

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-100863-111124

**VYTVORENIE OVLÁDAČA V PROSTREDÍ ROS PRE
MOBILNÉHO ROBOTA**
BAKALÁRSKA PRÁCA

2023

Filip Loppreis

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-100863-111124

**VYTVORENIE OVLÁDAČA V PROSTREDÍ ROS PRE
MOBILNÉHO ROBOTA**
BAKALÁRSKA PRÁCA

Študijný program: Robotika a kybernetika

Názov študijného odboru: kybernetika

Školiace pracovisko: Ústav robotiky a kybernetiky

Vedúci záverečnej práce: Ing. Michal Dobiš

Konzultant: Ing. Michal Dobiš

Bratislava 2023

Filip Lobpreis



ZADANIE BAKALÁRSKEJ PRÁCE

Autor práce:	Filip Lobpreis
Študijný program:	robotika a kybernetika
Študijný odbor:	kybernetika
Evidenčné číslo:	FEI-100863-111124
ID študenta:	111124
Vedúci práce:	Ing. Michal Dobiš
Vedúci pracoviska:	prof. Ing. Jarmila Pavlovičová, PhD.
Miesto vypracovania:	Ústav robotiky a kybernetiky
Názov práce:	Vytvorenie ovládača v prostredí ROS pre mobilného robota
Jazyk, v ktorom sa práca vypracuje:	slovenský jazyk
Špecifikácia zadania:	Mobilná robotika využívaná v kombinácii s logistickými alebo servisnými úkonmi sa stáva čoraz viac populárnejšou. Úlohou študenta je naštudovať si mobilné robotické zariadenie, ktoré bude mať k dispozícii na Národnom centre robotiky a k nemu príslušné materiály. Študent bude pracovať s reálnym hardvérom a otvoreným systémom, ktorý bude potrebné preštudovať a pochopiť jeho fungovanie. Cieľom práce bude následne vytvoriť nadradený ovládač implementovaný v ROS (Robotickom operačnom systéme), ktorý bude schopný riadiť daného robota.
Úlohy:	<ol style="list-style-type: none">1. Analyzujte súčasný stav riešenia a prostredie Robotického operačného systému.2. Analyzujte možnosti a metodiku implementácia riadiaceho balíka pre daný robot3. Navrhnite spôsob implementácie a architektúru riešenia4. Implementujte riadiaci systém pre mobilného roba5. Vypracujte dokumentáciu k dosiahnutým výsledkom.6. Vyhodnote dosiahnuté výsledky.
Termín odovzdania práce:	02. 06. 2023
Dátum schválenia zadania práce:	
Zadanie práce schválil:	

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Robotika a kybernetika
Autor:	Filip Lobpreis
Bakalárska práca:	Vytvorenie Ovládača v prostredí ROS pre mobilného robota
Vedúci záverečnej práce:	Ing. Michal Dobiš
Konzultant:	Ing. Michal Dobiš
Miesto a rok predloženia práce:	Bratislava 2023

Mobilný robot vykonáva také pohyby aké mu zadáme. Preto aby tento spôsob bol čo najefektívnejší, musí byť čitateľný dobre tak ako pre robot tak aj pre užívateľa. Našou úlohou je vytvoriť také prostredie na ovládanie robota, tak aby bol kód efektívny a jednoduchý. Tým pádom bude čitateľný pre ľudí aj robota. Rozhodli sme sa pre ROS druhej verzie. To prečo a ako sme spravili jednotlive časti sa dočítate d'alej.

Kľúčové slová: ROS, BlackMetal, uzly, témy, služby, akcie

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Robotics and cybernetics
Author:	Filip Lobpreis
Bachelor's thesis:	Implementation of ROS Driver for Mobile Robot
Supervisor:	Ing. Michal Dobiš
Consultant:	Ing. Michal Dobiš
Place and year of submission:	Bratislava 2023

The mobile robot performs such movements as we give it. Therefore, for this method to be as effective as possible, it must be readable both for the robot and for the user. The major task is to create such an environment for controlling the robot, so that the code is efficient and simple. Thus, it will be readable by both humans and robots. We decided for ROS2. You can read more about why and how we made the individual parts in the next sections.

Keywords: ROS, BlackMetal, nodes, topics, services, actions

Pod'akovanie

I would like to express a gratitude to my thesis supervisor.

Obsah

Úvod	1
1 ROS	2
1.1 Základné pojmy	2
1.1.1 Témy	2
1.1.2 Služby	3
1.1.3 Akcie	3
1.2 Parametre	4
1.3 ROS1	4
1.4 ROS2	5
1.5 Rozdiely	6
1.5.1 Štandard jazyka	6
1.5.2 Inicializácia nody(uzla)	6
1.5.3 Komunikácia	7
1.6 Parametre	7
1.6.1 Nodelet alebo komponent	7
1.6.2 Kompilácia	7
1.6.3 Vlákna	8
2 Robot	9
2.1 Hardware	10
2.2 Komunikácia s robotom	10
2.2.1 Logovanie	11
2.2.2 Ovládanie	11
2.3 Par slov k parametrom reťazca	12
3 Správanie programu	14
3.1 Uvod do citania grafu	15
3.2 Uzly	15
3.3 Vstup	15
3.4 Komunikacia s robotom	16
3.5 Odometria	16
3.6 Zdielanie polohy	16
4 Oprava chýb na robote	17

4.1	Nesprávna funkcia	17
4.2	Zašumený výstup	18
5	Získavanie rýchlosťí robota	20
	Záver	24
	Zoznam použitej literatúry	25

Zoznam obrázkov a tabuliek

Obrázok 1.1	Vizualizácia témy v ROSe [1]	2
Obrázok 1.2	Vizualizácia služby v ROSe [1]	3
Obrázok 1.3	Vizualizácia akcie v ROSe [1]	3
Obrázok 1.4	Porovnanie štruktúr ROS1 a ROS2 [3]	4
Obrázok 2.1	Zobrazenie spodnej časti mobilného robota [5]	9
Obrázok 2.2	Schéma zapojenia jednotlivých častí na robote	11
Obrázok 3.1	Graf vykonávania programu na ovládanie robota pomocou ROS2. . .	14
Obrázok 4.1	Ustálené hodnoty rýchlosťi ľavého motora.	18
Obrázok 4.2	Ustálené hodnoty rýchlosťi pravého motora.	19
Obrázok 4.3	Prechodová charakteristika rýchlosťi kolies [5].	19
Obrázok 5.1	Získanie prevodu z impulzov na rýchlosť v SI jednotkách.	20
Obrázok 5.2	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.9$. .	21
Obrázok 5.3	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.7$. .	21
Obrázok 5.4	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.75$.	22
Obrázok 5.5	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.8$. .	22
Obrázok 5.6	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.8$ a frekvenciou 10Hz.	23

Zoznam skratiek

API Application Programming Interface

DDS Data Distribution Service

IPC Inter Process Communication

JSON JavaScript Object Notation

LAN Local Area Network

ROS Robot Operating System

Úvod

Táto bakalárska práca popisuje ako naprogramovať ovládač pre mobilného robota pomocou druhej verzie Robotického Operačného Systému (ROS2). Ovládač obsahuje funkcie ROS2 napr. uzly, parametre, služby a témy. Hlavným cieľom tohto projektu je vytvoriť rozhranie tak, aby bolo možné ovládať robota pomocou jednoduchých príkazov cez ROS.

1 ROS

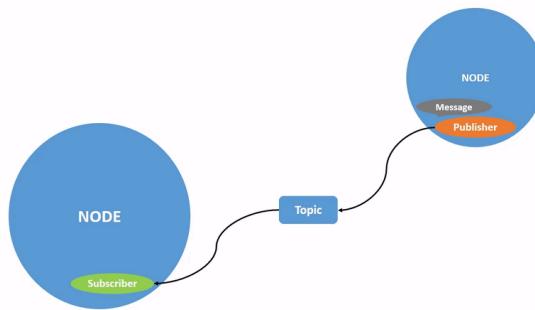
Robotický operačný systém (Robot Operating System) je súbor voľne dostupných softvérových knižníc a nástrojov, ktoré vytvárajú vhodné podmienky pre programátorov na písanie aplikácií pre mnohé druhy robotov. ROS má dve verzie. Vo všeobecnosti sa stretнемe s tým, že pod názvom ROS1 alebo ROS sa myslí ROS verzie 1. Pod názvom ROS2 sa myslí ROS verzie 2. Aby nenastali nejasnosti budeme v tomto dokumente označovať ROS verzie 1 ako ROS1 a ROS verzie 2 ako ROS2. V prípade, keď budme hovoriť o spoločných vlastnostiach a funkcionalitych, ROS1 a ROS2 budeme označovať dokopy ako ROS.

1.1 Základné pojmy

Komunikácia v ROSe je zabezpečená cez IPC (Inter Process Communication), TCP/IP UDP/IP komunikáciou pomocou troch zakladacích metód: **témy** (Topics), **služby** (Service) a **akcie** (Actions).

