VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

VYTVOŘENÍ DEMONSTRAČNÍCH ÚLOH PRO LEGO MINDSTORMS A ROBOTICKÝ OPERAČNÍ SYSTÉM

CREATING OF DEMONSTRATIONAL TASKS FOR LEGO MINDSTORMS AND ROBOTICS OPERATING SYSTEM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **MARTIN DAŇO**
AUTHOR

VEDOUCÍ PRÁCE **Ing. JAROSLAV ROZMAN, Ph.D.**
SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

163929

Ústav:	Ústav inteligentních systémů (UITS)
Student:	**Daňo Martin**
Program:	Informační technologie
Název:	**Vytvoření demonstračních úloh pro Lego Mindstorms a Robotický operační systém**
Kategorie:	Umělá inteligence
Akademický rok:	2024/25

Zadání:

1. Nastudujte Lego Mindstorms NXT/Ev3 a Robotický operační systém (ROS). Prozkoumejte možnosti propojení Lega a ROS, případně možnosti programování Lega programovacími jazyky třetích stran.
2. Navrhněte a vytvořte sadu úloh (např. ježdění po čáře) pro jednoduché roboty vytvořené z Lega a ovládané buďto přes ROS nebo jinými jazyky.
3. Vytvořené úlohy otestujte a navrhněte případné rozšíření.

Literatura:

- Macenski, T. Foote, B. Gerkey, C. Lalancette, W. Woodall, “Robot Operating System 2: Design, architecture, and uses in the wild,” Science Robotics vol. 7, May 2022.

Při obhajobě semestrální části projektu je požadováno:
První dva body zadání.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Rozman Jaroslav, Ing., Ph.D.**
Vedoucí ústavu:	Kočí Radek, Ing., Ph.D.
Datum zadání:	1.11.2024
Termín pro odevzdání:	14.5.2025
Datum schválení:	31.10.2024

Abstrakt

Táto práca sa zaoberá návrhom a implementáciou sady demonštračných úloh pre Lego Mindstorms NXT a EV3. Za pomoci programovacieho jazyka Python boli implementované programy vykonávajúce ovládaný pohyb robota, náhodný pohyb po miestnosti a sledovanie čiary. Na platforme ROS bol vytvorený program na mapovanie priestoru. Prínosom tejto práce je ukázať, že aj za pomoci Lego robotov je možné vytvoriť zložitejšie programy a programovať ich vo svojom obľúbenom programovacom jazyku.

Abstract

This thesis focuses on the design and implementation of a set of demonstration tasks for Lego Mindstorms NXT and EV3. Using Python programming language, a set of programs was developed that performs controlled robot movement, random movement around a room and line following. A special spatial mapping program was implemented on top of ROS. The contribution of this work is to demonstrate that even with Lego robots, it is possible to create complex programs with one's preferred programming language.

Kľúčové slová

Lego Mindstorms, NXT, EV3, NXT-Python, EV3-dev, ROS, Rviz, BrickLink Studio

Keywords

Lego Mindstorms, NXT, EV3, NXT-Python, EV3-dev, ROS, Rviz, BrickLink Studio

Citácia

DAŇO, Martin. *Vytvoření demonstračních úloh pro Lego Mindstorms a Robotický operační systém.* Brno, 2025. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Jaroslav Rozman, Ph.D.

Vytvoření demonstračních úloh pro Lego Mindstorms a Robotický operační systém

Prehlásenie

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením Ing. Jaroslava Rozmana, Ph.D. Uviedol som všetky literárne pramene, publikácie a ďalšie zdroje, z ktorých som čerpal.

.....
Martin Daňo
13. mája 2025

Poďakovanie

Ďakujem vedúcemu mojej práce Ing. Jaroslavovi Rozmanovi, Ph.D. za cenné rady a ochotu pri vytváraní tejto mojej záverečnej práci. Taktiež by som sa rád poďakovať kamarátovi Michalovi za pomoc pri testovaní funkčnosti sady úloh na jeho robotovi.

Obsah

Zoznam obrázkov

Kapitola 1

Úvod

Stavebnice LEGO sú celosvetovo známe a lákajú pozornosť nejedného dieťaťa a dospelého po celom svete. Zo stavebnice je možné postaviť naozaj rôznorodé diela. V mysliach mnohých detí určite vznikali vízie o možnosti ovládať svoje výtvory. Za pomoci stavebníc LEGO Mindstorms je to skutočne možné. Stavebnice LEGO Mindstorms obsahujú programovateľnú kocku, mikropočítač, ktorý môže fungovať ako mozog celého výtvoru. Za pomoci tejto kocky je možné vytvárať a spúšťať programy, a tak robotovi vdýchnuť život. V mladosti som sa taktiež veľmi rád hrával so stavebnicou LEGO a vytváral rôzne zložitejšie mechanizmy len za pomoci vlastnej hlavy. Avšak ovládanie mechanizmu na diaľku alebo určenie konkrétnej práce mechanizmu muselo vždy ostať len v mojej hlave a predstavách. Práve z tohoto dôvodu som pri prezeraní dostupných bakalárskych tém siahol po tej, v ktorej si môžem zaspomínať na detské časy, avšak s tou zmenou, že teraz za pomoci programovania môžem robotovi povedať, čo presne má vykonať. Na vytváranie programov LEGO ponúka jednoduchý software, ktorý však postačuje len na vytvorenie základných programov, na vytváranie zložitejších programov už nie je vhodný.

Cieľom tejto bakalárskej práce je zoznámenie sa s možnosťami, akými je možné kocky LEGO Mindstorms ovládať za pomoci iného ako oficiálneho LEGO softwaru. Taktiež je cieľom vytvoriť sadu demonštračných úloh vo vybranom programovacom jazyku. Na základe týchto úloh bude možné prezentovať rôzne možnosti, ako sa dajú programovacie kocky, senzory a motory dostupné v každej generácií LEGO Mindstorms využiť a ovládať tak chod robota.

Táto práca je rozdelená do viacerých spolu súvisiacich kapitol. Kapitola 2 je zameraná na históriu tejto rady stavebníc, na rôzne druhy dostupných motorov a senzorov, ako aj na priblíženie oficiálneho softwaru na vytváranie programov pre LEGO Mindstorms. V kapitole 3 je možné nájsť rôzne druhy programovacích jazykov a spôsobov, ako s nimi programovať kocky z generácie NXT a EV3. Kapitola 4 sa zaoberá vytváraním modelov daných LEGO robotov, či už skladaním Lega vo virtuálnom priestore, alebo za pomoci značkovacieho formátu URDF. Kapitola 5 sa zaoberá Robotickým Operačným Systémom, jeho históriou a verziami, pretože aj za pomoci ROSu je možné ovládať chod LEGO robotov. Kapitola 6 sa venuje analýze faktorov, ktoré viedli k tomu, že sa na vytváranie sady úloh nepoužíval oficiálny software, ale iný programovací jazyk. Taktiež je v tejto kapitole možné nájsť ciele, ktorých riešenie je v tejto práci popísané. Sú tu popísané programy, z ktorých sa má skladať sada vytvorených demonštračných úloh. Detaily implementácie je možné nájsť v kapitole 7, ktorá obsahuje základné princípy fungovania úloh, ako aj popis riešenia. V záverečnej kapitole je možné nájsť zhrnutie dosiahnutých výsledkov, ako aj návrhy na vylepšenia, ktoré môžu byť do budúcna implementované.

Kapitola 2

Programovacie kocky LEGO Mindstorms

LEGO Mindstorms je projekt spoločnosti LEGO, ktorá vytvorila v priebehu rokov viacero generácií setov, ktoré okrem bežných LEGO stavebníc, predovšetkým z rady Technic, obsahujú aj programovateľnú kocku, rôzne druhy senzorov a motorov. Ku kocke je možné pripojiť priložené senzory a motory a následne vytvárať rôzne programy, ktoré môžu oživiť postavené LEGO.

2.1 Vývoj LEGO Mindstorms

V 80. rokoch minulého storočia sa spoločnosť LEGO, na čele s Kirkom Kristiansenom, začala zaoberať myšlienkou vytvorenia takej súčiastky, na základe ktorej by deti mohli svoje výtvory ovládať a programovať [23]. S týmto nápadom prišiel pán Kristiansen na MIT za profesorom Seymourom Papertom, ktorý pár mesiacov predtým predstavil kontrolovaného robota LOGO. Dohodli sa na dlhodobej spolupráci a začali rozmýšľať ako spojiť LEGO a LOGO dokopy. Po rokoch spolupráce a pokusoch v roku 1998 vydali prvú programovateľnú kocku. Kocky dostali pomenovanie Mindstorms, po knihe od už spomínaného profesora Paperta, ktorá nesie rovnaký názov. LEGO vytvorilo štyri rôzne generácie LEGO Mindstorms.

Prvý model LEGO Mindstorms robotov nesie názov RCX. Tento model bol vydaný v roku 1998 [10]. RCX používal 16 MHz procesor a 32 kB RAM. Kocka je vybavená tromi vstupnými portami, ktoré sú určené na pripojenie senzorov a tromi výstupnými portami, ktoré slúžia na ovládanie motorov. Kocka taktiež obsahuje päťmiestny LCD displej a štyri tlačidlá. K základnej sade patria dva štandardné motory a jeden mikro motor. Zo senzorov je k dispozícií dotykový senzor, teplotný senzor, rotačný senzor a svetelný senzor, ktorý meria odraz svetla od povrchu.

V roku 2006 prišlo LEGO na trh s novším modelom z generácie Mindstorms, a to konkrétne s NXT, ktorý obsahuje 48 MHz procesor a 64 KB RAM. Tento model obsahuje tri servomotory a senzory zvuku, svetla, dotyku a senzor na určovanie vzdialenosti. Pri NXT 2.0, vydaného o tri roky neskôr, prišlo k pridaniu ďalšieho dotykového senzoru a zmene senzoru svetla na senzor rozpoznávania farby. Tento model má k dispozícii štyri vstupné porty, ktoré sú určené na pripojenie senzorov a tri výstupné senzory, ktoré slúžia k pripojeniu a ovládaniu motorov. Na pripojenie sú dostupné Bluetooth 2.0 a taktiež s USB-B portom, ktorý môže slúžiť na pripojenie k počítaču alebo na napájanie robota [23].

Ďalšou, v poradí treťou, programovateľnou kockou z rodiny Mindstorms je kocka EV3. Jedná sa o kocku, na ktorej beží operačný systém Linux. Má 300 MHz procesor, 64 MB RAM a 16 MB Flash pamäte. Model je predávaný s dvoma veľkými motormi, jedným stredným motorom a opäť so sadou senzorov. V sade sú taktiež dva dotykové senzory, ultrasonický senzor, farebný senzor a gyro senzor. Oproti predchádzajúcej generácií NXT sa na kocke EV3 nachádzajú až štyri výstupné porty, ktoré umožňujú ovládať pripojené motory. Rozdiel oproti predchádzajúcej generácií je tiež ten, že EV3 ponúka USB-host port, ktorý podporuje pripojenie Wi-Fi adaptéru, ako aj miesto na vloženie MicroSD karty. Taktiež existuje možnosť pripojiť kocku za pomoci Bluetoothu, ako tomu bolo už pri NXT, EV3 však už používa verziu Bluetooth 2.1.

Posledná, štvrtá, generácia LEGO Mindstorms nesie názov Inventor. Bola vydaná v roku 2020 a obsahuje 100 MHz procesor, 1 MB flash pamäte a 320 kB RAM. Obsahuje šesť portov, na ktoré je možné pripojiť štyri servomotory, senzor farby alebo senzor na meranie vzdialenosti. K pripojeniu sa využíva Micro USB alebo Bluetooth [3].

Ako už bolo povedané, LEGO Mindstorms Inventor je posledná z rady Mindstorms. Po 24 rokoch od vydania prvej programovateľnej kocky LEGO oznámilo, že na konci roku 2022 ukončí projekt LEGO Mindstorms a presmeruje svoju pozornosť pri výrobe LEGO robotov do projektu LEGO Education [22].

2.2 Motory a senzory

Ku každej z generácie LEGO Mindstorms sú k dispozícií motory a senzory. Pre verzie NXT a EV3 sú rovnaké dotykové senzory, senzor farby a ultrazvukový senzor.

Dotykový senzor funguje ako bežný mechanický tlačidlový spínač, kde pri uvoľnenom stave je obvod otvorený a pri stlačení sa obvod uzavrie [4]. Tento senzor sa pre obe generácie zásadne nemenil.

Senzor farby sa skladá z RGB LED a fotodiódy [24]. LED postupne vysiela červené, zelené a modré svetlo a fotodióda zaznamenáva intenzitu odrazeného svetla pre každú z farieb. Senzor má viac módov, ktoré dokážu rozlíšiť 6 základných farieb, merať množstvo odrazeného svetla spať do senzoru, merať množstvo okolitého osvetlenia alebo môžu použiť RGB LED ako farebnú lampu.

Ultrazvukový senzor má tri rôzne módy použitia [12]. V prvom dokáže zistiť, či sa v jeho blízkosti nachádza nejaký iný ultrazvukový senzor. V tomto móde senzor nevysiela žiaden signál, ale iba čaká, či zachytí signál z iného ultrazvukového zariadenia. V ostatných dvoch módoch, v ktorých senzor meria vzdialenosť od objektov v centimetroch alebo v palcoch, senzor generuje 40 kHz impulz, tento impulz je ľudským uchom nepočutý. Na základe tohoto signálu a následného odrazu signálu späť do senzoru dokáže senzor určiť, v akej vzdialenosti sa zaznamenaný predmet nachádza, keďže si zaznamenáva dĺžku časového úseku medzi odoslaním a prijatím signálu. Spôsob, akým sa určuje vzdialenosť, je vidieť v rovnici 2.1,

$$distance = \frac{v_{sound} * t}{2} \tag{2.1}$$

kde $v_{sound} \approx 343$ m/s.
Senzor dokáže merať vzdialenosť od 3 cm do 250 cm. Senzor dostupný v generácií NXT, má presnosť merania ±3 cm. Senzor, ktorý je k dispozícií v generácií EV3, má presnosť merania ±1 cm.

Generácia EV3 má oproti NXT v základnom balení k dispozícií ešte aj gyro senzor. Tento senzor meria rotačný pohyb senzoru okolo jednej osi. Poskytuje tak údaje o uhlovej

rýchlosti a už nahromadenú uhlovú zmenu od posledného resetu senzora. Pri generácií NXT bol k dispozícií zvukový senzor, ktorý bol však pri EV3 vyradený.

Pri NXT boli v základnej sade tri motory rovnakého druhu, a to NXT Interactive Servo Motor [16]. Pre tento druh motora je krútiaci moment, pôsobenie sily na bod vzdialený od osi otáčania, 15 N.cm. Pri novšej generácií EV3 prišlo LEGO s novými druhmi motorov. V základnej sade boli k dispozícií dva EV3 Large Motor a jeden EV3 Medium Motor. Pri Large Motoroch je krútiaci moment väčší ako pri NXT a to až 20 N.cm. Pri Medium Motore je to však len 6 N.cm. Tento motor je slabší, avšak dokáže vyprodukovať väčšiu rýchlosť ako veľký motor, kvôli svojej nižšej hmotnosti má rýchlejšiu odozvu, a tak je lepšie ovládateľný [11].

