

**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, PhD.
Konzultant: Ing. Michal Dobiš, PhD.

Bratislava 2023

Filip Lobpreis



ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Filip Lobpreis**

ID študenta: 111124

Študijný program: robotika a kybernetika

Študijný odbor: kybernetika

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 úkonomi 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 pochopíť jeho fungovanie. Cieľom práce bude následne vytvoriť nadradený ovládač implementovaný v ROS (Robotickom operačnom systéme), ktorý bude schopný riadiť daného robota.

Úlohy:

1. Analyzujte súčasný stav riešenia a prostredie Robotického operačného systému.
2. Analyzujte možnosti a metodiku implementácie riadiaceho balíka pre daný robot.
3. Navrhnite spôsob implementácie a architektúru riešenia.
4. Implementujte riadiaci systém pre mobilného robota.
5. Vypracujte dokumentáciu k dosiahnutým výsledkom.
6. Vyhodnoťte dosiahnuté výsledky.

Termín odovzdania bakalárskej práce: 02. 06. 2023

Dátum schválenia zadania bakalárskej práce: 08. 12. 2022

Zadanie bakalárskej práce schválil: doc. Ing. Eva Miklovičová, PhD. – garantka študijného programu

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, PhD.
Konzultant:	Ing. Michal Dobiš, PhD.
Miesto a rok predloženia práce:	Bratislava 2023

V tejto bakalárskej práci sa zaoberáme riadením mobilného robota v prostredí ROS2. Robot, ktorý budeme využívať je vybavený diferenciálnym podvozkom a dvoma opornými všesmerovými kolesami. Obsahuje dva enkódery pre pravé a ľavé koleso, ktoré merajú rýchlosť otáčania kolies v impulzoch za sekundu. Jeho ovládanie a komunikácia s ním prebieha cez TCP/IP spojenie. Pri komunikácii s robotom sa posielajú spravy v štruktúre JSON. Výsledným cieľom tejto bakalárskej práce je vytvoriť ovládač robota, ktorý bude sprístupňovať ovládanie robota v prostredí ROS2. V prvej kapitole sa venujeme analýze pôvodného stavu robota a jeho komunikačnému rozhraniu. V druhej kapitole popisujeme návrh implementácie ovládača robota v prostredí ROS2. Tretiu kapitolu sme venovali popísaniu zlepšení, ktoré sme vykonali na robote. Štvrtá kapitola sa venuje samotnej implementácii ovládača robota. V predposlednej kapitole opisujeme ako sme prekonali problémy, ktoré vznikli pri komunikácii s robotom. V poslednej kapitole opisujeme viacero využití ovládača robota v prostredí ROS2.

Kľúčové slová: ROS, ROS2, robot, klient, filter, ovládač, odometria

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, PhD.
Consultant:	Ing. Michal Dobiš, PhD.
Place and year of submission:	Bratislava 2023

This bachelor's thesis deals with the control of a mobile robot in the ROS2 environment. The robot we will be working with is equipped with a differential chassis and two support omnidirectional wheels. It contains two encoders for the right and left wheel, which measure the wheel velocities in pulses per second. The control and communication interface with the robot is implemented via TCP/IP connection. Messages in JSON structure are sent when communicating with the robot. The ultimate goal of this bachelor's thesis is to create a robot controller that will provide access to robot control in the ROS2 environment. In the first chapter, we analyse the original state of the robot and its communication interface. The second chapter describes the design of the robot controller implementation in the ROS2 environment. We devoted the third chapter to describing the improvements we made to the robot. The fourth chapter focuses on the actual implementation of the robot controller. In the penultimate chapter, we describe how we overcame issues that arose during communication with the robot. In the last chapter, we describe several uses of the robot controller in the ROS2 environment.

Keywords: ROS, ROS2, robot, client, filter, driver, odometry

Pod'akovanie

Chcel by som sa pod'akovat' môjmu školiteľovi pánu Michalovi Dobišovi.

Obsah

Úvod	1
1 Analýza pôvodného stavu robota	2
1.1 Robot a jeho ovládanie	2
1.2 Komunikácia s robotom	4
1.2.1 Logovanie	4
1.2.2 Ovládanie	4
1.2.3 Zapínanie robota	6
2 Návrh implementácie riadenia	7
2.1 Operačný systém	7
2.1.1 Windows	7
2.1.2 Ubuntu	8
2.2 Programovací jazyk a jeho prostredie	9
2.3 Robotický operačný systém	10
2.3.1 Základné pojmy	11
2.3.2 Témy	11
2.3.3 Služby	12
2.3.4 Akcie	12
2.3.5 Parametre	13
2.4 ROS1	13
2.5 ROS2	14
2.6 Rozdiely medzi ROS1 a ROS2	16
2.6.1 Štandard jazyka	16
2.6.2 Inicializácia uzla	16
2.6.3 Komunikácia	17
2.6.4 Parametre	17
2.6.5 Nodelet alebo komponent	17
2.6.6 Kompilácia	17
2.6.7 Vlákna	18
2.7 Ros_control	18
3 Zlepšenie stavu robota	21
3.1 Chyba v systéme	21
3.2 Chyba v dokumentácii	21

3.3	Oneskorenie komunikácie	21
3.4	Rozšírenie príkazov robota	22
3.5	Nesprávna spätná väzba	23
3.6	Zašumený výstup	24
4	Implementácia ovládača	26
4.1	Uzly	27
4.2	Vstup	27
4.3	Komunikácia s robotom	28
4.4	Odometria	28
4.5	Zdielanie polohy	29
5	Experimenty	30
5.1	Filtrovanie zašumeného signálu	31
5.2	Overenie meraní	37
6	Využitie	41
6.1	Joystick	41
	Záver	43
	Zoznam použitej literatúry	44
	Prílohy	46
	A Dokumentácia ku kódu	47
	B Kód ovládača	48
	C Videá ovládania robota	49

Zoznam obrázkov a tabuliek

Obrázok 1.1	Zobrazenie spodnej časti mobilného robota [1]	2
Obrázok 1.2	Schéma zapojenia jednotlivých častí na robote [1]	4
Obrázok 2.1	Vizualizácia témy v ROSe [13]	11
Obrázok 2.2	Vizualizácia služby v ROSe [13]	12
Obrázok 2.3	Vizualizácia akcie v ROSe [13]	13
Obrázok 2.4	Porovnanie štruktúr ROS1 a ROS2 [16]	14
Obrázok 2.5	Diagram zobrazujúci vnútornú implementáciu balíka ros2_control [19].	19
Obrázok 3.1	Ustálené hodnoty rýchlosťi ľavého a pravého motora.	25
Obrázok 3.2	Prechodová charakteristika rýchlosťi kolies [1].	25
Obrázok 4.1	Graf vykonávania programu na ovládanie robota pomocou ROS2. . .	26
Obrázok 5.1	Získanie prevodu z impulzov na rýchlosť v SI jednotkách.	30
Obrázok 5.2	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,9$. .	31
Obrázok 5.3	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,7$. .	32
Obrázok 5.4	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,75$.	33
Obrázok 5.5	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,8$. .	33
Obrázok 5.6	Získanie prevodu z impulzov za sekundu na rýchlosť v SI jednotkách. $\alpha = 0,8$ a frekvenciou 10Hz.	35
Obrázok 5.7	Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,8$ a frekvenciou 10Hz a prvou prepočítanou hodnotou.	36
Obrázok 5.8	Zobrazenie pohybu robota z odometrie bez použitia filtra.	37
Obrázok 5.9	Zobrazenie pohybu robota z odometrie s použitím najlepšieho filtra.	38
Obrázok 5.10	Odmeranie koncovej pozície robota na osi X po prejdení štvorcovej dráhy.	39
Obrázok 5.11	Odmeranie koncovej pozície robota na osi Y po prejdení štvorcovej dráhy.	39
Obrázok 5.12	Zobrazenie pohybu robota z odometrie s použitím slabšieho filtra. . .	40
Tabuľka 5.1	Porovnanie výsledkov experimentov prevodu impulzov za sekundu $\frac{1}{s}$ na metre za sekundu $\frac{m}{s}$	32

Zoznam skratiek

API	Application Programming Interface
BIOS	Basic Input Output System
DDS	Data Distribution Service
DOF	Degrees of Freedom
GCC	GNU Compiler Collection
GNU	GNU is not Unix
GUI	Grafical User Interface
IoT	Internet of Things
IPC	Inter Process Communication
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
LAN	Local Area Network
NCR	Národné Centrum Robotiky
QoS	Quality of Service
ROS	Robot Operating System
RTOS	Real time operating system

Úvod

Robotizácia sa v dnešnej dobe viac a viac rozširuje k bežným ľuďom a do ich príbytkov. Cieľom všetkých vedcov a inžinierov je vytvoriť roboty, ktoré budu schopné pomáhať ľuďom vo vykonávaní každodenných činností. Tým, že lúdia žijú v domoch a bytoch o rôznych veľkostiach, tak je potrebné vytvoriť roboty, ktoré sa budu vedieť pohybovať v užších, ale aj otvorených priestoroch. Ked' sa pozrieme na dnešné ľudom prístupné mobilné roboty, tak majú väčšinou diferenciálne podvozky, ktoré sú schopné pohybovať sa po rovine, na ktorej sa nenachádzajú veľké prekážky. To sú hlavne parkety, dlaždice alebo koberce. Tento trend sa dlhšiu dobu nezmení a bude sa len posúvať vpred.

V súčasnej dobe je pri vytváraní softvéru pre robota často využívaný Robotický Operačný Systém (ROS1) prvej verzie. Tento systém aktuálne prechádza zo staršej už spomenutej verzie ROS1 na novšiu ROS2. Tá dáva programátorom viac možností ako spravit systém užitočnejší a prístupnejší pre každodenného užívateľa.

Na základe tohto trendu bol vytvorený projekt na sprístupnenie ovládania mobilného robota do prostredia Robotického operačného systému druhej verzie. Toto sprístupnenie ovládania robota otvára dvere mnohým ďalším nápadom a projektom. Vytvorenie tohto rozhrania poskytuje možnosť implementácie rozhrania pre ovládanie robota pomocou hlasu, gestami alebo autonómnymi algoritmami.

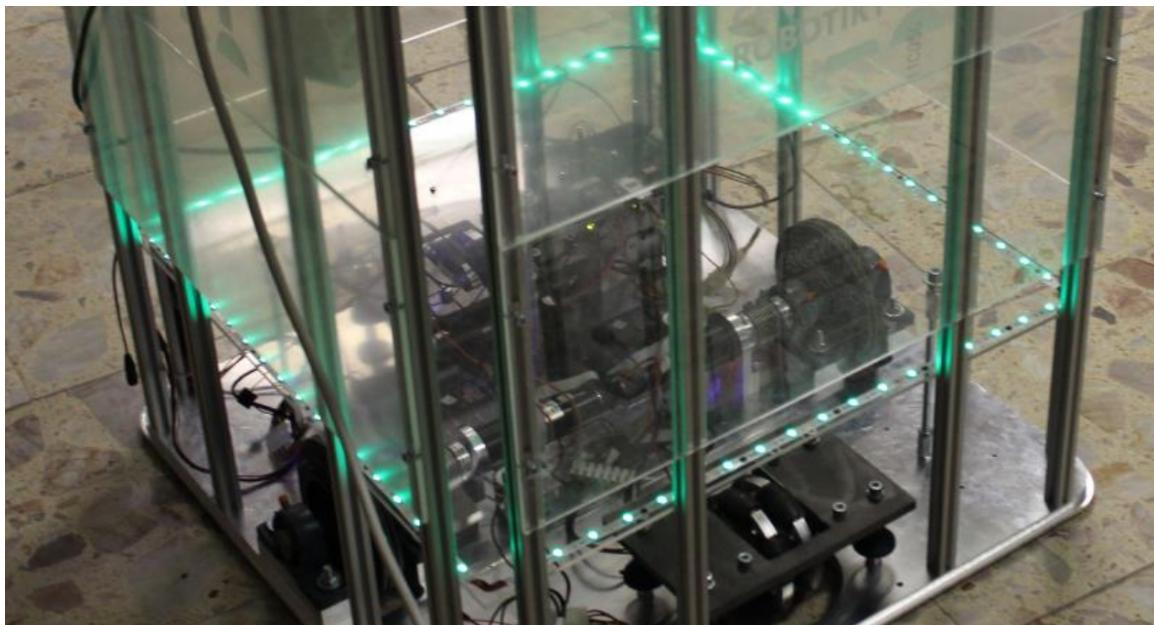
Na to aby sme vedeli využiť všetky tieto výhody, tak je potrebné mať ovládač, ktorý bude vytvárať rozhranie medzi robotickým operačným systémom a komunikačným rozhraním robota. V tejto práci sa budeme venovať implementácii viacerých procesov, ktoré budú spracovať informácie poslané robotickým operačným systémom. Tieto správy spracujeme do formátu JSON s tým, že prepočítame vstupné lineárne a uhlové rýchlosťi zo správy tejto témy na rýchlosť ľavého a pravého kolesa diferenciálneho podvozku. Výsledkom výpočtov zo získaných aktuálnych rýchlosťí robota bude jeho odometria, ktorá obsahuje údaje o rýchlosťi, smere a polohе robota.

1 Analýza pôvodného stavu robota

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 sú sprístupnené na mobilnom robote v záložke `$(HOME)/Desktop/Blackmetal [1]`.

1.1 Robot a jeho ovládanie

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 sa skladá z dvoch oceľových plátov, ktoré tvoria dno a strechu robota. Steny robota sú spravené z hliníku. Konkrétnie z hliníkových tyčí, ktoré sú pospájané plexisklovými plátkmi. Jeho vyhotovenie vidíme na nasledujúcom obrázku.



Obr. 1.1: Zobrazenie spodnej časti mobilného robota [1]

Na obrázku 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 modro na strane, do ktorej sa robot hýbe a na červeno na všetkých ostatných stranách. 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.