1.1.1 Témy

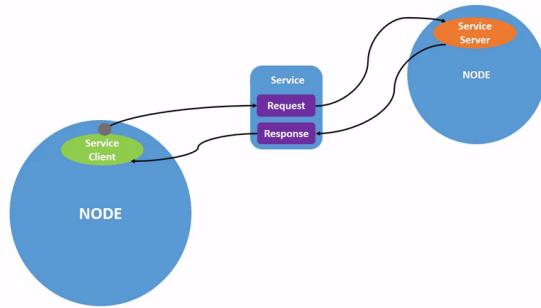
Témy sú sprostredkované pomocou IPC - Medzi procesová komunikácia z anglického Inter Process Communication. Je to najjednoduchší spôsob komunikácie. Vieme si ich prirovnáta k UDP/IP protokolu, s tým že neprebiehajú cez siet'. Definujeme si jedného poskytovateľa (publisher) a jedného alebo viacerých príjemcov (subscriber). Medzi týmito dvoma alebo viacerými účastníkmi sa následne posielajú správy (messages), ktoré sme si dopredu definovali.



Obr. 1.1: Vizualizácia témy v ROSe [1]

1.1.2 Služby

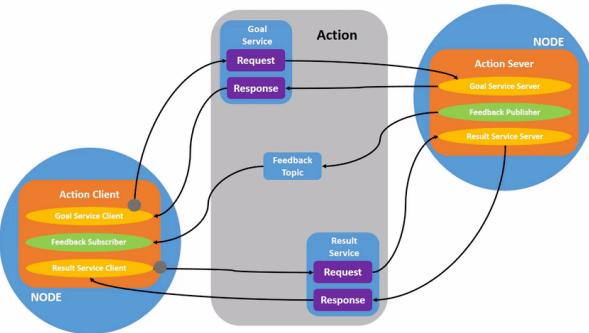
Služby sú sprostredkované pomocou TCP/IP protokolu. Poskytujú nám rovnaký spôsob komunikácie ako témy, až na to, že sa správy medzi servisom a klientom posielajú cez LAN (Local Area Network) a oboma smermi. Služby sa využívajú pri komunikácii medzi viacerými zariadeniami.



Obr. 1.2: Vizualizácia služby v ROSe [1]

1.1.3 Akcie

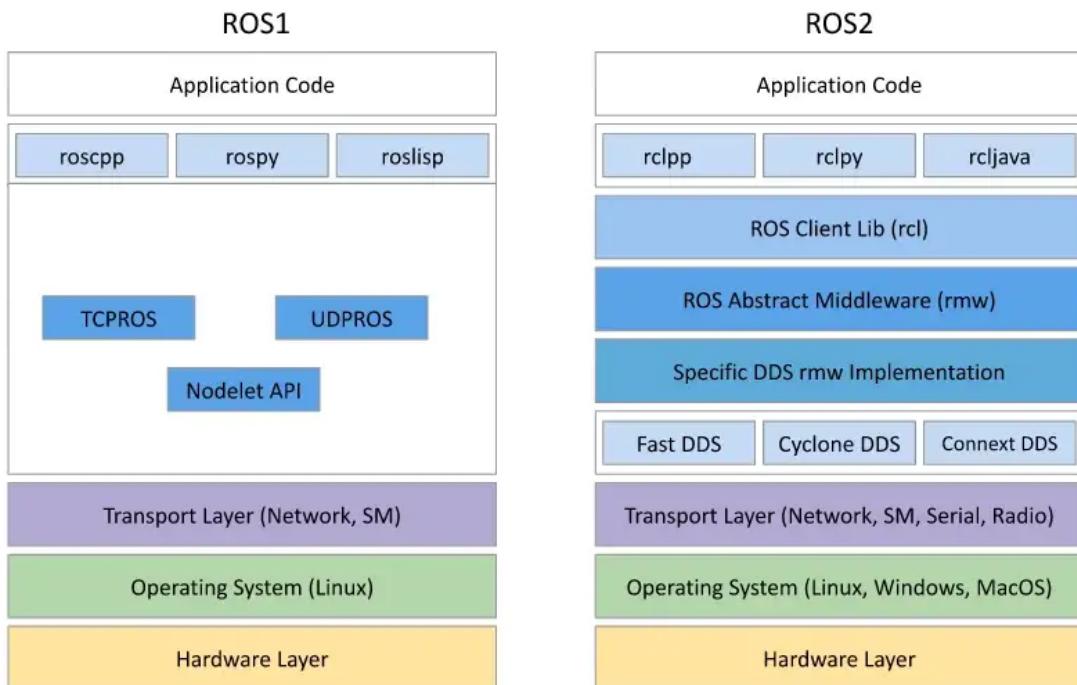
Akcie sú taktiež sprostredkované TCP/IP protokolom. Sú najzložitejším spôsobom komunikácie. Tento spôsob bol pridaný do ROS1 až neskôr. V druhej verzii ROSu je tento typ komunikácie medzi troma základnými. Sú založené na službách a prebiehajú asynchronne [2]. Máju 3 stavy vid' Obr. 1.3. Najprv pošle klient serveru, akú akciu má vykonat', server mu potvrďí, že túto požiadavku dostal. Server začne následne vykonávať danú akciu a posielat' klientovi priebežné správy o priebehu vykonávania žiadanej úlohy. Ked' server skončí pošle klientovi výsledok akcie a klient mu obratom potvrď obdržanie výsledku.



Obr. 1.3: Vizualizácia akcie v ROSe [1]

1.2 Parametre

Parametre sú spôsob, ako môže komunikovať užívateľ so základnými nastaveniami programu bez potreby zmenenia kódu a jeho následnej kompliacie. Definujú sa v *yaml* konfiguračnom súbore. V ňom si môžeme zadefinovať mená jednotlivých parametrov a ich základné hodnoty. Tie si programátor vie v programe vytiahnuť pomocou API, Application Programming Interface, (Aplikačné Programovacie Rozhranie) v ROSe.



Obr. 1.4: Porovnanie štruktúr ROS1 a ROS2 [3]

1.3 ROS1

ROS bol prvýkrát vydaný v roku 2007. Ide o softvér, ktorý sa začal vyvíjať so zámerom zjednodušiť programovanie a ovládanie robotov. Od doby, kedy vznikol prešiel mnohými verziami a úpravami. Jeho neoddeliteľnou súčasťou je štrukturovanie programu do uzlov (nodov), komunikácia medzi uzlami, podpora viacerých programovacích jazykov ako sú C, C++ alebo Python a vytváranie balíčkov dostupných širokej verejnosti.

Štrukturalizovanie základov ROS1 je spravené monoliticky čo najstabilnejším spôsobom. Na počiatku musí byť spustený hlavný program (roscore), ktorý zabezpečuje vytváranie jednotlivých uzlov. Komunikácia medzi uzlami je zabezpečená prostredníctvom prepojenia uzlov cez LAN/WLAN alebo IPC komunikáciu. Ak sú uzly spustené na iných zariadeniach, tak sa využíva len komunikácia cez siet'. Roscore d'alej poskytuje parametre jednotlivým uzlom z parametro-

vého servera. Jeho najdôležitejšou úlohou je zabezpečenie komunikácie uzlov v programe.

Aj napriek mnohým výhodám má ROS1 aj nedostatky, ktoré sa t'ahajú už od jeho počiatkov. Sú to napríklad:

- Nepostačujúca distribuovanosť systému. Všetky uzly sa spoliehajú na funkčnosť roscore-u,
- ROS1 je písaný v starom štandarde, to vnáša do programu technologický dlh a bezpečnostné riziká,
- Kvalita komunikácie sa nedá ovplyvniť,
- Preddefinované vláknové moduly [4].

Kvôli takýmto problémom a nedostatkom sa začala vyvíjať nová verzia ROSu, ROS2. Tá mala vyriešiť tieto problémy a zlepšiť funkcionalitu prvej verzie. V roku 2025 sa skončí podpora poslednej distribúcie ROS1 menom *Noetic*. Preto je odporúčané začínať nové projekty v ROS2.

1.4 ROS2

Ako už bolo spomenuté zámerom vývoja ROS2 bolo zlepšenie funkcionality a bezpečnosti systému. ROS2 nie je späť kompatibilný. Podstata toho, ako sú zoskupované uzly a ako spolu komunikujú je diametrálne odlišná od ROS1. Z tohto dôvodu bol vyvinutý takzvaný rosbridge, ktorý zabezpečuje kompatibilitu medzi verziami. Nie je to ale trvalé riešenie. Odporúčané je nástroj využívať a počas toho prepisovať kód z verzie 1 do verzie 2. Komunikácia prebieha v ROS2 rovnakým spôsobom ako v ROS1. Pomocou tém, služieb a akcií.

