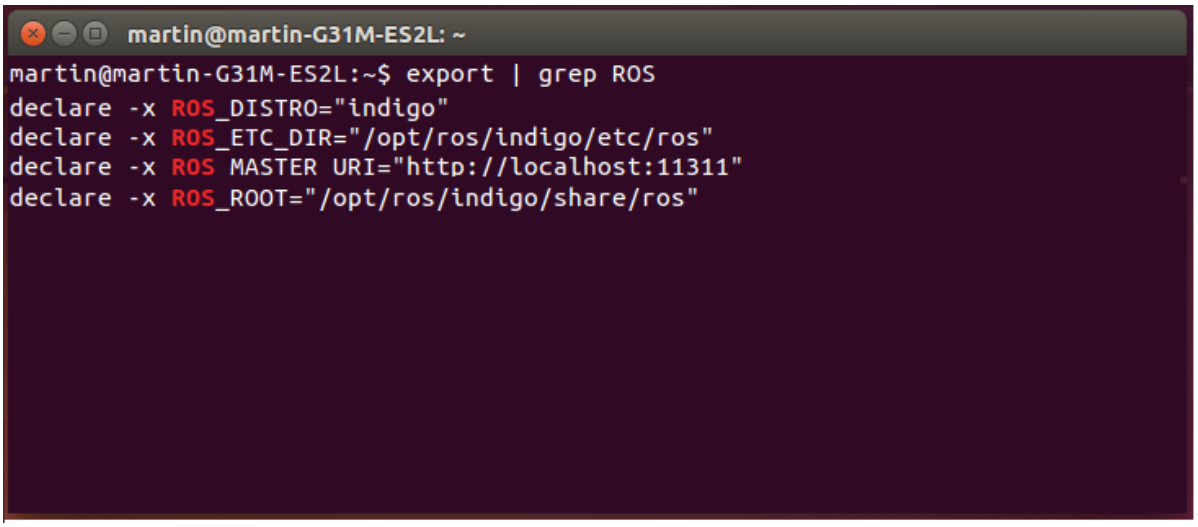
7. Nástroj *rosinstall* – je často používaný nástroj příkazového řádku v ROS, který je distribuován samostatně. Umožňuje snadné stažení zdrojových „stromů“ balíčků ROS.[37]
   Instalace nástroje *rosinstall*:

```
sudo apt-get install python-rosinstall
```

Po spuštění příkazu v terminálu:

```
export | grep ROS
```

by se měli zobrazit informace a nastavené cesty k jednotlivým komponentům. Např.:



Obr. 29 Terminál zobrazující výsledek po použití předchozího příkazu.

5.3 Propojení Gazeba s ROS

Po úspěšné instalaci robotického simulátoru Gazeba a frameworku ROS, nám zbývá jejich propojení pomocí instalačních balíčků *gazebo_ros_pkgs.*

Nejdříve otestujeme, zda máme správnou verzi Gazeba a zda je tato verze nainstalována na správném místě. Po zadání příkazů v terminálu:

```
which gzserver
which gzclient
```

by se nám mělo zobrazit následující umístění:

```
/usr/bin/gzserver
/usr/bin/gzclient
```

Po otestování Gazeba můžeme nainstalovat *gazebo_ros_pkgs* v tomto případě pro verzi ROS-Indigo:

```
sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control
```

5.3.1 Otestování Gazeba s integrací ROS

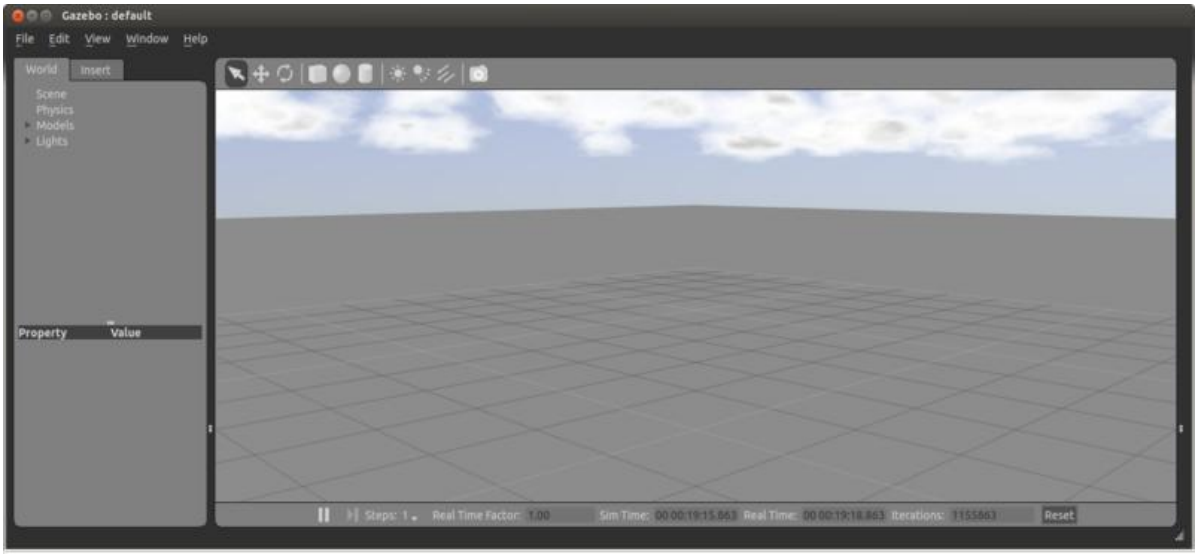
V první řadě se ujistíme, zda zdroj odpovídající instalačnímu souboru ROS, vypadá pro Ros-Indigo následovně:

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

Za předpokladu, že framework ROS a prostředí Gazebo byly správně nastaveny a sestaveny, můžeme spustit Gazebo jednoduchým příkazem *rosrun*, poté co spustíme *roscore:*

```
roscore &
rosrun gazebo_ros gazebo
```

Výsledkem by mělo být spuštění prázdného grafického rozhraní Gazeba, které je ukázáno na následujícím obrázku (*Obr. 30*).



Obr. 30 Prázdné GUI simulátoru Gazebo.

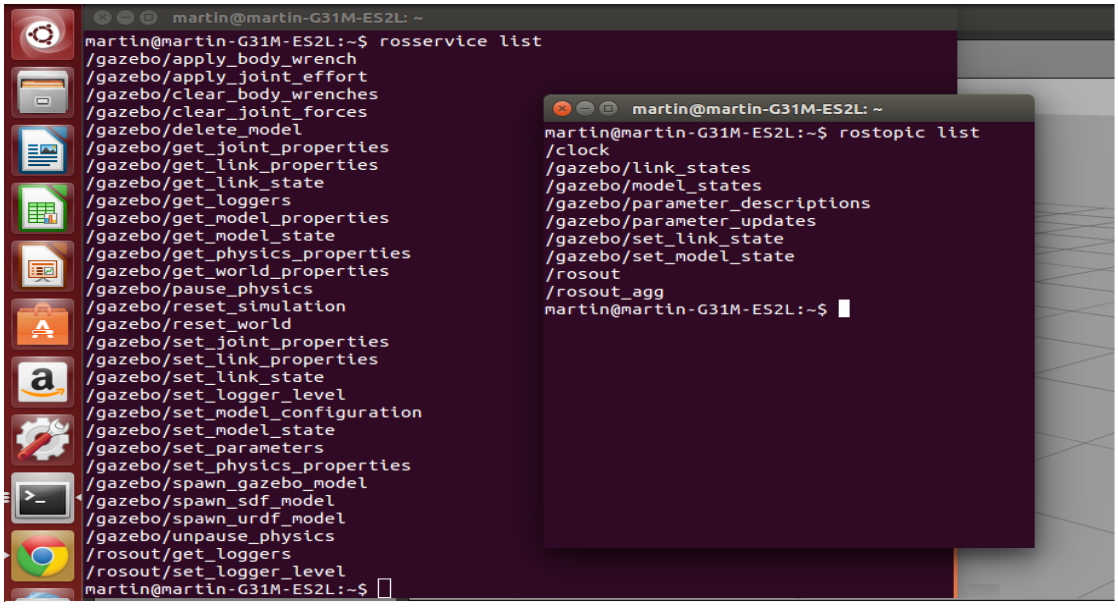
Pro další ověření správného připojení ROS je možné zobrazit témata, které jsou k dispozici (*viz Obr. 31*), příkazem:

```
rostopic list
```

a také služby, které v Gazebu existují (*viz Obr. 31*), příkazem:

```
rosservice list
```

Po provedení těchto příkazů, by se v terminálu měli zobrazit příslušné seznamy, jako jsou uvedeny na následujícím obrázku.