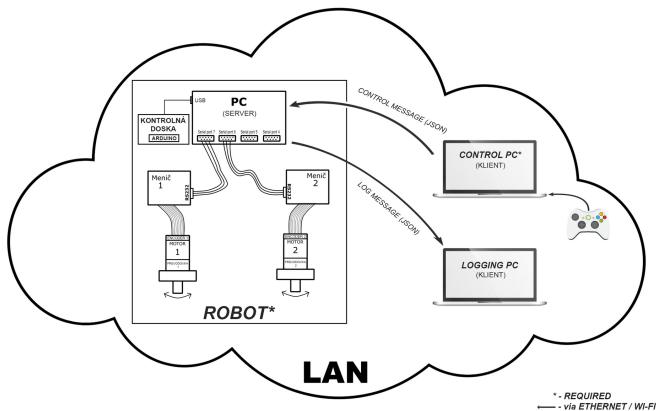
Hardvér robota sa skladá z:

- Riadiacej dosky Arduino Uno,
- Počítača ADVANTECH MIO-5272
Počítač obsahuje operačný systém založený na jadre Linux Ubuntu 16.04 [2].
- Extension board MIOe-210 [3]
- Meniče MAXON EPOS 24/5 (s číslom 275512)
Meniče sú napájané jednosmerným napäťom 11 - 24 V a 5 A [4].
- Enkódery MAXON Encoder MR Type L (s číslom 225787)
Rozlíšenie enkóderov je 1024 impulzov s troma kanálmi [5].
- Motory MAXON RE 40 (s číslom 148867)
Motory s výkonom 150W. Maximálna rýchlosť je 12 000 rpm a efektivita 91% [6].
- Prevodovka MAXON Planetary Gearhead GP 42 C (s číslom 202120)
Redukcia prevodovky je 43:1. Jej účinnosť je 72% [7].

Ovládanie robota je zabezpečené externými počítačmi:

- Control PC (Riadiaci klient) – Klient posielajúci príkazy na robot cez TCP/IP protokol.
- Logging PC (Logovací klient) – Klient prijímajúci stav robota cez TCP/IP protokol.

Títo klienti sú len reprezentáciou dvoch klientov pripájajúcich sa na dva separátne porty na robote. V realite to môže byť jeden a ten istý počítač. Na obrázku 1.2 vidíme komunikačné schému jednotlivých častí robota.



Obr. 1.2: Schéma zapojenia jednotlivých častí na robote [1]

1.2 Komunikácia s robotom

S robotom sa vieme spojiť pomocou dvoch portov. Jeden port je otvorený na prijímanie požiadavok (v anglickej literatúre zaužívaný pojem *request*) a ten druhý je na monitorovanie stavu robota. Port 664 je otvorený pre jedného užívateľa, ktorý môže len sledovať stav robota. Druhý port je na prijímanie požiadavok 665 a je taktiež otvorený len pre jedného užívateľa.

1.2.1 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. Tieto správy budú posielat' pokým robot nezastavíme príkazom, stlačením tlačidla vypnutia, alebo stlačením bezpečnostného tlačidla. Ak nejaká z týchto udalostí nastane, server prestane posielat' správy.

Potom môžeme 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. Momentálne tieto správy slúžia len na to, aby sme vedeli, že tento robot je aktívny a vie primat' a spracúvať informácie.

1.2.2 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**. Podľa dokumentácie [8] sa robot má ovládať správami nasledujúceho typu:

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

Význam jednotlivých parametrov:

- **UserID**

- Znázorňuje ID užívateľa, ktorý je pripojený na robota. Predvolená hodnota je 1.
- 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. V prípade, že sa budú môcť pripojiť viacerí užívatelia, tak sa bude musieť vyriešiť, koho príkaz bude mať akú prioritu.

- **Command**

- Číselná hodnota znázorňujúca príkaz, ktorý ma robot vykonáť
- Hodnota tohto kľúča určuje, aký príkaz sa pošle robotu na vykonanie. Dva príkazy nie sú implementované, a teda robot má možnosť na rozšírenie svojej funkcionality.

Príkazy, ktoré vie robot spracovať sú:

0. Prázdný príkaz slúžiaci na overenie spojenia
1. Núdzové zastavenie
2. Normálne zastavenie
3. Príkaz nastavujúc rýchlosť kolies mobilného robota
4. Prázdný príkaz
5. Prázdný príkaz
6. Príkaz pýtajúci si aktuálne rýchlosť pravého a ľavého kolesa.
7. Pripravenie motorov robota
8. Príkaz pýtajúci si aktuálnu pozíciu pravého a ľavého kolesa.

- **RightWheelSpeed** – Nastavenie rýchlosť pre pravé koleso. Tento parameter je povinný len pre príkaz 3.
- **LeftWheelSpeed** – Nastavenie rýchlosť pre ľavé koleso. Tento parameter je povinný len pre príkaz 3.

Spracovanie týchto správ je spravené v samostatnom vlákne. Toto vlákno je vytvorené pomocou metódy **polling**. Neustále dookola sa volá funkcia *read* (čítaj). Táto funkcia je nastavená tak, aby nezablokovala vlastné vlákno.

Študenti, ktorí navrhovali systém posielania požiadavok (request) a odpovedí (response) robili tieto správy ručne. Preto nastávajú situácie, kedy robot pošle správu, ktorá nezodpovedá štandardu písania textu v JSON štardarde. Z tohto dôvodu sme nemohli použiť už existujúci kód na analýzu tohto typu textu (parser), ktorý by nám zjednodušil prehľadávanie týchto správ.

1.2.3 Zapínanie robota

Pri každom zapnutí robota prechádza systém robota do BIOS-u (Basic Input Output System). V tomto stave sa nedá s robotom spraviť nič. Musíme preto napojiť monitor a klávesnicu na robot, prejsť cez BIOS a naštartovať operačný systém. Keď tento postup nespravíme, tak operačný systém na robote nebude vedieť komunikovať s Arduino doskou, ktorá ovláda robot. Tento problém sme skúmali podrobnejšie a zistili sme, že je spôsobený samotným operačným systémom. Je to teda chyba, ktorú my nevieme opraviť.

2 Návrh implementácie riadenia

Našou úlohou v tejto bakalárskej práci je naprogramovať a otestovať nami vytvorený ovládač pre mobilného robota. Na výber máme viacero technológií, ktoré môžeme použiť. Na vytvorenie ovládača budeme potrebovať operačný systém na počítači, ktorý budeme používať na programovanie a testovanie. Uvažovali sme nad dvoma. Sú to operačný systém *Windows* alebo operačný systém *Ubuntu*, ktorý je založený na jadre Linux. Ďalšiu vec, ktorú budeme potrebovať je programovací jazyk. Existuje veľa programovacích jazykov, v ktorých toto zadanie vieme naprogramovať a to hlavne *C*, *C++*, *Python*, *Java* alebo *Matlab*. Keďže, zo zadania vypláva, že máme použiť Robotický operačný systém, tak si môžeme vybrať z doch možností. Sú to Robotický operačný systém prvej verzie (*ROS1*) a Robotický operačný systém druhej verzie (*ROS2*). Na vytvorenie ovládača môžeme využiť aj už naprogramované balíčky. Jednou z možností je *ros_control* pre Robotický operačný systém prvej verzie a *ros2_control* pre Robotický operačný systém druhej verzie. Samozrejme na implementáciu a otestovanie potrebujeme samotného robota. Pred tým ako začneme preberať spomenutý ovládač a robota, ktorého budeme riadiť, tak si predstavíme technológie, ktoré budeme používať pri vytváraní ovládača.

2.1 Operačný systém

Na začiatku našej práce sme sa museli rozhodnúť, či budeme písat program na náš ovládač v operačnom systéme Windows 11 alebo v operačnom systéme Ubuntu, ktorý je založený na jadre Linux. Ako programátori sme už dlhšie pracovali s operačným systémom Ubuntu a poznáme aj jeho výhody, aj nevýhody. To isté platí aj pre operačný systém Windows. Toto sú fakty, ktoré sme zvažovali pri výbere operačného systému, ktorý sme používali počas celej bakalárskej práce.

2.1.1 Windows

Windows je najpoužívanejším operačným systémom na počítače na svete. Je to hlavne preto, že je jednoduchý na používanie pre menej techniky zdatných užívateľov. Windows ale nie je bezpečným operačným systémom a taktiež aj jeho stabilita a rýchlosť nie sú na najvyššej úrovni. Tento operačný systém používajú ľudia hlavne na voľnočasové aktivity ako je hranie hier alebo v práci ako napríklad v kanceláriách na úpravu tabuľiek. Od zavedenia ROS sú robotickí vývojári nútene používať Linuxové počítače, pretože ROS bol primárne zameraný na operačný systém Ubuntu. [9] Preto pre nás ako robotníckych programátorov tento operačný systém nie je výhodný. Nastavovanie prostredia pre programovanie je v tomto operačnom systéme oveľa náročnejšie ako v operačnom systéme s Linuxovým jadrom.

2.1.2 Ubuntu

Ubuntu je stabilný a bezpečný operačný systém, ktorý je založený na Linuxe. Väčšina ľudí hovorí o Linuxe ako o operačnom systéme. Pravdou to ale nie je. Linux je totižto len **kernel** (jadro) operačného systému. Kernel je hlavný program, ktorý sa stará o správu hardvéru, riadenie procesov a správu pamäte. Zároveň poskytuje možnosť operačnému systému komunikovať s hardvérom a poskytuje jednotlivým programom prístup k hardvéru a jeho časťam [10].

Linux je používaný skoro vo všetkých mobilných zariadeniach, serveroch, cloudoch (dátové servery, na ktoré sa pripájame cez internet) a ďalších elektronických zariadeniach, ktoré potrebujú rýchlosť, stabilitu a bezpečnosť. Je taktiež často používaný v mobilných robotoch. Nad operačným systémom Windows má niekoľko výhod.

- **Open source** Linux je open source jadro pre operačný systém, čo znamená, že kód, ktorý tvorí jeho základ, je k dispozícii pre všetkých zadarmo a každý môže prispieť k jeho vylepšeniu. Tento otvorený prístup umožňuje programátorom prispôsobiť si Linux pre svoje potreby a vytvárať programy, ktoré sú optimálne pre ich prácu. Na Linuxe sú založené aj operačné systémy reálneho času alebo softvére pre smart mobilné telefóny ako je napríklad operačný systém Android.
- **Programovacie nástroje** Linux obsahuje mnoho vynikajúcich programovacích nástrojov, ktoré sú k dispozícii zadarmo a sú široko používané programátormi. Napríklad k dispozícii v Linuxe sú (GCC) GNU Compiler Collection (Kolekcia GNU prekladačov) (GNU je skratka pre GNU is not Unix) a GNU Debugger (GDB) sú vynikajúce nástroje pre C/C++ programátorov.
- **Bezpečnosť** Systémy s jadrom Linux sú len zriedkavo nakazené škodlivými programami, to robí z operačných systémov založených na Linuxe veľmi bezpečnými operačnými systémami. [11] V Linuxe je ľahšie infikovať sa škodlivými programami, pretože programy majú obmedzené oprávnenie a prístup k ďalším súborom. Systém je navrhnutý tak, aby bol odolný voči útokom.
- **Stabilita** Linux má tendenciu byť stabilnejší ako Windows, pretože nie je závislý na ovládačoch a softvéri od tretích strán. Linux poskytuje jednotný prístup k správe a aktualizácii softvéru. Pri vylepšovaní systému nie je za potreby reštartovať počítač alebo server, čo má za dôsledok zvýšenie spoločnosti zariadenia.
- **Flexibilita** Linux je veľmi flexibilný operačný systém, ktorý umožňuje programátorom prispôsobiť si svoje pracovné prostredie a používať programy, ktoré najlepšie vyhovujú ich potrebám. Linux dovoluje upraviť samotné jadro operačného systému a vybrať si

z neho len tie časti, ktoré užívateľ potrebuje. Tie, ktoré nepotrebuje sa v systéme nenachádzajú. Týmto spôsobom sa zvyšuje rýchlosť systému a zabezpečuje sa jeho bezpečnosť.

- **Podpora a komunita** Linux má veľkú a aktívnu komunitu. Ta obsahuje ako aj programátorov, tak aj menej technicky zdatných ľudí. Táto komunita poskytuje podporu a rieši problémy. Vďaka tomu, že Linux je open source, mnoho ľudí prispieva k jeho vylepšeniu a vytvára nové programy, čo znamená, že je vždy niečo nové na objavovanie.
- **Komplexnosť** Napriek všetkým týmto výhodám má Linux aj svoje nevýhody. Najväčšou nevýhodou, ktorú je treba spomenúť je komplexnosť operačného systému. Linux je komplexnejší ako operačný systém Windows, hlavne kvôli svojej flexibilnosti. Tým, že si vieme nastaviť vlastné prostredie, typ systému správy súborov, programy na sťahovanie, publikovanie a správy balíčkov. Ľudia, ktorí nevedia tento systém opraviť, často siahajú po preinštalovaní systému.

Vďaka svojim výhodám sme sa rozhodli použiť operačný systém Ubuntu verzie 22.04.

2.2 Programovací jazyk a jeho prostredie

Medzi programovacie jazyky, ktoré by sme mohli použiť patria: *C++, C, Python, Matlab, Java*. Aby sme sa rozhodli, ktorý jazyk použijeme pri vytváraní ovládača, tak sme si spísali ich výhody a nevýhody.

- **C a C ++:** C a C ++ sú hlavné jazyky používané pri vývoji ROS aplikácií. Sú to výkonné a efektívne jazyky, ktoré možno použiť na nízko úrovňové programovanie, ako sú riadiace slučky, ovládače a systémové programovanie. C++ je najčastejšie používaný jazyk v ROS2.
- **Java:** Java je ďalší jazyk, ktorý sa môže použiť na implementáciu aplikácií v Robotickom operačnom systéme. Je menej bežné používaný ako jazyk C++, ale má výhodu, a to takú, že je prenositeľný jazyk, ktorý môže byť spustený na akejkoľvek platforme, ktorá podporuje JVM (Java Virtual Machine) (Virtuálny stroj podporujúci Javu). Java je vhodná pre vývoj vysoko úrovňových komponentov robotického systému, ako sú GUI (z anglického Grafical User Interface) (užívateľské rozhranie) a sietová komunikácia.
- **Python:** Python je populárny jazyk pre vývoj ROS2 kvôli svojej čitateľnosti a jednoduchosti použitia. Je vhodný pre vývoj vysoko úrovňových komponentov robotického systému, ako sú algoritmy na rozhodovanie, spracovanie dát a vizualizácia.

- **Matlab:** Matlab je vysoko úrovňový programovací jazyk a prostredie používané vo vedeckom prostredí, analýze dát a vizualizácií. Je to výkonný jazyk pre numerické výpočty a často sa používa na simulovanie a testovanie algoritmov pre roboty. Avšak, nie je bežne používaný v ROS2 vývoji. Na používanie tohto rozhrania je spravený modul *ROS Toolbox*, ktorý zabezpečuje jednoduchú implementáciu uzlov a ich komunikácie.
- **Výkon:** C a C++ poskytujú najlepší výkon pre ROS aplikácie kvôli ich nízkoúrovňovej implementácii a priamemu prístupu k pamäti. Python, Matlab a Java sú pomalšie ako C a C ++ kvôli ich interpretácii.
- **Čas vývoja:** Python a Java sa vyvíjajú rýchlejšie ako C a C++ kvôli ich jednoduchosti a jednoduchosti použitia. Matlab môže byť tiež rýchlejší pri vývoji pre určité úlohy vďaka svojim zabudovaným knižniciam a nástrojom. Medzi tieto nástroje patri napríklad *Simulink* alebo *Stateflow*.
- **Kompatibilita:** Všetky uvedené jazyky môžu byť použité s Robotickým operačným systémom a môžu komunikovať s robotom pomocou TCP/IP komunikácie.