Táto podobnosť končí na najvyššej vrstve. Ako sme videli na Obr. 1.4. Štruktúra ROS2 je rozdelená do viacerých vrstiev. Najdôležitejšie je pre nás vedieť, že komunikácia je spracovávaná modelom DDS (Služba distribúcie údajov) z anglického (Data Distribution Service). Tento model zlepšuje výkon, stabilitu a bezpečnosť modelu oproti ROS1. Je založený na TCP/UDP protokole. Z obrázku vyčítame aj lepšie rozloženie modulov. To zabezpečuje jednoduchšie prispôsobovanie systému pre nové funkcionality. Podpora operačných systémov sa v ROS2 rozšírila aj o Windows, Mac OS či operačné systémy reálneho času. Operačné systémy nie sú jediné rozšírenie ohľadom kompatibility. S ROS2 je možné programovať už aj v Java či Matlabe. Tvorcovia mysleli aj na programátorov a pridali rozšírené možnosti testovania, debugovania či nasadzovania programu do reálneho využitia.

ROS2 má necentralizovanú štruktúru, a preto pri spúštaní programov už nie je potrebné mať spustený roscore. Ak teda spadne jeden proces druhé budú fungovať nadalej. V ROS1

sme vedeli ovplyvniť počet uchovaných správ pokým nepretiekol zásobník, ktorý ich uchovával na neskoršie použitie. V ROS2 vieme zmeniť kvalitu komunikácie. Vieme si zadefinovať, či by sme radšej stratili niektoré správy, ale dostali by sme všetky rýchlo. Alebo aby sa zabezpečilo, že dostaneme všetky správy, ktoré boli vyslané, aj keby to trvalo dlhšie. Dokonca si vieme zadefinovať maximálny čas, ktorý budeme čakať na ďalšiu správu.

Pri všetkých týchto zlepšeniach nemôžeme zabudnúť spomenúť aj nasledovný nedostatok. Ked'že ROS2 je mladší ako ROS1 nájdeme k nemu menej dokumentácie. Pridaním veľkého počtu funkcionálít začal vznikať problém pre začiatočníkov s porozumením niektorých kódov. Avšak tento problém je nedostatkom, ktorý časom zanikne. V čase písania tejto prace pribudli na stránke dokumentácie 2 strany popisujúce pokročilejšie Funkcionality druhej verzie ROSu.

1.5 Rozdiely

Čo je určite dobrou správou pre všetkých programátorov, ktorí robili v prvej verzii a sú zvyknutí na jej štandardy a funkcionality sa nemajú čoho obávať. Prechod z ROS1 na ROS2 je dosť priamočiary. Čo sa zmenilo je spôsob písania kódu, ale koncepty ostali všetky rovnaké. V tejto sekcii nebudeme písat konkrétné kódy, budeme len opisovať čo je podobné a čo zasa rozdielne medzi verziami spomínaného systému. Ked'že celý projekt bol písaný v programovačom jazyku C++ tak sa aj tieto zmeny budu týkať hlavne C++.

1.5.1 Štandard jazyka

Pokým ROS1 bola písaná v štandarde C++03 tak ROS2 je už písaná v novom štandarde. A to hlavne C++11, ale pouziva aj nejake casti z C++14 a C++17. To zahrňa inicializovanie templatov a ich používanie. Definície a deklarácie templatov sú na knihu samú o sebe, preto do detailov nebudeme zachádzat. Stačí nám vedieť, ako ich inicializovať. V prvej verzii sme definovali všeobecného publishera (publikovateľa) a definovali sme mu len cez akú tému má posielat správy. V druhej verzii naväzujeme publishera na špecifický tip správy akú posielame. Nemôže sa teda stať, že takýto program by sme skompilovali a následne, ked' ho spustíme, tak by spadol z dôvodu, že čítame iný typ spravy ako posielame.

1.5.2 Inicializácia nody(uzla)

Tak isto ako v prvej verzii aj v druhej verzii musíme definovať uzol (node). Rozdiel je v tom, že prvá verzia obsahovala NodeHandle a druhá verzia obsahuje priamo Node. V druhej verzii je zaužívaným štandardom túto nodu prededit' a použiť polymorfizmus pri objekte, ktorý bude existovať počas celej doby vykonávania programu. Pri prvej verzii tomu tak nebolo. Museli sme vytvoriť už spomenutý NodeHandle. Ten sa nemusel využiť ako base trieda a nemusel ani existovať počas celého behu programu.

1.5.3 Komunikácia

DDS (Služba distribúcie údajov) je protokol strednej vrstvy (middleware) implementovaný nad UDP [2]. Je používaný v ROS2 na komunikáciu medzi uzlami. Je to systém správ publikovania (publish) / odoberania (subscribe), ktorý umožňuje uzlom komunikovať medzi sebou bez toho, aby poznali identitu ostatných uzlov. Druh komunikácie je v ROS2 rozšírený ešte o akcie vid' 1.1.3.

1.6 Parametre

ROS1 používa parametrový server, ktorý sa nachádza v roscore-e. Každý uzol si mohol vytiahnuť parametre, ktoré boli zapísané v konfiguračnom súbore. ROS2 žiadny roscore nemá, preto sa parametre musia distribuovať iným spôsobom. Parametre v druhej verzii ROSu patria jednotlivým uzlom. To znamená, že jednotlivé parametre sa dajú vytiahnuť len daným uzlom. Tieto parametre taktiež existujú len počas existencie daného uzlu. Parametre sú d'alej distribuované pomocou už spomínaného DDS protokolu.

1.6.1 Nodelet alebo komponent

ROS1 ponúka možnosť definície uzlov ako uzlík (nodelet). Je to definovanie uzlu ako zdielanej knižnice (shared library). Je to spôsob ako ul'ahčiť prácu CPU. Keď sa definuje uzol ako uzlík, tak jeden proces môže spracovať programy z viacerých takýchto uzlíkov. Táto funkcia sa nachádza aj v ROS2. Volá sa komponent (component). Vylepšením oproti nodelet-om je zjednotenie aplikačnej implementácie (API). Pokým nodelet-y mají vlastný spôsob implementácie v ROS1 tak v ROS2 je implementácia uzla a komponentu rovnaká. Pri komponente sa musí len naviac definovať, že daný komponent existuje pomocou makra. Použitie komponentov zjednodušuje prácu CPU a používa sa hlavne v zariadeniach, ktoré majú obmedzený výkon výpočtovej techniky.

1.6.2 Kompilácia

Zmenou verzii sa zmenil aj spôsob kompliacie programu. ROS1 bol kompliovaný pomocou `catkin build` systému. Catkin je založený na programe `cmake`. Jeho nastavenie dependencií je konfigurované pomocou súboru `package.xml`. ROS2 prešiel na viac nastaviteľný systém `Colcon`. Tento systém je na rozdiel od catkin-u založený na Python-e a jeho dependencie sa nastavujú pomocou `setup.py` súboru. V prípade colcon-u si môžeme definovať spôsob kompliacie to znamená, že môžeme nastaviť, ako sa budú spracovať dependencie. Ponúkané možnosti sú `catkin_make`, `catkin_make_isolated`, `catkin_tools` a `ament_cmake`. Jednou s najviac používaných možností je `ament_cmake`. Je založený na programe `cmake` a spolupracuje so systémom `colcon`. Z tohto dôvodu mu vieme definovať dependencie pomocou xml súboru ako tomu bolo v ROS1 pričom možnosť definície pomocou Python skriptu

ostáva. Je to jeden zo spôsobov, ako zmenšiť rozdiel medzi ROS1 a ROS2.

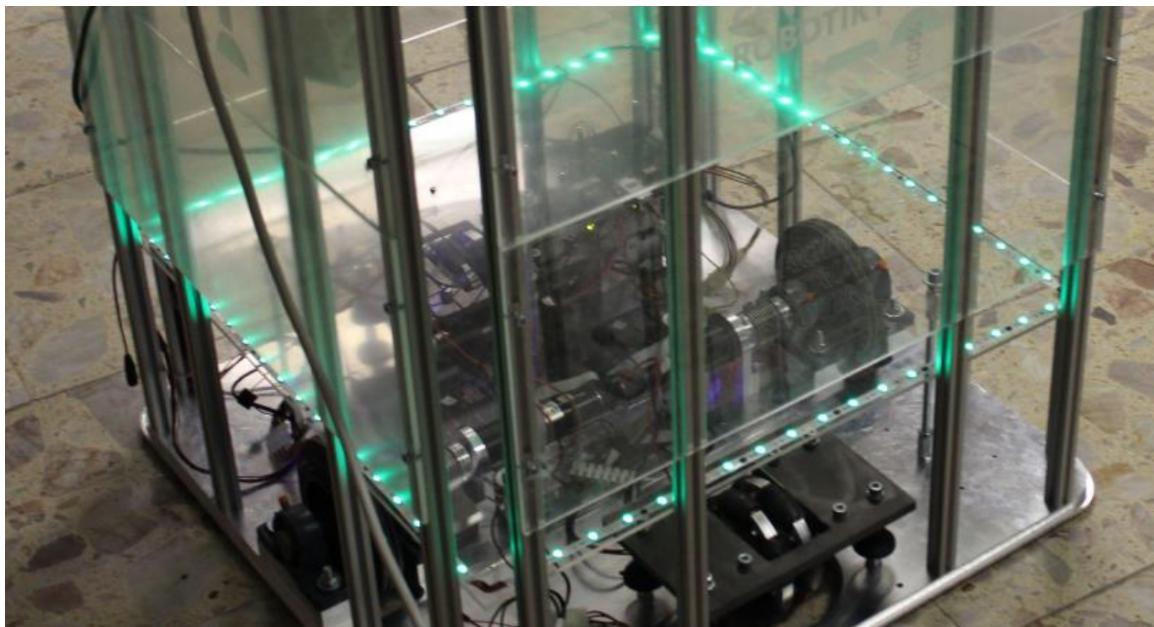
1.6.3 Vlákna

ROS1 dovoľuje programátorom vybrať si medzi jedno vláknovým a viac vláknovým vykonávaním programu. Tvorcovia ROS2 si dali zaležať na modularite aj tejto oblasti kódu. V druhej verzii ROS-u si vieme zadefinovať typ vykonávania programu separátne pre každý uzol a vieme si tento typ zadefinovať aj sami [4].