Obr. 31 Vypsání příslušných seznamů rostopic a rosservice.

5.3.2 Jiné cesty ke spuštění Gazeba s ROS

Pro spuštění Gazeba existuje několik příkazů *rosrun*:

- Spuštění serveru i klienta dohromady:

```
rosrun gazebo_ros gazebo
```

- Spuštění samostatného serveru Gazeba:

```
rosrun gazebo_ros gzserver
```

- Spuštění samostatného klienta Gazeba:

```
rosrun gazebo_ros gzclient
```

- Spuštění pouze Gazebo serveru, v režimu ladění:

```
rosrun gazebo_ros debug
```

- Spuštění Gazeba pomocí *roslaunch*:

```
roslaunch gazebo_ros empty_world.launch
```

5.3.3 Roslaunch

Existuje mnoho způsobů, jak spustit Gazebo, načíst modely světa, nebo modely robotů do simulovaného prostředí. Tato kapitola se bude zabývat spouštěním/načítáním těchto modelů do Gazeba prostřednistvím *roslaunch*. To zahrnuje ukládání URDF souborů v ROS balíčcích a je nutné udržovat aktuální cesty různých zdrojů vzhledem k pracovnímu prostoru.[38]

Roslaunch je standardní metoda pro spuštění ROS uzlů a chování robotů v ROS. Pro spuštění prázdného Gazebo světa stačí spustit příkaz, po kterém by se mělo spustit prázdné GUI Gazeba (*viz Obr. 30*):

```
roslaunch gazebo_ros empty_world.launch
```

K příkazu *roslaunch* můžeme připojit argumenty, které ovlivní chování simulátoru:

- Pauza – spustí Gazebo v pozastaveném stavu (výchozí nastavení).
- Použití simulovaného času (use_sim_time) – tento příkaz říká Gazebu, aby publikoval simulační čas. Publikace probíhá přes ROS témata/hodiny.
- GUI – spustí okno grafického uživatelského rozhraní Gazeba.
- „Headless“ – zakáže všechny funkce, které volají různé komponenty do simulátoru.
- Debug – spustí gzserver (Gazebo server) v režimu ladění.[38]

Příklad příkazu *roslaunch* s připojenými argumenty:

```
roslaunch gazebo_ros empty_world.launch paused:=true
use_sim_time:=false gui:=true throttled:=false headless:=false
debug:=true
```

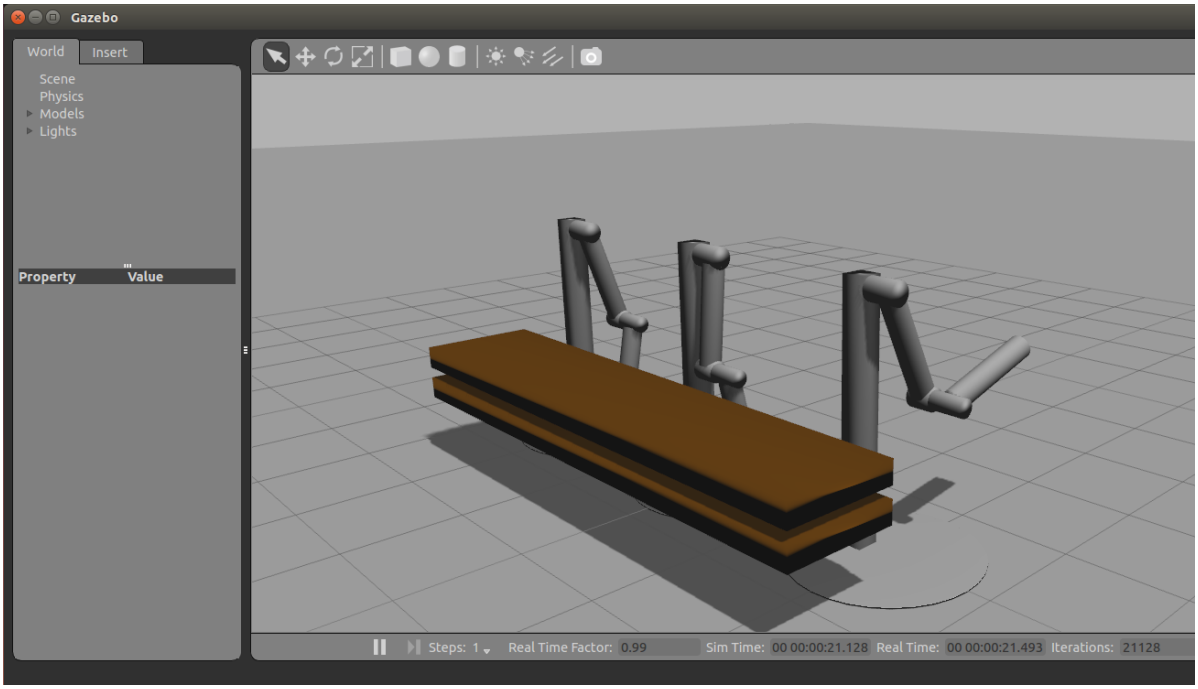
V balíčku *gaezbo_ros_package* jsou již zahrnuta spustitelná dema světů, které se dají spustit pomocí *roslaunch.* Spouštění launch souborů obsažených v balíčcích probíhá pomocí následné syntaxe příkazu:

```
roslaunch <jméno-balíčku> <jméno-launch-souboru>
```

Příklady příkazů pro spuštění vytvořených Gazebo demo světů:

```
roslaunch gazebo_ros willowgarage_world.launch
roslaunch gazebo_ros mud_world.launch
roslaunch gazebo_ros shapes_world.launch
roslaunch gazebo_ros rubble_world.launch
```

Například demo *mud_world.launch* spustí Gazebo s jednoduchým kloubovým mechanismem.



Obr. 32 Spuštění Gazebo z mud_world.launch.

Spouštěcí soubor mud_world.launch má následující strukturu:

```
<launch>

  <!– empty_world.launch spouští svět s názvem mud.world  -->

  <include file="$(find gazebo_ros)/launch/empty_world.launch">

    <arg name="world_name" value="worlds/mud.world"/> <!-- Poznámka:
world_name musí respektovat cestu - GAZEBO_RESOURCE_PATH proměnných prostředí
-->

    <arg name="paused" value="false"/>

    <arg name="use_sim_time" value="true"/>

    <arg name="gui" value="true"/>

    <arg name="headless" value="false"/>

    <arg name="debug" value="false"/>

  </include>

</launch>
```

V tomto spouštěcím souboru se většina funkcí dědí z *empty_world.launch*. Jediným parametrem, který se musí změnit, je parametr *word_name*, ten nahrazuje původní *empty.world* za *mud.world*. Ostatní argumenty jsou jednoduše nastaveny na výchozí hodnoty.

Světový soubor *mud.world* obsahuje popis prostředí a zahrnuje modely, které jsou do prostředí vloženy. Zde je zobrazeno prvních pár komponent, které *mud.world* obsahuje:

```
  <sdf version="1.4">
    <world name="default">
      <include>
        <uri>model://sun</uri>
      </include>
      <include>
        <uri>model://ground_plane</uri>
      </include>
      <include>
        <uri>model://double_pendulum_with_base</uri>
        <name>pendulum_thick_mud</name>
        <pose>-2.0 0 0 0 0 0</pose>
      </include>
      ...
    </world>
  </sdf>
```

Světové soubory (world file) se nacházejí ve složce */worlds*. Na tuto složku odkazuje cesta Gazebo zdrojů (Gazebo resource path). Umístění této složky závisí na tom, kam bylo Gazebo nainstalováno. Pro zjištění tohoto umístění slouží následující příkaz:[38]

```
env | grep GAZEBO_RESOURCE_PATH
```

6 Robotická úloha

Tato kapitola se bude zabývat výběrem, simulací a implementací robotické úlohy. Robotickou úlohou je myšlena některá z úloh, která se plní na robotických soutěžích, jako například Line Following, Ketchup House, Bear Rescue Advanced, RoboCarts, atd.