2.3 Programovanie s oficiálnym softwarom

Ako už bolo vyššie spomenuté, základnou a hlavnou myšlienkou celého vývoja kocky Mindstorms bolo, aby boli výtvory ovládateľné, a tak užívatelia mohli vytvárať ešte zaujímavejšie a zložitejšie mechanizmy, ktorým následne mohli určiť, čo a ako budú vykonávať. Preto bolo potrebné vytvoriť jednoduché a ľahko ovládateľné prostredie. Prostredie, umožňujúce vytvárať programy, určené pre LEGO robotov, ktoré práve záujemca postavil, a má záujem ho ovládať za pomoci vlastného programu. Tieto prostredia boli navrhnuté tak, aby s nimi dokázal pracovať každý užívateľ.

Softwary, ktoré boli dostupné ku každej generácií Mindstorms, boli pre každú generáciu odlišné. Počas 24 rokov, ktoré Mindstorms boli na trhu, vznikli tak štyri rôzne softwary, pre každú generáciu jeden. Software sa pre každú verziu vizuálne líši, čo však ostáva stále veľmi podobné, je spôsob, akým vytváranie programov v softwari funguje. V každom softwari funguje vytváranie programu za pomoci výberu úkonu, ktorý sa má vykonať, poprípade zmeny parametrov. Následne je tu možnosť pridať ďalší blok, ktorý reprezentuje určitý kus kódu. Takýmto spôsobom software dovoľuje na seba nabaľovať rôzne druhy blokov, vytvárať tak programy, ktoré bude LEGO robot vykonávať.

Oficiálny software pre LEGO Mindstroms NXT alebo aj EV3 je grafické a ikonkami riadené prostredie, v ktorom nie je potrebné žiadne písanie kódu, ale program je vytváraný len za pomoci drag-and-drop. Tieto programy potom komunikujú s programovateľnou kockou, pre ktorú bol program vytvorený, cez Bluetooth, USB alebo pri EV3 cez Wi-Fi adaptér.

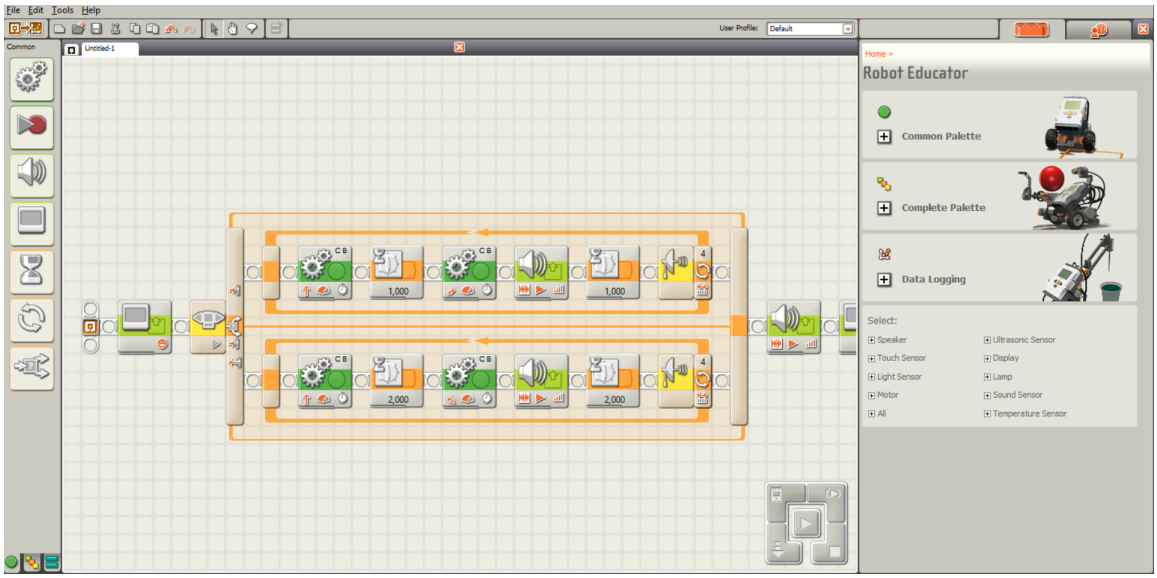
Po úspešnej inštalácií softwaru je k dispozícií možnosť vytvoriť si vlastný program výberom z jednotlivých blokov, ktoré poskytuje generácia Mindstormu, s ktorou práve pracujeme. Blok predstavuje určitú modifikovateľnú časť kódu, používanú na vytvorenie programu. Bloky obsahujú parametre, na základe ktorých sa dá upravovať chovanie bloku. LEGO Mindstorms NXT ponúka tieto bloky:

- `Move` – ponúka parametre smeru a rýchlosti otáčania motora
- `Play/Record` – možnosť nahrať a prehrať pohyby motorov, vykonané ručne[1] robotom
- `Sound` – prehranie vybraného zvuku
- `Display` – zobrazenie vybraného útvaru na displeji
- `Wait` – čakanie, či už časovú jednotku alebo na údaj zo senzoru

[1] Napríklad užívateľ robota vezme do ruky a vykoná s ním pohyb manuálnym otáčaním motorov. Robot si ho zapamätá a následne na požiadavku pohyb znovu vykoná.

- `Loop` – základná jednotka na tvorbu cyklov
- `Switch` – základná jednotka na tvorbu podmienok

Každý z týchto druhov blokov umožňuje upraviť ich parametre. Pri používaní motorov a senzorov je potrebné vždy nastaviť, k akému portu sú pripojené.



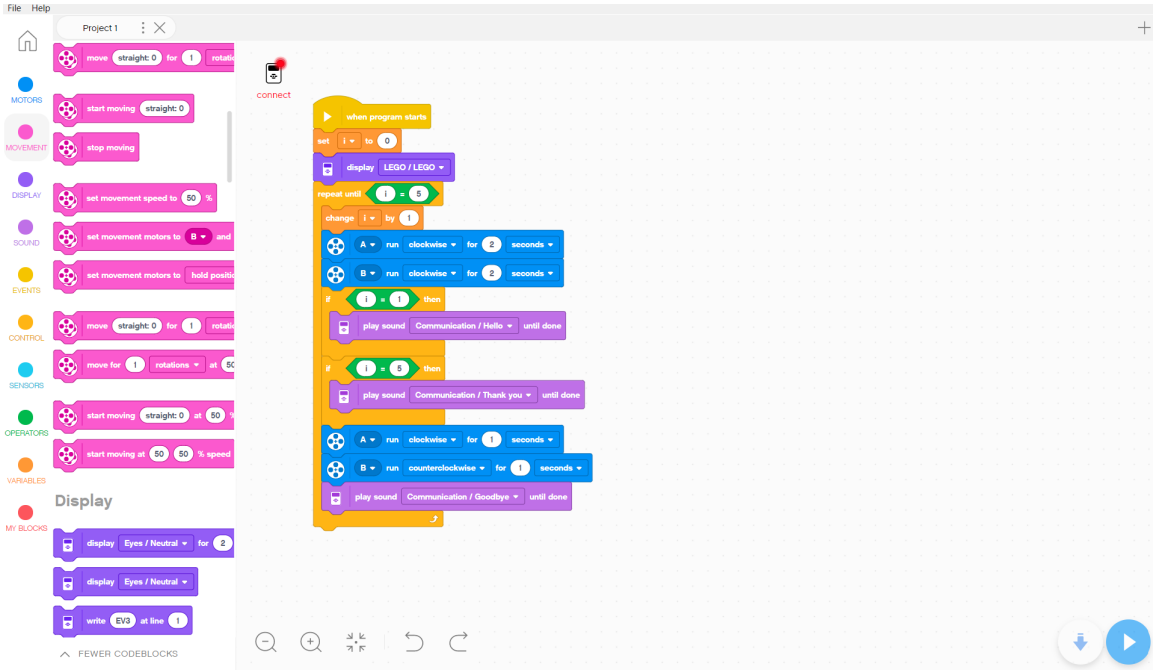
Obr. 2.1: Oficiálny software NXT

Software ponúkaný ku generácií EV3 priniesol možnosť používať tieto bloky:

- `Motors` – výber motora s nastavením parametrov
- `Movement` – výber druhu pohybu a nastavenie jeho rýchlosti a trvania
- `Display` – zobrazenie zvoleného obrázku na displeji
- `Sound` – prehranie vybraného zvuku
- `Events` – výber udalosti, ktorá spúšťa časť programu keď nastane
- `Control` – základné možnosti tvorby cyklov a podmienok
- `Sensors` – výber z dostupných senzorov
- `Operators` – výber matematickej operácie
- `Variables` – možnosť vytvoriť premennú
- `My blocks` – zoskupenie časti blokov programu, podobné vytvoreniu funkcie

Na Obr. 2.1 vidíme jednoduchý program vytvorený v softwari určenom pre LEGO Mindstorms NXT a na Obr. 2.2 vidíme triviálny program vytvorený v softwari určenom pre LEGO Mindstorms EV3.

Program zobrazený na Obr. 2.1 zabezpečuje, že robot, na ktorom bude spustený daný program, sa bude otáčať do toho smeru, aké tlačidlo na hlavnej kocke bolo stlačené. Pokiaľ bolo stlačené pravé tlačidlo, bude sa otáčať v smere hodinových ručičiek a pokiaľ ľavé



Obr. 2.2: Oficiálny software EV3

tlačidlo, tak sa bude otáčať proti smeru hodinových ručičiek. Robot bude teda jazdiť stále dokola, s občasnými prestávkami a s hraním hudby v smere, respektíve proti smeru hodinových ručičiek. Keď však robot zaznamená pred sebou prekážku, program skončí s cyklom a robot sa už len rozlúči a program skončí.

Za pomoci oficiálnych softwarov je možné robota naprogramovať a určiť mu, čo, kedy a ako bude vykonávať. Každý jeden software, ktorý bol pre Mindstorms vytvorený, je navrhnutý intuitívne, aby v ňom dokázal pracovať každý. Toto prostredie je vhodné na menšie programy.

Kapitola 3

Programy tretích strán

Ľudia chcú LEGO Mindstorms využívať aj na zložitejšie úkony, ktoré môžu byť náročné na postavanie v oficiálnom softwari. Alebo chcú programovať robota vo svojom obľúbenom programovacom jazyku, a tak sa vyhnúť nutnosti robiť s oficiálnym softwarom Mindstorms. Pri vytváraní programov na LEGO Mindstorms pokročilí užívatelia veľmi často narážali na problémy s oficiálnym softwarom, ktorý má obmedzené možnosti debugovania, obmedzené možnosti rozšírení a vytváraní robustných projektov. Existujú teda aj iné, komunitou vytvorené možnosti, ako tieto programovateľné kocky ovládať. Boli totižto vytvorené knižnice, pomocou ktorých dokážete tieto kocky programovať aj za použitia iných programovacích jazykov.

3.1 Nástroje dostupné pre NXT

Pokiaľ kocku nechceme programovať za pomoci oficiálneho softwaru pre NXT, vieme si vybrať z množstva programovacích jazykov, ktorými vieme kocky Mindstorms z generácie NXT programovať. Pri výbere programovacieho jazyka C je možnosť použiť programovacie prostredie RobotC, toto prostredie bolo vyvinuté na Carnegie Mellon University v USA [18]. Toto prostredie je vhodné pre užívateľov, ktorí by chceli využívať jazyk C na programovanie NXT robotov. Pri používaní tohoto prostredia je taktiež k dispozícií možnosť krokovania programu. Programy vytvorené v prostredí RobotC sa kompilujú priamo pre kocky NXT, tým pádom bežia priamo na kocke. Ďalšia možnosť programovacieho jazyka, ktorým je možné NXT programovať, je Java. Na programovanie v Jave slúži napríklad leJOS [13]. Použitie jazyku Java umožňuje tvorenie objektovo orientovaných programov, využívanie výnimiek, synchronizáciu, a paralelizáciu, ktorá môže pomôcť pri súbežnom čítaní alebo obsluhovaní viacerých senzorov a motorov. Ďalšie jazyky, v ktorých sa dá programovať NXT, je napríklad Python, C++ alebo Haskell. Pre každý z nich existuje možnosť, ako v danom programovacom jazyku programovať Mindstorms NXT.

3.2 Nástroje dostupné pre EV3

Pri kockách z generácie EV3 existuje taktiež veľké množstvo možností, aký programovací jazyk používať. Rovnako, ako pri predchádzajúcej generácii, je tu možnosť programovať v jazyku C pomocou prostredia RobotC. Použitie RobotC má pre programátorov veľa výhod oproti použitiu oficiálneho softwaru. Veľkou výhodou je možnosť využívať breakpointy, sledovať premenné a krokovať programy. Výhodou je aj fakt, že narozdiel od oficiálnych

softwarov sa pre programovanie NXT aj EV3 dá využívať rovnaké prostredie. Kocky z generácie EV3 je možné programovať pomocou jazyka Java za použitia operačného systému leJOS, podobne ako v prípade kociek NXT.

Ev3dev je open-source operačný systém založená na distribúcii Debian GNU/Linux, ktorý bola vytvorená pre kocky LEGO Mindstorms EV3. Tento operačný systém sa bootoje z MicroSD karty, ktorá je vložená v robotovi. Pri spustení, ak je prítomná SD karta s operačným systémom, spustí sa prednostne tento systém pred vstavaným systémom. Operačný systém vznikol v roku 2013, a to preto, aby užívatelia nemuseli používať len proprietárny operačný systém, ktorý poskytovala spoločnosť LEGO. Cieľom tejto distribúcie bolo umožniť používanie rôznych programovacích jazykov na písanie kódu. Systém ev3dev poskytuje veľké množstvo knižníc, ktoré zlepšujú a zjednodušujú prácu s robotom. V neposlednom rade sa zlepšila aj možnosť sieťovej komunikácie, či už cez USB, Bluetooth, alebo cez Wi-Fi za pomoci adaptéru. Debian umožňuje programovať EV3 za pomoci Pythonu, C, C++, Javy, Rustu a iných známych programovacích jazykov [8].

3.3 Python

Pri tvorbe tejto záverečnej práce sa pri programovaní robotov oboch generácií, LEGO Mindstorms NXT a LEGO Mindstorms EV3, používal programovací jazyk Python. Python som zvolil kvôli veľkému množstvu knižníc, ktoré sú k dispozícií, a pre ľahkú čitateľnosť kódu.

3.3.1 NXT-Python

NXT-Python[1] je balíček, ktorý využíva programovací jazyk Python 3 na prácu a ovládanie robotov s programovateľnou kockou LEGO Mindstorms NXT [2]. Tento balíček podporuje ovládanie kocky cez oba dostupné spôsoby poskytované pre danú kocku, cez Bluetooth, ako aj cez USB.

Pôvodný projekt pod názvom NXT_Python bol založený v roku 2006 Douglasom P. Lauom. Táto prvá verzia knižnice podporovala USB a Bluetooth komunikáciu a bola určená pre staršie verzie Pythonu. Pôvodný autor ukončil vývoj v máji 2007. Od roku 2007 až do roku 2015 Marcus Wanner udržiaval knižnicu až do verzie 2.2.2. V roku 2015 prešla knižnica veľkými zmenami, kedy NXT-Python prešiel na Python 3, o čo sa zaslúžil Elvin Luff. Luff pracoval na knižnici do roku 2021. Od 6.11.2021 má na starosti projekt NXT-Python Nicolas Schodet, ktorý pravidelne opravuje rôzne chyby a pridáva nové funkcie. Posledná vydaná verzia vyšla 12.1.2025 a podporuje Python 3.9 až 3.12.