2 Robot

Robot, s ktorým sme pracovali bol výsledkom tímového projektu viacerých študentov z roku 2019. Pri vysvetľovaní a opisovaní robota sa budeme odvolávať na dokumenty, stránky a kód, ktorý napísali. Všetky tieto údaje si sprístupnené na mobilnom robote v záložke [§ \(HOME\) /Desktop/Blackmetal \[5\]](#).

Robot je v tvare kvádra. Jeho šírka je 60cm a je vyzdvihnutý nad zem o 1.5cm. Nachádza sa na kolesách o polomere 8cm. Jeho kostra, až na ocel'ové pláty, ktoré držia robot, je spravená z hliníku. Konkrétnie z hliníkových tyčí, ktoré sú pospájané plexisklovými plátkmi. Jeho podobizeň vidíme na nasledujúcom obrázku.



Obr. 2.1: Zobrazenie spodnej časti mobilného robota [5]

Na obrázku ďalej vidíme olemovanie robota pásom s LED-kami. Tie svietia nasledovným spôsobom. Ked' sa robot nehýbe všetky LED-ky svietia na zeleno. Ked' sa robot pohne do nejakej strany, LED-ky znázornia jeho pohyb tým, že svietia na strane, do ktorej sa robot hýbe. Ked' nastane situácia, kedy počítač ovládajúci motory prestane komunikovať s Arduinom, ktoré sa stará o detekciu stavov robota tak LED-ky začnú blikat' červeno-modrými farbami.

Ako bolo spomenuté LED-ky znázorňujú pohyb robota. Ten sa pohybuje za pomocí diferenciálneho podvozku s dvoma podpornými všesmerovými kolesami. Motory robota sú pripojené na meniče. Tie sú ovládané priamo príkazmi z počítača.

2.1 Hardware

Hardware robota sa skladá z:

- kontrolnej dosky Arduino Uno,
- Počítača ADVANTECH MIO-5272 [6]
Počítač obsahuje operačný systém Ubuntu 16.04.
- Extension board MIOe-210 [7]
- Meniče MAXON EPOS 24/5 (s číslom 275512) [8]
Sú napájané jednosmerným napäťom 11 - 24 V a 5 A.
- Enkódery MAXON Encoder MR Type L (s číslom 225787) [9]
Rozlíšenie enkóderov je 1024 impulzov s troma kanálmi.
- Motory MAXON RE 40 (s číslom 148867) [10]
Motory s výkonom 150W. Maximálna rýchlosť je 12 000 rpm a efektivita 91%.
- Prevodovka MAXON Planetary Gearhead GP 42 C (s číslom 202120) [11]
Redukcia prevodovky je 43:1. Jej účinnosť je 72%.

Ovládanie robota je zabezpečené externými počítačmi

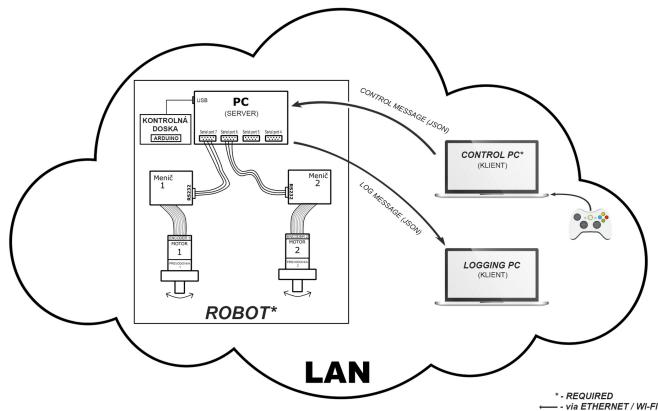
- Control PC (Kontrolný počítač) – Počítač posielajúci príkazy na robot cez TCP/IP protokol.
- Logging PC (Logovací počítač) – Počítač prijímajúci stav robota cez TCP/IP protokol.

Tieto počítače sú len reprezentácia servera. V realite to môže byť jeden a ten istý počítač.

Na obrázku Obr. 2.2 vidíme zapojenie jednotlivých častí robota. Čo sme nespomenuli a je na obrázku je XBox ovládač je to kvôli tomu, že tímový projekt bol zameraný na ovládanie robota pomocou tohto ovládača. My ho ale používat' nebudeme.

2.2 Komunikácia s robotom

S robotom sa vieme spojiť pomocou dvoch portov. Jeden port je otvorený na prijímanie požiadavok (requestov) a ten druhý je na monitorovanie stavu robota. Port 664 je otvorený pre tisíc užívateľov, ktorí môžu len sledovať stav robota. Druhý port je na prijímanie requestov 665 a je otvorený len pre jedného užívateľa.



Obr. 2.2: Schéma zapojenia jednotlivých častí na robote

2.2.1 Logovanie

Spomínaný port 664 je otvorený jednému užívateľovi. Ked' sa užívateľ pripojí začne dostávať nepretržité správy typu **JSON** (JavaScript Object Notation), ktoré hlásia stav robota. Správy, ktoré dostávame sú nasledujúceho formátu

```
{"state":1,"direction":1}
```

Hodnoty sa pri stave (state) a ani pri smere (direction) nemenia. Sú to stále jednotky. Pokým robota nezastavíme bud' príkazom, alebo stlačením tlačidla vypnutia, tak sa tieto správy budú posielat'. Môžeme potom začať polemizovať o tom či by nebolo lepšie už tieto správy využiť na to čo reálne spomenutý **JSON** reťazec ukazuje. A to udávať smer a stav robota.

2.2.2 Ovládanie

Port 665 je sprístupnený na prijímanie a odosielanie požiadavok a ich odpovedí. Príkazy sa na počítač posielajú cez siet' z externého počítača vo formáte **JSON**. Študenti, ktorí navrhovali systém posielania požiadavok (request) a odpovedí (response) robili tieto správy ručne. Preto nastávajú situácie, kedy robot pošle správu, ktorá nespadá do štandardu písania **JSON** textu. Z tohto dôvodu sme nemohli použiť už existujúci kód, ktorý by nám zjednodušil prehľadávanie týchto správ. Podľa dokumentácie sa robot mal ovládať správami typu [12]

```
{"UserID":1,"Command":3,"RightWheelSpeed":50,"LeftWheelSpeed":50}
```

Význam jednotlivých parametrov:

- **UserID** – Znázorňuje ID užívateľa, ktorý je pripojený na robot. Predvolená hodnota je 1.
- **Command** – Číselná hodnota znázorňujúca príkaz, ktorý ma robot vykonat'
 0. Prázdny príkaz slúžiaci na overenie spojenia
 1. Núdzové zastavenie
 2. Normálne zastavenie
 3. Príkaz nastavujúc rýchlosť kolies mobilného robota
 4. Prázdny príkaz
 5. Prázdny príkaz
 6. Príkaz pýtajúci si aktuálnu rýchlosť pravého a ľavého kolesa. Tento príkaz neboli sprave navrhnutý v kóde robota. Vracal nám žiadanú hodnotu namiesto aktuálnej. Museli sme ho prepísat'.
 7. Pripravenie motorov robota
 8. Príkaz pýtajúci si aktuálnu pozíciu pravého a ľavého kolesa.
- **RightWheelSpeed** – Nastavenie rýchlosť pre pravé koleso
- **LeftWheelSpeed** – Nastavenie rýchlosť pre ľavé koleso

Z tohto kusu kódu je jasné, že sa majú posielat' celé čísla a na základe tohto vstupu sa bude robot hýbať. Čo sme zistili až po skompilovaní a spustení tímového projektu je, že sa majú posielat' desatinné čísla z intervalu 0 až 1. Toto nebolo písané v dokumentácii, ktorá nám bola dodaná na začiatku programu. Môžeme preto príklad prepísat' na reťazec, ktorý by fungoval

```
{"UserID":1,"Command":3,"RightWheelSpeed":0.50,"LeftWheelSpeed":0.50}
```

2.3 Par slov k parametrom reťazca

UserID

Táto možnosť je v momentálnom stave robota nevyužitá. Počet zariadení, ktoré sa môžu pripojiť na port, cez ktorý sa dá robot ovládať je 1. Je to ale dobrá možnosť na rozšírenie kódu. Keď sa budú môcť pripojiť viacerí užívatelia, tak sa bude musieť vyriešiť, koho príkaz bude mať akú prioritu.