Vybraná robotická úloha bude nejdříve provedena v robotickém simulátoru Gazebo s využitím frameworku ROS. Poté bude řídící mechanismus, vytvořený v této simulaci, implementován, přes Framework ROS, na reálného robota.

6.1 Line Following

V této práci se budeme zabývat robotickou úlohou Line Following (sledování čáry/cesty), která spočívá v tom, že vybraný robot bude sledovat předdefinovanou trasu určenou vodící čárou. Tato robotická úloha se pravidelně objevuje na různých robotických soutěžích. Hodnotí se rychlost detekce vodící čáry, schopnost sledovat různě tvarovanou cestu (pomocí vodící čáry), plynulost a rychlost pohybu po čáře.

Pro tuto úlohu se využívá mnoho modelů robotů, ať už jsou to roboty „stavebnicové“, například LEGO Mindstorms NXT, ze kterých lze postavit spoustu typů robotů (*viz Obr. 33*), nebo modely robotů jako je Turtlebot (*viz Obr. 34*), jejichž vzhled závisí na různých soupravách/kitech.



Obr. 33 Možné typy robotů sestavené z LEGO Mindstorms NXT.[39]

6.2 Simulace robotické úlohy

Pro simulaci vybrané úlohy, musíme udělat několik věcí. V první řadě musíme vybrat robota, který bude pro tuto robotickou úlohu vhodný. Dále musíme vytvořit prostředí, ve kterém bude tato úloha probíhat, tzn. prostředí s vymodelovanou cestou (vodící čára), kterou bude robot následovat. V neposlední řadě musíme vytvořit a sestavit ROS balíčky, díky kterým bude simulace probíhat.

6.2.1 Model robota

Jako model robota vybereme robota *Turtlebot*. *Turtlebot* je nízkonákladový osobní robotický kit s open source softwarem. Byl vytvořen společností Willow Garage. *Turtlebot* souprava/kit se skládá z modilní základny Kobuki (používaná na autonomní vysavače), 3D senzoru, senzoru Kinect, laptopu a hardwarové sady.



Obr. 34 Turtlebot.[40]

Turtlebot je obsažen i v databázi modelů poskytované Gazebem. Jeho popis v SDF formátu vypadá následovně:

*model-1_4.sdf ×

```
<?xml version="1.0" ?>
<sdf version="1.4">
  <model name="turtlebot">
    <link name="rack">
      <inertial>
      <collision name="plate_0_collision">
        <pose>-0.043340 0 0.084757 0 0 0</pose>
        <geometry>
          <box>
            <size>0.233502 0.314845 0.006401</size>
          </box>
        </geometry>
      </collision>
      <visual name="plate_0_visual">
        <pose>-0.043340 0 0.084757 0 0 0</pose>
        <geometry>
          <mesh>
            <uri>model://turtlebot/meshes/plate_0_logo.dae</uri>
          </mesh>
        </geometry>
      </visual>
```

Obr. 35 Náhled SDF formátu Turtlebota.

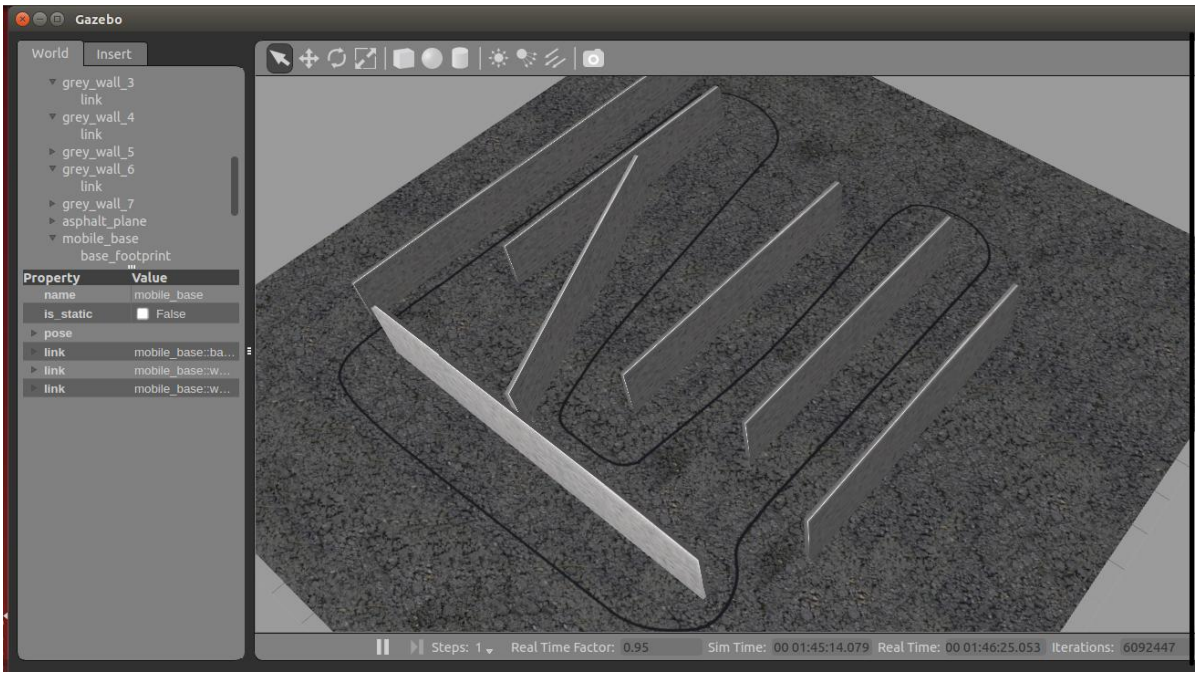
6.2.2 Prostředí – svět

Dostáváme se k vymodelování prostředí/světa, ve kterém se bude Turtlebot pohybovat. Toto vymodelování může probíhat dvěma způsoby.

Prvním způsobem je vymodelování prostředí přímo v simulátoru Gazebo. Tento způsob spočívá ve vytváření objektů, nebo vkládání objektů z databáze Gazeba. Po uložení takto vymodelovaného světa, se vytvoří soubor s příponou *.world*, který bude obsahovat popis světa ve SDF formátu.

Druhým způsobem je opačná strategie, vymodelování světa pomocí skriptů v SDF formátu. Tento model se bude také nacházet v souboru s příponou *.world* a bude spuštěn při startu Gazeba.

V našem případě zvolíme druhý způsob. Vytvoříme si soubor s názvem *maze_with_path.world* (tento soubor bude součástí příloh na CD), ve kterém si nadefinujeme naše prostředí a vložíme do něj vodící čáru. Tento soubor později vložíme do složky *worlds,* která bude obsažena ve vytvořeném balíčku *turtlebot_gazebo viz. kapitola 6.2.3.*



Obr. 36 Spuštění Gazeba s vymodelovaným světem maze_with_path.world.

6.2.3 Vytvoření ROS balíčků

Ze všeho nejdříve vytvoříme pracovní prostor (*catkin workspace*), ve kterém budeme dále pracovat. *Catkin workspace* vytvoříme následnými příkazy:

```
mkdir -p ~/catkin_ws/src    <--příkaz mkdir vytvoří složku src-->
cd ~/catkin_ws/src          <--příkaz cd zajišťuje pohyb mezi složkami-->
catkin_init_workspace
```

Catkin workspace je zatím prázdný, obsahuje pouze vytvořenou složku *src*, která je ovšem prázdná. Nyní můžeme pracovní prostor sestavit pomocí catkin_make:

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

Po provedení těchto příkazů můžeme vidět, že v adresáři catkin_ws nám přibyly další dvě složky *buil* a *devel*. Ve složce *devel* můžeme vidět několik souborů nastavení s příponou *.sh*. Před pokračováním nastavíme cestu nového zdroje:

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

Pro zkontrolování správné cesty k balíčkům provedeme příkaz, který nám vypíše nastavenou cestu k balíčkům. Cesta by měla vypadat následovně:

```
echo $ROS_PACKAGE_PATH
/home/youruser/catkin_ws/src:/opt/ros/indigo/share:/opt/ros/indig
o/stacks
```

Než přejdeme k vytváření balíčků, ukážeme si jednoduchou strukturu pracovního prostoru:[41]

```
workspace_folder/          -- WORKSPACE
   src/                    -- SOURCE SPACE
      CMakeLists.txt       -- 'Toplevel' CMake file
      package_1/
        CMakeLists.txt     -- CMakeLists.txt file for package_1
        package.xml        -- Package manifest for package_1
      ...
      package_n/
        CMakeLists.txt     -- CMakeLists.txt file for package_n
        package.xml        -- Package manifest for package_n
```

První balíček, který vytvoříme, se bude jmenovat *turtlebot_gazebo* a druhý *turtlebot_follower*. Vytvoříme je následujícím způsobem:

```
cd ~/catkin_ws/src
catkin_create_pkg turtlebot_gazebo std_msgs rospy roscpp
catkin_create_pkg turtlebot_follower std_msgs rospy roscpp

catkin_create_pkg <název_balíčku> <závislot1> <závislot2>
```

Nakonec provedeme příkaz *catkin_make* pro sestavení balíčků.