Pri práci s týmto balíčkom je potrebné vedieť, že napísaný program sa počas celého procesu vykonáva na počítači, ktorý riadi kocku NXT a zasiela jej len príkazy. To znamená, že počas celého behu programu musí byť počítač s robotom spojený, či už cez USB alebo Bluetooth. Pre ovládanie cez USB sa používa knižnica PyUSB, ktorá je nainštalovaná automaticky. Pokiaľ chceme, aby bol robot s počítačom spojený cez Bluetooth, je potrebný balíček PyBluez.

Programovateľná kocka NXT má štyri vstupné porty a tri výstupné porty. Vstupné porty sú určené na pripojenie senzorov a výstupné na pripojenie motorov. Pre korektné používanie motorov a senzorov knižnicou NXT-Python je potrebné presné definovanie portu, ku ktorému sú dané motory a senzory pripojené. Knižnica NXT-Python umožňuje vytvoriť program, ktorý bude využívať senzory a motory, ktoré sú pre danú generáciu k dispozícií.

[1] https://ni.srht.site/nxt-python/latest/index.html
[2] https://github.com/schodet/nxt-python

Tento balíček taktiež podporuje aj pripojenie iných senzorov, ako len tých, ktoré sú k dispozícií v LEGO Mindstorms NXT sade. Program môže získavať dáta zo senzorov, analyzovať ich a na základe ich hodnôt určiť, čo sa bude ďalej vykonávať. Taktiež je tu možnosť zapínať motory na určitú rýchlosť, na určitý počet otáčok alebo na počet sekúnd, počas ktorých je potrebné mať motor zapnutý.

Pri každom senzore je k dispozícii viacero funkcií, ktoré je pri tvorbe programu možné použiť. Pri dotykovom senzore sa dá zisťovať, či je senzor zatlačený, alebo je v stave pokoja. Obdobne to je aj pri senzore na meranie teploty, ktorý zisťuje teplotu v °C alebo v °F. Pri senzore na detekciu farby sa dá zistiť presná intenzita odrazeného svetla alebo získať farba, ktorej sa snímaný povrch podobá najviac. Taktiež sa dá nastaviť farba, akou bude senzor svietiť. Pri ultrazvukovom senzore sa dá získať vzdialenosť senzoru od prekážky pred ním. Táto vzdialenosť sa udáva v centimetroch.

3.3.2 Python-ev3dev

Knižnica Python-ev3dev vznikla v roku 2014, kedy ju vytvoril Denis Demidov [6]. Táto knižnica poskytovala základné triedy, ktoré boli potrebné na prácu s motormi a senzormi. V roku 2017 bola knižnica prepísaná na Python-ev3dev2 [3], ktorá vylepšila dokumentáciu, pridala nové triedy a zlepšila typové anotácie. V súčasnosti knižnica podporuje Python 3.

Pri písaní kódu pre LEGO Mindstorms EV3 s použitím Python-ev3dev sa dajú používať všetky štandardné senzory a motory, ktoré sú dostupné pre danú generáciu. Pracovať sa dá aj s neoficiálnymi senzormi alebo so senzormi z generácie NXT. Pri senzoroch, ktoré sa nachádzajú v štandardnom balení, je možné používať veľké množstvo metód, ktoré sú k dispozícii. Kocka EV3 má štyri vstupné porty, ktoré sú určené na pripojenie senzorov a následne aj štyri výstupné porty, ktoré slúžia na pripojenie motorov. Všetky pripojené senzory a motory, s ktorými program potrebuje pracovať, je potrebné priradiť k portu, do ktorého sú zapojené.

Pri tvorbe programu pre EV3 je neoddeliteľnou súčasťou aj práca so senzormi a motormi, ktoré má robot k dispozícií. Použitím dotykového senzoru sa dá zistiť, či je senzor stlačený, poprípade počkať, pokiaľ bude uvedený do zopnutého stavu alebo do rozopnutého stavu. Za pomoci ultrazvukového senzoru sa dá zisťovať vzdialenosť od predmetu, ktorý sa nachádza pred senzorom. Túto vzdialenosť senzor vracia ako hodnotu v centimetroch alebo v palcoch. Taktiež je k dispozícii metóda, na základe ktorej dokáže senzor zistiť, či sa v jeho blízkosti nachádza nejaký iný ultrazvukový senzor. Za pomoci senzoru farby sa dá zistiť percentuálna intenzita svetla, ktorá je buď intenzitou svetla odrazeného povrchu späť do senzoru, alebo intenzitou okolitého svetla. Taktiež môžeme za pomoci senzoru zistiť presné množstvo RGB zložiek farby snímaného povrchu. Pri použití gyro senzoru vieme získať hodnoty uhlovej rýchlosti, ale aj veľkosť uhlu, o ktorý sa senzor posunul od posledného volania metódy reset. Python-ev3dev taktiež umožňuje zmeny farieb LED umiestnených pod tlačidlami, ako aj ich intenzitu. Taktiež je možné pracovať aj s reproduktormi, ktorým je možné poslať sekvenciu tónov a pri spustení programu ich kocka prehrá.

K dostupným motorom sa dá pristupovať samostatne a ovládať ich oddelene. Balíček však umožňuje taktiež pracovať s dvomi veľkými motormi v móde tanku, kedy sa motory ovládajú jedným príkazom súčasne. Pri motoroch je možné nastaviť im, aby pracovali určitý počet sekúnd, urobili určitý počet otočiek alebo aby sa otočili o daný uhol danou rýchlosťou.

Pri generácií EV3 sa program vykonáva priamo na kocke. To znamená, že program sa na kocku nahrá či už cez USB, Bluetooth, alebo cez Wi-Fi a následne sa tento program na

[3] https://github.com/ev3dev/ev3dev-lang-python

kocke môže vykonávať. V spojení s používaním Pythonu a množstvom iných balíčkov, ktoré sú k dispozícií, je možné využiť k vytvoreniu zložitých programov bez zbytočnej námahy.

Kapitola 4

Modely

Pri vytváraní zložitejších LEGO robotov a robotov ako takých, je veľmi užitočné vytvorenie modelu robota. Modely robotov slúžia na veľmi rýchle prototypovanie a validáciu vytváraných alebo navrhnutých robotov. V prípade použitia správneho softwaru sa dá veľmi rýchlo určiť, či je daný model robota správne navrhnutý, či je konkrétny robot zostrojiteľný, poprípade určiť, čo a ako treba pozmeniť. Všetky tieto zmeny a úpravy je možné vykonať bez nutnosti stavby robota v reálnom svete. Taktiež sú veľmi užitočné pri odhadovaní množstva súčiastok, ktoré budú na stavbu robota potrebné. Niektoré softwary ponúkajú taktiež možnosť vytvárať presný postup, plán stavby robota krok po kroku, na základe ktorého je možné navrhnutého robota zostrojiť. V neposlednom rade modely robotov sú často používané pri rôznych simuláciách a zobrazovaniach skutočného sveta vo virtuálnom priestore.

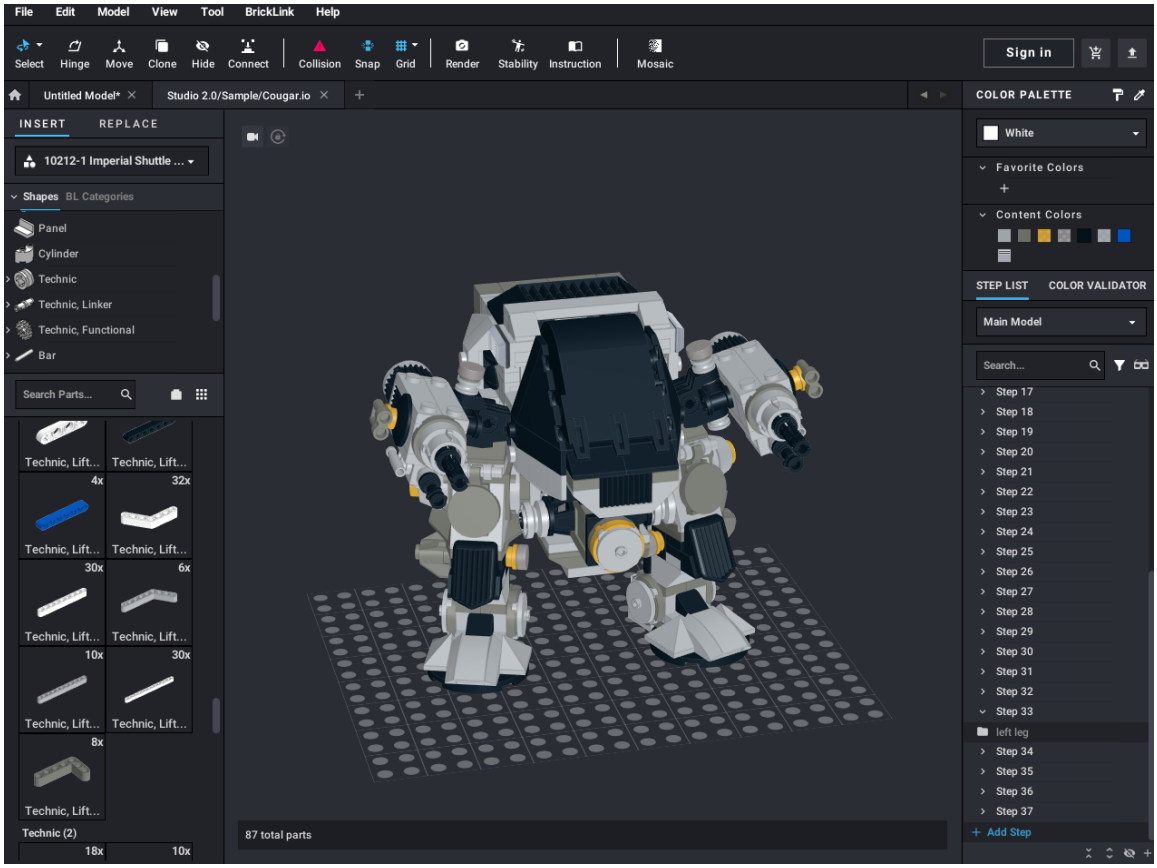
4.1 LEGO model

Oficiálnym softwarom spoločnosti LEGO na vytváranie LEGO modelov bol od roku 2003 LEGO Digital Designer. Išlo o bezplatný program, ktorý umožňoval vytvárať 3D návrhy a modely LEGO stavieb. LEGO v roku 2016 oficiálne oznámilo, že LDD už nebudú ďalej podporovať. Od tej doby vyšlo ešte pár nových verzií, avšak tento software už nie je možné stiahnuť na oficiálnej stránke Lega. V roku 2022 totiž spoločnosť LEGO oznámila, že LEGO Digital Designer bude nahradený programom BrickLink Studio[1] ako oficiálnym softwarom na vytváranie LEGO modelov. Tento software bol vytvorený spoločnosťou BrickLink, ktorá je od roku 2019 súčasťou spoločnosti LEGO. Slúži ako online priestor na nákup a predaj nových alebo použitých LEGO dielikov, setov a figúrok [2].

BrickLink Studio, pôvodne Stud.io, je freeware, ktorý slúži na vytváranie 3D modelov z LEGO dielikov. Program má k dispozícií všetky dostupné LEGO dieliky a súčiastky, ktoré sú uvedené na trhu. Program funguje na základe výberu konkrétneho dieliku a následného spájania dielikov do celkov. Dieliky sa dajú rôzne otáčať, spájať a upravovať tak, aby do seba zapadali. Ako je vidieť na Obr. 4.1, k dispozícií je taktiež upráva farby dielikov alebo ich presné spojenie s iným dielikom za pomoci príslušných nástrojov. Program je navrhnutý tak, že pokiaľ dochádza k určitému prekrytu súčiastok, zmenia svoju farbu ako indikátor toho, že model je neskonštruovateľný. Program taktiež ponúka možnosť vytvárať kroky, ako bol robot vytvorený, čo sa dá použiť pri vytváraní návodu, ako daný model skonštruovať v reálnom svete. V BrickLink Studiu sa taktiež nachádza služba, ktorá umožňuje zistenie

[1] https://www.bricklink.com/v3/studio/download.page

presného počtu súčiastok, ktoré boli na modeli použité, a taktiež zistiť približnú cenu týchto súčiastok, ktoré je možné si za pomoci spoločnosti BrickLink objednať.



Obr. 4.1: Model vytvorený v programe BrickLink Studio

Po dokončení modelu je k dispozícií viacero možností, ako daný model exportovať. K dispozícií sú napríklad formáty:

- `.lfml` – umožňuje výmenu modelov so staršou verziou, LEGO Digital Designer
- `.dae` – univerzálny 3D formát, podporuje geometriu, vhodný aj mimo LEGO ekosystém
- `.io` – uloženie celého modelu, automatická možnosť znovu otvoriť v BrickLink Studiu
- `.csv` – export potrebných dielikov

Taktiež je k dispozícií možnosť exportovať už spomínaný plán postupu stavby robota vo formáte `.pdf`. Program tiež ponúka možnosť vytvoriť animáciu stavby a výsledného vzhľadu modelu, ako aj exportovanie obrázku v zvolenom formáte.

4.2 URDF

Unified Robot Description Format (URDF) je jazyk, ktorý slúži na popis modelov robotov [15]. Hlavnou snahou je, aby tento jazyk bol čo najvšeobecnejší, avšak nie je možné, aby

popisoval všetky druhy robotov. Hlavným obmedzením v tomto bode je, že môžu byť reprezentované iba stromové štruktúry, čím sa vylučujú všetky paralelné roboty (Paralelný robot je pohyblivá platforma spojená so stálou platformou za pomoci minimálne dvoch nezávisle pohybujúcich sa nôh [7]). V URDF sa roboty skladajú z nepohyblivých častí, linkov, ktoré sú spojené kĺbmi. URDF je vyžadované pre ďalšie používanie modelu robota, ako napríklad vizualizáciu robota v programe Rviz alebo rôzne simulácie za pomoci Gazebo.

Vývojári Robot Operating System ROS vytvorili v roku 2009 formát, ktorý bol založený na základe značkovacieho jazyka XML [21]. Tento formát slúži na podrobný popis robotov, na popísanie jeho vizuálu, popísanie jeho kolíznych hodnôt, ako aj na určenie rôznych spojov robota a na pohyb okolo určitého kĺbu. Pre jednoduchšie používanie URDF, používanie rôznych makier, vytváranie konštánt alebo výpočty matematických výrazov sa v roku 2012 vytvorila knižnica Xacro. Xacro je knižnica na spracovávanie makier v jazyku XML. Táto knižnica výrazne pomohla pri vývoji a údržbe zložitejších modelov robotov, ktorý boli vytvorení za pomoci URDF.

Každý model robota, ktorý je vo formáte URDF, musí obsahovať koreňový element `<robot>`. Tento element má povinný atribút name. Vnútri tohto elementu sa nachádzajú linky a kĺby, z ktorých sa robot skladá.