Command: 4

Tento príkaz je prázdny. My sme ho ale neskôr prepísali na príkaz, cez ktorý sa dá nastaviť žiadana pozícia kolies robota (natočenie). Táto funkcia nie je v takom stave ako sme si priali.

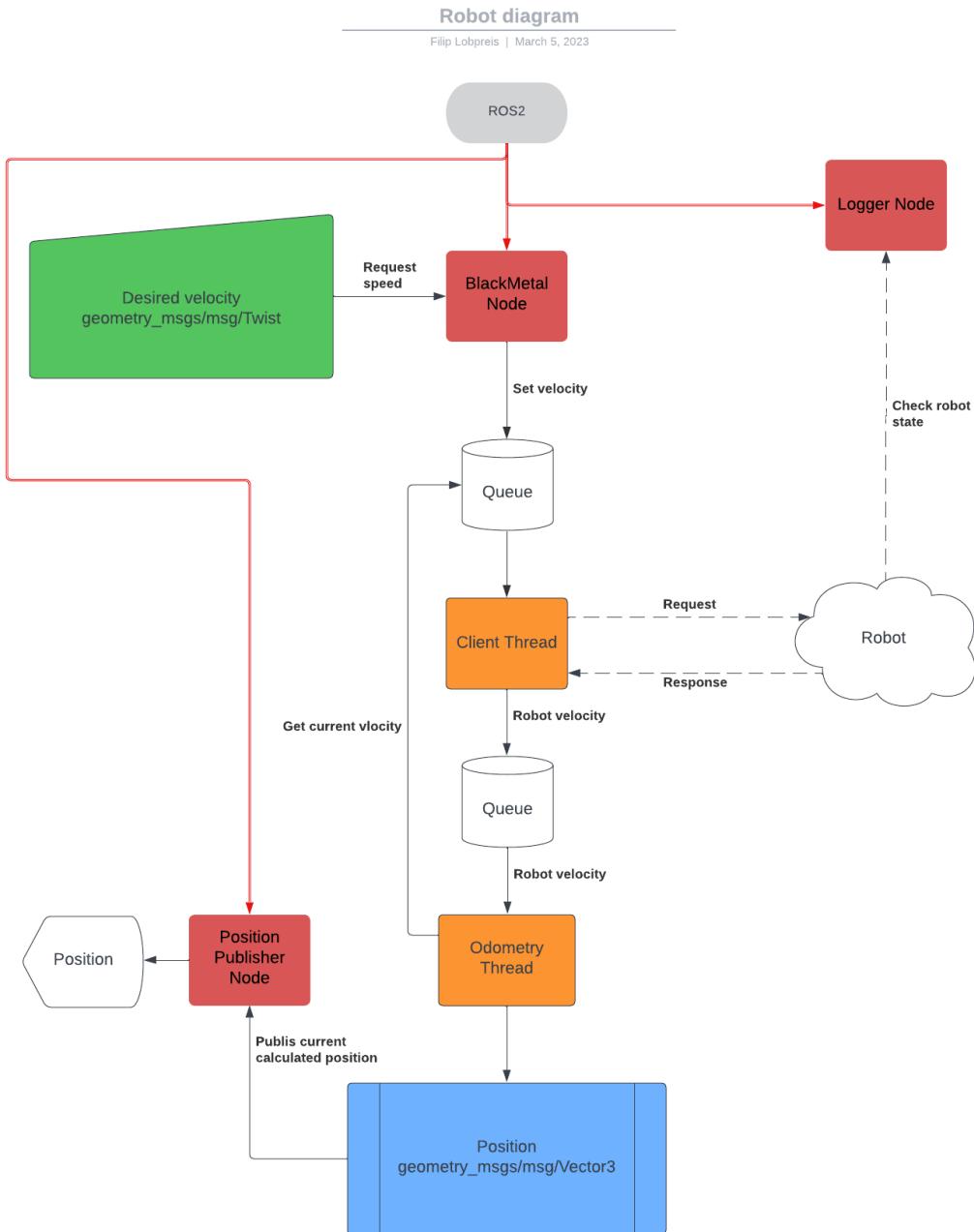
Command: 8

Príkaz na nastavenie polohy kolies neboli originálne naprogramované na robote. Pridali sme ho za cieľom presného dostavania robota na preddefinované miesto.

RightWheelSpeed/LeftWheelSpeed

Nastavovanie rýchlosťi pravého a ľavého kolesa nie sú povinné parametre. Musíme ich zadávať len v prípade posielania rýchlosťí cez príkaz s číslom 3.

3 Správanie programu



Obr. 3.1: Graf vykonávania programu na ovládanie robota pomocou ROS2.

3.1 Uvod do citania grafu

Na obrazku Obr. 3.1 mozeme vidieť viaceru objektov roznych farieb. Objekty zobrazene cervenou farbou su uzly spracovavane a vytvarane v ramci ROS2. Kazdy tento objekt bezi v separatnom procese. Objekty zobrazene oraznovou farbou su objekty, ktore maju svoje vlastne vlakno. Tieto objekty boli vytvorene uzlom BlackMetal. Datove struktury Queue, zobrazene bielou farbou su vytvorene tak aby zabezpecovali bezchybnu komunikaciu medzi viacerimi vlaknami. Objekt so zelenou farbou je vstup do programu. Je zadavany uzivatelow a reprezentuje ziadanu rychlosť robota. Modry objekt je vystupom probtramu. Je to topic, na ktorý sa publikuje aktualna pozicia robota. Robot samotny je zobrazeny bielou farbou vo forme maleho oblaciaka. Prerusovane ciary na diagrame znazornuju sietovu komunikaciu programu a robota. Cervene dvojite ciary udavaju presne ktore objekty patria ROS-u. Nakoniec cierne plne ciary reprezentuju tok dat medzi objektami.

3.2 Uzly

Na obrazku Obr. 3.1 mozeme vidieť postup vykonavania programu na ovladanie robota BlackMetal pomocou ROS2. Na zaciatku programu sa vytvoria 3 uzly. Prvy uzol **Position Publisher Node** je uzol, na ktorý sa publikuje vypocitana pozicia robota. Pocita sa na zakalde obdrzanych dat z enkoderov robota. Uzol **Logger Node** sluzi na zaznamenavanie stavu robota 2.2.1. Posledny uzol **BlackMetal Node** ovlada robota na zaklade zadanych dat uzivatelow.

3.3 Vstup

Uzol *BlackMetal Node* vytvorí príjemcu, ktorý počúva na téme *geometry_msgs/msg/Twist*. Tento vstup je vo forme príkazu zadaneho v príkazovom riadku. Vyzera nasledovne:

```
ros2 topic pub /cmd_vel geometry_msgs/msg/Twist
    "linear:
        x: 0.0,
        y: 0.0,
        z: 0.0,
    angular:
        x: 0.0,
        y: 0.0,
        z: 0.0" -1
```

Tento príkaz publikuje jednu správu (-I) o lineárnych a uhlových rýchlosťach na tému */cmd_vel* Táto sprava je typu *geometry_msgs/msg/Twist*. Je nasledne spracovaná a ulozená do rady *Queue*. Tato rada je prioritne založená. To znamena, že požiadavka s nízsim kodom má

vyzsiu prioritu. Poziadavky a ich kody mozeme videt v sekciu 2.2.2.

3.4 Komunikacia s robotom

Ako je naznacene na Obr. 3.1, klient si vo svojom vlastnom vlakne vytiahne prvu spravu z rady a pretransformuje ju do formy JSON. Tento typ spravy mozeme videt v 2.2.2. Prikaz je poslany robotu a ten obratom da vediet, ci danu poziadavku obdrzel. Ak tato sprava ziadala rychlosť kolies, tak robot dalsou spravou odpovie na danu poziadavku. Typ tejto odpovede mozeme videt v 4.1. V tomto pripade sa sprava spracuje a uloži sa do dalsej rady.

3.5 Odometria

Odometria, pocitanie polohy na zaklade rychlosť kolies, sa vykonava rovako ako komunikacia s robotom v separatnom vlakne. Tu je potreba si uvedomiť jednu skutočnosť. To je ta, že keď posielame ziadost na nastavenie rychlosť kolies robota, tak robot si hodnoty v ziadosti prepočita a data da nasledne enkoderom. Keď si ale tieto rychlosť vyziadame z enkoderov, tak sa robot týchto dat nechyta a my si ich musíme prepočítať na metre za sekundu. Zaroven si tento objekt neustale pyta od robota rychlosť kolies. Ako sme už spomenuli v 3.4, tieto spravy majú nizšiu prioritu ako nastavenie rychlosť kolies alebo bezpocnostné zastavenie robota. Preto sa može stať, že spravy posielane robotu nebudu dodržiavať presné stanovené frekvencie v čase keď mu bude užívateľ posielat prikazy.