V balíčku *turtlebot_gazebo* vytvoříme pomocí příkazu *mkdir* složky *launch* a *worlds*. Do složky *worlds* vložíme náš vytvořený svět (*maze_with_path.world*) z předchozí kapitoly a ve složce *launch* vytvoříme launch soubor, kterým se bude spouštět simulátor Gazebo s naším vytvořeným světem a vloženým turtlebotem. Tento soubor nazveme *turtlebot_maze_world.launch* (tento soubour bude součástí příloh na CD).

V našem druhém balíčku (*turtlebot_follower*) vytvoříme složky *launch* a *src*. Ve složce *launch* vytvoříme launch soubor s názvem *follower.launch* (tento soubor bude součástí příloh na CD), který po spuštění zajistí spojení s turtlebotem a přenese na něj soubor *follower.cpp*, který vytvoříme ve složce *src*. Soubor *follower.cpp* (tento soubor bude součástí příloh na CD) je kód psaný v jazyce C++ a zajišťuje algoritmus pro sledování čáry/cesty.

6.2.4 Simulace

Všechny aspekty potřebné pro simulaci jsme již splnili, takže můžeme simulaci otestovat.

Nejdříve v terminálu nastavíme cestu zdroje příkazem:

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

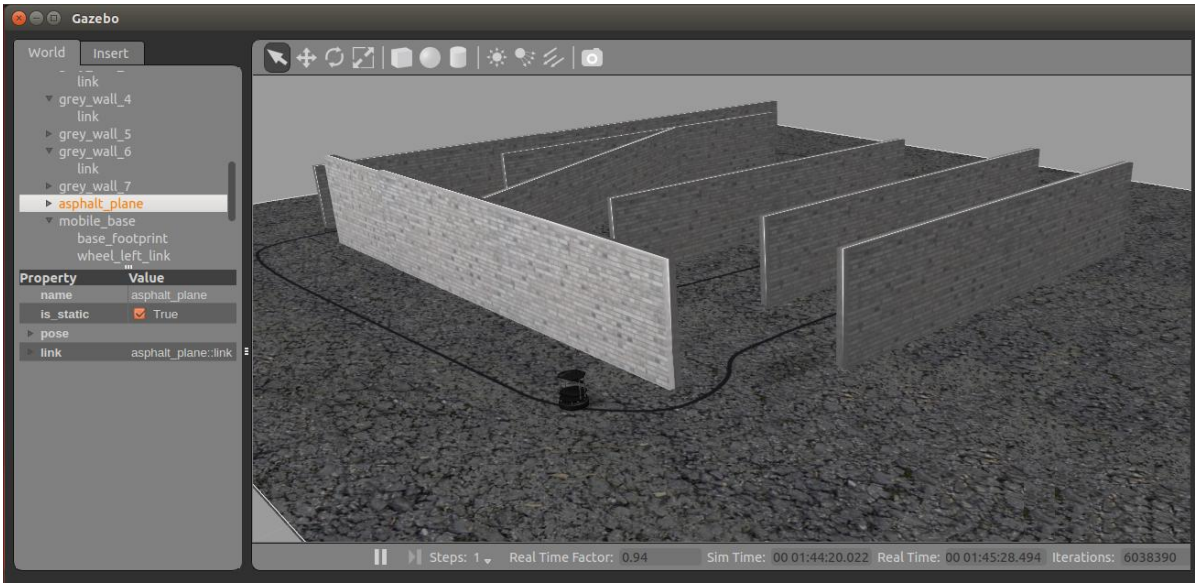
Následně zkusíme spustit Gazebo s vytvořeným světem a turtlebotem. Provedeme to pomocí příkazu:

```
roslaunch turtlebot_gazebo turtlebot_maze_world.launch
```

Pokud máme vše správně nastavené (cestu zdroje, cestu balíčku), můžeme k dokončování příkazů použít stisknutí klávesy *Tab*. Stačí napsat pár počátečních písmen příkazu a po stisknutí tabulátoru by se měl příkaz automaticky dokončit.

```
rosla...<tab>
roslaunch turt...<tab> turtlebot_g...<tab>
roslaunch turtlebot_gazebo t...<tab>
roslaunch turtlebot_gazebo turtlebot_maze_world.launch
```

Po provedení příkazu se spustí Gazebo, jehož prostředí vypadá následovně:

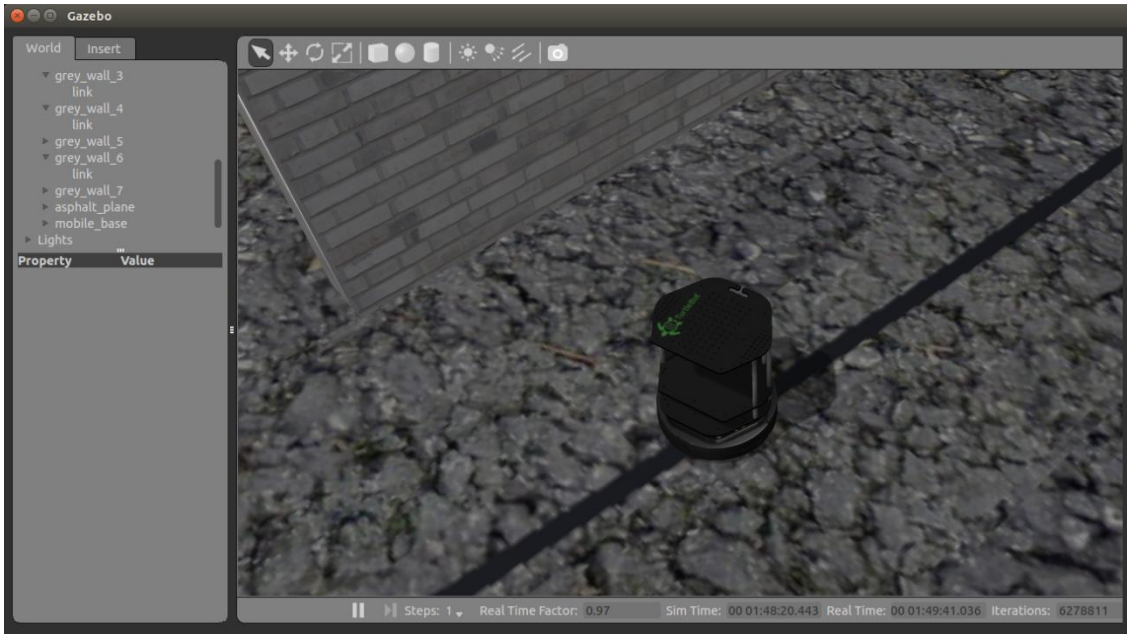


Obr. 37 Výsledek provedení příkazu roslaunch.