Element `<link>` predstavuje časť robota. Tento element určuje všetky vizuálne, ale aj fyzikálne vlastnosti tejto časti. Rovnako, ako aj element `<robot>`, aj tu je jediný potrebný atribút name, ktorý jednoznačne identifikuje časť v rámci modelu. Tento element má aj tri druhy potomkov:

- `<inertial>` – určovanie fyzikálnych vlastnosti, ako napríklad hmotnosť alebo ťažisko
- `<visual>` – tvar a materiál vizuálnej časti robota
- `<collision>` – určenie kolíznej geometrie robota

Každý z týchto potomkov môže byť predkom pre ďalších potomkov. Všetky môžu totižto obsahovať podelement `<origin>`, ktorý v atribútoch určuje posun a rotáciu. Vo `<visual>` a `<collision>` sa bežne používa element `<geometry>`. V tomto elemente sa definuje, aký geometrický útvar sa vykresľuje, respektíve aký geometrický útvar bude použitý pri vytváraní kolíznej plochy robota. Pri tomto elemente je možné vybrať si z týchto listov, ktoré vykreslia:

- `<box>` – kváder o rozmeroch, ktoré zadáte v atribútoch
- `<cylinder>` – cylinder s výškou a polomerom zadanými v atribútoch
- `<sphere>` – guľa s polomerom zadaným v atribúte
- `<mesh>` – externé 3D modely, napríklad vo formáte `.stl` alebo `.dae`

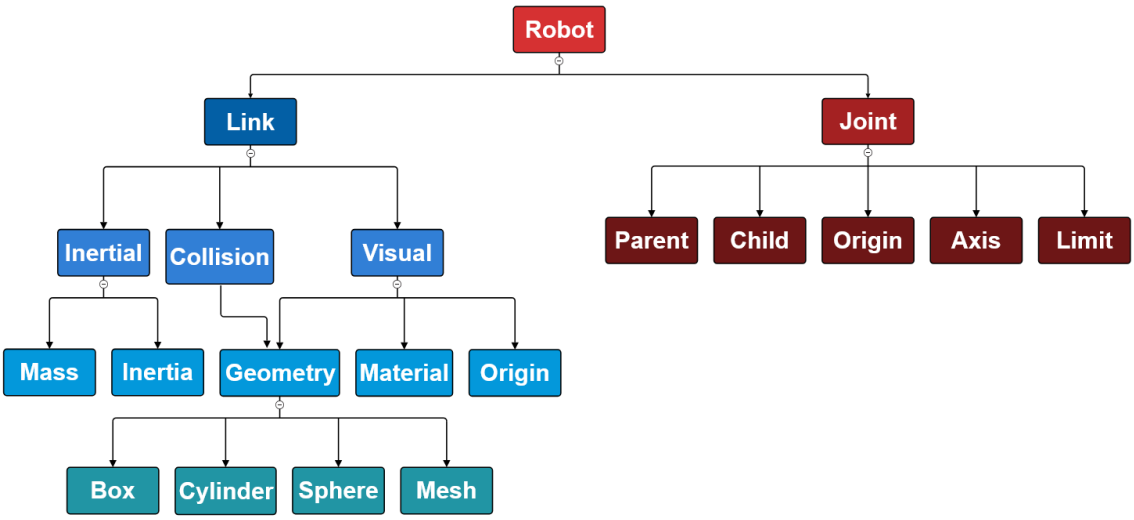
Po vizuálnej stránke dôležitým potomkom je taktiež `<material>`. V tomto podelemente sa môžu nachádzať ďalšie listy `<color>` a `<texture>`, ktoré dovoľujú určiť farbu a textúru zobrazovanej geometrie.

Časti robota však nemôžu existovať samostatne, a je potrebné spojiť ich do jedného celku. Kvôli tomu v URDF existuje element `<joint>`. Tento element má povinne dva atribúty, a to jednoznačné meno kĺbu a následne typ kĺbu. Na základe tohoto typu sa určuje, ako sa bude spojenie dvoch spájaných častí správať. Najčastejšie sa využívajú typy:

- revolute – rotačný kĺb, ktorý má obmedzený uhlový rozsah

- continuous – rotačný kĺb bez obmedzenia uhlového rozsahu
- prismatic – posuvný kĺb, ktorý má obmedzený posuvný rozsah
- fixed – spoj, ktorý je nehybný

Pri kĺboch sú vždy potrebné minimálne dvaja potomkovia, a to `<parent>` a `<child>`, do ktorých je potrebné uviesť jednoznačné názvy častí (linkov). Ďalej môžeme používať podelement `<origin>`, v ktorom sa určuje pozícia a orientácia daného kĺbu. V podelemente `<axis>` sa určuje os rotácie kĺbu a v `<limit>` sa nastavuje rozsah pohybu kĺbu. Kĺby sú pre robotov veľmi dôležité, keďže na základe nich vieme modelovať ich pohyb. Či už ide o modelovanie motorov, ktoré poháňajú kolesá, alebo kĺby pri ohýbaní určitých častí robotov, bez kĺbov by robot bol statický. Na Obr. 4.2 vidíme triviálnu stromovú schému URDF súboru.



Obr. 4.2: Stromový diagram URDF

URDF taktiež ponúka možnosť pracovať s elementami ako `<transmission>`, `<gazebo>` alebo `<sensor>`. Element `<transmission>` slúži na modelovanie väzby medzi motorom a kĺbom, v ktorom vieme pracovať aj s paralelnými pohonmi a prevodmi pomerov [9]. Tento element dokáže určiť, koľkokrát je potrebné, aby sa otočil motor alebo ozubené koliesko, aby sa kĺb pohol napríklad o 1°. Tag `<gazebo>` rozširuje URDF o parametre, ktoré sú potrebné do simulácií, avšak URDF ich nepokrýva. Jedná sa napríklad o trenie alebo tlmenie materiálu. Element `<sensor>` je zatiaľ podľa dostupných zdrojov len v experimentálnej forme[2]. Na základe tohoto elementu by malo byť možné priamo v URDF namodelovať všetky potrebné informácie o senzoroch.

Modelácia robotov v URDF je veľmi užitočná, pokiaľ je cieľom vytvárať simulácie reálneho sveta v programe Gazebo, alebo vytvoriť vizualizáciu robota za pomoci programu Rviz. Je potrebné dbať na to, aby bol model správne proporčne vytvorený, teda aby odpovedal skutočným rozmerom, a taktiež, aby jeho kolízne plochy v modeli boli rovnaké ako v reálnom svete.

[2] https://wiki.ros.org/urdf/XML/sensor/proposals

Kapitola 5

Robot Operating System

Robot Operating System (ROS) je sada open-source knižníc a nástrojov, ktoré sú špecializované na návrh a vývoj aplikácií určených pre robotov. ROS je framework určený na písanie softwaru robotov. Cieľom je zjednodušiť úlohu tvorby komplexného a robustného správania robotov naprieč širokou škálou robotických platforiem. [17] ROS zasahuje široké spektrum užívateľov, keďže ho využívajú amatéri ale aj komerčné spoločnosti. Využitie taktiež nachádza na akademických pôdach a vo výskumných ústavoch. Jeho služby sú teda vhodné aj pre začiatočníkov, ktorí s vývojom programov pre roboty len začínajú, alebo sa majú záujem niečo nové dozvedieť a naučiť, ako aj pre vysoko profesionálne prostredie, ktoré dokáže využiť všetky vymoženosti, ktoré ROS ponúka.

Prototyp Robot Operating Systému vznikol v roku 2007, a to na Stanfordskej Univerzite [5]. Morgan Quigley napísal v rámci projektu STanford AI Robot jadro Switchyard, ktorý dodnes tvorí kostru ROSu. Nápad nadchol spoločnosť Willow Garage Inc., ktorá tento koncept zobrala pod svoje krídla a v mnohom ho rozšírila a upravila. V roku 2010 bola vydaná prvá verzia ROS 1.0, ktorá vo veľkej miere pomohla ROSu preraziť. Od tohto vydania sa ROS stal súčasťou a štandardom pri vývoji robotických softwarov. Od roku 2013 sa OSRF stáva hlavným správcom ROS kódov, ktoré prevzala po Willow Garage, ktorá ukončila svoju činnosť. Pod túto značku prešlo taktiež veľké množstvo hlavných vývojárov systému ROS. ROS bol pôvodne zamýšľaný len pre robotov typu PR2. Keďže sa však uchytil a postupom času sa začal používať pre riadenie viacerých druhov robotov rôznych značiek, nedokázal tento systém pokryť úplne všetky požiadavky rýchlo sa rozširujúcej komunity užívateľov. Niektoré z požiadavok bolo možné implementovať a vylepšiť aj s použitím ROSu, avšak s nárastom novo vyvinutých technológií a požiadavok od užívateľov sa OSRF rozhodla spustiť práce na novej generácií ROSu, ROS 2.0. Prvá oficiálna verzia ROS 2 vyšla koncom roku 2017.

5.1 ROS 1

ROS 1 je open–source nástroj na vývoj programov pre roboty, ktorý beží nad operačným systémom. Tento systém poskytuje komunikačný model medzi procesmi, abstrakciu hardwaru, ale aj správu rôznych knižníc.

ROS 1 pracuje na základe konceptu komunikačného modelu *publisher-subscriber*, kde procesy, nazývané *uzly* spolupracujú za pomoci *tém* a *služieb*.

- Uzol – označovaný ako Node je samostatne bežiaci program, ktorý vykonáva svoju špecifickú úlohu

- Téma – označovaná ako Topic predstavuje publish – subscribe vzor, uzol publikuje správu a ľubovolný počet odberateľov ich môže asynchrónne prijímať
- Služba – označovaná ako Service, spôsob implementácie RPC volaní, jeden uzol vie poslať druhému správu na základe ktorej čaká odpoveď

Centrálnym prvkom celého ROSu je *ROS Master* [1]. ROS Master poskytuje registráciu mien a vyhľadávanie zvyšku komunikačného modelu. Bez jeho prítomnosti by sa uzly v ROS 1 nevedeli navzájom nájsť, a tak si vymieňať správy medzi sebou alebo vyvolávať služby. Master registruje Nodes, Topics a Services. Taktiež poskytuje mechanizmy, ktoré párujú publisherov a subscriberov. Dôležitú časť tvorí taktiež Parameter Server, ktorý ukladá dáta, ku ktorým máju uzly prístup počas celého behu programu.

Uzly medzi sebou komunikujú správami, pomocou ktorých si vymieňajú dáta. Tieto správy môžu byť tvorené rôznymi základnými typmi, ako napríklad `int32`, `float64` alebo `string`. Taktiež sú podporované rôzne zložitejšie typy správ, veľmi používaným je typ `geometry_msgs/Twist`, v ktorom je možné posielať dáta o uhlovej, ako aj lineárnej rýchlosti. Tento typ sa častokrát používa pri riadení pohybov robota.

Pred pustením akéhokoľvek uzlu je potrebné spustiť príkaz `roscore`, ktorý zabezpečí spustenie základných komponentov potrebných pre fungovanie. Príkaz spustí `ROS Master` a `Parameter Server`. Po príkaze `roscore` môže prebiehať medzi uzlami priama komunikácia za pomoci peer-to-peer spojenia, keďže za pomoci `ROS Master` sa dokázali uzly medzi sebou lokalizovať.

Pri používaní Robot Operating System je možné naraziť na pojem `Package`, alebo balíček. Jedná sa o základnú organizačnú jednotku programu v ROSe. Ide o adresár, ktorý obsahuje všetky zdrojové súbory, metadáta potrebné na kompilovanie, ako aj konfigurácie a súbory `.launch`. Balíček môže obsahovať rôzne súbory a adresáre, avšak každý z nich musí obsahovať súbor `package.xml` a `CMakeLists.txt`. Základný balíček sa teda skladá z:

- `package.xml` – metadáta o balíčku ako meno, verzia, popis, licencia a potrebné balíčky
- `CMakeLists.txt` – skript pre Catkin[1], ktorý špecifikuje, ako sa balík kompiluje, linkuje a inštaluje
- `src/` – adresár ktorý obsahuje zdrojové kódy
- `launch/` – adresár pre launch súbory
- `msg/` – adresár na definíciu správ

Pomocou systému ROS sa dajú vytvárať robustné programy určené pre robotov. Táto verzia ROSu podporuje programovanie vo viacerých programovacích jazykoch, a to za pomoci klientských knižníc. Najpoužívanejšie sú však jazyky C++, pre ktorý existuje knižnica roscpp a Python, s knižnicou rospy. Veľkou výhodou je taktiež dostupnosť veľkého množstva knižníc, ktoré je možné použiť pri SLAM, plánovaní pohybu robota, pri rôznych simuláciách v Gazebo, ale aj v rôznych iných oblastiach robotiky. V neposlednom rade je tu taktiež možnosť vizualizácie v programe Rviz, ktorý nám poskytuje možnosť vidieť stav robota a robotických komponentov v danej chvíli.

[1] Catkin je build systém založený na CMake

5.2 ROS 2

Robot Operating System 2 vychádza zo staršej generácie ROS 1, ktorá taktiež slúžila na vývoj aplikácií určených pre robotov [14]. S novou generáciou však prichádza množstvo zmien. Pri vývoji bolo hlavným cieľom zanechať to, čo bolo pri predchádzajúcej verzií kľúčové a čo urobilo ROS populárnym, avšak odstrániť obmedzenia, ktoré ROS mal, aby bol vhodnejší na nasadenie do priemyslu.

Prvou veľkou zmenou je multiplatformová podpora. Pri predošlej verzii bol ROS viazaný primárne len na Linux. Vo verzii ROS 2 je oficiálne podporovaný Linux, Windows aj MacOS, čo otvára dvere pre ďalších potencionálnych užívateľov [20].

Obrovskou zmenou je používanie DDS protokolu. Za pomoci tohoto protokolu bola odstránená nutnosť používať `ROS Master`. Miesto neho sa používa DDS discovery protokol, ktorý zaručí, že uzly sa navzájom objavia bez nutnosti ROS Mastera. Upustilo sa taktiež od používania TCPROS/UDPROS, miesto nich DDS poskytuje rozhranie pre komunikáciu, kde je možné používať ako TCP, tak aj UDP, a to spoločne s QoS[2] vrstvou. Na základe QoS vieme určiť garanciu doručenia, kontrolu oneskorenia, prioritu a iné. Pomáha tak výrazne pri spracovaní dát v reálnom čase, čo bolo veľmi žiadúce v priemysle. Taktiež sa podporilo zabezpečenie, kde v predchádzajúcej verzií neboli žiadne vstavané mechanizmy autentifikácie, autorizácie ani šifrovania. ROS 2 používa DDS-Security, ktorý vedie k výraznému zvýšeniu ochrany v priemyselnom nasadení.

Tieto a ďalšie zmeny urobili ROS 2 flexibilnejším, bezpečnejším a mocnejším prostredím, ktoré je vhodné aj na použitie v priemysle alebo na vývoj robotických aplikácií, ktoré pracujú v reálnom čase.