3.6 Zdielanie polohy

Dalsiu vec, ktorú je treba vysvetliť ku grafu Obr. 3.1 je zdielanie polohy. Odometria po každom dopocitani polohy robota publikuje tuto informaciu na temu */position* tato sprava je typu *geometry_msgs/msg/Vector3*. Obsahuje 3 hodnoty, ktoré sú x, y a z. Suradnice reprezentujú aktuálnu polohu robota.

4 Oprava chýb na robote

4.1 Nesprávna funkcia

Ako bolo spomenuté vyššie, pri poslaní príkazu s číslom 6 nám robot vráti aktuálne rýchlosť kolies. Počas skúšaní tejto funkcionality sme narazili na problém. Ked' sme sa robota spýtali na jeho rýchlosť dostali sme reťazec, ktorý obsahoval náhodne veľké čísla. Tieto čísla sa menili, ked' sme zadávali nejaké hodnoty pre rýchlosť kolies aby sa robot hýbal.

```
{ "LeftWheelSpeed"=236223201280  
  "RightWheelSpeed"=4294967296 }
```

Tu vidíme príklad obdržanej spravy. Ako si môžeme všimnúť. Pri tomto type správ nie je dodržaná správna forma reťazca typu JSON. Namiesto ':' máme '=' a medzi argumentmi sa nenachádza čiarka. Jeden z nápadov, ktorý sme mali bolo premeniť tieto čísla na desatinné čísla, ked'že sme mu posielali taký formát čísel. Problém je v tom, že ked' posielame request na nastavenie rýchlosť kolies, tak kód na robote funguje tak, že si ich premení na celé čísla v rozsahu 0 až 1000. To je hodnota, na ktorú nastaví rýchlosť otáčania kolies respektíve rýchlosť otáčania motora. Na druhú stranu, ked' si vypýtame od robota rýchlosť kolies. On zoberie informáciu z enkóderov a pošle nám to bez spracovania. Aj napriek týmto poznatkom sa nám nepodarilo získať z týchto dát žiadane rýchlosťi.

Po dôkladnom preštudovaní kódu sme zistili, že hodnoty ktoré nám posielala robot nie sú ani vytahované z enkóderov správnou funkciou. Preto sme ju zmenili a začali sme dostávať hodnoty, s ktorými by sa mohlo dať pracovať.

Funkcie z knižnice zabezpečujúce komunikáciu z enkóderov motorov pochádzajú z firmy Maxon [13]. Funkcie, ktoré končia koncovkou 'Target' majú návratné hodnoty reprezentujúce žiadane hodnoty. Funkcie s koncovkou 'Is' vracajú aktuálne hodnoty. Z tohto dôvodu sme museli prepísať funkciu, ktorá sa vykonávala, ked' sme chceli získať aktuálne hodnoty rýchlosťi motora poslaním príkazu 6.

```
BOOL VCS_GetTargetVelocity(  
    HANDLE KeyHandle,  
    WORD NodeId,  
    long* pTargetVelocity,  
    DWORD* pErrorCode);  
  
BOOL VCS_GetVelocityIs(  
    HANDLE KeyHandle,  
    WORD NodeId,
```

```

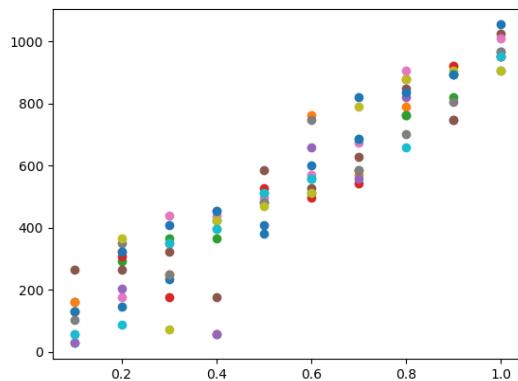
long* pVelocityIs,
DWORD* pErrorCode);

```

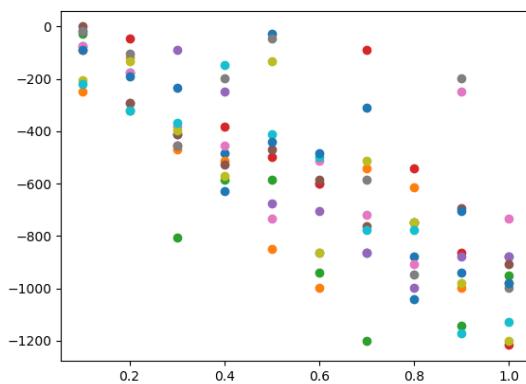
Ako môžeme vidieť v týchto predpisoch funkcií, bolo treba zmeniť názov funkcie a ostatné parametre ostali rovnaké. Nebolo treba meniť implementáciu kódu.

4.2 Zašumený výstup

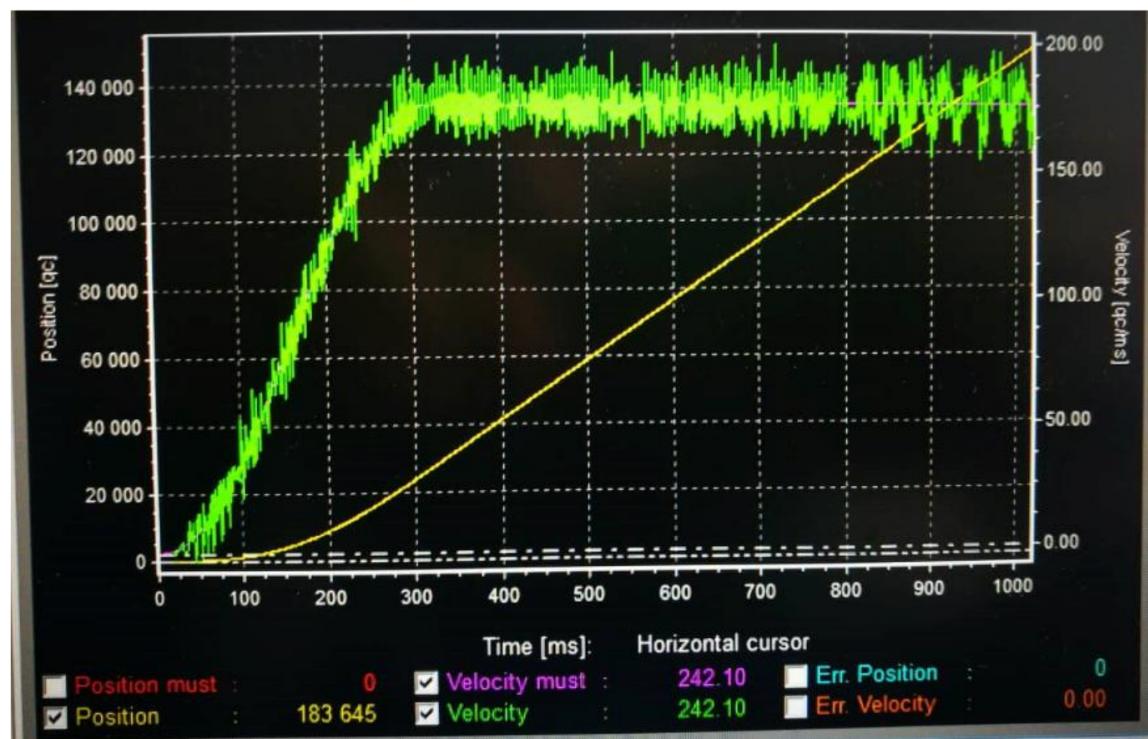
Po prepísaní funkcie na získavanie rýchlosťí robota sme spravili pári meraní, aby sme zistili, aké presné informácie o rýchlosťach motorov dostávame. Aby nám robot neodbiehal postavili sme ho na vyvýšené miesto, tak aby sa kolesá nedotýkali zeme. V takomto postavení sa robot nepohne z miesta a my môžeme bez problémov odmerať prechodové a prenosové charakteristiky rýchlosťi pravého a ľavého motora.



Obr. 4.1: Ustálené hodnoty rýchlosťi ľavého motoru.