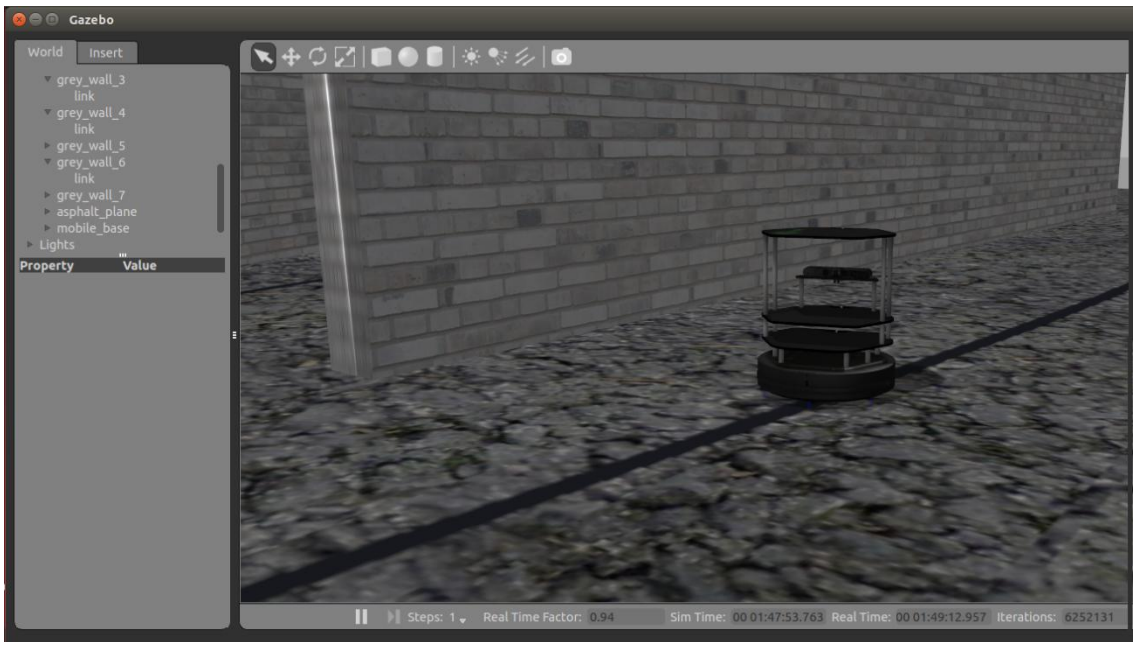
Pokud máme spuštěné Gazebo, můžeme spustit i *follower.launch* pro rozpohybování robota. Příkaz vypadá následovně:

```
roslaunch turtlebot_follower follower.launch
```

Provedením tohoto příkazu se dá robot (turtlebot) do pohybu, začne hledat a sledovat cestu (vodící čáru).

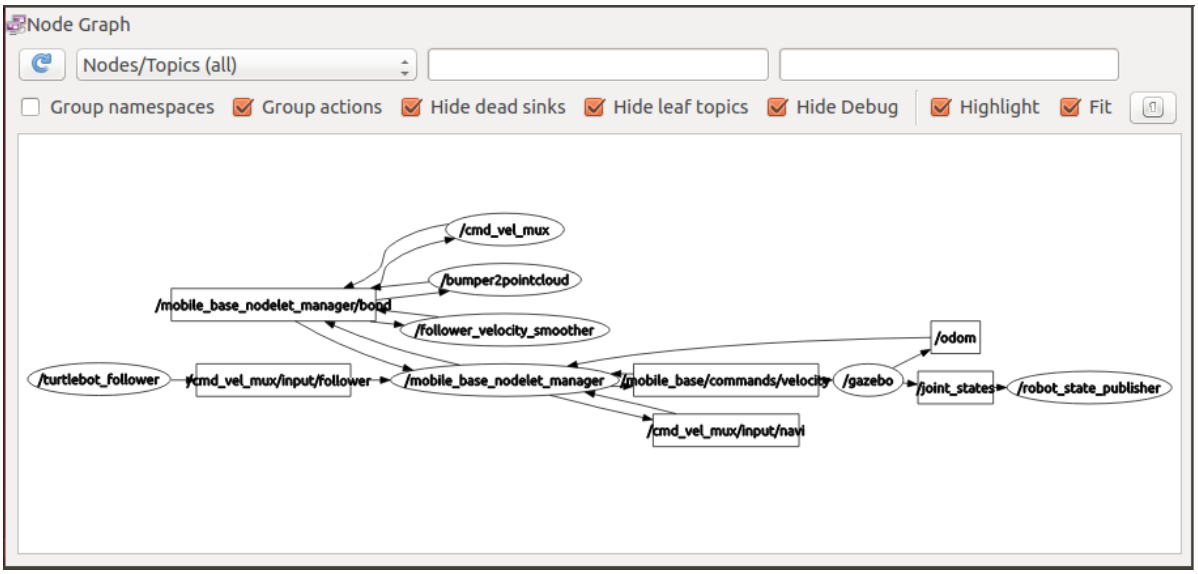


Obr. 38 Turtlebot v Gazebu.



Obr. 39 Turtlebot v Gazebu sledující čáru/cestu.

Pomocí nástroje *rqt_graph* můžeme zobrazit graf, kde je zobrazeno, jak spolu jednotlivé uzly komunikují. Do nového terminálu stačí zadat příkaz *rqt_graph.*



Obr. 40 rqt graf zobrazující propojení/komunikaci uzlů prostřednictvím témat.

Ze simulace robotické úlohy „Sledování čáry“ (Line Following) prostřednictvím Gazeba a frameworku ROS, byl pořízen videozáznam. Tento videozáznam je součástí příloh na CD, které je dodané k této práci.

6.3 Implementace řídícího algoritmu

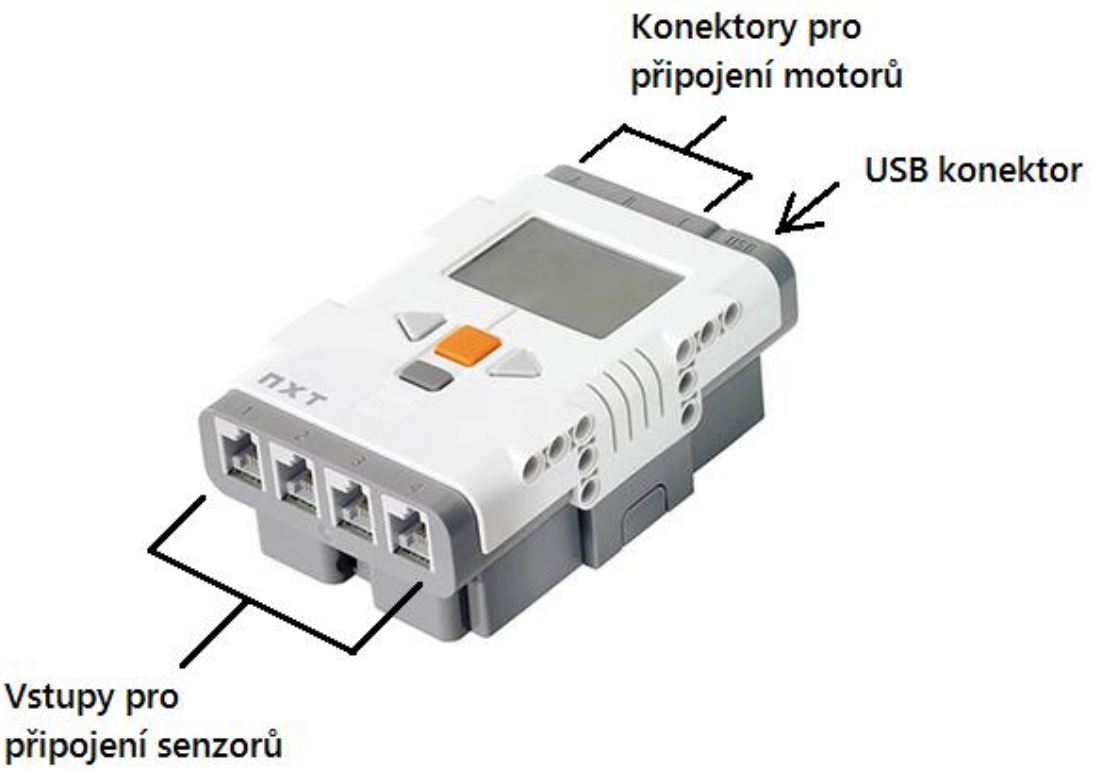
V této kapitole se budeme zabývat implementací řídícího algoritmu z předešlé simulace robotické úlohy. Implementace bude probíhat pomocí frameworku ROS Electric, který bude nainstalován na operačním systému Ubuntu 12.04 LTS, a robota LEGO Mindstorms NXT.