Na základe vyššie spomenutých rozdielov sme sa rozhodli použiť programovací jazyk C++. Toto rozhodnutie sme spravili hlavne kvôli jeho možnostiam v implementácii tried, výkonusu a podpore v komunite.

2.3 Robotický operačný systém

Robotický operačný systém (ROS) (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 budeme hovoriť o spoločných vlastnosťach a funkcionalityach, ROS1 a ROS2 budeme označovať dokopy ako ROS.

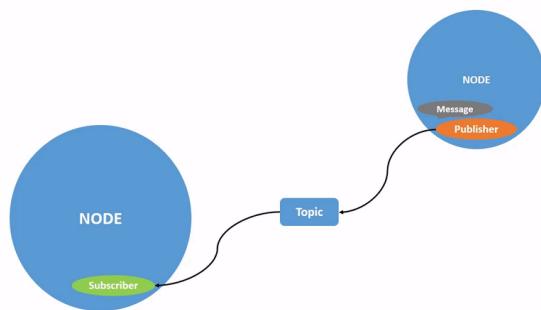
ROS je open source, čo znamená, že ide o softvér uvoľnený pod licenciou, kde má používateľ práva na použitie, štúdium, zmenu a redistribúciu. Mnohé open source licencie však určujú určité obmedzenia na tieto práva pre tento softvér, ale v podstate môžeme predpokladať tieto práva. Najbežnejšie licencie pre ROS2 softvérové balíčky sú Apache 2 a BSD, hoci vývojári majú slobodu používať aj iné [12]. Využitia Robotického operačného systému v reálnom svete sa môžu objaviť v automobiloch, mobilných robotoch, dronoch alebo manipulátoroch.

2.3.1 Základné pojmy

Komunikácia v ROSe je zabezpečená cez IPC (Inter Process Communication), TCP/IP a UDP/IP komunikáciou pomocou troch zakladacích metód: **témy** (Topics), **služby** (Services) a **akcie** (Actions). Všetky tieto metódy slúžia na posielanie správ na synchronizáciu vykonávania programu a komunikáciu jednotlivých častí programu. Na to, aby sme ich vedeli správne využiť musíme poznat' ich základnú implementáciu a funkcionality. Správanie týchto metód je nasledovné.

2.3.2 Témy

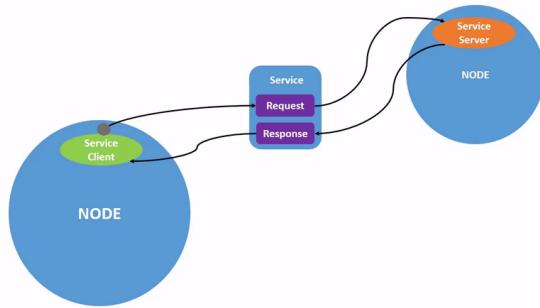
Témy sú sprostredkované pomocou IPC, medzi procesová komunikácia z anglického Inter Process Communication, alebo UDP/IP komunikácie. Je to najjednoduchší spôsob komunikácie. Komunikácia pomocou tém prebieha následovne. Definujeme si jedného poskytovateľa (publishers) a jedného alebo viacerých príjemcov (subscribers). Medzi týmito dvoma alebo viacerými účastníkmi sa následne posielajú správy (messages), ktoré sme si dopredu definovali. Parametre a obsah týchto správ je definovaný priamo v téme. Nové témy si vieme sami vytvoriť. V prípade, že chceme vytvoriť novú tému, musíme si vytvoriť nový balíček, v ktorom si spravíme súbor s príponou *.msg*.



Obr. 2.1: Vizualizácia témy v ROSe [13]

2.3.3 Služby

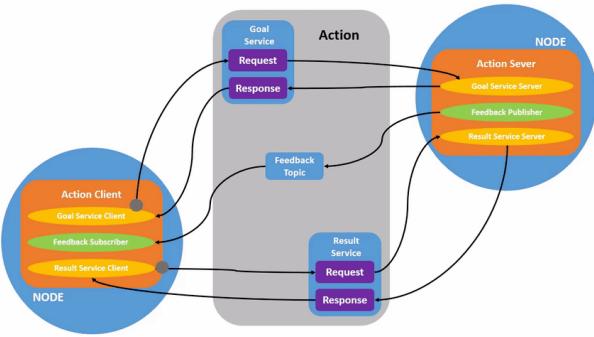
Služby sú sprostredkované pomocou TCP/IP protokolu. Poskytujú nám rovnaký spôsob komunikácie ako témy, až na to, že sa správy medzi servisom a klientom posielajú cez LAN (Local Area Network). Tieto správy sa posielajú oboma smermi. Služby sa využívajú pri komunikácii medzi viacerými zariadeniami. Jedno zariadenie posielá požiadavku (request) a druhé zariadenie túto požiadavku prijme a spracuje. Po spracovaní pošle naspäť odpoveď (response). Odpoveď môže obsahovať jednoduchú potvrdenie, alebo inú komplexnejšiu štruktúru. Nové služby si vieme sami vytvoriť. V prípade, že chceme vytvoriť novú službu, musíme si vytvoriť nový balíček, v ktorom si spravíme súbor s príponou `.srv`.



Obr. 2.2: Vizualizácia služby v ROSe [13]

2.3.4 Akcie

Akcie sú taktiež sprostredkované TCP/IP protokolom. Sú najzložitejším spôsobom komunikácie. Tento spôsob bol pridaný do ROS1 až neskôr. V ROS@ je tento typ komunikácie medzi troma základnými formami komunikácie uzlov. Sú založené na službách a prebiehajú asynchronne [12]. Máju 3 stavy vidieť Obr. 2.3. Najprv pošle klient serveru, akú akciu má vykonávať, server mu potvrdí, 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 potvrdí obdržanie výsledku. Nové akcie si vieme sami vytvoriť. V prípade, že chceme vytvoriť novú akciu, musíme si vytvoriť nový balíček, v ktorom si spravíme súbor s príponou `.action`.



Obr. 2.3: Vizualizácia akcie v ROSe [13]

2.3.5 Parametre

Parametre sú spôsob, ako môže komunikovať užívateľ so základnými nastaveniami uzlov bez potreby zmenenia kódu a jeho následnej komplilácie, čo pri väčších projektoch môže zabrať veľmi dlho. Konfigurácie sa definujú 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.

2.4 ROS1

ROS bol prvýkrát vydaný v roku 2007. Jeho tvorcovia *Keenan Wyrobek* and *Eric Berger* zo Stanford univerzity začali ROS ako osobný projekt za účelom odstránenia procesu znovaobjavovania kolesa [14]. Ide o softvér, ktorý sa začal využívať so zámerom zjednodušiť a zjednotiť programovanie a ovládanie robotov. Od doby, kedy vznikol prešiel mnohými verziami a úpravami. Jeho neoddeliteľnou súčasťou sú 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. Tvorcovia pritom dbali na to, aby tento systém vytvorili 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 ďalej poskytuje parametre jednotlivým uzlom z parametrového servera. Jeho najdôležitejšou úlohou je zabezpečenie komunikácie uzlov v programe.

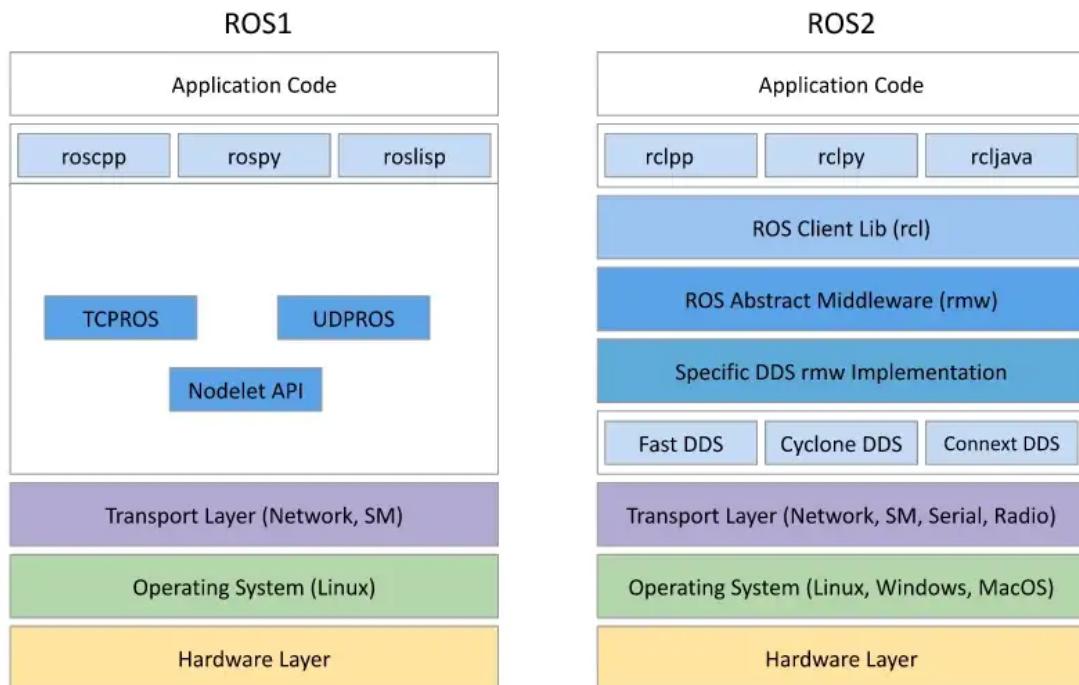
Testovanie prebieha pomocou použitia Google testov. Debugovanie je možné uskutočniť pomocou debuggera gnu-gdb. Pri spustení programu cez spúšťací súbor (launch file) je potrebné pridať príkaz na spustenie spomenutého debugovacieho programu.

Aj napriek mnohým výhodám má ROS1 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 [15],
- Možnosť užívateľa predefinovať základne prvky ROS-u.

Kvôli takýmto problémom a nedostatkom sa začala vyvíjať nová verzia 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.

2.5 ROS2



Obr. 2.4: Porovnanie štruktúr ROS1 a ROS2 [16]

Ako už bolo spomenuté zámerom vývoja ROS2 bolo zlepšenie funkcionality a bezpečnosti systému. Jeden z dôsledkov tohto vývoja je, že ROS2 nie je späť kompatibilný software. 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. Odporučané 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 2.3.2, služieb 2.3.3 a akcií 2.3.4.

Táto podobnosť končí na najvyššej vrstve. Ako sme videli na Obr. 2.4. Štruktúra ROS2 je rozdelená do viacerých vrstiev. Najdôležitejšie je pre nás vedieť, že komunikácia je spracovávaná modelom DDS (Služba distribúcii údajov) z anglického (Data Distribution Service). DDS obsahuje podobné dizajnové vzory ale poskytuje väčšiu ovládateľnosť komunikácie systémov, zatiaľ čo odstraňuje všetky vrstvy, ktoré zabraňujú debugovaniu [17]. Tento model zároveň zlepšuje výkon, stabilitu a bezpečnosť modelu oproti ROS1. Je založený na TCP/IP a UDP/IP protokole. Z obrázku 2.4 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 RTOS (Operačné systémy reálneho času) z anglického Real time operating system. 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.

Testovanie programu prebieha rovnako ako v ROS1. Jediný rozdiel je v jazyku v ako sa nastavia jednotlive rozšírenia.

ROS2 má necentralizovanú štruktúru, a preto pri spúštaní programov už nie je potrebné mať spustený roscore. Ak teda spadne jeden uzol všetky ostatné uzly budú fungovať nadľahčie. V ROS1 sme vedeli ovplyvniť počet uchovaných správ komunikácie pokým nepretiekol zásobník, ktorý ich uchovával na neskoršie použitie. V ROS2 vieme implementovať túto schopnosť použitím takzvanej *QoS* triedy (kvalita komunikácie), z anglického Quality of Service. Pomocou tejto triedy 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.

Ak by bol užívateľ veľmi schopný programátor a potreboval by si zmeniť triedy, ktoré definujú základnú funkcionality ROS-u, tak aj toto je možné. Jednou z takýchto funkcionálít je, že užívateľ si vie predefinovať triedu, ktorá bude alokovať miesto na (IPC) komunikáciu (medziprocesovú komunikáciu). K tomuto bodu je dodat, že tento prípad je špecifický a väčšina programátorov sa s takouto možnosťou do kontaktu nedostane.

Pri všetkých týchto zlepšeniach nemôžeme zabudnúť spomenúť aj nasledovný nedostatok. Keďže 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 práce pribudli na stránke dokumentácie minimálne 2 strany popisujúce pokročilejšie Funkcionality druhej verzie ROSu.

2.6 Rozdiely medzi ROS1 a ROS2

Čo je určite dobrou správou pre všetkých programátorov, ktorí robili v prvej verzii a sú zvyknutí na jej štandardy a funkcionality. Tak títo 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. Keďže celý projekt bol písaný v programovačom jazyku C++, tak sa aj tieto zmeny budu týkať hlavne jazyka C++.

2.6.1 Štandard jazyka

Pokým ROS1 bola písaná v štandarde C++03, tak ROS2 je už písaná v novom štandarde. A to hlavne C++11, ale používa aj nejaké časti z C++14 a C++17. To zahrňa inicializovanie templatov a ich používanie. Tým, že ROS2 je stále nová a stále vyvíjajúca sa platforma, tak môžeme očakávať aj časti kódu, ktoré budú podporovať najnovší C++ štandard a to štandard z rokov 2020 a 2023.

Definície a deklarácie templatov sú na knihu samú o sebe, preto do detailov nebudeme zachádzať. 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, keď ho spustíme, tak by spadol z dôvodu, že čítame iný typ správy ako posielame.

2.6.2 Inicializácia uzla

Tak isto ako v prvej verzii aj v druhej verzii musíme definovať uzol (node). Rozdiel je v tom, že prvá verzia obsahovala NodeHandle (Ovládač uzla) a druhá verzia obsahuje priamo Node (Uzol). V druhej verzii je zaužívaným štandardom tento uzol odvodiť od základnej triedy Node použitím polymorfizmu, ktorý bude existovať počas celej doby vykonávania programu. Pri prvej verzii tomu tak nebolo. Tam sme museli vytvoriť už spomenutý NodeHandle. Ten sa nemusel využiť ako základná trieda a nemusel ani existovať počas celého behu programu.