5.3 Rviz

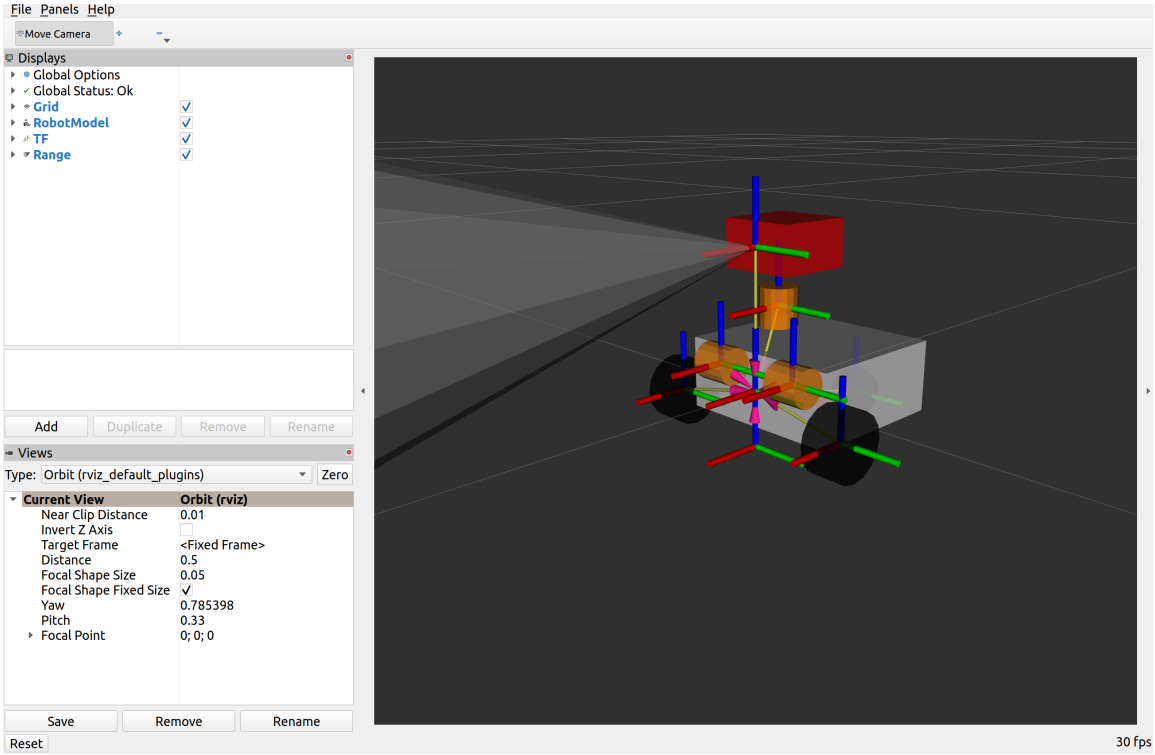
ROS vizualization, Rviz, je univerzálny 3D vizualizačný prostriedok pre roboty a senzory [17]. Ide o nástroj, ktorý je používaný v ROSe na 3D vizualizáciu modelov robotov a celého robotického systému. Zobrazované dáta umožňuje sledovať v reálnom čase, podporuje teda sledovanie, v akom stave sa práve robot nachádza.

Rviz funguje ako samostatný uzol v ROSe, prijíma správy, ktoré boli publikované na témach a zobrazuje ich v 3D priestore. Displej umožňuje zobrazenie rôznych druhov vizualizácií, ako napríklad modelu robota, zobrazovanie transformačných rámcov, vykresľovanie mapy, zobrazenie rozsahu senzora a rôzne ďalšie možnosti typov displeja, ktoré program Rviz umožňuje pridať do vizualizácie.

`RobotModel` umožňuje vizualizáciu modelu robota, ktorý je modelovaný za pomoci URDF. Rviz poskytuje možnosť vidieť stavy kĺbov robota, ich pohyb, ako aj farby, ktoré boli na robotovi použité. Pri použití `joint_state_publisher_gui` sa zobrazí grafické rozhranie, ktoré umožňuje prácu s pohyblivými kĺbmi robota, a tak dáva možnosť sledovať, ako sa robot správa pri pohyboch kĺbmi.

`Tf` umožňuje zobrazenie viacerých 3D transformačných rámcov, ich orientáciu a pozíciu v sledovanom čase. Červenou farbou je zobrazená os X, zelená farba reprezentuje os Y a modrá farba os Z. `Range` zobrazuje rozsah, ktorý je snímaný určitým senzorom robota. Môže teda zobrazovať rozsah, ktorý robot sníma napríklad Lidar senzorom. Na Obr. 5.1 je možne vidieť tieto zobrazenia.

[2] Quality of service



Obr. 5.1: Vizualizácia modelu robota v programe Rviz

Rviz taktiež umožňuje zobraziť vytváranú mapu pridaním typu displeja `Map`, ako aj zobrazenie skenov vytvorených laserom, a to za pomoci typu displeja `LaserScan`. Možností, ktoré Rviz ponúka na zobrazenie, je naozaj mnoho, a preto je často používaný medzi užívateľmi ROS. Využitie nachádza pri prezentovaní výsledkov práce s robotom a predvádzaní funkčnosti, ako aj pri ladení programu. Rozdiel medzi Rvizom a programom Gazebo, ktorý je taktiež veľmi často využívaný v ROSe na simulácie je uvedený v knihe Programming Robots with ROS. Táto kniha hovorí že, Rviz zobrazuje, čo si robot myslí, že sa deje. Na druhej strane Gazebo, rovnako, ako aj fyzický robot, zobrazuje, čo sa s robotom deje v skutočnosti [17].

Kapitola 6

Analýza a ciele

V tejto kapitole sa analyzujú možnosti tvorby programov pre programovateľné kocky LEGO Mindstorms. Zaoberá sa nevýhodami oficiálnych softwarov na programovanie kociek LEGO a dôvodmi používania programov tretích strán. Taktiež sa v tejto kapitole nachádzajú ciele, ktoré boli stanovené po konzultácií s vedúcim bakalárskej práce, a ktorým sa venuje kapitola 7.

6.1 Analýza existujúcich možností

V sekcií 2.3 je popísané programovanie kociek z generácie Mindstorms za pomoci oficiálnych softwarov, ktoré fungujú na princípe drag-and-drop. Tieto softwary však neposkytujú ani zďaleka všetky prostriedky, ktoré sú pri vývoji zložitejších programov potrebné. Nie je možné paralelne riadiť viacero úloh v jednom momente. Taktiež nie je k dispozícií možnosť pracovať so zložitejšími konštrukciami, ako napríklad zanorenými podmienkami alebo cyklami. Vývojové prostredie pre EV3 má taktiež niekoľko chýb, ktorým sa dá pri dlhodobej práci v tomto prostredí len ťažko vyhnúť [19].

Oproti tomu, programy tretích strán umožňujú plnohodnotné používanie obľúbeného programovacieho jazyka. Taktiež je tu k dispozícií veľké množstvo knižníc, ktoré je pri vývoji programov možné použiť. Pomocou programovocích jazykov ako je Python, C a iné, je možné taktiež využívať breakpointy a krokovanie programu. K dispozícii sú taktiež fóra a komunita ľudí, ktorí dokážu veľmi rýchlo poradiť a pomôcť pri vyriešení problému.

Oficiálny software	
Výhody	**Nevýhody**
Vhodné na základné učenie programovania	Proprietárny systém
Jednoduché ovládanie prostredia	Nevhodné na zložité programy
Optimalizované pre LEGO hardware	Obmedzené len na LEGO ekosystém
	Dve generácie = dva softwary

Tabuľka 6.1: Výhody a nevýhody používania oficiálnych softwarov

Tabuľka 6.1, Tabuľka 6.2 a skutočnosti uvedené v nich ma viedli k tomu, aby som pri práci nepoužíval oficiálne softwary, ale využil možnosť programovať roboty za pomoci niektorých z programov tretích strán, ktoré boli spomenuté vyššie, a to konkrétne v sekcií 3.

Programy tretích strán	
Výhody	**Nevýhody**
Textové programovanie vybraným jazykom	Potreba znalosti syntaxe jazyka
Využívanie balíčkov a knižníc	Zložitejšia inštalácia a aktualizácia
Podpora rôzneho hardwaru	Riziko poškodenia hardwaru
Rovnaké vývojové prostredie	Dokumentácia na rôznych stránkach
Možnosť krokovania programu	

Tabuľka 6.2: Výhody a nevýhody používania programov tretích strán

6.2 Cieľ riešenia

Cieľom tejto práce bolo vytvoriť sadu programov pre LEGO robotov s použitím LEGO Mindstorms NXT a LEGO Mindstorms EV3, avšak bez použitia oficiálnych softwarov na to určených. Po dohode s vedúcim mojej bakalárskej práce som si vybral na implementáciu jazyk Python. Pri generácií NXT som použil NXT-Python a pri novšej generácií EV3 som využil Python-ev3dev. S vedúcim mojej bakalárskej práce sme stanovili sadu úloh, ktoré by mali roboty vedieť vykonávať pomocou mnou vytvorených programov. Pre generácie NXT a EV3 sa jedná o úlohy:

- ovládanie robota za pomoci klávesnice (W,A,S,D)
- náhodný pohyb robota po miestnosti
- jazdenie po sledovanej čiare

Taktiež sme sa dohodli na vytvorení programu za pomoci ROSu, ktorý prechádza po miestnosti a v programe Rviz sa vykresľuje mapa sledovaného priestoru v reálnom čase. Program vie zobraziť polohu robota a jeho model na mape, ako aj stále prekresľovať a upravovať mapovaný priestor. Tento program som si dekomponoval, aby som zistil problémy, ktoré bude pri implementácii tejto časti cieľa potrebné vyriešiť. Dekompozícia vyzerá nasledovne:

- vytvorenie modelu robota v URDF
- vybratie vhodnej mapovacej funkcie
- vytvorenie pohybu robota po miestnosti
- zobrazenie modelu robota a jeho pohyb v programe Rviz
- vykresľovanie vytváranej mapy

Kapitola 7

Implementácia

Táto kapitola zobrazuje podrobný rozbor riešenia pre jednotlivé podúlohy, ktoré sú cielené pre generácie LEGO Mindstorms NXT a EV3. Pri vytváraní sád úloh bolo potrebné zachovať všetky body, ktoré sú uvedené v sekcii 6.2.

7.1 Návrh implementácie riešenia

Medzi problémy, ktoré bolo pri jednotlivých podúlohách treba vyriešiť, patrí napríklad:

- návrh robotov, obsahujúcich všetky potrebné senzory pre vytvorenú sadu úloh
- akým spôsobom čítať vstup od užívateľa a ovládať tak robota
- ako zabezpečiť plynulý náhodný pohyb po miestnosti, bez zaseknutia o pásy robota
- ako zistiť farbu čiary, ktorú má robot sledovať
- ako vytvoriť mapovanie za pomoci len ultrazvukového senzoru

Každý z týchto podproblémov bude detailnejšie popísaný v nasledujúcich sekciách.

7.2 Návrh a konštrukcia robotov

Základom celej práce sú dva LEGO roboty, ktoré obsahujú všetky senzory, ktoré sú nutné pre vykonávanie sady úloh. Nie je totižto žiadané, aby užívateľ počas používania sady úloh musel výrazným spôsobom zasahovať do robota. Je teda potrebné, aby roboty mali k dispozícii ultrazvukový senzor, senzor farby a dva dotykové senzory. Pri robotovi z generácie EV3 je taktiež potrebný gyro senzor, ktorý je využívaný pri práci s Robot Operating System.

Ako kostra pre vytvorenie robotov slúžili oficiálne LEGO Mindstorms modely robotov, ako pre NXT[1], tak aj pre EV3[2]. Tieto modely bolo však potrebné za pomoci programu BrickLink Studio[3] upraviť tak, aby bolo možné umiestniť všetky potrebné senzory a rovnako rýchlo a bezproblémovo vymeniť batérie, ktoré poháňajú programovateľnú kocku LEGO Mindstorms. Prvým krokom bolo vymodelovať kostru originálneho robota, z ktorej zostalo umiestnenie motorov a pripevnenie kocky. Následne bolo nutné vymyslieť, ako a kam

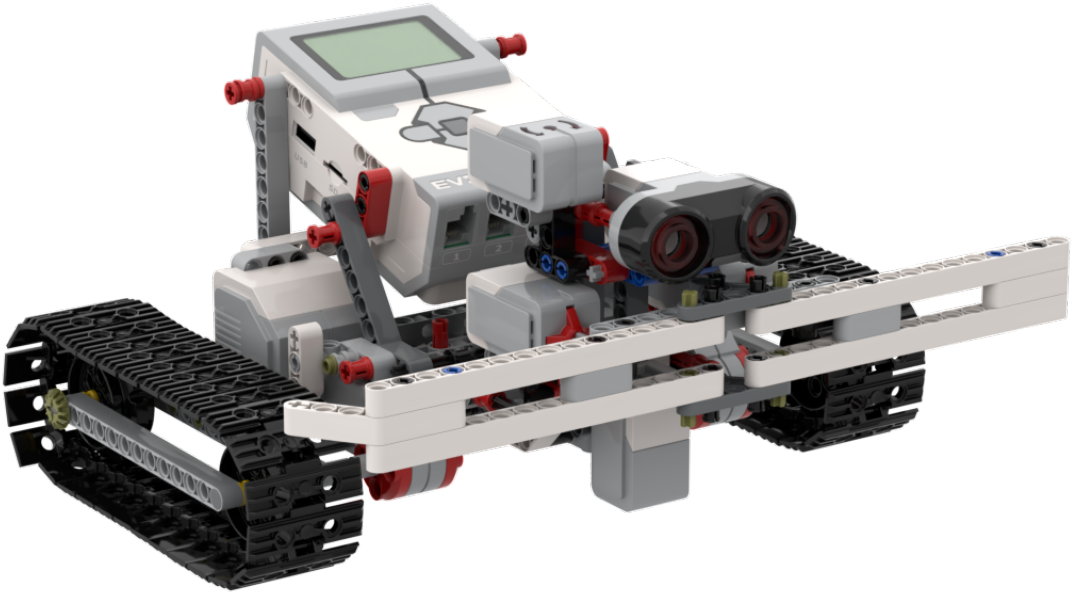
[1] https://www.lego.com/cdn/product-assets/product.bi.core.pdf/4589647.pdf
[2] https://pybricks.com/ev3-micropython/examples/tank_bot.html
[3] https://www.bricklink.com/v3/studio/download.page

pripevniť senzor na detekciu farby, ktorý musí byť otočený smerom dole a musí byť v blízkosti sledovaného povrchu. Taktiež bolo umiestniť oba dotykové senzory tak, aby som k nim vedel pripevniť nárazník, ktorý bude detekovať rôzne nárazy. Taktiež bolo nutné na model pridať ultrazvukový senzor, aby robot vedel snímať okolie. Pri robote z generácie EV3 som musel pridať ešte piaty senzor, a to gyro senzor, ktorý je potrebný len pri práci s ROSom. Všetky tieto senzory museli byť čo najviac stabilné a bolo nutné ich vystužiť tak, aby sa pri pohybe a presune robota nechveli. Po namodelovaní všetkých senzorov bolo ešte treba vytvoriť vizuálne prvky modelov a upraviť farbu jednotlivých súčiastok. Výsledný model

Obr. 7.1: Model robota pre generáciu NXT

robota pre LEGO Mindstorms NXT, pre ktorý bola sada demonštračných úloh vytvorená, je možné vidieť na Obr. 7.1. Na Obr. 7.2 a Obr. 7.3 je možné vidieť výsledný model robota pre LEGO Mindstorms EV3 z programu BrickLink Studio.