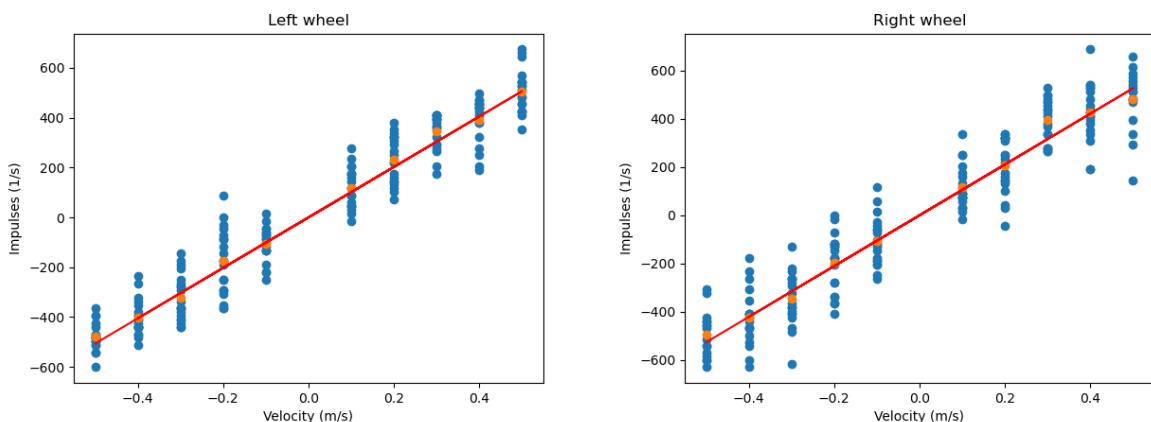
Obr. 4.2: Ustálené hodnoty rýchlosť pravého motora.



Obr. 4.3: Prechodová charakteristika rýchlosť kolies [5].

5 Získavanie rýchlosťí robota

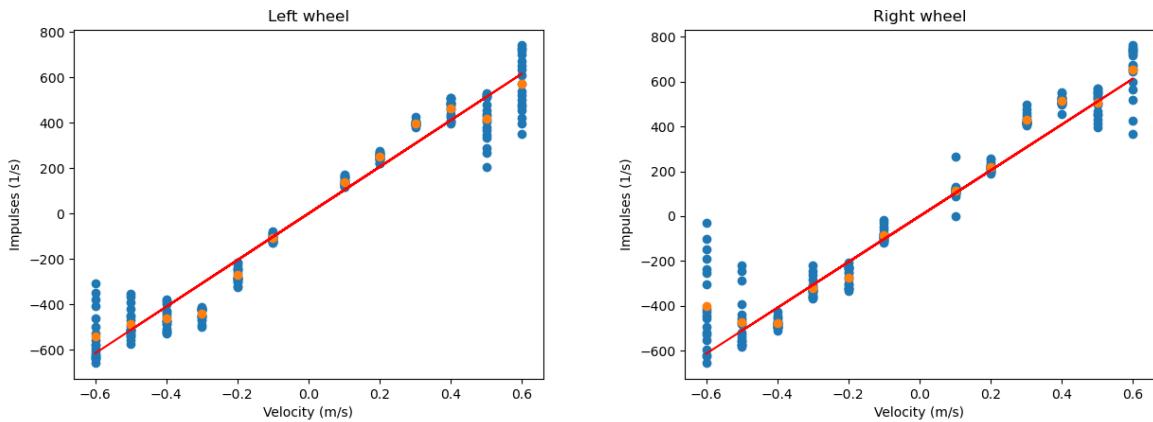
Ako bolo spomenuté v predchádzajúcej kapitole, rýchlosťi kolies sa dajú získať z enkódarov. Tieto dátu sa posielajú v sprave, ktorá pripomína JSON formát. Z týchto vzoriek poslaných robotom nevieme priamo vypočítať polohu. Musíme si tieto dátu premeniť z impulzov za sekundu $\frac{1}{s}$ na metre za sekundu $\frac{m}{s}$. Tento prevod nebude jednoznačný, pretože jazdy enkódery posielajú dátu inak zašumené. Preto je potrebne zistiť, ako sa zmení rýchlosť pri zmene impulzov za sekundu. Tento prevod je nožne získať z merania, kde po nastavení rýchlosť zoberieme veľa dát z enkóderov a zistíme, ako sa zmení rýchlosť pri zmene impulzov za sekundu. Prvý nápad na získanie čo najlepšej prevodovej charakteristiky bol cez všetky dátu položiť lineárnu regresiu. To sa ukázalo ako zlé riešenie, lebo dátu, ktoré dostávame majú veľmi veľký rozptyl. Jednou z nasledujúcich úvah bolo spraviť klízavý priemer. Toto riešenie malo tiež svoje chyby a to v tom, že zmeny zaznamenaných impulzov za sekundu sa zmenili v závislosti od rýchlosťi a smeru. V tomto bode sme vyskúšali počítať biometriu z obdržaných dát. Táto implementácia bola veľmi nepresná. Tento pokus nám potvrdil, že potrebujeme filter obdržaných impulzov. Výsledky pokusu, kde sme zistovali prevodovú charakteristiku z impulzov za sekundu na rýchlosť v SI jednotkách nájdeme na nasledovných grafoch.



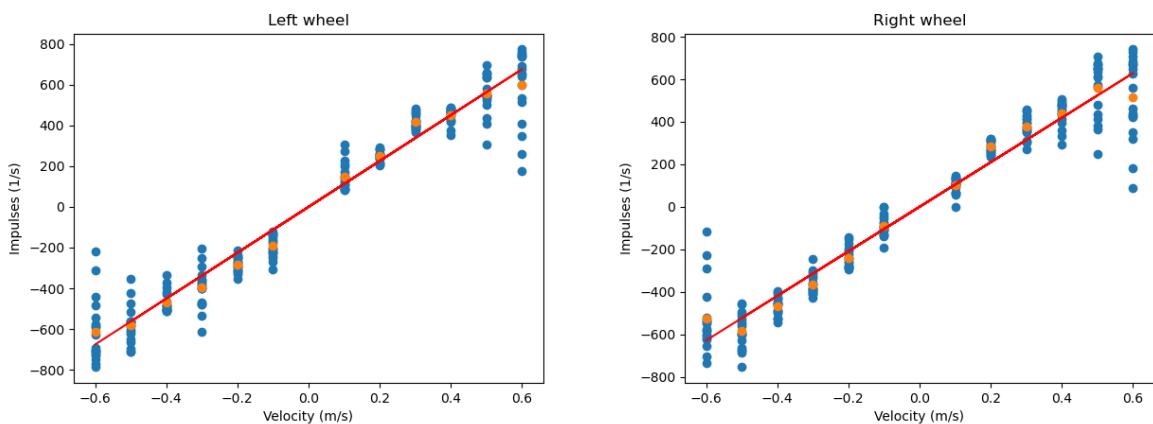
Obr. 5.1: Získanie prevodu z impulzov na rýchlosť v SI jednotkách.

Implementovali sme si preto dolnopriepustný kvadratický filter. Jeho parameter α sme získaли viacerými meraniami. Začali sme najsilnejším filtrom s hodnotou $\alpha = 0.9$.

Ako môžeme vidieť na obrázkoch Obr. 5.2 a Obr. 5.1 aplikácia filtra výrazne pomohla proti sumu signálu. Problémom pri silnom filtri je to, že ak na začiatku merania dostaneme zlu hodnotu, tak sa tato hodnota t'ažko mení na správnu. Tento problém sme riešili postupným menením parametra filtru. Ďalšiu hodnotu sme použili $\alpha = 0.7$.

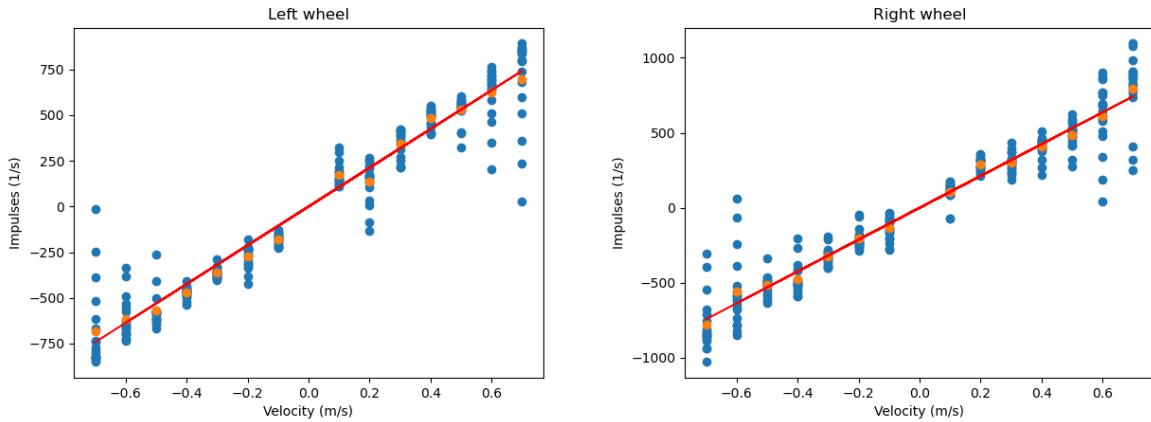


Obr. 5.2: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.9$.