2.6.3 Komunikácia

DDS (Služba distribúcie údajov) je protokol strednej vrstvy (middleware) implementovaný nad UDP [12]. Tento protokol je použitý v IoT (internet vecí) (Internet of Things) sfére. Využíva sa napríklad v oblastiach ako sú telekomunikácie, vizualizácia, automatizácia, alebo zdravotníctvo [18]. DDS 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' kapitolu 2.3.4.

2.6.4 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. V prípade, že sa tieto parametre nepodari vytiahnuť z konfiguračného súboru. Či už z dôvodu, že daný súbor neexistuje, alebo iného dôvodu, tak sa aplikujú základné hodnoty, ktoré si zvolil užívateľ pri používaní funkcie na ich zistovanie.

2.6.5 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. Sú to napríklad mikroprocesory, ktoré ovládajú roboty.

2.6.6 Kompilácia

Zmenou verzii sa zmenil aj spôsob kompliacie programu. ROS1 bol kompliovaný pomocou catkin build systému. Catkin je založený na programe cmake. Jeho nastavenie závislostí 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 závislosti sa nastavujú pomocou setup.py súboru. V prípade colcon-u si môžeme definovať spôsob kom-

pilácie to znamená, že môžeme nastaviť, ako sa budú spracovávať závislosti. 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 závislosti pomocou XML súboru ako tomu bolo v ROS1 pričom možnosť definície pomocou Python skriptu ostáva. Je to jeden zo spôsobov, ako zmeniť rozdiel medzi ROS1 a ROS2.

2.6.7 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 [15].

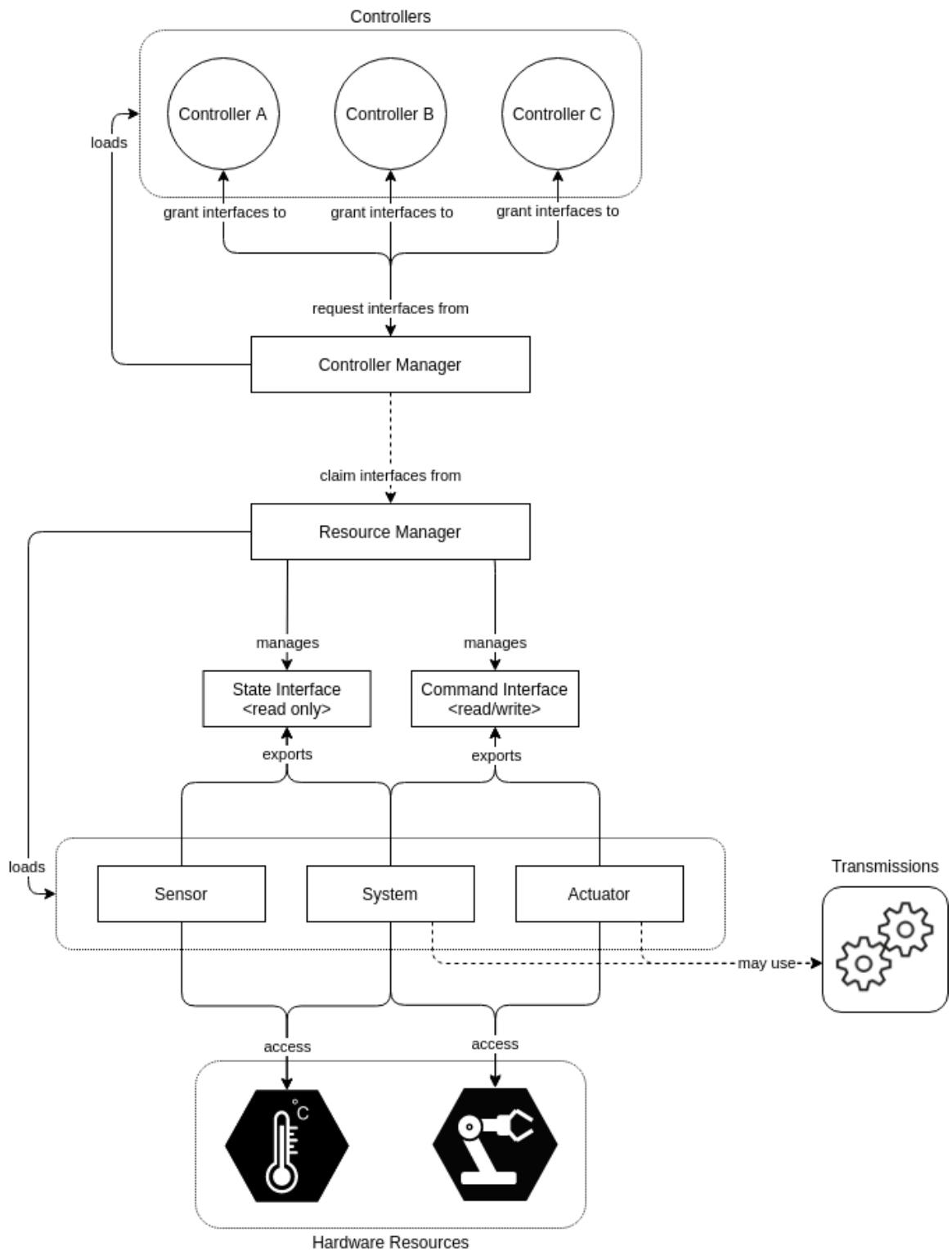
2.7 Ros_control

`Ros_control` je balík naprogramovaný v prostredí ROS1 aj ROS2. Tým, že sme si vyбрали ROS2 ako prostredie pre naše riešenie, tak by sme použili balík `ros2_control`. Obrazok 2.5 zobrazuje architektúru balíka `ros2_control`.

Delegátor snímačov **Controller Manager** (CM) prepája snímače a abstrakciu hardvéru balíka `ros2_control`. Slúži taktiež ako vstupný bod pre užívateľov ROS služieb [19]. Delegátor snímačov na jednej strane ovláda snímače a rozhrania, ktoré potrebujú. Na druhej strane má prístup k hardvérovým komponentom, t.j. ich rozhraniam. Delegátor snímačov spája požadované a poskytované rozhrania, poskytuje prístup ovládačom k hardvéru, keď ho potrebujú alebo hlási chybu, ak existuje konflikt prístupu [19].

Delegátor zdrojov **Resource Manager** (RM) spravuje fyzický hardvér a ich ovládače (hardvérové komponenty) pre balík `ros2_control` [19]. V tejto triede je možnosť definovať takzvané *callbacky* (ukazovatele na funkcie, ktoré môže užívateľ definovať) `read()` a `write()`, ktoré zabezpečujú komunikáciu hardvérových komponentov [19].

Hardvérové komponenty **Hardware components** (HC) sú triedy, ktoré realizujú komunikáciu s fyzickým hardvérom a jeho reprezentáciou v balíku `ros2_control` [19].



Obr. 2.5: Diagram zobrazujúci vnútornú implementáciu balíka `ros2_control` [19].

V tomto balíku sú definované tri základné typy komponentov:

- **Systém** Komplexný (multi-DOF (Degrees of Freedom) (z anglického veľa stupňov volnosti)) robotický hardvér ako industriálny robot. Hlavný rozdiel medzi akčným členom a možnosťou použitia komplexného prechodu aký potrebujeme pri robotickom ramene ruky. Komponent poskytuje čítanie a zápis do hardvéru. Používa sa v len prípade jednej logickej komunikácie kanálov hardvéru (napríklad KUKA-RSI) [19].
- **Senzor** Robotický hardvér prizvaný na zaznamenanie okolitého prostredia. Senzor ako komponent je príbuzný s kĺbom (napríklad prekladač) alebo rameno (napríklad snímač sily a krútiaceho momentu). Tento komponent ma len schopnosť čítania vlastností [19].
- **Akčný člen** Jednoduchý (1 DOF) robotický hardvér ako motor, chlopňa alebo podobný člen. Akčný člen implementuje len jeden kĺb. Tento komponent má schopnosť čítať aj zapisovať vlastnosti. Čítanie nie je požadované, ak nie je uskutočnitelné (napríklad DC ovládač motora s Arduino doskou). Tento akčný člen môže byť tiež použitý s viac stupňovými kĺbmi voľnosti robota, ak jeho hardvér umožňuje tento dizajn (napríklad CAN-komunikácia s každým motorom nezávisle) [19].

Aj napriek všetkým týmto výhodám nemôžeme tento softvér použiť. Je to kvôli tomu, že na robote sa systém ROS nenachádza a nemá preto prístup k jednotlivým časťiam hardvéru. Ak by tento systém na robote bol implementovaný, mohli by sme použiť funkciu **Delegátora zdrojov** na čítanie a zapisovanie dát do jednotlivých komponentov. Mienime si preto vytvoriť vlastný systém na čítanie, spracovanie a ovládanie robota.

3 Zlepšenie stavu robota

3.1 Chyba v systéme

Prvýkrát, keď sme prišli k robotu do NCR (Národné Centrum Robotiky), tak sme si určili ako prvú úlohu zálohovať systém. Ak by sa teda stala nejaká chyba a robot by prestal fungovať, tak by sme sa vedeli dostať do posledného funkčného stavu robota. Zálohovanie systému nanešťastie nebolo možné hned po zapnutí robota. Bolo to spôsobené poškodeným balíkovým systémom. Tento problém bol zapríčinený nesprávnou inštaláciou a následným nesprávnym odstránením ROS1 (Robot Operating System). Verzia, ktorá bola pôvodne nainštalovaná na robote bola *Lunar Loggerhead*. Toto by problém nespôsobilo, čo ale problém spôsobilo, bolo jeho nesprávne odinštalovanie. Táto akcia mala za dôsledok vypisovanie nasledovnej chybovej hlášky pri zálohovaní.

```
E: Unable to~correct problems, you have held broken packages.
```

Tento problém sme vyriešil vymazaním všetkých knižníc, ktoré boli nainštalované spolu s ROS1. Toto vyriešilo problém. Na vymazanie týchto knižníc sme použili nasledovný príkaz.

```
sudo apt autoremove && sudo dpkg --remove $(dpkg --get-selections | grep hold)
```

Tento príkaz najprv odstráni všetky balíčky, ktoré nie sú používané žiadnou aplikáciu v systéme. Následne vyhľadá všetky knižnice, ktoré taktiež nie sú používané žiadnym balíčkom a nasilu odinštaluje. Po vykonaní tejto operácie sme ešte museli odstrániť všetky referencie na ROS1 v systéme. Toto sme vykonali odstránením prepojení na už neaktívne repozitáre, ktoré sa nachádzali v súbore */etc/apt/sources.list*.

3.2 Chyba v dokumentácii

V kapitole 1.2.2 sme uviedli pôvodný stav ovládania robota pomocou správ typu JSON. Z kódu 1.2.2 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 prepísať na reťazec, ktorý by fungoval:

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

3.3 Oneskorenie komunikácie

Pri blízkom pohľade na kód robota sme odhalili závažnú chybu. Vo vlákne, ktoré zabezpečuje prijímanie správ sa nachádza cyklus, ktorý prijíma dátu z TCP/IP spojenia a bezpečne ich zapisuje do zdielanej štruktúry. Prijímanie správ je realizované pomocou funkcie *recv* z knižnice *sys/socket.h*. Táto funkcia je za normálach podmienok blokujúca, to znamená, že vlákno, ktoré

ju volá, sa zablokuje, až pokým nemá k dispozícii dátu na čítanie. Na robote je toto blokovanie vypnutý s tým, že po každom zavolaní tejto funkcie sa čaká *100 milisekúnd*. Tento problém je ľahko opraviteľný. Čo sme spravili bolo, že sme odstránili funkciu *usleep*, ktorá zabezpečovala čakanie. Vo funkciu *recv* sme d'alej vymazali makro *MSG_DONTWAIT*, ktoré spôsobuje, že sa funkcia *recv* nezablokovala.

3.4 Rozšírenie príkazov robota

Prvotne sme mali v plane získavať pozíciu robota pomocou príkazov, ktoré sa nachádzajú v knižnici enkóderov. Tieto príkazy mali vracať počet impulzov IRC snímačov prejdených od spustenia robota. Pre zaobchádzanie s touto funkcionálitou sme definovali dva príkazy:

Command: 4

Tento príkaz je prázdny. My sme ho ale neskôr prepísali na príkaz, cez ktorý sa dá nastaviť žiadana pozícia kolies robota (ich natočenie) pomocou funkcie z knížnice EPOS [20]. Táto funkcionálita nie je v takom stave ako sme si priali. Je to spôsobené hlavne nepostačujúcou dokumentáciou enkóderov na robote. Síce sme našli v dokumentácii funkciu, ktorá by mala túto možnosť povolovať. Čo sa ale stane pri poslaní príkazu je to, že kolesá sa začnú točiť rýchlosťou 0,5 metra za sekundu.

Command: 8

Príkaz na zistovanie polohy kolies neboli pôvodne naprogramovaný na robote. Pridali sme ho s cieľom presného dostavenia sa robota na preddefinované miesto. Táto funkcionálita nefunguje správne rovnako ako v predchádzajúcim príklade, keď si vypýtame polohu kolies od robota, dátu ktoré obdržme sú, že jedno koleso je priamo nastavené na hodnotu, ktorú sme si vyžiadali a to druhé koleso vráti náhodnú hodnotu. Počas toho sa ale kolesá robota stále točia.

Tento postup sa ukázal ako neuskutočiteľný, lebo príkazom 8 sme z jedného enkódery získavalí informácie priamo nastavenej polohy a z druhého enkódera sme dodržiavali čisto náhodné dátu. Z tohto dôvodu sme sa rozhodli využívať spätnú väzbu, ktorá obsahuje rýchlosť robota.

3.5 Nesprávna spätná väzba

Ako bolo spomenuté vyššie, pri poslaní príkazu s číslom 8 nám robot vráti aktuálne rýchlosť kolies. Počas skúšania tejto funkcionality sme narazili na problém. Ked' sme sa robota spýtali na jeho rýchlosť. Dostali sme ret'azec, ktorý obsahoval náhodne veľké čísla. Tieto čísla sa menili, ked' sme zadávali nejaké hodnoty pre rýchlosť kolies aby sa robot hýbal. Ich magnitúda ale ostávala nezmenená. V nasledujúcom príklade môžeme vidieť ako tento ret'azec vyzeral:

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

Ako si môžeme všimnúť'. Pri tomto type správ nie je dodržaná správna forma ret'azca typu *JSON*. Namiesto ':' máme '=' a medzi argumentmi sa nenachádza čiarka. Hned' ako prvú vec sme chceli tento štandard napraviť'. Bohužiaľ na tomto robote už bolo spravených niekol'ko projektov a museli by sme prejst' každý z nich a zistit' či používajú túto spätnú väzbu. Ak by ju používali museli by sme tieto kódy upravit'.