Po ukončení procesu modelovania som zostavil roboty podľa návodov z programu BrickLink Studio.



Obr. 7.2: Model robota pre generáciu EV3 spredu

Obr. 7.3: Model robota pre generáciu EV3 zozadu

7.3 Práca s NXT a EV3

Ďalším dôležitým krokom pri tejto práci bolo zistiť, ako správne komunikovať s robotmi. Za pomoci NXT-Python sa dá pracovať s programovacou kockou z generácie NXT. Pri obsluhovaní programovacej kocky EV3 bol použitý Python-ev3dev. Pri NXT, ako aj pri EV3 sú k dispozícií rôzne funkcie a možnosti, ako je možné získavať údaje zo senzorov a ovládať motory robotov. Pri generácií EV3 bola na riadenie robota použitá trieda `MoveTank`. Táto trieda umožňuje určovať rýchlosti pravému a ľavému motoru samostatne. Použitím tejto triedy je možné využívať metódy:

- `on()` – motory idú zvolenými rýchlosťami, pokiaľ nie sú zastavené
- `on_for_degrees()` – motory idú zvolenými rýchlosťami určitý počet stupňov
- `on_for_seconds()` – motory idú zvolenými rýchlosťami zadaný počet sekúnd
- `stop()` – oba motory prestanú bežať

V NXT-Python neexistuje podobná metóda, za pomoci ktorej by bolo možné ovládať asynchrónne viacero motorov naraz, a tak som sa ju rozhodol implementovať sám. Implementácia tejto funkcie používa asynchrónne programovanie v jazyku Python za pomoci knižnice `asyncio`. Týmto spôsobom je možné synchronizovane ovládať pohyby dvoch motorov. Motorom je možné nastaviť rýchlosť otáčania a aj veľkosť uhla otočenia jednotlivých motorov. Vďaka tejto funkcií je možné ovládať oba motory aj pri generácií NXT bez čakania, čo bolo využité aj pri riešení ďalších úloh.

Veľkým rozdielom medzi NXT a EV3, ako už bolo spomenuté v sekcií 3.3.2, bol fakt, že pri NXT beží program na počítači, zatiaľ čo pri generácií EV3 beží program priamo na kocke. Pre ovládanie robotov sa v oboch prípadoch používa GUI vytvorené za pomoci knižníc `tk` a `ttk`. Pri NXT sa vstupy od užívateľa môžu priamo spracovávať na počítači, ktorý vykonáva daný program na ovládanie robota z generácie NXT. Na základe týchto vstupov sa na počítači rozhodne, čo má robot vykonávať, a robot dostáva cez Bluetooth alebo USB príkazy, ktorý má vykonať. Pri generácií EV3 je potrebné zasielať na kocku vstupy od užívateľa, aby robot vedel určiť, čo má vykonávať. Z toho dôvodu sa pri generácií EV3 využíva Klient-Server komunikácia. Táto komunikácia prebieha pomocou TCP protokolu. Na sieťovú komunikáciu pomocou TCP protokolu je použitá knižnica `socket`, ktorá je dostupná v základnej inštalácii jazyku Python. Server počúva na dohodnutom porte a prijíma príkazy, ktoré mu klient odosiela. Klient beží na počítači, za pomoci ktorého sa robot dá ovládať. Program, ktorý sa správa ako klient, funguje tak, že na základe vstupu, ktorý načíta od užívateľa cez GUI, odosiela príslušnú správu na server. Program, ktorý beží na EV3 a správa sa ako server, prijatú správu vyhodnotí a na základe nej sa rozhodne, čo má robot práve v tomto momente vykonať.

7.4 Ovládaný pohyb

Prvou zo sady úloh, ktoré boli vytvorené v rámci tejto práce, je program, ktorý zabezpečí možnosť ovládania robotov na základe vstupu užívateľa na klávesnici. Dôležitou časťou riešenia tejto úlohy bolo zistiť, ako získavať vstupy z klávesnice od užívateľa. K tomu sú použité knižnice `tk` a `ttk`. V rámci tohto programu sa zobrazuje menu, ktoré obsahuje šesť tlačidiel:

- `Forward` – W, pohyb robota vpred
- `Back` – S, pohyb robota vzad
- `Left` – A, pohyb robota vľavo
- `Right` – D, pohyb robota vpravo
- `Brake` – SPACE, akútne zastavenie motorov
- `EXIT` – ESC, korektné ukončenie programu

Program taktiež podporuje aj ovládanie robota za pomoci klávesnice. Následne bolo potrebné vedieť ovládať motory robotov. Pri používaní NXT–Python a Python–ev3dev nebolo ovládanie takéhoto druhu pohybu zložité na implementáciu. Stačí nastaviť rýchlosť otáčania jednotlivých motorov na základe toho, aký vstup prišiel od užívateľa. Algoritmus 1 ukazuje základné funkčné body tejto riešenej úlohy.

Algoritmus 1: Ovládanie robota

Vstup: premenná *char* – stlačený znak z klávesnice
Výstup: Ovládanie pohybu robota

```
1:  ovladam ← True // premenna na ovládanie cyklu
2:  while ovladam do
3:      if char == 'w' then
4:          motory.run(100, 100); // pohyb vpred
5:      end if
6:      if char == 's' then
7:          motory.run(−100, −100); // pohyb vzad
8:      end if
9:      if char == 'a' then
10:         motory.run(−100, 100); // pohyb vľavo
11:     end if
12:     if char == 'd' then
13:         motory.run(100, −100); // pohyb vpravo
14:     end if
15:     if char == ' ' then
16:         motory.brake(); // zastavenie motorov
17:     end if
18:     if char == char(27) then
19:         ovladam ← False; // ukončenie ovládania
20:     end if
21: end while
```

7.5 Náhodný pohyb po miestnosti

Ďalšou úlohou zahrnutou do sady demonštračných úloh ktoré bolo cieľom vytvoriť, bolo vytvorenie programu, ktorý zabezpečí náhodný pohyb po miestnosti. V tejto časti sa pracuje

s ultrazvukovým senzorom, ku ktorému boli neskôr pridané aj dotykové senzory. Pri prvotných pokusoch totižto nastal problém, že robot bol príliš robustný a v niektorých prípadoch nastala situácia, že ultrazvukový senzor nezachytil prekážku, ktorá sa nenachádzala priamo pred ním, ale niekde v oblasti pohyblivého pásu. To zapríčinilo, že robot narazil a snažil sa stále pokračovať vpred, keďže ultrazvukovým senzorom nedetekoval žiadnu prekážku. Tento nedostatok bolo možné jednoducho odstrániť pridaním dotykových senzorov a pripevnením nárazníka.

Na získanie potrebných dát zo senzorov sú pri ultrazvukovom aj dotykovom senzore využívané funkcie, ktoré sú k dispozícii v používaných knižniciach NXT–Python a Python–ev3dev. Program konkrétne využíva funkcie na získanie vzdialenosti predmetu od ultrazvukového senzoru a funkciu na určenie stavu dotykového senzoru. Algoritmus 2 zobrazuje základný popis princípu fungovania tejto úlohy.

Algoritmus 2: Náhodný pohyb

Vstup: premenná *má cestovať* – určuje, kedy má robot náhodne cestovať
Výstup: Náhodný pohyb robota

```
1:  while má cestovať do
2:      v ← get_distance(); // načítanie vzdialenosti od predmetu
3:      z ← is_pressed1(); Or is_pressed2() // zistenie stavov
            dotykových senzorov
4:      if v < 25 Or z then
5:          motory.brake(); // zastavenie robota
6:          motory.run(−100, −100); // robot cúva
7:          smer ← random(0, 1); // generovanie smeru otáčania
8:          uhol ← random(30, 150); // generovanie veľkosti uhlu
                otáčania
9:          if smer == 0 then
10:             motory.turn(100, −100, uhol); // otáčanie robota o uhol
                    vľavo
11:         end if
12:         else
13:             motory.turn(−100, 100, uhol); // otáčanie robota o uhol
                    vpravo
14:         end if
15:     end if
16:     motory.run(100, 100); // pohyb robota vpred
17: end while
```

Program obsahuje menu, ktoré umožňuje spustiť náhodný pohyb, zastaviť tento pohyb alebo ukončiť daný program. Menu je vytvorené za pomoci knižnice `tk`.

Pokiaľ je stlačené tlačidlo náhodného pohybu, robot zistí za pomoci ultrasonického senzoru vzdialenosť od predmetu nachádzajúceho sa pred ním. Pokiaľ je táto vzdialenosť väčšia ako 25 cm a nie je zatlačený ani jeden dotykový senzor, tak robot cestuje dopredu. Keď však príde do stavu, kedy je predmet vo vzdialenosti menej ako 25 cm, alebo je zatlačený aspoň jeden z dotykových senzorov, tak robot mierne zacúva. Tento pohyb je nutný z toho dôvodu, aby robot zbytočne nenarážal do detekovaných prekážok. Po cúvaní je potrebné zabezpečiť náhodné otočenie robota. Toto náhodné otočenie je generované za pomoci dvoch

funkcií `randint`, ktoré vygenerujú smer otáčania a následne veľkosť uhlu, o ktorý sa bude robot otáčať. Po tomto otočení robot znovu zisťuje, či sa môže pohybovať smerom vpred.

Počas behu je možné robota v akomkoľvek bode zastaviť prostredníctvom GUI, čím sa preruší jeho pohyb. Taktiež je v GUI k dispozícií tlačidlo na vypnutie programu, ktoré zastaví všetku činnosť robota a korektne vypne danú úlohu.

7.6 Pohyb po čiare

Poslednou zo sady úloh, ktoré bolo potrebné splniť pre obe generácie robotov, bolo vytvoriť program, ktorý umožní robotovi sledovať čiaru a pohybovať sa pozdĺž nej. Pre správny chod tejto úlohy je potrebné mať pri oboch generáciách senzor farby umiestnený približne vo vzdialenosti 1 cm od sledovaného povrchu, na ktorom sa nachádza čiara.

Pri generácií NXT a knižnici NXT–Python sa využíva funkcia `get_reflected_light()`, ktorá vracia intenzitu odrazeného svetla od 0 po 1023. Taktiež sa využíva funkciu na asynchrónne ovládanie viacerých motorov, ktorá je popísaná v sekcií 7.3. Pri EV3 a knižnici Python–ev3dev sa využíva metóda `reflected_light_intensity`. Za jej využitia je možné získať sledovanú farba , vďaka ktorej sa dá získať percentuálna intenzita odrazeného svetla od skúmaného povrchu. Na ovládanie oboch motorov som používal už vyššie spomínanú triedu `MoveTank`, za pomoci ktorej bolo možné riadiť pohyby robota pri sledovaní čiary.

Táto úloha, rovnako, ako aj predchádzajúce, má k dispozícií menu, ktoré je vytvorené za pomoci knižníc `tk` a `ttk`. V menu je možné vybrať si:

- `Calibrate` – začne sa vykonávať kalibrácia
- `Start following line` – robot začne sledovať nájdenú čiaru
- `Stop following` – zastavenie robota
- `Exit` – korektné ukončenie programu

Taktiež je možné v priebehu chodu programu meniť štyri rôzne parametre. K dispozícií je možnosť meniť rýchlosť, akou sa robot pohybuje a zatáčanie robota. V ďalších parametroch sa dá upravovať minimálna a maximálna intenzita odrazeného svetla, ktoré boli získané robotom pri kalibrácií.

Mnou vytvorený program môže vykonávať dva rôzne druhy pohybu, a to kalibráciu a sledovanie čiary. Pri kalibrácií robot vykoná mierne otočenie okolo vlastnej osi, pričom zbiera hodnoty intenzity svetla z farebného senzoru. Z týchto dát určí minimálnu a maximálnu nameranú hodnotu, ktorá odpovedá čiare, ktorú senzor zachytil a povrchu, na ktorom sa čiara nachádza. Práve vďaka takémuto spôsobu kalibrácie je možné za pomoci programu sledovať čiaru rôznej farby, nielen čiernu na bielom podklade. Tieto získané hodnoty je možné v menu meniť, avšak ich výrazná zmena môže zásadne ovplyvniť chovanie robota.

Po kliknutí na tlačidlo `Start following line` v Gui robot začne vykonávať ďalší druh pohybu, a to sledovanie čiary. Základný princíp tohoto programu na sledovanie čiary je možné vidieť na Algoritmus 3.

V algoritme je možné vidieť, že práve snímaná intenzita odrazeného svetla sa využíva na výpočet toho, ako veľmi sa farba líši od priemeru – od farby, ktorú chceme sledovať. Zo získaného rozdielu sa následne vytvorí normalizovaný rozdiel. Na základe tohoto normalizovaného rozdielu program určí, akou rýchlosťou sa majú točiť motory robota, aby zostali na sledovanej čiare. Na výpočet tejto rýchlosti motorov sa v algoritme používa funkcia

Algoritmus 3: Sledovanie čiary

Vstup: premenná *sledovať čiaru* – určujúca, či má robot sledovať čiaru
Výstup: Sledovanie čiary robotom

```
1:  while sledovať čiaru do
2:      farba ← get_reflected_color(); // načítanie intenzity farby
3:      priemer ← (min_farba + max_farba) / 2;
4:      rozdiel ← farba − priemer; // rozdiel skúmanej farby od
            priemeru
5:      max_rozdiel ← max_farba − priemer; // maximálny možný
            rozdiel
6:      norm_rozdiel ← rozdiel / max_rozdiel; // normalizácia rozdielu
7:      rychlost_praveho ← vypocet_rychlosti(norm_roziel);
8:      rychlost_laveho ← vypocet_rychlosti(−norm_roziel);
9:      motory.run(rychlsot_praveho , rychlost_laveho); // pohyb robota
            po čiare
10: end while
```

`vypocet_rychlosti`, ktorá potrebuje ako parameter normalizovaný rozdiel práve sledovanej farby a priemernej farby. Je potrebné zistiť, či je tento rozdiel kladný alebo záporný, na základe toho sa nastaví premenná sign, ktorá určí výsledné znamienko. Tento normalizovaný rozdiel sa dosadí do rovnice 7.1:

$$s = sign * ((|norm\_rozdiel| - 1)^3 + 1) \tag{7.1}$$

Výsledok rovnice slúži na to, aby program dokázal určiť, ako prudko je potrebné meniť smer jazdy robota. Pokiaľ je rozdiel minimálny, bude aj potreba zatáčania nízka, akonáhle sa však zväčší rozdiel, musí sa zväčšiť aj nutnosť zatáčania robota. Pri riešení tejto úlohy som vyskúšal viacero rovníc, na základe ktorých sa určovala sila zatáčania. Rovnicu 7.1 som použil z toho dôvodu, že mala najlepšie výsledky zo všetkých skúmaných rovníc.