Obr. 41 Logo operačního systému Ubuntu 12.04 LTS.[36]

6.3.1 Robot LEGO Mindstorms NXT

LEGO Mindstorms NTX je robotická stavebnice, která byla představena roku 2009. Tato stavebnice je velmi flexibilní, dají se z ní sestavit různé druhy robotů, ať už kolové, pásové nebo humanoidního typu. Stavebnice NTX obsahuje, jako řídící jednotku, 32-bitový mikrokontrolér Atmel. Komunikace s ním může probíhat pomocí USB sběrnice nebo bezdrátovou technologií Bluetooth.

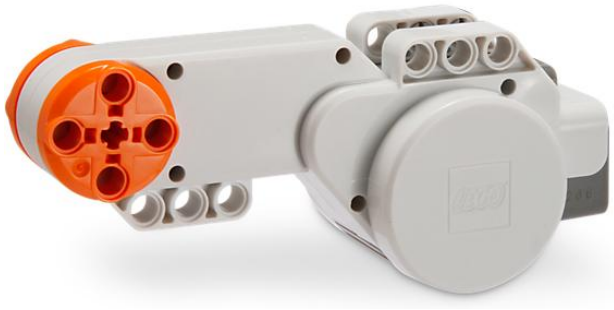


Obr. 42 Řídící jednotka robotické stavebnice LEGO NXT.[42]

Řídící jednotka obsahuje 4 vstupy (označeny 1, 2, 3, 4), pro připojení různých senzorů. V našem případě použijeme barevný RGB senzor (*viz Obr. 43*), pro sledování čáry. Dále obsahuje tři konektory (označené A, B, C) pro připojení motorů, v našem případě použijeme dva DC motory (*viz Obr. 44*) pro pohyb kol. A nakonec obsahuje USB konektor pro komunikaci.

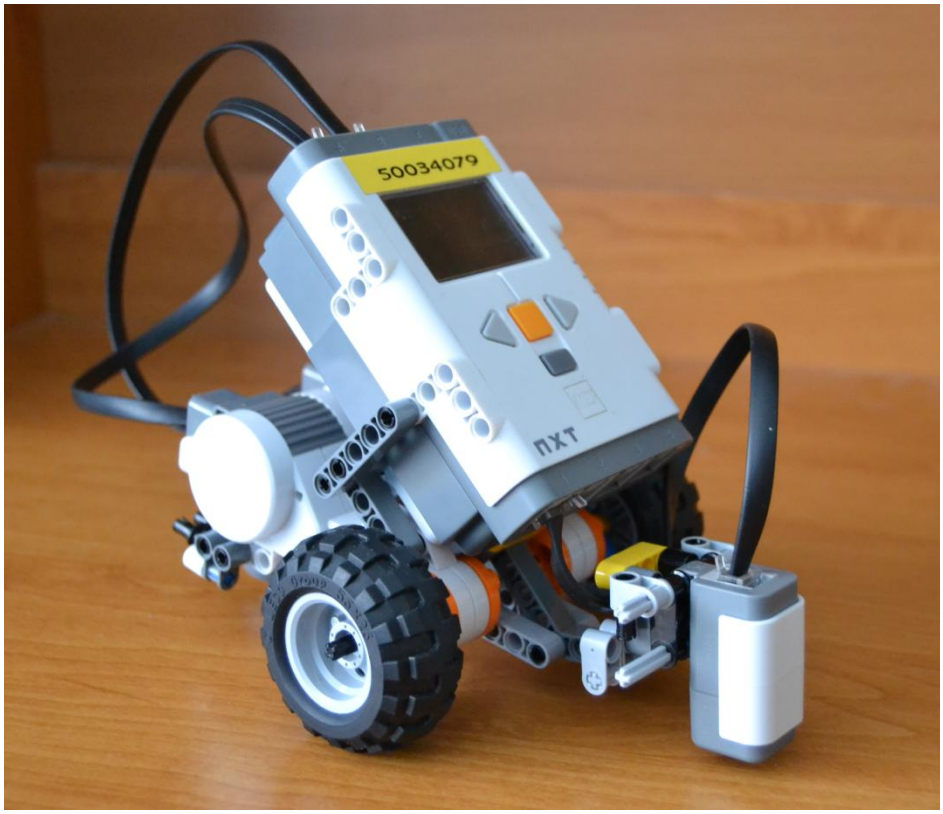


Obr. 43 Barevný RGB senzor LEGO NXT.[42]



Obr. 44 DC motor LEGO NXT.[42]

Pomocí robotické stavebnice LEGO Mindstorms NXT, si můžeme sestavit vhodného robota pro naší robotickou úlohu. Náš sestavený robot je zobrazen na následujícím obrázku:



Obr. 45 Sestavený robot ze stavebnice LEGO Mindstorms NXT.

6.3.2 Propojení robotu NXT s frameworkem ROS

Jako první nainstalujeme framework *ROS electric* na *Ubuntu 12.04 LTS*. Nejdříve nastavíme počítač pro přijetí softwaru z ROS.org. To provedeme následujícím příkazem:

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise pangolin
main" > /etc/apt/sources.list.d/ros-latest.list'
```

Dále provedeme nastavení klíče:

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Nyní můžeme spustit samotnou instalaci:

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

Pro nastavení prostředí je vhodné, aby při každém novém spuštění shell příkazu, byly automaticky přidány proměnné prostředí ROS do bash relace.

```
echo "source /opt/ros/electric/setup.bash" >> ~/.bashrc.
~/.bashrc
```

Když máme hotovou instalaci frameworku ROS Electric, můžeme se pustit do instalace softwaru NXT-ROS. Když je náš počítač nastaven pro přijetí softwaru z ROS.org a je nastaven i klíč, provedeme příkaz pro instalaci NXT-ROS:

```
sudo apt-get update
sudo apt-get install ros-electric-nxtall
```

Nyní máme nainstalované potřebné komponenty a můžeme přejít k vytváření ROS balíčků.

6.3.3 Vytvoření ROS balíčků

Jako první vytvoříme „poznámkový“ balíček, který nazveme *nxt_robot_sensor*. Vytvoříme ho následujícím způsobem:

```
roscd nxt     <--umístění do složky nxt-->
roscreate-pkg nxt_robot_sensor rospy nxt_ros <--vytvoření balíčku-->
rosmake       <--sestavení balíčku-->
```

Ve verzi ROS Electric nepoužíváme k sestavení *catkin* (catkin_make), ale starší *rosbuild* (rosmake).

NXT-ROS vazba musí vědět, které senzory jsou připojeny ke kostce (řídící jednotce) robotu a na jakém portu jsou připojeny. To je popsáno v konfiguračním souboru. V našem případě použijeme barevný RGB senzor, který bude připojen na port 1 a dva DC motory, které budou připojeny na portech A a B.

Vytvoříme tedy konfigurační soubor s názvem *robot.yaml*, který bude uvnitř balíčku *nxt_robot_sensor*. Tento soubor bude mít následující obsah:

```
nxt_robot:
      - type: motor
        name: r_wheel_joint
        port: PORT_A
        desired_frequency: 20.0
nxt_robot:
      - type: motor
        name: l_wheel_joint
        port: PORT_B
        desired_frequency: 20.0
nxt_robot:
      - type: color
        frame_id: color_link
        name: color_sensor
        port: PORT_1
        desired_frequency: 10.0
```

Když máme hotový konfigurační soubor, musíme ještě vytvořit spouštěcí soubor (ROS launch), který bude spouštět NXT-ROS. Spouštěcí soubor bude umístěn na stejném místě jako *robot.yaml*. Nazveme ho *robot.launch*.

Pro kontrolu komunikace mezi robotem a počítačem, a pro kontrolu správného zapojení a čtení senzorů, připojíme robota USB kabelem k počítači a ve dvou terminálech spustíme tyto příkazy:

```
roslaunch nxt_robot_sensor robot.launch
rostopic echo color_sensor
```

První příkaz spustí komunikaci prostřednictvím NXT-ROS, druhý příkaz zobrazuje vjemy získané z barevného RGB senzoru (*viz Obr. 46).*



Obr. 46 Zobrazení vjemů (barev) z RGB senzoru.

Zbývá nám tedy vytvořit balíček, který bude zajišťovat algoritmus sledování čáry. Pojmenujeme ho *nxt_follower* a vytvoříme ho následovně:

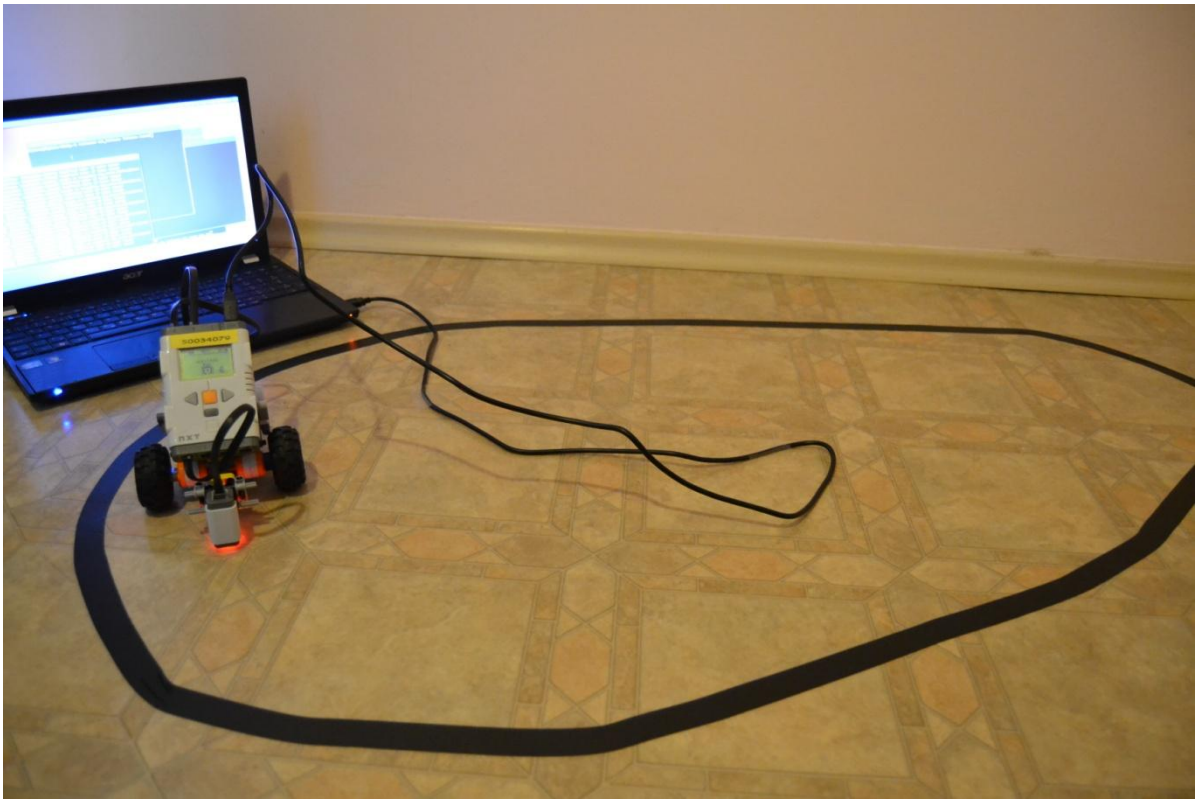
```
roscd nxt     <--umístění do složky nxt-->
roscreate-pkg nxt_follower roscpp nxt_ros <--vytvoření balíčku-->
rosmake       <--sestavení balíčku-->
```

Do složky *src* (tato složka byla vytvořena sestavením balíčku) vložíme soubor *nxt_follower.cpp*, který obsahuje kód korespondující s kódem obsaženým v souboru *follower.cpp* (*viz kapitola 6.2.3*). V tomto kódu bude pouze nahrazen robot Turtlebot robotem NXT.

Nakonec už jen stačí vytvořit složku *launch*, ve které se bude nacházet spouštěcí soubor *nxt_follower.launch*.

6.3.4 Provedení robotické úlohy

Nyní máme vše hotovo a můžeme provést robotickou úlohu. Jelikož se jedná o robotickou úlohu „sledování čáry“, je potřeba nějakou vodící čáru vytvořit (*viz Obr. 47*).



Obr. 47 Vytvoření vodící čáry.

Spuštěním následujících příkazů se dá robot do pohybu, začne hledat a sledovat vodící čáru. V prvním terminálu spustíme:

```
roslaunch nxt_robot_sensor robot.launch
```

Ve druhém terminálu spustíme:

```
roslaunch nxt_follower nxt_follower.launch
```

Z provedení robotické úlohy „sledování čáry“, prostřednictvím frameworku ROS, softwaru ROS-NXT a robota LEGO Mindstorms NXT, byl pořízen videozáznam, který bude součástí příloh na CD.

7 ZÁVĚR

V této práci byla rozebrána problematika robotiky, robotických simulací a implementací řídících algoritmů.

V první části jsou popsány vybrané Open Source robotické simulátory, a to robotické simulátory V-rep, Webots a Gazebo. Tyto simulátory byly vybrány pro jejich velmi široké využití. Poskytují velmi realistické simulace a to díky fyzikálním enginům, které v simulovaných prostředích zajišťují fyzikální zákony, jako je gravitace, tření, odpor vzduchu nebo proudění částic. Tyto robotické simulátory nabízejí, kromě možnosti vytvoření vlastního robota, i použití široké škály modelů z různých databází. Jako nejflexibilnější robotickým simulátorem, byl vybrán robotický simulátor Gazebo.

V druhé části této práce byl přiblížen framework ROS (Robotic Operation Systém). Je zde uvedení jeho historie a postupného vývoje. Dále je zde ukázán koncept, podle kterého ROS pracuje. ROS má v dnešní době již devět verzí, poslední verze *ROS Jade* byla zveřejněna 23.5.2015. V této práci jsou použity dvě verze ROS, a to *ROS Indigo* a *ROS Electric.* Dále bylo v této části práce ukázáno propojení robotického simulátoru Gazebo s frameworkem ROS. Oba tyto prostředky byly nainstalovány na operačním systému Ubuntu 14.04 LTS.

Třetí část práce se zabývá simulací a implementací robotické úlohy. Jako robotická úloha byla vybrána úloha „Line Following“ (sledování čáry/cesty). Tato úloha je velmi známá a vyskytuje se takřka na každé robotické soutěži. Úloha spočívá v tom, že vybraný robot musí detekovat a následně sledovat předdefinovanou vodící čáru pomocí vybraných senzorů. V našem případě byl pro simulaci vybrán robot *turtlebot,* který byl simulován prostřednictvím robotického simulátoru Gazebo a frameworku ROS. Simulátor Gazebo zajišťoval grafické rozhraní simulace a framework ROS, který byl nainstalován ve verzi Indigo, zajišťoval svými ROS „balíčky“ samotnou simulaci (řídící algoritmus, popis robota a senzorů, komunikaci mezi nimi). Implementace řídícího algoritmu probíhala ve verzi ROS Electric. Jako reálný robot pro implementaci byl vybrán robot postavený ze stavebnice LEGO Mindstorms NXT, který sledoval vodící čáru pomocí barevného RGB senzoru. Tento sestavený robot, byl řízen pomocí softwaru NXT-ROS. Tento software zajistil, pomocí rozhraní USB, přenesení vytvořených ROS „balíčků“ s řídícím algoritmem na reálného sestaveného robota. Při implementaci bylo pořízeno video robotické úlohy, které je dostupné v přílohách vložených na CD. Toto CD je přiloženo k této diplomové práci.

Robotický operační systém (framework) ROS se již zasloužil o vytvoření spousty robotických aplikací a to hlavně proto, že dokázal spojit práci mnoha lidí. V dnešní době se ROS stále vyvíjí a rozšiřuje, přidávají se k němu stále nové a nové robotické instituce, ústavy nebo nadšenci pro robotiku, a každý něčím přispívá k vytvoření nejpropracovanějšího robotického operačního systému ROS.

SEZNAM CITOVANÝCH ZDROJŮ

[1] Wikipedie, Otevřená encyklopedie. *Simulace* [online]. 2008, 16.5.2013
[cit. 2015-05-10]. Dostupné z: http://cs.wikipedia.org/wiki/Simulace

[2] Wikipedie, Otevřená encyklopedie. *Počítačová simulace* [online]. 2008, 16.5.2013
[cit. 2015-05-10]. Dostupné z: http://cs.wikipedia.org/wiki/Simulace

[3] Coppelia Robotics. *V-REP home* [online]. 2013
[cit. 2015-05-10]. Dostupné z: http://www.coppeliarobotics.com/index.html

[4] Open Source Robotics Foundation. *Gazebo* [online]. 2014
[cit. 2015-05-10]. Dostupné z: http://gazebosim.org/

[5] Cyber Botics. *Introduction to Webots.* [online]. 2013
[cit. 2015-05-10]. Dostupné z: https://www.cyberbotics.com/

[6] AMS. *Process simulate.* [online]. 2015 [cit. 2015-05-10].
Dostupné z: http://www.ams-innovation.com/index.php/2014-10-09-15-29-46

[7] Coppelia Robotics. *V-REP home* [online]. 2013 [cit. 2015-05-10].
Dostupné z: http://www.coppeliarobotics.com/index.html

[8] Coppelia Robotics. *Dynamic Module.* [online]. 2013 [cit. 2015-05-10].
Dostupné z: http://www.coppeliarobotics.com/helpFiles/en/dynamicsModule.htm

[9] ŠŤASTNÝ, M. *SW pro simulaci autonomních robotů*. [PDF dokument]. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. 55 s.

[10] Coppelia Robotics. *Writing code in and around V-REP*. [online]. 2013 [cit. 2015-05-10].
Dostupné z: http://www.coppeliarobotics.com/helpFiles/en/writingCode.htm#sixMethods

[11] Cyber Botics. *Webots overview.* [online]. 2015
[cit. 2015-05-15]. Dostupné z: https://www.cyberbotics.com/overview

[12] Robotica Unileon. *Nao tutoriál*. [online]. 18.2.2015 [cit. 2015-05-15].
Dostupné z: http://robotica.unileon.es/mediawiki/index.php/Nao_tutorial_1:_First_steps

[13] Robot Advance. *Art-robotis-darwin.* [online]. 2012 [cit. 2015-05-15].
Dostupné z: http://www.robot-advance.com/art-robotis-darwin-op-deluxe-edition-577.htm

[14] Cyber Botics. *Guide.* [online]. 22.5.2013
[cit. 2015-05-15]. Dostupné z: http://www.cyberbotics.com/guide.pdf

[15] Open Source Robotics Foundation. *Gazebo.* [online]. 2014
[cit. 2015-05-15]. Dostupné z: http://gazebosim.org/

[16] SimTK. *Simbody*. [online]. 2015 [cit. 2015-05-17].
Dostupné z: https://simtk.org/home/simbody/

[17] DartSim. *Dynamic Animation and Robotics Toolkit.* [online]. 2015
[cit. 2015-05-17]. Dostupné z: http://dartsim.github.io/

[18] Open Source Robotics Foundation. *SDF*. [online]. 2014
[cit. 2015-05-17]. Dostupné z: http://sdformat.org/

[19] Open Source Robotics Foundation. *Gazebo Architekture.* [online]. 2014 [cit. 2015-05-17]. Dostupné z: http://gazebosim.org/tutorials?tut=architecture&cat=get_started

[20] Open Source Robotics Foundation. *Gazebo Components.* [online]. 2014 [cit. 2015-05-17]. Dostupné z: http://gazebosim.org/tutorials?tut=components&cat=get_started

[21] ROS. *About ROS.* [online]. 2013 [cit. 2015-05-18].
Dostupné z: http://www.ros.org/about-ros/

[22] DIRK, Thomas. *ROS Introduction*. WILLOW GARAGE. [online]. [cit. 2014-05-16].
[cit. 2015-05-18]. Dostupné z: http://wiki.ros.org/ROS/Introduction

[23] The Orocos Project. *Home*. [online]. 2015
[cit. 2015-05-18]. Dostupné z: http://www.orocos.org/

[24] CARMEN-team. *Robot Navigation Toolkit.* [online]. 2015
[cit. 2015-05-18]. Dostupné z: http://carmen.sourceforge.net/

[25] The Player Project. *Player.* [online]. 16.2.2014
[cit. 2015-05-18]. Dostupné z: http://playerstage.sourceforge.net/

[26] YARP. *Welcome to YARP*. [online]. 2015
[cit. 2015-05-18]. Dostupné z: http://wiki.icub.org/yarpdoc/

[27] Microsoft. *Developer studio.* [online]. 2015 [cit. 2015-05-19].
Dostupné z: https://msdn.microsoft.com/en-us/library/bb648760.aspx

[28] Oxford Mobile Robotics Group. MOOS *Home page.* [online]. 2015 [cit. 2015-05-19].
Dostupné z: http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage

[29] Orca Robotics. *Orca: Components for Robotics.* [online]. 1.4.2005
[cit. 2015-05-19]. Dostupné z: http://orca-robotics.sourceforge.net/

[30 WOODALL, William. *Distributions.* WILLOW GARGE. [online]. 22.5.2015
[cit. 2015-05-19]. Dostupné z: http://wiki.ros.org/Distributions

[31] ROS. *History*. [online]. 2013 [cit. 2015-05-19].
Dostupné z: http://www.ros.org/history/

[32] ROS. *Is ROS for me?*. [online]. 2013 [cit. 2015-05-20].
Dostupné z: http://www.ros.org/is-ros-for-me/

[33] AaronMR. *ROS Concepts*. WILLOW GARAGE. [online]. 21.6.2013
[cit. 2015-05-20]. Dostupné z: http://wiki.ros.org/ROS/Concepts

[34] POOLEY, Acorn. *Catkin.* WILLOW GARAGE. [online]. 25.5.2014
[cit. 2015-05-20]. Dostupné z: http://wiki.ros.org/catkin/conceptual_overview

[35] Open Source Robotics Foundation. *Gazebo ROS overview*. [online]. 2014 [cit. 2015-05-20]. Dostupné z: http://gazebosim.org/tutorials?tut=ros_overview&cat=connect_ros

[36] Ubuntu. *Co je Ubumtu.* [online]. 2014 [cit. 2015-05-20].
Dostupné z: http://www.ubuntu.cz/

[37] GERKEY, Brian. *Ubuntu instar of ROS Indigo.* WILLOW GARAGE. [online]. 3.11.2015 [cit. 2015-05-20].
Dostupné z: http://wiki.ros.org/indigo/Installation/Ubuntu

[38] Open Source Robotics Foundation. *Using roslaunch.* [online]. 2014 [cit. 2015-05-20].
Dostupné z: http://gazebosim.org/tutorials?tut=ros_roslaunch&cat=connect_ros

[39] LEGO, Mindstorms. *Home.* [online]. 2015 [cit. 2015-05-20].
Dostupné z: http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com

[40] Turtlebot. *Turtlebot.* [online]. 2015 [cit. 2015-05-22].
Dostupné z: http://www.turtlebot.com/

[41] COLEMAN, Davet. *Creating Package.* WILLOW GARAGE. [online]. 30.5.2013
Dostupné z: http://wiki.ros.org/ROS/Tutorials/CreatingPackage [cit. 2015-05-22].

[42] LEGO, Mindstorms. *Products*. [online]. 2015 [cit. 2015-05-22].
Dostupné z: http://www.lego.com/en-us/mindstorms/products/31313-mindstorms-ev3

SEZNAM ODBORNÉ LITERATURY

[1] DIRK, Thomas. Rosbag tool. WILLOW GARAGE. Robot Operating System [online]. [cit. 2014-05-16]. Dostupné z: http://wiki.ros.org/rosbag.

[2] FONTANA, Flavio. ROS on Windows. WILLOW GARAGE. Robot Operating System [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/win_ros/hydro/Msvc%20SDK.

[3] OPITZ, Tobias. Books. WILLOW GARAGE. Robot Operating System [online]. 2014 [cit. 2014-05-22]. Dostupné z: http://wiki.ros.org/Books.

[4] THRUN, S., BURGARD, W., FOX, D. Probabilistic robotics. The MIT Press, 2005, ISBN 0-262-20162-3.

SEZNAM PŘÍLOH NA CD

- Hlavní dokument DP, 2015, Šťastný Martin, Modelování a simulace robotických simulací
- Videozáznam simulace (video_simulace)
- Videozáznam implementace (video_implementace)
- Soubory.zip (maze_with_path.world, turtlebot_maze_world.launch, follower.launch, follower.cpp)