V dokumentácii robota bohužiaľ nebolo písané v akom formáte sa tieto rýchlosť kolies majú nachádzat'. Preto jeden z nápadov ako zistit' presne v akom formáte sa posielali tieto čísla bolo vyskúšať pár možností:

- *long* - celé číslo s malým endianom
- *long* - celé číslo s veľkým endianom
- *float* - desatinné číslo s malým endianom
- *float* - desatinné číslo s veľkým endianom

Ked'že robot má počítač so 64 bytovým procesorom [2], tak *long* aj *float* budú mať 64 bitovú dĺžku. Po skúsení všetkých štyroch možností sa ukázalo, že ani jedna nebola správna a problém je niekde inde.

Aby sme pochopili, ako sa máme správať k prijatým dátam musíme vedieť ako funguje program na robote. Skopírovali sme si preto kód z robota a pustili sme sa do jeho analýzy. Pri poslaní požiadavky na nastavenie rýchlosť kolies si ich robot premení na celé čísla v rozsahu 0 až 1000. To je hodnota, na ktorú nastaví rýchlosť otáčania pravého a ľavého kolesa respektíve rýchlosť otáčania ich motorov. Na druhú stranu, ked' si vypýtame od robota rýchlosť kolies. On zoberie informáciu z enkóderov a pošle nám ju 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 posiela robot nie sú ani vytahované z enkóderov správou 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 [20]. Táto dokumentácia nebola moc nápmocná. Opisy jednotlivých funkcií boli len ich rozložené názvy na osobitné slová. Aj napriek tomu sa nám podarilo nájsť funkcie, ktoré sme potrebovali. Funkcie obsahujúce slovo ‘Target’, majú návratné hodnoty reprezentujúce žiadane hodnoty. Funkcie s príponou ‘-Is’ vracajú aktuálne hodnoty. Z tohto dôvodu sme museli prepísat funkciu na robote, ktorá sa vykonávala. Pre získanie aktuálnych hodnôt rýchlosť motoru poslaním príkazu 6 sme museli zmeniť nasledujúcu funkciu:

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

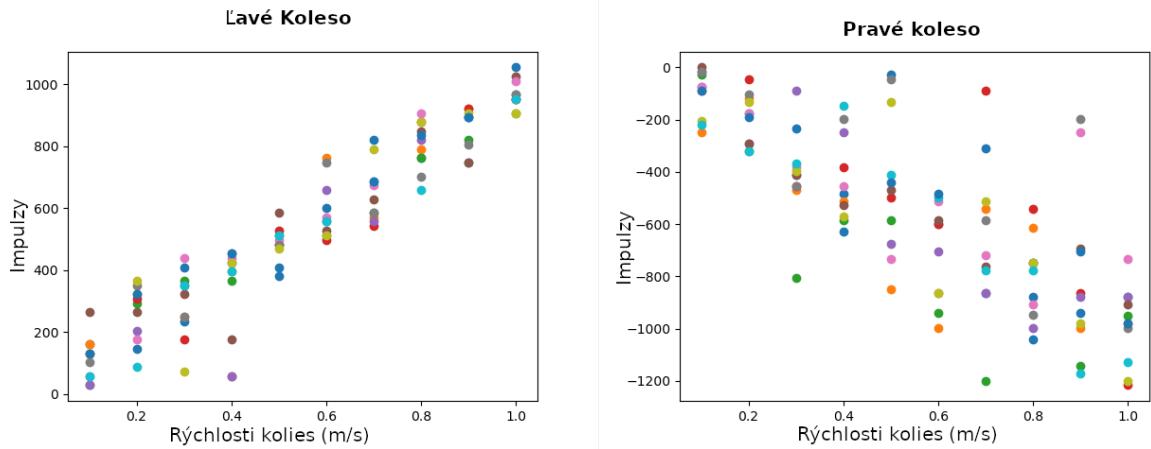
BOOL VCS_GetVelocityIs(
    HANDLE KeyHandle,
    WORD NodeId,
    long* pVelocityIs,
    DWORD* pErrorCode);
```

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

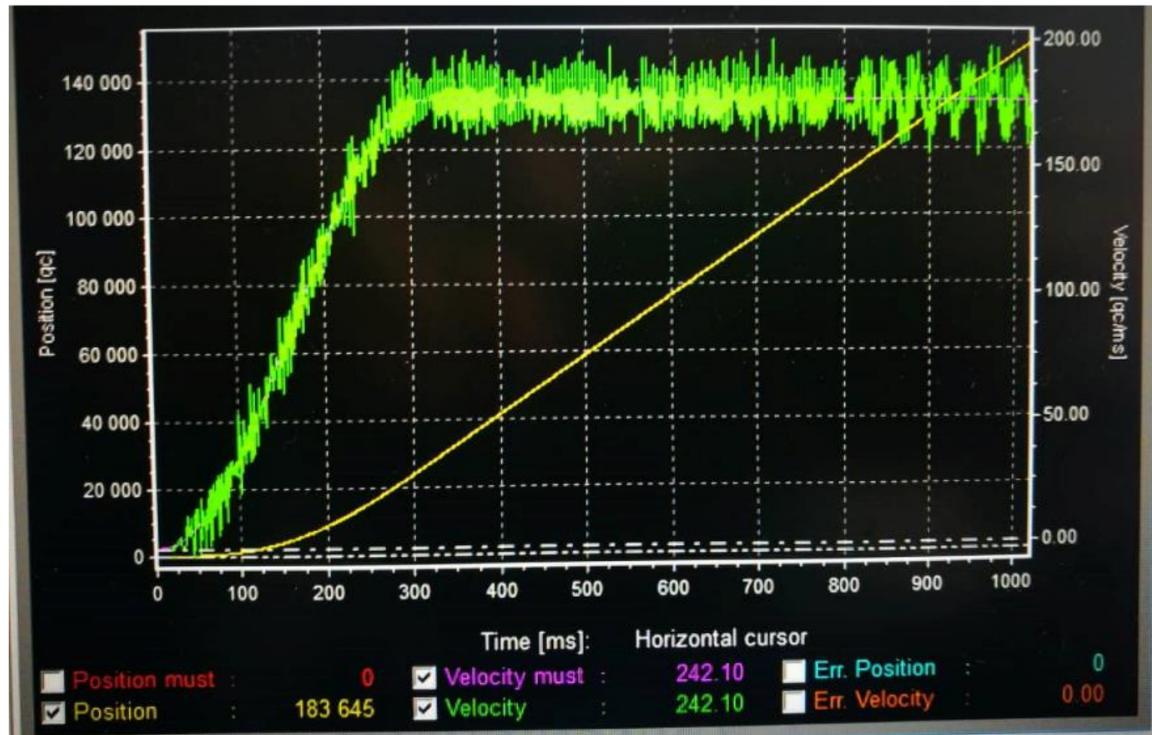
3.6 Zašumený výstup

Po prepísaní funkcie na získavanie rýchlosťí robota sme spravili pári meraní, aby sme zistili, aké presné informácie o rýchlosťach motorov dostávame. Vykonali sme meranie prechodovej a prevodovej charakteristiky oboch motorov, tým, že bol zabezpečený pohyb kolies vo vzduchu s vypodložením robota.

Na obrázku 3.2 vidíme zašumený signál rýchlosťí, ktoré sme obdržali z enkóderov. Predpokladá sa, že toto zašumenie je spôsobené nesprávnym vedením káblov robota, prípadne nepostačujúcim izolovaním ich niektorých častí. Keď sa pozrieme na obrázok 1.1, tak môžeme vidieť nesprávny manažment káblov. Túto teóriu potvrdzuje aj fakt, že spätná väzba rýchlosťi, keď robot stojí neobsahuje rušenie a je zaručene 0. Problémy zo zašumeným signálom riešili už v spomínanom tímovom projekte [1]. Im sa podarilo spraviť graf prechodovej charakteristiky rýchlosťí kolies zobrazený na Obr. 3.2.

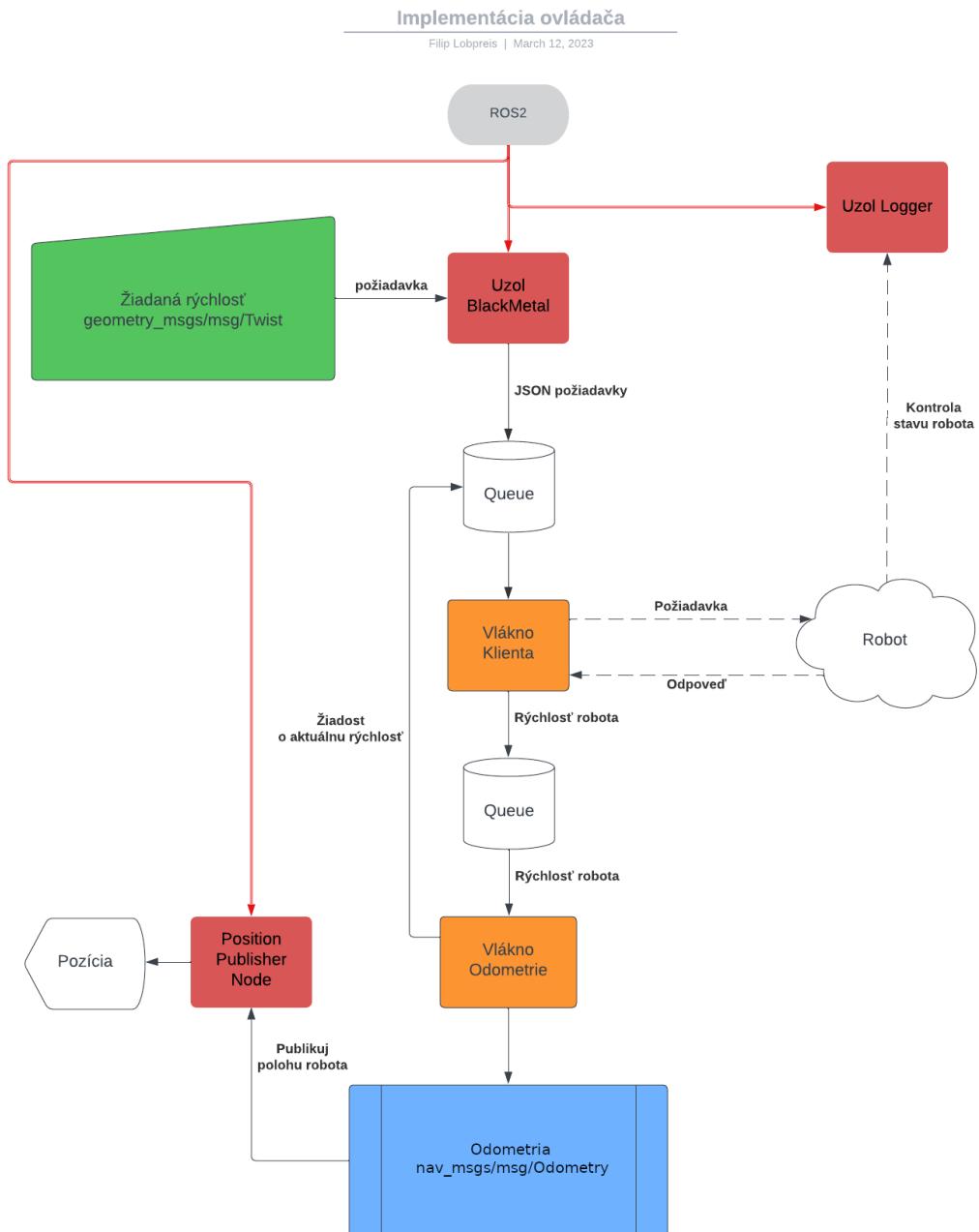


Obr. 3.1: Ustálené hodnoty rýchlosťi ľavého a pravého motora.



Obr. 3.2: Prechodová charakteristika rýchlosťi kolies [1].

4 Implementácia ovládača



Obr. 4.1: Graf vykonávania programu na ovládanie robota pomocou ROS2.

Cieľom kapitoly je navrhnúť a implementovať ovládač v prostredí ROS2, ktorý komunikuje s robotom cez protokol TCP/IP. Na obrázku 4.1 môžeme vidieť viacero objektov rôznych

farieb. Objekty zobrazené červenou farbou sú uzly spracovávané a vytvárané v rámci ROS2. Každý tento objekt sa vykonáva v osobitnom procese. Objekty zobrazené oranžovou farbou sú objekty, ktoré majú svoje vlastné vlákno. Tieto objekty boli vytvorené uzlom BlackMetal. Dátové štruktúry Queue, zobrazené bielou farbou sú vytvorené tak, aby zabezpečovali bezchybnú komunikáciu a synchronizáciu medzi viacerými vláknami. Objekt so zelenou farbou je vstup do programu. Je zadávaný užívateľom a reprezentuje žiadanú rýchlosť robota. Modrý objekt je výstupom programu. Je to téma, na ktorú sa publikuje aktuálna pozícia robota. Robot samotný je zobrazený bielou farbou vo forme malého obláčika. Prerušované čiary na diagrame znázorňujú sietovú komunikáciu ovládača a robota. Červené dvojité čiary udávajú, ktoré objekty patria ROS-u. Nakoniec čierne plné čiary reprezentujú tok dát medzi objektmi.

4.1 Uzly

Na obrázku Obr. 4.1 môžeme vidieť postup vykonávania programu na ovládanie robota BlackMetal pomocou ROS2. Na začiatku programu sa vytvoria 3 uzly.

- **Position Publisher** – uzol, na ktorý sa publikuje vypočítaná pozícia robota. Tento uzol existuje v programe len na overenie publikovaných informácií a ich následné uchovávanie,
- **Logger** – uzol slúžiaci na zaznamenávanie stavu robota 1.2.1,
- **BlackMetal** – uzol ovládajúci robot podľa zadaných dát užívateľom.

4.2 Vstup

Uzol **BlackMetal** vytvorí príjemcu, ktorý počúva na téme **/cmd_vel** a príma správy typu *geometry_msgs/msg/Twist*. Tento vstup je vo forme príkazu zadaného v príkazovom riadku. Vyzerá 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 (-1) na tému */cmd_vel*. Obsahuje lineárne a uhlové rýchlosťi. Z tejto správy sa využijú dva údaje lineárna rýchlosť po osi *x* a uhlová rýchlosť po osi *z*. Ostatné lineárne rýchlosťi neovplyvňujú chod robota rovnako ako ďalšie uhlové rýchlosťi. Táto správa typu *geometry_msgs/mgs/Twist* je následne spracovaná a uložená do rady *Queue*. Táto rada je prioritne založená. To znamená, že požiadavka s nižším kódom ma vyššiu prioritu. Požiadavky a ich kódy môžeme vidieť v sekciách 1.2.2 a 3.4.

4.3 Komunikácia s robotom

Ako je naznačené na Obr. 4.1, klient si vo svojom vlastnom vlákne vytiahne prvú správu z rady a pretransformuje ju do formy JSON. Tento typ správy môžeme vidieť v sekcii 1.2.2. Príkaz je poslaný robotu a ten obratom dá vedieť, či danú požiadavku obdržal. Ak obsah tejto správy žiadal o vrátenie rýchlosťí kolies, tak robot ďalšou správou odpovie na danú požiadavku. Typ tejto odpovede môžeme vidieť v 3.5. V tomto prípade sa správa spracuje a uloží sa do ďalšej rady.

4.4 Odometria

Odometria, počítanie polohy na základe rýchlosťí kolies, sa vykonáva rovnako ako komunikácia s robotom, v separátnom vlákne. Tu je potreba si uvedomiť jednu skutočnosť. A to je tá, že keď posielame žiadosť na nastavenie rýchlosťí kolies robota, tak robot si hodnoty v žiadosti prepočíta. Prepočítané dátia následne poskytne enkóderom. Keď si ale tieto rýchlosťi vyžiadame z enkóderov, tak sa robot týchto dát nechytá a my si ich musíme prepočítať na metre za sekundu. Aby sme dostali rýchlosťi kolies od robota, tak si ich musíme od neho vypýtať. Ako sme už spomenuli v kapitole 4.3, tieto správy majú nižšiu prioritu ako nastavenie rýchlosťí kolies alebo bezpečnostné zastavenie robota. Preto sa môže stať, že správy posielané robotu nebudú dodržiavať presne stanovenú frekvenciu v čase, keď mu bude užívateľ posielat príkazy. Toto nám až tak neprekáža, lebo server na robote sme upravili, tak aby tieto spravy prijímal neustále (kapitola 3.3).

Po obdržaní rýchlosťí robota v impulzoch za sekundu si tieto dátia kvôli zašumeniu preženime cez filter a následne ich spracujeme. Spracovanie prebieha nasledovne:

1. Prepočítanie rýchlosťí kolies robota z impulzov za sekundu na metre za sekundu,
2. Prepočítanie lineárnej a uhlovej rýchlosťi robota na jeho ďažisko,
3. Zistenie aktuálnej zmeny času oproti predchádzajúcemu meraniu,
4. Zapísanie aktuálneho času, lineárnej a uhlovej rýchlosťi do spravy,

5. Prepočítanie aktuálnej polohy z nameraných rýchlosťí.

Za cieľom počítania odometrie sme si odmerali polomer kolies a vzdialenosť stredov týchto kolies. *Polomer kolies (R)* sme určili na **0,08m**. *Polomer stredov kolies L* robota sme určili na **0,56m**. Z dokumentácie robota [5] sme si zistili *rozsah enkóderov (E)* (**1024** impulzov na otočku) Tieto údaje sme použili na prepočítanie impulzov za sekundu na metre za sekundu.

$$V_{mps} = \frac{2\pi R}{E} v_{imp} \quad (4.1)$$

Týmto spôsobom si vieme prepočítať rýchlosť pre pravé a ľavé koleso robota. V druhom bode sme si prepočítali lineárnu a uhlovú rýchlosť pomocou rýchlosťi pravého a ľavého kolesa.

$$V_{mps} = \frac{2\pi R}{E} v_{imp} \quad (4.2)$$

$$\omega = \frac{v_r + v_l}{L} \quad (4.3)$$

Po prepočítaní rýchlosťí kolies sme si zistili aktuálne natočenie robota

$$\gamma = \gamma + \omega dt \quad (4.4)$$

Následne sme dopočítali polohu robota v Karteziánskej súradnicovej sústave pomocou vztahov:

$$X = v \cos(\gamma) dt \quad (4.5)$$

$$Y = v \sin(\gamma) dt \quad (4.6)$$

4.5 Zdieľanie polohy

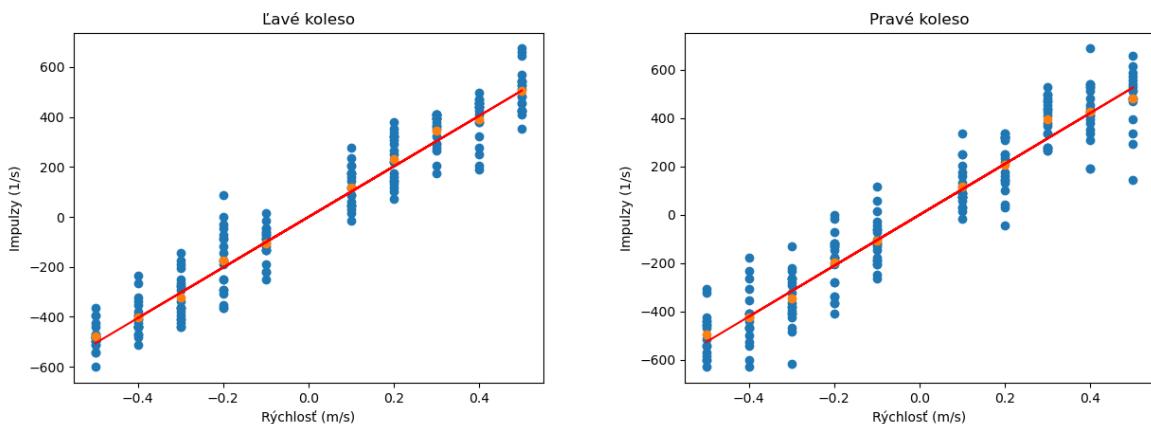
Ďalšiu vec, ktorú je treba vysvetliť ku grafu Obr. 4.1 je zdieľanie polohy. Odometria po každom prepočítaní polohy robota publikuje túto informáciu na tému */odom* táto správa je typu *geometry_msgs/msg/Odometry*. Správa obsahuje štyri hlavné časti:

- **hlavička** Hlavička obsahuje čas a identifikačný reťazec odosielajúceho rámca
- **ID dcérskeho rámca** Identifikačný reťazec prímačujúceho rámca
- **rýchlosť** Lineárne a uhlové rýchlosťi
- **poloha** Polohu v Karteziánskej súradnicovom systéme so súradnicami *x*, *y* a otočením vo forme *kavaterniónu*.

5 Experimenty

Ako bolo spomenuté v predchádzajúcej kapitole, rýchlosť kolies sa dajú získať z enkóderov. Tieto dáta sa posielajú v správe, ktorá pripomína JSON formát. Z týchto vzoriek poslaných robotom nevieme priamo vypočítať polohu. Musíme si tieto dáta premeniť z impulzov za sekundu $\frac{1}{s}$ na metre za sekundu $\frac{m}{s}$. Tento prevod nebude jednoznačný, pretože každý z enkóderov posielá dáta inak zašumené. Preto je potrebné zistiť, ako sa zmení rýchlosť pri zmene impulzov za sekundu. Tento prevod (sklon preloženej lineárnej regresie cez namerané data) je možné získať z merania, kde po nastavení rýchlosťí zoberieme veľa dát z enkóderov a zistíme, ako sa zmení rýchlosť pri zmene impulzov za sekundu.