Funkciu potrebnej rýchlosti otáčania motorov vidíme v rovnici 7.2, kde na základe vypočítaného s a od užívateľa získanej rýchlosti a otáčania, ktorú môže nastavovať v menu, program určí, akou rýchlosťou sa musia otáčať motory, aby robot dokázal sledovať čiaru.

$$v = rychlost + s * otacanie \tag{7.2}$$

Na základe týchto rovníc program spracúva práve skúmanú intenzitu odrazu, za pomoci ktorej určí, ako rýchlo má poháňať jednotlivé motory, aby bolo možné pokračovať v sledovaní čiary.

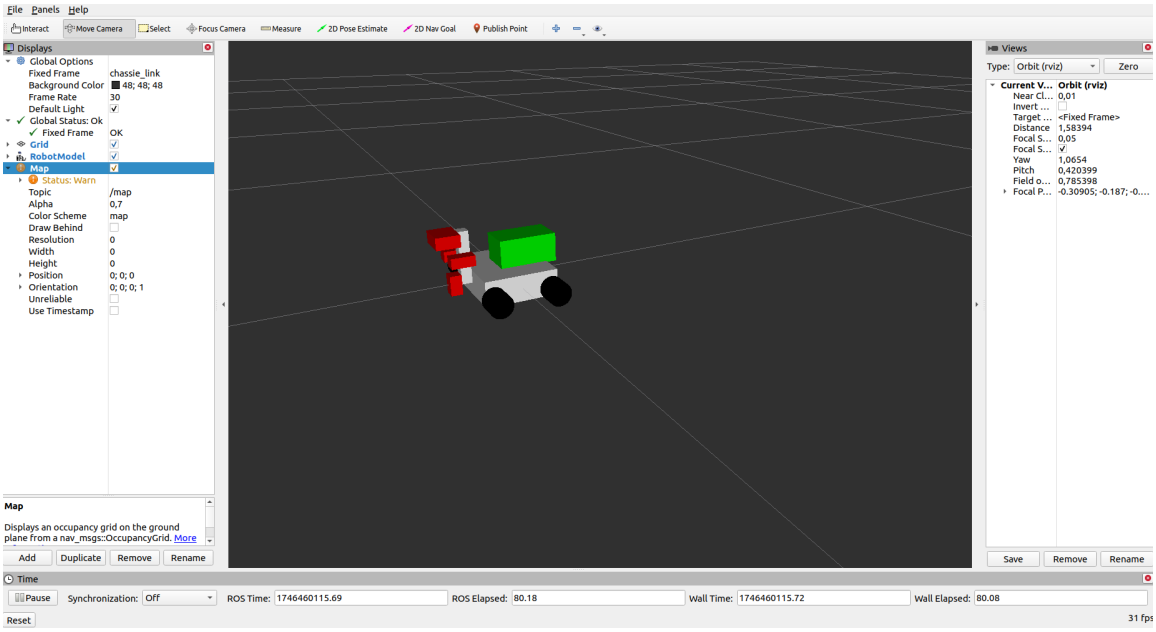
7.7 Mapovanie priestoru

Posledným cieľom, ktorý bolo potrebné v tejto práci splniť, bolo pozrieť sa na prácu s Robot Operating System. S vedúcim práce sme sa dohodli, že implementujem vytváranie mapy za pomoci jedného z generácie robotov LEGO Mindstorms. Rozhodol som sa pracovať s LEGO Mindstorms EV3 kvôli dostupnosti gyro senzoru, ktorý sa pri tomto programe využíva na zisťovanie aktuálneho uhlu otočenia robota. Na základe toho sa uisťuje, že robot mať vždy pravidvé informácie o svojom otočení okolo osi Z. Na vykresľovanie mapy sa používa vizualizačný program Rviz.

7.7.1 URDF model

Prvou časťou z dekompozície, ktorú bolo potrebné vyriešiť, bolo vytvorenie modelu robota, ktorý je možné zobraziť v programe Rviz. Najprv bolo potrebné previesť model robota vytvoreného v programe BrickLink Studio, ktorý je zobrazený na Obr. 7.2, do formátu, ktorý je možné vizualizovať v Rvize. Model robota bol exportovaný z BrickLink Studio vo formáte `.dae` a ten sa za pomoci balíčka `collada_urdf` previedol do formátu `.urdf`, ktorý je možné vizualizovať v Rvize. Tento postup bol popísaný na oficiálnej stránke ROS[4]. Takto vytvorený model sa však v Rvize nezobrazil, preto som sa rozhodol vytvoriť model ručne za pomoci značkovacieho jazyka URDF. Pre jednoduchší vývoj modelu a prácu s matematickými operáciami som pracoval s knižnicou Xacro, ktorá mi umožnila definovať si rozmery jednotlivých častí robota a následne pracovať len s názvami. Na základe toho bolo jednoduchšie upravovať rozmery jednotlivých častí robota. Taktiež mi to umožnilo vykonávať matematické operácie, čo výrazne uľahčilo prácu pri vývoji modelu. Pri vývoji modelu bol kladený dôraz na presnosť rozmerov robota, aby veľkosti častí modelu odpovedali skutočnému robotovi.

Na Obr. 7.4 je možné vidieť výsledný model robota vytvoreného za pomoci URDF zobrazeného v programe Rviz.



Obr. 7.4: Model robota EV3 vytvorený pomocou URDF

Model sa skladá zo štyroch kolies, kostry robota, programovateľnej kocky a sady senzorov. Oproti realite sa líši chýbajúcim nárazníkom a vymenením pásov za kolesá. Tieto odchýlky od reality nemajú žiaden zásadný vplyv na chod programu. Prítomnosť pásov je nahradená dvoma pármi kolies, kde predné kolesá sú pripojené na kĺby, ktoré predstavujú motory zabezpečujúce pohyb robota, rovnako ako je tomu pri použití pásov. Taktiež tu môžeme vidieť všetky senzory, z ktorých najpotrebnejší je ultrazvukový, na základe ktorého sa budú získavať údaje potrebné pre vytváranie mapy.

[4] http://wiki.ros.org/collada_urdf#collada_to_urdf_command-line_tool

7.7.2 Mapovanie

Po vytvorení modelu, ktorý je možné použiť pri vykresľovaní mapy, bolo potrebné vytvoriť program zabezpečujúci vytváranie tejto mapy. Spočiatku som sa snažil použiť na implementáciu `Gmapping`, ktorý by mi vo veľkej miere pomohol pri vytváraní a vykresľovaní mapy. Avšak napriek snahe sa mi nepodarilo správne nastaviť všetky parametre, ktoré boli potrebné pre správne fungovanie a vykresľovanie mapy. K používaniu `Gmappingu` je totižto potrebné používať senzor, ktorý sníma väčší rozptyl. Kvôli tomu som sa pokúsil vytvoriť objekt LaserScan, s ktorým Gmapping pracuje. Avšak objekt LaserScan predpokladá širší uhol snímania (napríklad zo zariadenia Xbox kinect), a teda nie je dobre prispôsobený na snímanie jedným lúčom. Z tohto dôvodu mapa vygenerovaná programom Gmapping za pomoci LaserScan dát reprezentujúcich vzorku jedného lúča neodpovedala realite. Preto som sa rozhodol implementovať si vlastnú funkciu, ktorá bude vytvárať mapu.

Špeciálny mapovací program pracuje s bitmapou, ktorej pixely zodpovedajú častiam priestoru v realite. V bitmape sa ukladajú hodnoty 0 alebo 100 na základe toho, či sa na skúmanom mieste nachádza, alebo nenachádza prekážka. Program počúva na téme, kde robot publikuje údaje o vzdialenosti predmetov získaných z ultrazvukového senzoru a na základe toho do bitmapy vloží na príslušnú pozíciu údaj o prekážke. Na všetky políčka v bitmape medzi aktuálnou polohou robota a pozíciou prekážky robot na základe `Bresenhamovho algoritmu`[5], ktorý slúži na rasterizáciu úsečky, vloží príslušnú hodnotu 0, ktorá odpovedá tomu, že sa na danom mieste nenachádza žiadna prekážka. Hodnotu 100 vloží na miesto, ktoré reprezentuje prekážku. Takto vytvorená bitmapa sa potom pri každej zmene publikuje na tému `/map`, za pomoci ktorého prebieha vykresľovanie mapy v programe Rviz. Princíp fungovania vytvárania mapy je popísaný v Algoritmus 4.

Algoritmus 4: Vytváranie mapy

Vstup: *robot* – obsahuje x, y súradnice robota na mape,
prekazka – obsahuje x, y súradnice pozície prekážky na mape
Výstup: Vytváranie mapy

1: $bitmap \leftarrow Bitmap2d()$; `// inicializácia bitmapy`
2: $map\_pub = Publisher('/map')$; `// vytvorenie publisheru na mapu`
3: **for** $(x, y) \in$ bresenham(robot, prekazka) **do**
4: $bitmap[x,\ y] \leftarrow 0$; `// rasterizácia sledovanej úsečky bresenhamovým algoritmom`
5: **end for**
6: $x,\ y \leftarrow prekazka$; `// nastavenie prekážky`
7: $bitmap[x,\ y] \leftarrow 100$; `// vloženie prekážky do bitmapy`
8: publish bitmap on map_pub; `// posielanie bitmapy na zobrazenie`

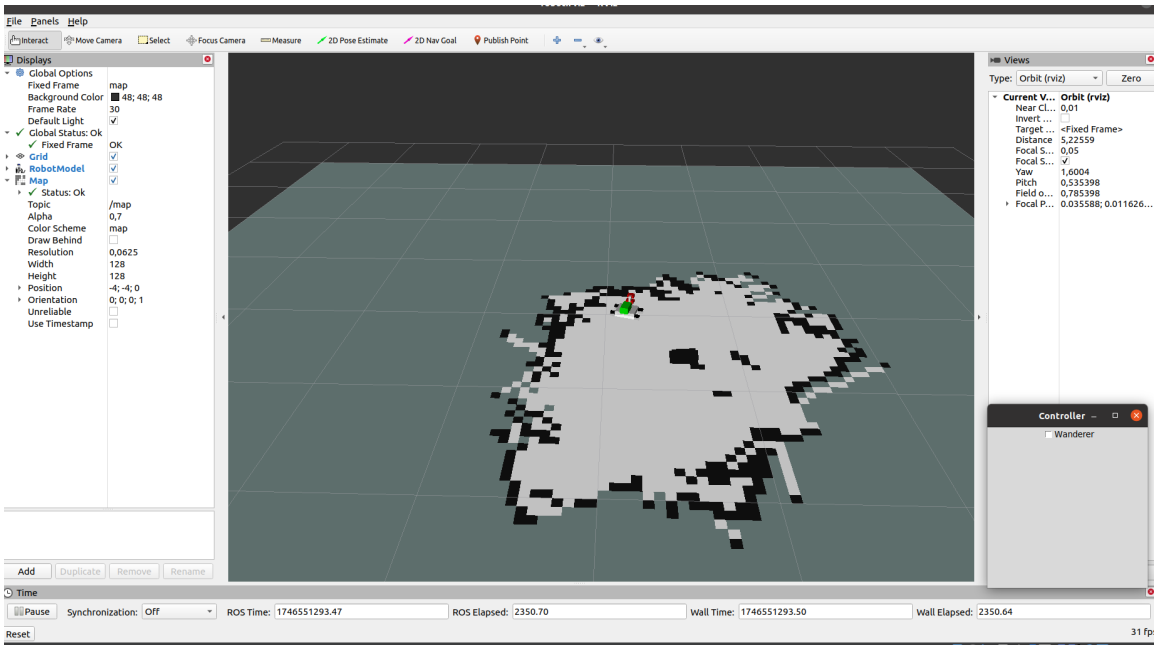
Pre správne vytváranie mapy je nutné zabezpečiť, aby sa robot pohyboval po miestnosti, zbieral dáta o umiestnení prekážok za pomoci ultrasonického senzoru a tie publikoval už na spomínanú tému, na ktorej počúva funkcia vytvárajúca mapu. V programe existujú dva druhy pohybu, akým sa môže robot po priestore pohybovať. Pri spustení programu sa mimo iného zobrazí menu, v ktorom je možné vybrať, či sa má robot pohybovať náhodne, alebo má byť ovládaný kurzorom. Pokiaľ je zvolená možnosť náhodného pohybu po priestore, robot vykazuje správanie podobné, ako bolo popísané v sekcií 7.5. Pohybuje sa teda

[5] https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

náhodne po miestnosti a skenuje priestor, v ktorom sa pohybuje. Pokiaľ je zvolená možnosť ovládania kurzorom, tak program Rviz publikuje na tému pozíciu kliknutia kurzoru, na základe ktorého robot upraví svoj smer pohybu tak, aby sa ku kliknutému miestu dostal najkratšou možnou trasou. Smer sa upraví na základe aktuálnej pozície robota, jeho smerovania a bodu kliknutého v programe Rviz. Za pomoci týchto údajov sa prepočíta, o aký uhol sa má robot otočiť, aby mohol robot priamo pokračovať ku miestu, ktoré mu bolo určené kurzorom na mape. Pokiaľ však v priebehu cesty narazí na prekážku, zostane stáť a čaká na ďalšie podnety k pohybu.

Veľmi užitočnú funkciu v programe zohráva taktiež gyro senzor, na základe ktorého dokáže program veľmi presne určiť, o aký uhol sa robot otočil, čo pomáha pri vytváraní presnejšej mapy. Taktiež je potrebné stále aktualizovať pozíciu, kde sa nachádza robot, na čo slúži samostatné vlákno, ktoré stále zverejňuje pozíciu robota, na základe ktorej je možné vykresľovať jeho pohyb v programe Rviz.

Súčasťou tejto úlohy bol vytvorený skript, ktorý dokáže diaľkovo spúšťať uzly z balíčku na kocke EV3. Tento skript dokáže spúšťať `catkin`, preniesť celý balíček na diaľku na programovateľnú kocku EV3 a pomocou launch súboru spustiť príslušný program na kocke.



Obr. 7.5: Mapa vytvorená pomocou robota EV3 vizualizovaná v programe Rviz

Výsledkom tejto úlohy bol program, ktorý mapuje priestor, po ktorom sa robot pohybuje. Na Obr. 7.5 je možné vidieť vizualizáciu programom vytvorenej mapy, ktorá zodpovedá skúmanému priestoru. V tomto priestore sa nachádzala prekážka, ktorá je jasne identifikovateľná vo vytvorenej mape. Počas celého behu programu sa mapa prekresľuje, a tak dokáže pozmeniť stav už vykreslených políčok mapy.

Lokalizácia pomocou `amcl` v tejto práci nebola riešená, pretože taktiež vyžaduje LaserScan dáta, ktorých použitie pomocou ultrazvukového senzoru nefungovalo ani pri pokuse o použitie `Gmapping`.

7.8 Testovanie

Testovanie je neoddeliteľnou súčasťou vývoja programov a softwarov. Preto bolo potrebné vykonávať testovanie aj pri mnou vytvorených sadách úloh pre generácie robotov LEGO Mindstorms NXT a EV3. Testovanie úloh prebiehalo vo viacerých fázach:

- priebežné testovanie funkčnosti jednotlivých úloh
- testovanie správnosti spracovávania dát
- test splnenia cieľov zadania
- testovanie finálnej verzie sady úloh

Všetky z týchto druhov testovania boli využité v priebehu vývoja a implementácie sady úloh. Testovanie slúžilo na získavanie informácií o správnosti a funkčnosti programu. V prebehu implementácie úloh sa na základe testovania podarilo odhaliť niekoľko chýb. Po odhalení chyby a analýze problému boli chyby odstránené. Po upravení programu a odstránení chyby prebiehalo testovanie opakovane, až do bodu kým testovanie neodhalilo žiadne ďalšie chyby.