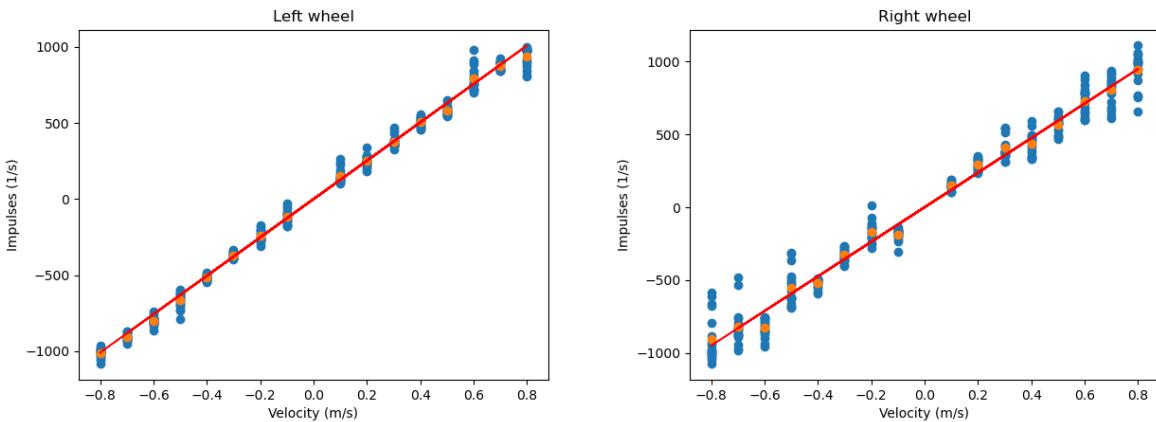
Obr. 5.3: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.7$.

Pri použití hodnoty $\alpha = 0.7$ sme zistili, že sa hodnoty rýchlosť aj pri aplikácii filtra výrazne menili. Spravili sme preto ďalšie meranie, kde sme použili hodnotu $\alpha = 0.75$.



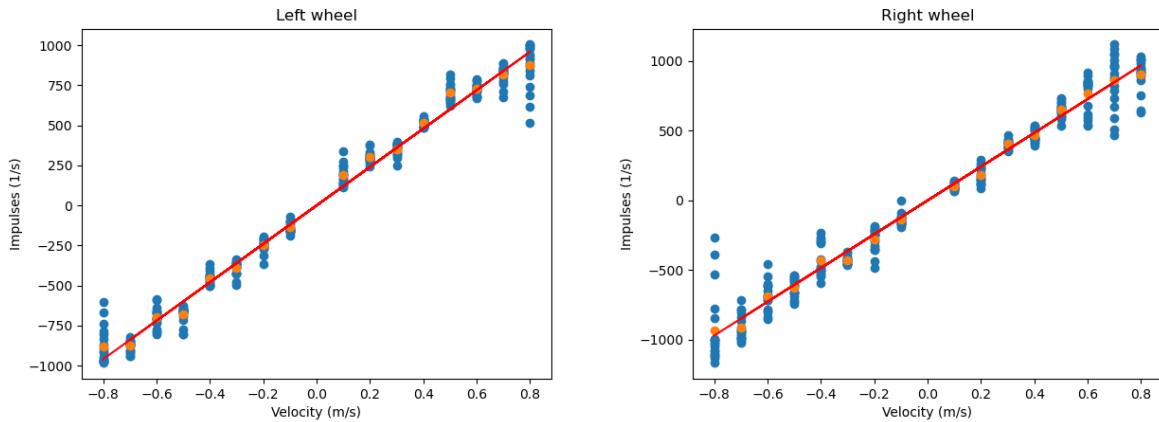
Obr. 5.4: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.75$.

Použitie silnejšieho filtra súčasťom pomohlo rýchlejšiemu ustáleniu hodnoty, ale skúsili sme ešte silnejší filter s hodnotou $\alpha = 0.8$.



Obr. 5.5: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.8$.

Ako môžeme vidieť na Obr. 5.5 ustálenie hodnôt je veľmi jednoznačne. Vybrali sme si preto filter s hodnotou $\alpha = 0.8$. Zatiaľ všetky dátá čo sme merali holo s frekvenciou 4Hz (1 vzorka za 250 milisekúnd). Pre presnejší výsledok sme tuto frekvenciu ešte zvýšili. Z testov robota sme vyzorovali, že najsilnejšia frekvencia, ktorú môže robot sprostredkovať je 10Hz (1 vzorka za 100 milisekúnd). Spravili sme si preto test na prevod rýchlosť ešte raz s rovnakou hodnotou $\alpha = 0.8$, ale s frekvenciou 10Hz.



Obr. 5.6: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0.8$ a frekvenciou 10Hz.

Najlepšia možná hodnota tohto parametru nám vyšla $\alpha = 0.8$. Pri implementácii tohto filtra sme museli myslieť na dôležitú vec. Ked' sa zmenia jednorazovo impulzy na hodnotu 0 a hned' späť, tak nám táto vzorka pokazí výsledok. Musíme preto túto vzorku ignorovať. Ďalšia prekážka, ktorú sme mali pred sebou bola zmena rýchlosťi. Obyčajná implementácia filtra by nám spomalila zmenu vypočítanej rýchlosťi a teda aj veľkú odchýlku v polohe. Tento problém sme opravili prestavením počiatočnej hodnoty filtra na prvú hodnotu po zmene rýchlosťi. Toto riešenie sa ukázalo ako najlepšie so skúšaných riešení.

Záver

V prvej časti bakalárskej prace bolo našou úlohou zoznámiť sa s robotom a neimplementovať ovládač na neho pomocou ROSu. V stave v akom sme ho dostali sme museli opraviť niektoré softvérové chyby na robote. Tie sa týkali hlavne komunikácie s robotom. V tejto práci sme sa dostali do stavu, kedy vieme robotu poslať, akou rýchlosťou sa majú hýbať jednotlive kolesa a on nám dáva spätnú väzbu, že akou rýchlosťou naozaj ide. To ale ešte nefunguje, tak ako má kvôli zašumenému signálu. V druhej časti bakalárskej prace by sme chceli opraviť alebo aspoň zredukovať toto zašumenie. Poprípade nájsť iný spôsob na kontrolovanie pozície robota v priestore.

Zoznam použitej literatúry

1. *ROS2 documentation* [online] [cit. 2022-12-23]. Dostupné z : <https://docs.ros.org/en/humble/index.html>.
2. RICO, Francisco Martín. *A Concise Introduction to Robot Programming with ROS2*. 1. vyd. Chapman a Hall/CRC Press, 2023. ISBN 978-1-003-28962-3.
3. *ROS2 from the Ground Up* [online] [cit. 2022-12-23]. Dostupné z : <https://medium.com/@nullbyte.in/ros2-from-the-ground-up-part-1-an-introduction-to-the-robot-operating-system-4c2065c5e032>.
4. *Changes between ROS 1 and ROS 2*. Dostupné tiež z: <http://design.ros2.org/articles/changes.html>.
5. BC. MAREK PACALAJ, BC. TOMÁŠ KÚTIK, BC. DOMINIK GULA, BC. DÁVID PAVLIČ, BC. DANIEL ĎURKOVIČ. *Mobilný podstavec pre robota*. 2019.
6. *MIO-5272* [online] [cit. 2022-12-26]. Dostupné z : [https://www.mouser.sk/datasheet/2/638/MIO-5272_DS\(01.17.18\)20180118153722-1570123.pdf](https://www.mouser.sk/datasheet/2/638/MIO-5272_DS(01.17.18)20180118153722-1570123.pdf).
7. *MIO-210* [online] [cit. 2022-12-26]. Dostupné z : [https://advdownload.advantech.com/productfile/PIS/MIOe-210/Product%20-%20Datasheet/MIOe-210_220_230_110_120_PWR1_DS\(03.26.14\)20140327095019.pdf](https://advdownload.advantech.com/productfile/PIS/MIOe-210/Product%20-%20Datasheet/MIOe-210_220_230_110_120_PWR1_DS(03.26.14)20140327095019.pdf).
8. *EPOS2 Positioning Controllers* [online] [cit. 2022-12-26]. Dostupné z : https://www.maxongroup.com/medias/sys_master/root/8831294472222/2018EN-457-458-459-461.pdf.
9. *Encoder MR Type L, 256–1024 CPT, 3 channels, with line driver* [online] [cit. 2022-12-26]. Dostupné z : <https://innodrive.ru/downloads.php?file=/wp-content/uploads/files/maxon/sensor/15032-EN-21-479.pdf>.
10. *Details RE 40 Ø40 mm, Graphite Brushes, 150 Watt* [online] [cit. 2022-12-26]. Dostupné z : <https://www.maxongroup.com/maxon/view/product/motor/dcmotor/re/re40/148867>.
11. *Details Planetary Gearhead GP 42 C Ø42 mm, 3 - 15 Nm, Ceramic Version* [online] [cit. 2022-12-26]. Dostupné z : <https://www.maxongroup.com/maxon/view/product/gear/planetary/gp42/203120>.
12. BC. MAREK PACALAJ, BC. TOMÁŠ KÚTIK, BC. DOMINIK GULA, BC. DÁVID PAVLIČ, BC. DANIEL ĎURKOVIČ. *Dokumentacia k softwaru robota BlackMetal*. 2019.

13. MAXON. *EPOS Command Library: Document ID: rel6806*. 2019.