Prvá úvaha na získanie čo najlepšej prevodovej charakteristiky bolo cez všetky dáta položiť lineárnu regresiu. To sa ukázalo ako zlé riešenie, lebo dáta, ktoré dostávame majú veľmi veľký rozptyl. Jednou z nasledujúcich úvah bolo spraviť kľavý 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 Obr. 5.1. V tomto bode sme vyskúšali počítať odometriu z obdržaných dát. Táto implementácia bola veľmi nepresná. Zároveň nám tento pokus potvrdil, že potrebujeme filtrovať dátu, ktoré dostávame od robota, a ktoré reprezentujú jeho rýchlosť v impulzoch. 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.

Ako prvú vec sme si vykreslili všetky **nazbierané dátu**. Tie sú zobrazené **modrou** farbou. Cez ne sme spravili **lineárnu regresiu**. Je zobrazená ako **červená** úsečka. Z nazbieraných dát sme si nakoniec spravili aritmetický priemer, aby sme videli, ako presne aproximuje nami vypočítaná lineárna regresia priemer nazbieraných dát. **Priemery** jednotlivých rýchlosťí sú zobrazené ako **oranžové** body. Ked' sme cez tieto dáta polozili priamku lineárnej regresie dostali

sme sklon prevodu ľavého enkóderu. Jeho hodnota je **1012,54**. Pre pravý enkóder to je hodnota **1053,68**. Tento parameter môžeme použiť v predpočítavaní nastavenej rýchlosťi robota.

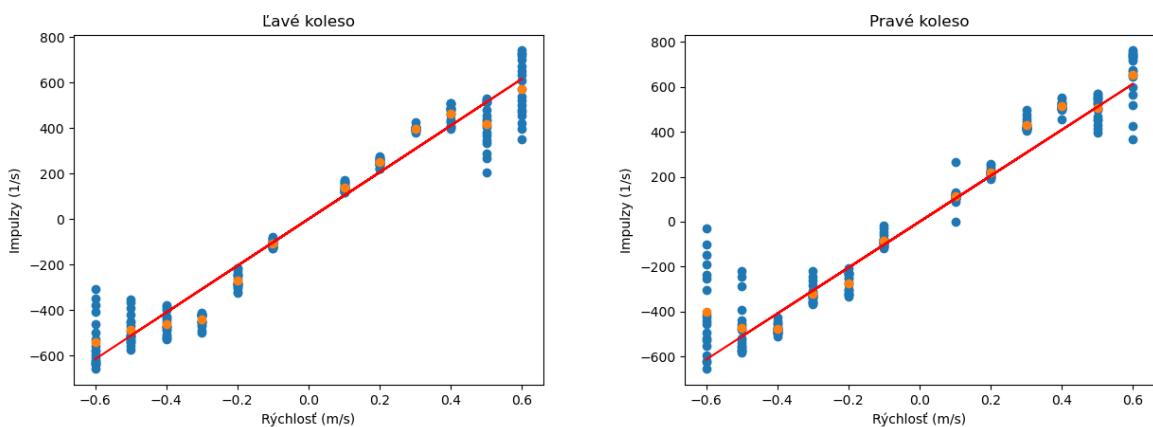
Môžeme si všimnúť, že vypočítané priemery takmer presne ležia na vypočítanej lineárnej aproximácii. Problémom je, ako už bolo spomenuté, že dátu, ktoré dostávame od robota sú veľmi zašumené. Preto z nich nevieme priamo počítať polohu robota. Na zobrazenie veľkosti odchýlky sme spravili meranie. Nechali sme robot aby prešiel dráhu štvorca so stranou dlhou 1 meter a rýchlosťou $0,5 \frac{m}{s}$. Výsledok bol veľmi nepresný. Metrom sme si odmerali jeho x -ovú a y -ovú súradnicu s počiatkom v bode, kde sme na robote spustili náš ovládač. Jeho skutočná poloha bola v bode (-0,3m, 0m). Čo nám ale vypočítala odometria je, že sa robot nachádzal 4 metre od počiatku súradnicového systému.

5.1 Filtrovanie zašumeného signálu

Implementovali sme si preto dolnopriepustný kvadratický filter. Fungovanie tohto filtra spočíva v skombinovaní nového **vstupného parametra δ** a starého parametra π_{n-1} uloženého vo filtro v danom pomere. Tento pomer je daný parametrom **alpha α** . Výsledok tohto výpočtu je **nová hodnota filtra π_n**

$$\pi_n = \alpha * \pi_{n-1} + (1 - \alpha) * \delta \quad (5.1)$$

Jeho parameter α sme získali viacerými meraniami. Začali sme s vysokou hodnotou filtra, α rovnou 0,9.



Obr. 5.2: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,9$.

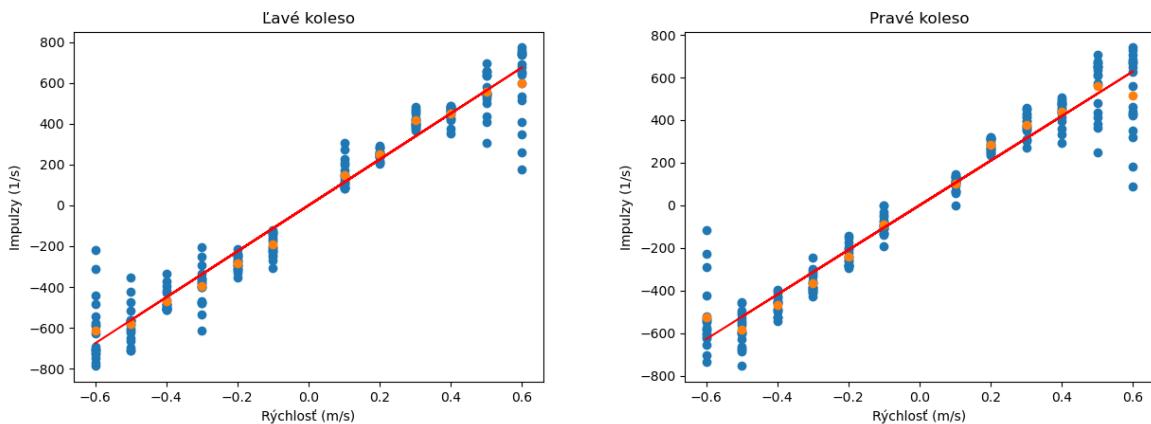
Ako môžeme vidieť na obrázkoch Obr. 5.2 a Obr. 5.1 aplikácia filtrovania výrazne pomohla proti šumu signálu. Problémom pri silnom filtrovании je to, že ak na začiatku merania dostaneme zlú hodnotu, tak sa táto hodnota t'ažko mení na správnu. Tento efekt si môžeme všimnúť

Tabuľka 5.1: Porovnanie výsledkov experimentov prevodu impulzov za sekundu $\frac{1}{s}$ na metre za sekundu $\frac{m}{s}$

α	Frekvencia (Hz)	L'avý enkóder	Pravý enkóder
0,9	250	1028,04	1021,47
0,7	250	1126,16	1028,62
0,75	250	1145,83	1060,25
0,8	250	1028,81	1070,02
0,8	100	1198,86	1212,52

skoro pri každej meranej rýchlosťi. Najviditeľnejší dopad môžeme vidieť pri pravom kolese na Obr. 5.2 pri rýchlosti $-0,6 \frac{m}{s}$. Tento problém sme riešili postupným menením parametra filtra. Aby sme predišli veľkému množstvu meraní, tak sme použili metódy binárneho vyhľadávania.

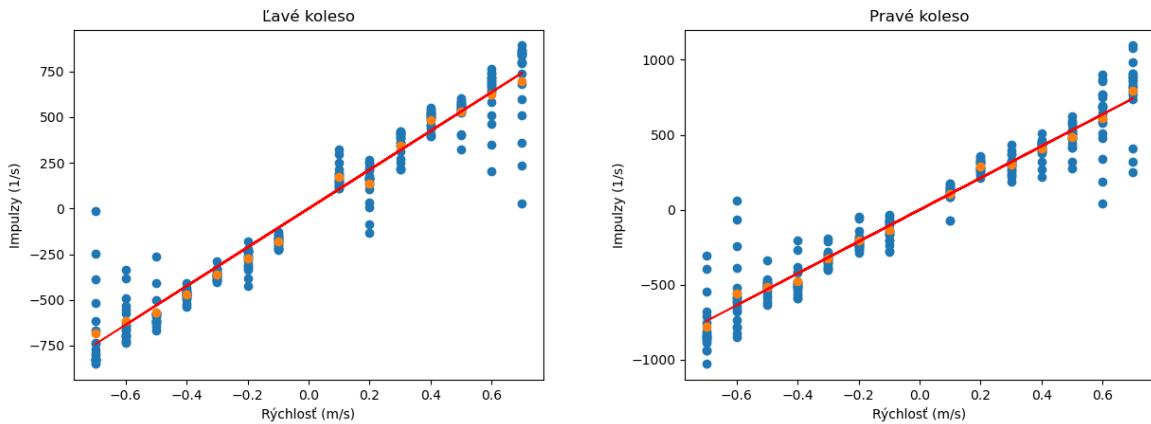
V tomto prípade sme začali s veľkou hodnotou a postupne sme skákali do stredu nášho intervalu. Pre toto meranie nám vyšiel koeficient sklonu lineárnej approximácie pre l'avý enkóder **1028,04** a pre pravý nám vyšiel **1021,47** tento sklon. Kvôli vyššiemu spomenutým dôvodom sme si ako ďalšiu hodnotu koeficientu α sme si zvolili 0,7.



Obr. 5.3: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,7$.

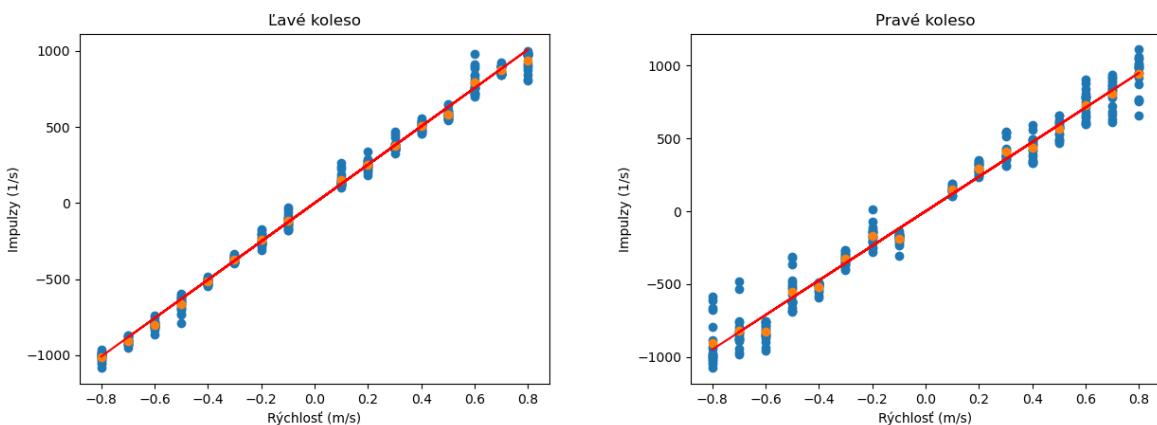
Pri použití koeficientu α s hodnotou 0,7 (Obr. 5.3) sme zistili, že sa hodnoty rýchlosťi aj pri aplikácii filtra výrazne menili. Ustálené hodnoty zobrazené oranžovou farbou sú podobne ako pri meraní s filtrom s alphou α rovnou 0,9 mimo lineárnej approximácie. Tá má sklon pre l'avý enkóder **1126,16** a pre pravý enkóder to je **1048,62**. Neprekrytie priemeru a approximácie je zapríčinené iným dôvodom ako pri silnejšom filtrovi. Pokým pri silnejšom filtrovi sme

dostali zlú počiatočnú hodnotu, tak už bolo zložité ju zmeniť. Pri slabšom filtro, ak dostávame rozdielne vstupné hodnoty tak sa výstupná hodnota filtra ľahko mení. To má za dôsledok posun priemeru vstupných hodnôt. Tento efekt sa dá odstrániť zosilnením filtra, čiže zväčšením koeficientu alpha α . Spravili sme preto ďalšie meranie, pri ktorom sme použili hodnotu koeficientu α rovnú 0,75.



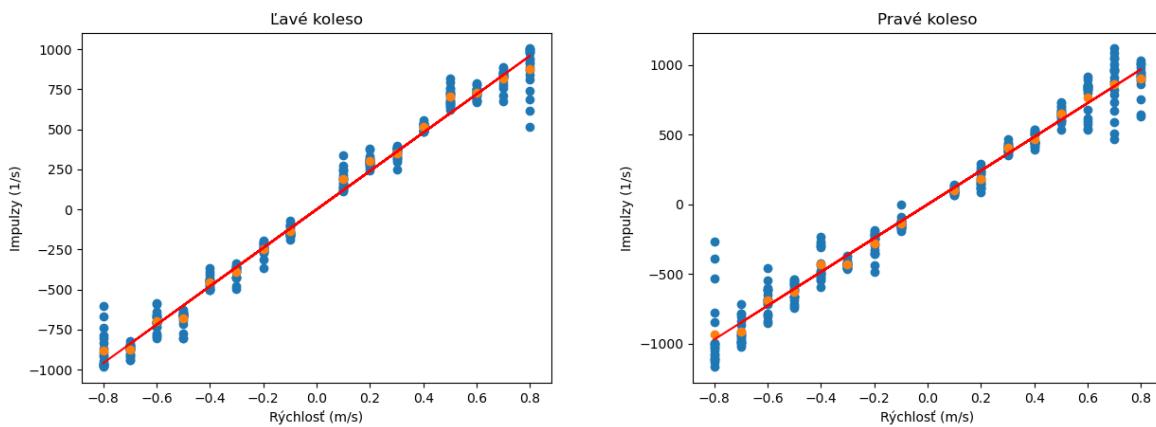
Obr. 5.4: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,75$.

Obrázok 5.4 zobrazuje graf výsledkov nameraných pri aplikácii filtra s koeficientom alpha α o veľkosti 0,75. V tomto prípade sa priemerné hodnoty na rozdiel od filtrov s koeficientami alpha 0,9 a 0,7 dostali takmer priamo na úsečku lineárnej regresie prevodu z impulzov za sekundu na metre za sekundu. Pričom sklon approximácie sa zmenil pre ľavý enkóder na hodnotu **1145,83** a pre pravý enkóder je hodnota sklonu approximácie **1060,25**. Použitie silnejšieho filtra sice pomohlo rýchlejšiemu ustáleniu hodnoty, ale pre istotu sme skúsili ešte silnejší filter s hodnotou alpha α rovnou 0,8.



Obr. 5.5: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,8$.

Ako môžeme vidieť na obrázku 5.5 ustálenie hodnôt je veľmi jednoznačné. Vybrali sme si preto filter s hodnotou koeficientu alpha α rovnou 0,8. Zatiaľ všetky dátá čo sme merali boli s frekvenciou 4Hz (1 vzorka za 250 milisekúnd). Pre presnejší výsledok sme túto frekvenciu ešte zvýšili. Z testov robota sme vyzorovali, že najfrekventovanejšia frekvencia, ktorú môže robot sprostredkovat' je 10Hz (1 vzorka za 100 milisekúnd). Spravili sme si preto test na prevod rýchlosťi ešte raz s rovnakou hodnotou koeficientu alpha α rovnou 0,8, ale s frekvenciou 10Hz namiesto už spomenutých 4Hz.

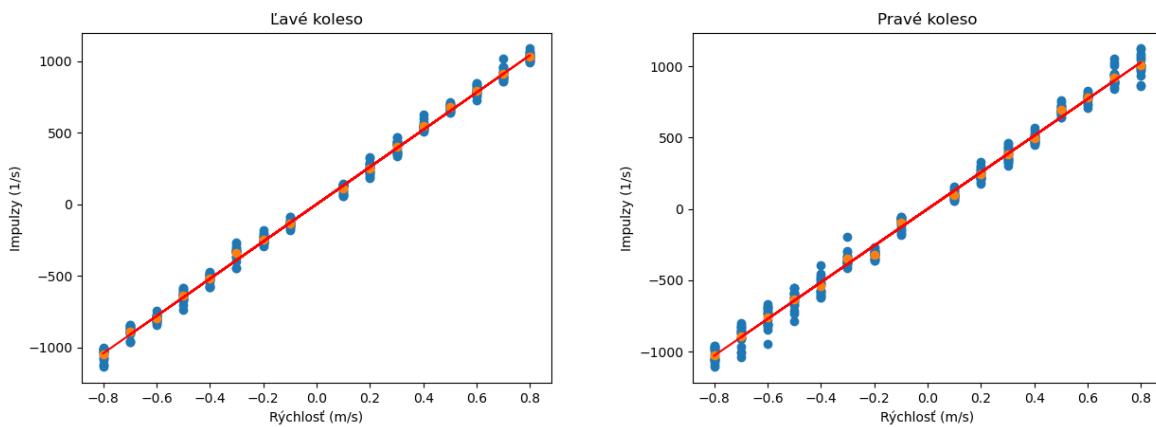


Obr. 5.6: Získanie prevodu z impulzov za sekundu na rýchlosť v SI jednotkách. $\alpha = 0,8$ a frekvenciou 10Hz.

Výsledok spomínaného merania vidíme na obrázku 5.6. Nie všetky získane hodnoty impulzov z robota sa nachádzajú pri úsečke lineárnej aproximácie. Aj napriek tomuto faktu sa väčšina priemerov získaných dát nachádza na tejto úsečke. Hodnota sklonu lineárnej aproximácie pre pravý enkóder je **1028,81** a pre pravý enkóder má aproximácia hodnotu **1070,02**. Disperzia získaných dát je chyba, ktorú odstránime ako ďalšiu.

Koeficient alpha α s najlepšími parametrami nám vyšiel s hodnotou **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 by spôsobila 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.

Toto riešenie má jeden problém. Ak prvá prijatá hodnota po zmene rýchlosťi nie je v blízkom okolí žiadanej hodnoty, tak pôsobením filtra bude trvať dlho pokým sa hodnoty vypočítané filtrom dostanú na správnu hodnotu. Tento problém vieme vyriešiť predpočítavaním hodnôt za pomoci vypočítaných lineárnych aproximácií. Problém so zlou počiatočnou hodnotou môžeme vidieť aj na posledom grafe 5.6. Tento problém sme vyriešili predpočítavaním prvej hodnoty filtra po zmene rýchlosťi. Ked'že sme už mali koeficienty lineárnej regresie, tak sme ich využili na predpočítanie počiatočnej hodnoty.



Obr. 5.7: Získanie prevodu z impulzov na rýchlosť v SI jednotkách. $\alpha = 0,8$ a frekvenciou 10Hz a prvou prepočítanou hodnotou.

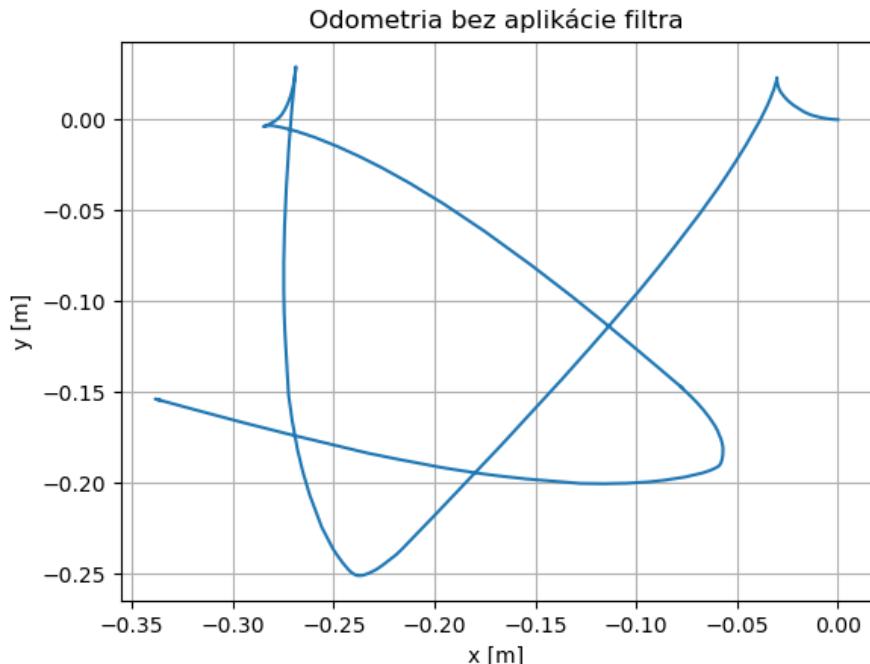
Meranie so zmenou predpočítavania prvej hodnoty sme zopakovali viackrát, aby sa ustálili hodnoty aproximácie. Po opakovanom experimente sa nám hodnoty sklonov lineárnych aproximácií ustálili na hodnotách:

- Ľavý enkóder: **1198,86**
- Pravý enkóder: **1212,52**

Výsledok tohto postupu vidíme na nasledujúcom grafe Obr. 5.7.

5.2 Overenie meraní

Výsledky posledných meraní filtrov sme si vyskúšali na znázornení prejdenej dráhy v tvare štvorca a následne sme si zobrazili na grafe túto trasu. Pre porovnanie účinnosti filtra sme porovnali tri grafy. Prvý graf zobrazuje prejdenú dráhu počas toho ako je filter vypnutý. Druhý graf znázorňuje prejdenú dráhu s aktívnym filtrom s koeficientom α rovným 0,8. Tretí graf reprezentuje dráhu prejdenú so slabším filtrom s hodnotou koeficientu α 0,7.