7.8.1 Priebežné testovanie funkčnosti jednotlivých úloh

Tento spôsob testovania slúži na zaručenie správneho postupu pri vývoji a implementácií konkrétnych úloh. Na základe jednotlivých výsledkov testov prebiehala úprava kódu a zmeny potrebné na správne fungovanie programu. Pri testovaní jednotlivých úloh bolo potrebné skúmať, či program funguje korektne a vykonáva to, čo mu bolo zadané.

Pri priebežnom testovaní úlohy, ktorá zabezpečí ovládaný pohyb robota po miestnosti bolo nutné testovať správnosť odosielaných a prichádzajúcich príkazov pre robota. Testovanie sa taktiež venovalo správnemu fungovaniu získavania správ o stlačení tlačidla užívateľom. Taktiež bolo otestované správanie robotov, keď boli stlačenie viaceré tlačidlá v jednom momente.

Pri testovaní programu na náhodný pohyb bolo nutné overiť správne určovanie a získavanie dát z ultrazvukového senzoru. Priebežným testovaním bola zistená nutnosť pridať dva dotykové senzory a nárazník, aby sa zabránilo zaseknutiu robota o pohyblivé pásy. Taktiež bolo nutné testovať správnosť chodu programu, pokiaľ robot narazil nárazníkom do prekážky.

Pri sledovaní čiary prebiehalo priebežné testovanie za pomoci vytvorenej dráhy, ktorá sa skladala z rôznych druhov oblúkov, zákrut a zmien smeru. Za pomoci tejto dráhy sa priebežne testovala funkčnosť programu, ako aj použitie vhodnej funkcie, ktorá určuje silu otáčania.

Priebežným testovaním programu, ktorý slúži na vytváranie mapy, bola overená funkčnosť pobyhu robota v programe Rviz. Priebežné testovanie viedlo k nepoužívaniu `Gmapping` a vytvorenie vlastnej funkcie na tvorbu mapy. Testovanie viedlo k overeniu správnosti používania Bitmapy ako využívania `Bresenhamovho algoritmu`. Taktiež bola v priebehu tvorby programu otestovaná správnosť publikovania bodov, na ktoré bolo v programe Rviz kliknuté. Toto testovanie viedlo k zlepšeniu fungovania mapového programu a možnosti ovládať robota za pomoci kliknutia kurzorov v programe Rviz.

7.8.2 Spracovávanie dát + splnenie cieľov zadania

Počas celej implementácie úloh bolo nutné testovať a zisťovať správnosť získaných dát zo senzorov, keďže tieto dáta tvorili výraznú časť funkčnosti programov. Taktiež boli testované funkcie a rovnice, ktoré využívali získané dáta počas behu programov. Bolo potrebné otestovať správnosť spracovávania týchto dát. Taktiež bolo nutné otestovať správnosť spracovávania dát, ktoré boli získané z použitých GUI, ktoré má užívateľ k dispozícií.

Po dokončení každej úlohy zo sady demonštračných úloh bolo nutné zistiť, či výsledný program spĺňuje požiadavky, ktoré boli stanovené. Taktiež bolo nutné overiť funkčnosť daného programu, otestovanie správneho a jednoduchého spúšťania konkrétnej úlohy.

7.8.3 Testovanie finálnej verzie sady úloh

Po implementácií všetkých úloh, ktoré sa mali nachádzať v sede demonštračných úloh prebiehalo celkové testovanie sady úloh. Toto testovanie bolo založené na zistenie funkčnosti všetkých programov ako aj funkčnosti menu pri generácií NXT. Taktiež bola zisťovaná užívateľská prívetivosť programov, ako aj schopnosť užívateľov ovládať dané programy. Na základe finálneho testovania boli odhalené chyby v Gui, ktoré boli následne odstránené. Po odstránení chyby testovanie finálnej verzie prebehlo dôkladne ešte raz. Toto testovanie už neodhalilo žiadne ďalšie chyby.

7.8.4 Testovanie na iných zariadeniach

Pri testovaní boli taktiež využité iné zariadenia z generácie NXT a EV3 ako tie, na ktorých prebiehal vývoj programov. Cieľom týchto testov bolo zistenie, či programy bezproblémovo fungujú aj na iných zariadeniach. Pri testovaní sady úloh pre generáciu NXT mi bol vedúcim mojej práce v škole zapožičaný druhý model robota, na ktorom som bez akýchkoľvek problémov otestoval všetky úlohy a funkcionality mojich programov. Pri generácií EV3 som o pomoc pri testovaní na inom zariadení poprosil kamaráta Michala, ktorý vlastní robota z generácie EV3. Za jeho pomoci som otestoval funkčnosť všetkých vytvorených programov aj na inom ako vývojovom zariadení. Výsledkom testovania na rozdielnych zariadeniach bolo zistenie, že sada úloh bezproblémovo funguje aj na iných ako vývojových zariadeniach.

7.9 Možné rozšírenia

Pri študovaní materiálov a implementácií týchto demonštračných úloh boli zistené informácie, na základe ktorých by bolo možné vytvoriť rozšírenia tejto práce.

7.9.1 Nové demonštračné úlohy

Táto práce je určite možná rozšíriť o vytvorenie veľkého množstva nových úloh. Na základe zistení funkčnosti senzorov a motorov by bolo možné využiť tento hardware aj na vývoj iných demonštračných úloh. Za pomoci senzoru farby a motorov by bolo možné vytvoriť program, ktorý dokáže triediť rôzne Lego súčiastky na základe ich farby. Tieto súčiastky by bolo možné triediť na 6 základných farieb, ktoré senzor sníma. Taktiež by bolo možné vytvoriť program, ktorý bude vedieť zodvihnúť predmet, uchopiť ho a preniesť ho na iné miesto, kde by ho následne odložil. Tento program by potreboval ultrazvukový senzor a motory, ktoré by ovládali ramená a pásy. Tieto ramená by mali na starosti uchopenie predmetu a pásy by sa starali o prenesenie predmetu na iné miesto.

7.9.2 Použitie ROS2 + Gazebo

Môj program na mapovanie priestoru funguje za pomoci ROS. K dispozícií je už však aj ROS2. Bolo by teda možné pretvoriť celý tento program na mapovanie do ROS2, ktorý je novší a poskytuje viac možností pre ovládanie robota.

Ďalšie rozšírenie práce môže byť použiť pri mapovaní priestoru nie len vizualizačný program Rviz ale taktiež využiť možnosť pracovať s programom Gazebo. V tomto programe by bolo možné simulovať prostredie a pohyby robota po miestnosti.

7.9.3 Použitie iného programovacieho jazyku

Pre implementáciu tejto práce bol použitý programovací jazyk Python a ROS. Bolo by však možné rozšíriť túto prácu o implementáciu úloh aj za pomoci iného programovacieho jazyku ako napríklad Java alebo C. Taktiež by bolo možné vytvoriť všetky tieto demonštračné úlohy len za pomoci ROS alebo ROS2.

Kapitola 8

Záver

V tejto práci bolo cieľom vytvoriť sadu demonštračných úloh za pomoci programovacieho jazyka Python pre LEGO Mindstorms, a to konkrétne pre generácie NXT a EV3. Taktiež bolo cieľom práce vytvoriť pre generáciu EV3 program vytvárajúci mapu priestoru s použitím ROSu. Všetky tieto ciele práce sa podarilo v rámci tejto práci splniť. Priebežným vytváraním jednotlivých úloh pre konkrétne generácie robotov a plnením bodov, ktoré boli spomenuté v návrhu implementácie, sa podarilo vytvoriť všetky úlohy, z ktorých sa mala skladať sada úloh pre generáciu NXT a EV3.

Počas tvorenia tejto práce bolo čerpané z množstva materiálov, ktoré sa venovali konkrétnym generáciám LEGO Mindstorms a spôsobu ich programovania. Taktiež boli využité materiály zaoberajúce sa Robot Operating Systemom a jeho využitím. Na základe týchto naštudovaných dát boli následne vytvorené konkrétne úlohy zo sady demonštračných úloh pre obe generácie, a to sledovanie a pohyb po čiare, náhodný pohyb po miestnosti, ako aj ovládanie robota za pomoci klávesnice. Na vytvorenie týchto sád úloh bol používaný programovací jazyk Python, a to konkrétne pri generácií NXT bola používaná knižnica `NXT-Python` a pri generácií EV3 knižnica `Python-ev3dev`. Pri vytváraní programu na tvorbu mapy bol používaný ROS, v ktorom mapovanie priestoru prebiehalo za pomoci mnou vytvorenej funkcie, ktorá používala bitmapu. Pri vykresľovaní dostupnosti políčok sa využíval `Bressenhamov algoritmus`, ktorý sa využíva na rasterizáciu úsečky.

Práca na tejto bakalárskej práci mi dala množstvo zaujímavých a užitočných informácií a skúseností. Medzi najdôležitejšie určite patria získané informácie a znalosti o práci s ROSom, ako aj možnosť vyskúšať si prácu s ním. Táto skúsenosť rozšírila moje obzory a otvorila možnosti práce s robotmi a vytváraním programov na ich ovládanie aj do budúcnosti.

Tento projekt by sa do budúcna dal rozšíriť. Dalo by sa vytvoriť a vymyslieť množstvo nových a zaujímavých úloh, ktoré by mohli roboty vykonávať. Mohli by napríklad rozdeľovať kocky Lega na základe ich farby za pomoci senzoru farby. Taktiež by bolo možné implementovať program, za pomoci ktorého by dokázal robot prenášať objekty z jedného miesta na druhé. Možností pri vymýšľaní ďalších úloh je skutočne veľa. Ďalším možným rozšírením tejto práce by mohlo byť využívanie programu Gazebo pri simulovaní prostredia a pohybu robota po miestnosti.

Literatúra

[1] AmmarAzab. *ROS/Concepts* online. 2022-09-20. Dostupné z: http://wiki.ros.org/ROS/Concepts. [cit. 2025-02-11].

[2] BrickLink. *BrickLink* online. Dostupné z: hhttps://www.bricklink.com/v2/main.page. [cit. 2025-02-05].

[3] Camellia. *LEGO Mindstorms* online. 2023-03-23. Dostupné z: https://www.camellia.xin/LOG/Lego/LegoLog.html. [cit. 2025-01-25].

[4] carrick149. *Using NXT Components With a Micro Controller* online. Dostupné z: https://www.instructables.com/How-to-use-LEGO-NXT-sensors-and-motors-with-a-non-/. [cit. 2025-01-29].

[5] DailyAutomation. *Úvod do ROS* online. Dostupné z: https://www.dailyautomation.sk/uvod-do-ros. [cit. 2025-02-10].

[6] Demidov, D. *Python-ev3dev2* online. 2020. Dostupné z: https://pypi.org/project/python-ev3dev2/. [cit. 2025-02-03].

[7] E MotionSupply.com. *What is a Parallel Robot?* online. 2025. Dostupné z: https://www.e-motionsupply.com/Parallel_Robot_s/3343.htm. [cit. 2025-02-28].

[8] ev3dev.org. *Ev3dev* online. Dostupné z: https://www.ev3dev.org/. [cit. 2025-02-02].

[9] Gelsvartas, J. *URDF Transmissions* online. 2017. Dostupné z: https://wiki.ros.org/urdf/XML/Transmission. [cit. 2025-02-07].

[10] Lego. *LEGO Mindstorms RCX* online. Dostupné z: https://www.lego.com/cdn/product-assets/product.bi.core.pdf/4157492.pdf. [cit. 2025-01-25].

[11] Lego. *Lego Mindstorms EV3 user guide* online. 2013. Dostupné z: https://www.lego.com/cdn/cs/set/assets/bltbef4d6ce0f40363c/LMSUser_Guide_LEGO_MINDSTORMS_EV3_11_Tablet_ENUS.pdf. [cit. 2025-01-29].

[12] Lego. *LEGO MINDSTORMS EV3 Hardware Developer Kit* online. 2023. Dostupné z: https://www.mikrocontroller.net/attachment/338591/hardware_developer_kit.pdf. [cit. 2025-01-29].

[13] leJOS. *The leJOS NXJ Tutorial* online. Dostupné z: https://www.ctestlabs.org/robotteams/leJOSNXJTutorial.pdf. [cit. 2025-02-02].

[14] MACENSKI, S.; FOOTE, T.; GERKEY, B.; LALANCETTE, C.; WILLIAM et al. Robot Operating System 2: Design, architecture, and uses in the wild. *Science Robotics* online, Máj 2022. Dostupné z: https://doi.org/10.1126/scirobotics.abm6074.

[15] PECKA, M. *XML Robot Description Format (URDF)* online. 2023-03-24. Dostupné z: https://wiki.ros.org/urdf/XML/model. [cit. 2025-02-05].

[16] PHILOHOME. *NXT motor internals* online. 2020. Dostupné z: https://philohome.com/nxtmotor/nxtmotor.htm#hubdismantle. [cit. 2025-01-29].

[17] QUIGLEY, M.; GERKEY, B. a SMART, W. D. Programming Robots with ROS. In:. 1. vyd. Sebastopol: O'Reilly Media, Inc., 2015. ISBN 978-1-4493-2389-9.

[18] ROBOTS, G. *Introduction to programming NXT Robots in C with RobotC* online. Dostupné z: https://www.generationrobots.com/en/content/58-robotc-logiciel-programmation-robots-lego-mindstorms-nxt. [cit. 2025-02-02].

[19] SKRZYPEK, D.; ZALEWSKA, A. a SYROCKA, O. *5 Pesky Bugs of EV3 Classroom and How to Fix Them* online. 01. júna 2021. Dostupné z: https://www.robocamp.eu/en/blog/lego-mindstorms-ev3-classroom-bugs/. [cit. 2025-02-20].

[20] THOMAS, D. *Changes between ROS 1 and ROS 2* online. Jún 2017. Dostupné z: https://design.ros2.org/articles/changes.html. [cit. 2025-02-15].

[21] TOLA, D. a CORKE, P. Understanding URDF: A Dataset and Analysis. *IEEE Robotics and Automation Letters* online. IEEE, Marec 2024, zv. 9, č. 5, s. 4479 – 4486. Dostupné z: https://doi.org/10.1109/LRA.2024.3381482.

[22] TURNER WHARFE, C. *LEGO is discontinuing MINDSTORMS in 2022* online. 2022-10-26. Dostupné z: https://www.brickfanatics.com/lego-discontinuing-mindstorms-end-of-2022/. [cit. 2025-01-18].

[23] WATTERS, A. *Lego Mindstorms: A History of Educational Robots* online. 2015-04-10. Dostupné z: http://hackeducation.com/2015/04/10/mindstorms. [cit. 2025-01-14].

[24] WAYNE a LAYNE. *General information about how the LEGO NXT sensors and motors work* online. Dostupné z: https://www.wayneandlayne.com/bricktronics/design-and-theory/. [cit. 2025-01-29].