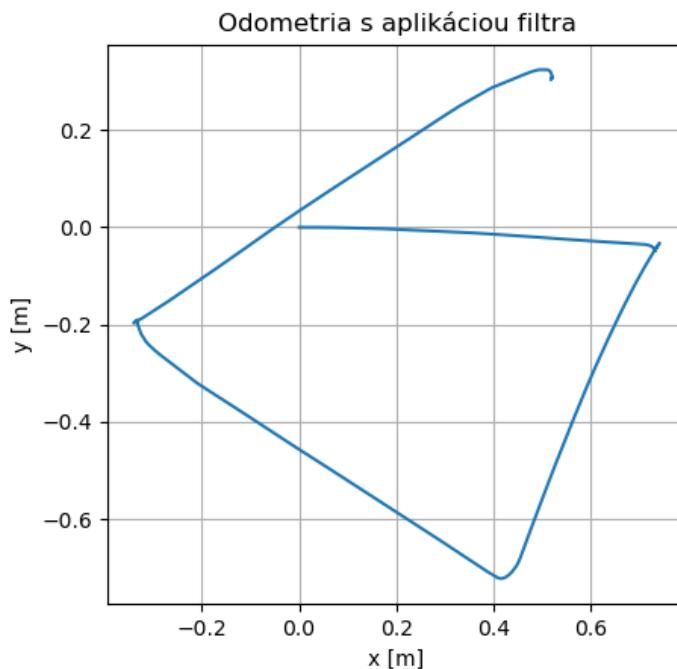


Obr. 5.8: Zobrazenie pohybu robota z odometrie bez použitia filtra.

Ako prvý pokus sme vypli filtrovanie prichádzajúcich dát. To malo za dôsledok, že všetky skoky, ktoré môžeme vidieť napríklad na obrázku 5.1 sa ukázali na grafe. Obrázok 5.8 reprezentuje dráhu, ktorú robot prešiel bez aplikácie filtra. Na grafe môžeme vidieť zašumenie nazbieraných dát z robota. To sa zobrazuje najmä v častiach, keď má robot ist' rovno. V prípade, že filter nepožívame, tak rovné čiary sa premenia na krivky. To spôsobuje nepresné uvedenie aktuálnej polohy. Tieto nepresnosti sa prejavujú aj pri otáčaní robota. V prípade, že sa robot má otočiť o 90 stupňov, respektíve o $\frac{\pi}{2}$ radiánu, tak robot toto otočenie reprezentuje iným uhlom, poprípade aj posunutím.

Aby sme videli ako ovplyvňuje filter s nami navrhnutými parametrami, tak sme spravili ďalšie meranie s filtrom s najlepšími hodnotami. Druhý pokus sme zvolili najlepšiu hodnotu filtra s koeficientom α rovným 0,8. Obrázok 5.9 reprezentuje dráhu, ktorú robot prešiel, keď

sme na prichádzajúce dátu aplikovali tento filter.

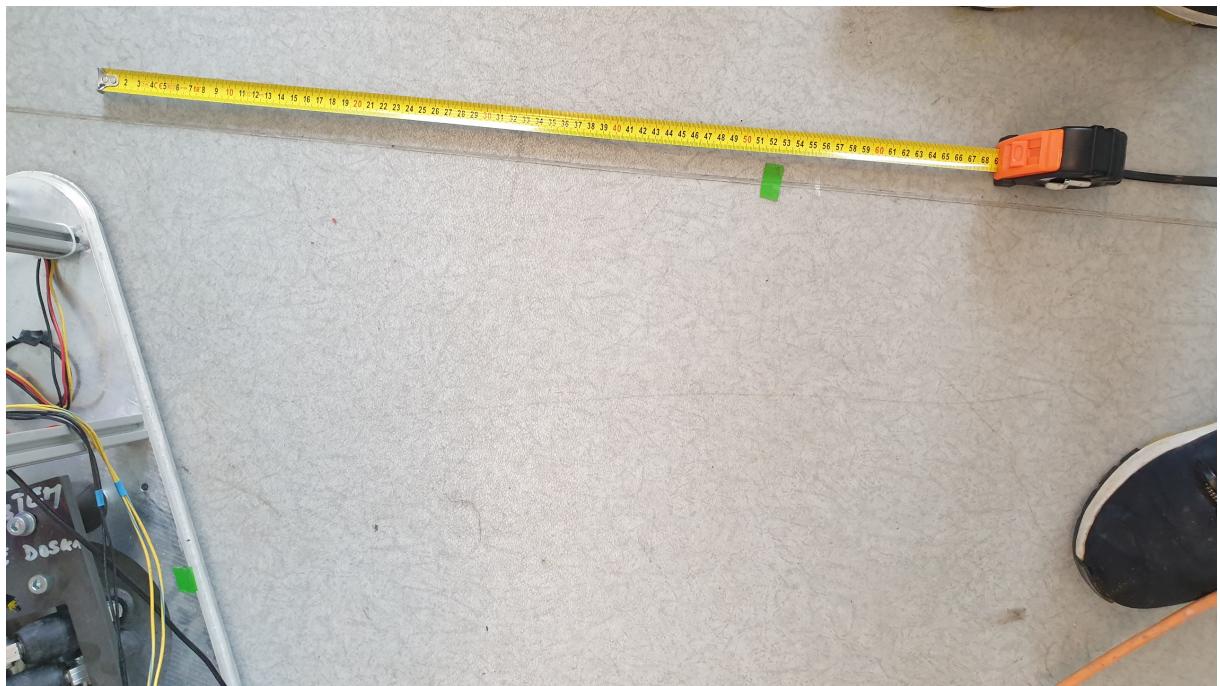


Obr. 5.9: Zobrazenie pohybu robota z odometrie s použitím najlepšieho filtra.

Ako môžeme vidieť na tomto grafe tak síce filter zlepšil výsledok, ktorý sme dostali bez filtra, ale stále tvar, ktorý je znázornený na grafe nie je štvorec. Môže to byť spôsobené viacerými faktormi.

- **Prešmyk** Robot prešmykuje pri zrýchľovaní. To má za dôsledok nepresné uvedenie polohy robota.
- **Oneskorenie komunikácie** Dáta z ovládača prijíname vo while cykle so softvérovým oneskorením (100ms) plus [1]. Tým, že sa komunikácia oneskoruje, tak sa vnáša chyba do výpočtu aktuálnej polohy na základe rýchlosťí robota.

Pre porovnanie odmeraných výsledkov odometrie a reálneho prejdenia dráhy robotom sme odmerali vzdialenosť robota od začiatočnej po konečnú pozíciu robota. Obrázky zobrazujúce toto meranie môžeme vidieť na Obr. 5.10 a Obr. 5.11. Odometria odmerala koniec trasy na bod **[0,52; 0,30]**, kde prvá hodnota zobrazuje súradnicu na osi X a druhá hodnota zobrazuje súradnicu na osi Y. Obe hodnoty boli merané v metroch. Odmerané hodnoty metrom sme vyčíslili na **[0,52; 0,34]**.

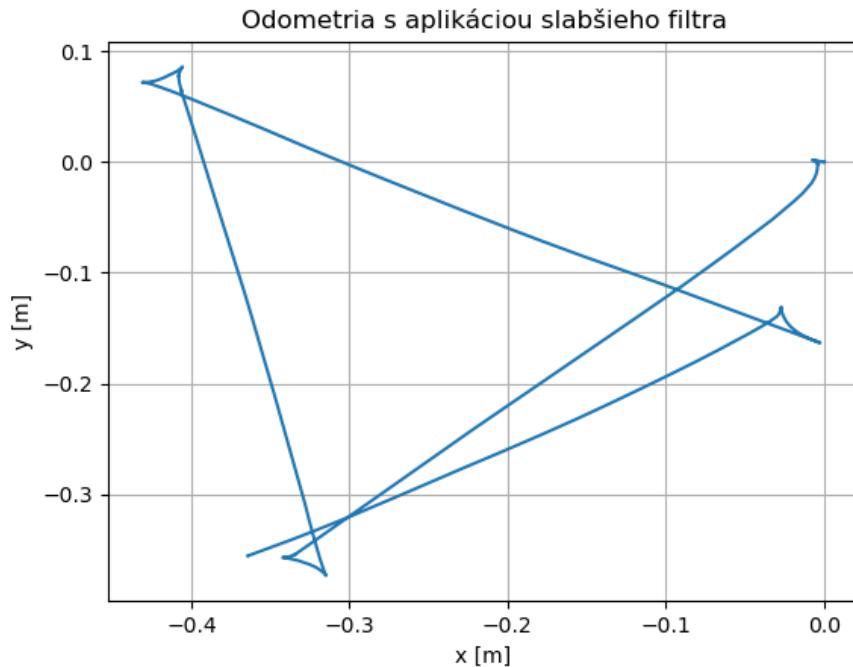


Obr. 5.10: Odmeranie koncovej pozície robota na osi X po prejdení štvorcovej dráhy.



Obr. 5.11: Odmeranie koncovej pozície robota na osi Y po prejdení štvorcovej dráhy.

Tretie a posledné meranie, ktoré sme spravili bolo meranie so slabším filtrom ako sme sa rozhodli použiť. Obsahovalo koeficient α s hodnotou 0.7. Pri takejto konfigurácii filtra sme dostali graf zobrazený na obrázku 5.12. Toto zhoršenie spôsobilo slabšie filtrovanie prichádzajúcich dát a tým pádom aj zhoršenie presnosti určovania polohy robota.



Obr. 5.12: Zobrazenie pohybu robota z odometrie s použitím slabšieho filtra.

Ak by sme použili silnejší filter dostali by sme lepsi výsledok. V tom prípade by sme, ale nemali relevantný výsledok vzhl'adom na robot. Filtrované dáta by boli až veľmi skreslené žiadoucou hodnotou.

Experimentmi sme sa dostali k najvhodnejšiemu parametru α , ktorý sme využili ako vstup do hodno pripustného filtra. Jeho implementáciu sme navrhli tak, aby sme vedeli zistiť čo najpresnejšiu hodnotu aktuálnych rýchlosí robota. Využitie filtra nám pomohlo presnejšie určovať aktuálnu polohu robota.

6 Využitie

Robot, ktorý sme dostali na tento projekt bol pôvodne ovládateľný len pomocou JSON reťazcov. My sme na ňom museli vykonať pár zmien, aby sme vedeli počítať odometriu a obdržali sme malú spätnú väzbu. Pri odometrii sme čelili pár prekážkam, ktoré sa nám podarilo z časti odstrániť. Skončili sme s ovládačom naprogramovaným v jazyku C++ v prostredí ROS2.

Ďalšou otázkou môže byť ako sa tento ovládač môže využiť? Odpoved' na túto otázku nie je jedna, ale rovno viacero. Prostredie ROS2 nám otvára mnoho d'alších možností ako posielat príkazy na robot. Balíky naprogramované v prostredí ROS posielaniu dátu na tému */cmd_vel*. Ich implementácia je už dokončená a my môžeme tieto kódy len jednoducho pripojiť k programu v spúšťacom súbore. Možnosťami sú napríklad:

- **Manuálne ovládanie** Ovládanie pomocou
 - **klávesnice** použitie uzlu *teleop_twist_keyboard*,
 - **joystick** použitie uzlu *teleop_twist_joy*,
 - **príkazov** skripty poprípade priame zadávanie príkazov do konzoly.
 - **hlasu** použitie umelej inteligencie na pretvorenie hlasu na text,
 - **gestami** použitie počítačového videnia na detekciu pohybu užívateľa.
- **Automatické ovládanie** Ovládanie pomocou
 - **algoritmov** použitie algoritmov pre nájdenie najlepšej trasy,

6.1 Joystick

Joystick je ovládač obyčajne používaný pri herných konzolách. My sme ho použili na manuálne ovládanie robota. Ako bolo spomenuté vyššie (Kapitola 6) joystick je možné použiť s uzlom *teleop_twist_joy*. Tento uzol sprostredkuje využitie hardware-u ovládača. Pohyby vykonané na ovládači sa transformujú do typu správy *geometry_msgs/Twist* a publikujú sa priamo na tému *cmd_vel*. Táto téma je priamo pripojená na náš uzol *BlackMetal* (Kapitola 4.1). Prijatú správu nami vytvorený ovládač spracuje a pošle ju robotu. Ovládač je možné spustiť s viacerými typmi ovládačov sú to *atk3*, *ps3-holonomic*, *ps3*, *xbox*, *xd3* [21]. Po pripojení ovládača k nášmu programu a prepojení programu s robotom sme mohli začať s jeho ovládaním.

Pri používaní ovládača nastáva jeden problém a to je, že pri hocjakom pohybe joysticke sa pošle požiadavka na zmenu rýchlosťi robota. To spôsobuje zahltenie odosielacej rady. To má

za dôsledok zlyhanie rady odometrie. Aby sme tomuto problému predišli, spravili sme filter prichádzajúcich správ. Keďže vieme, že robot čaká 10 sekúnd a následne pri žiadnej sprostredkovanej komunikácii zastaví, tak sme nastavili limit na posielanie rovnakých požiadaviek na 9 sekúnd.

Záver

Hlavným cieľom bakalárskej prace bolo vytvorenie ovládača pre mobilného robota v prostredí ROS2. Aby sme mohli začať s programovaním tohto ovládača museli sme opraviť pár chýb, ktoré sa na robte vyskytli. Boli to chyba v systéme (3.1), oneskorenie komunikácie robota (3.3) a nesprávna spätná väzba robota (3.5).

Hlavnú úlohu ovládača tvoril uzol, ktorý zabezpečoval prijímanie a spracovanie dát od užívateľa (4.1). Jeho ďalšou úlohou bolo vytvorenie klienta a odometrie. Klient zaistoval výmenu požiadavok a odpovedí so serverom, ktorý sa nachádzal na robote. Odometria na druhú stranu zabezpečovala spracovanie priyatých sprav od robota a ich následnú publikáciu na tému /odom. Pre validáciu presnosti odometrie sme spravili s robotom test ako presne vypočítanie dráhu štvorca, ktorú prejde robot. Ovládač je možné použiť na ovládanie robota pomocou preddefinovaných uzlov alebo pomocou vlastného uzla, ktorý bude publikovať spravy na tému /cmd_vel. Vytvorený ovládač sme otestovali s uzlom *teleop_twist_joy*.

Pri prijímaní dát z robota sme zistili, že enkódery posielajú zašumený signál. Tento problém sme vyriešili implementáciu kvadratického dolno-priepustného filtra. Dolno-priepustný filter zabezpečil stabilitu dát, z ktorých sme následne počítali aktuálnu polohu robota.

Nami navrhnutý ovládač je len jednou z možností ako navrhnutý tento ovládač. Pre budúce ovládače robotov by som navrhoval použitie druhej verzie robotického operačného systému. Jeho použitie umožňuje jednoduchšie a bezpečnejšie programovanie, pričom zároveň poskytuje možnosť jednoduchého rozširovania ovládača o ďalšie uzly.

Zoznam použitej literatúry

1. BC. MAREK PACALAJ, BC. TOMÁŠ KÚTIK, BC. DOMINIK GULA, BC. DÁVID PAVLIČ, BC. DANIEL ĎURKOVIČ. *Mobilný podstavec pre robota*. 2019.
2. *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).
3. *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).
4. *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.
5. *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>.
6. *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>.
7. *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>.
8. BC. MAREK PACALAJ, BC. TOMÁŠ KÚTIK, BC. DOMINIK GULA, BC. DÁVID PAVLIČ, BC. DANIEL ĎURKOVIČ. *Dokumentacia k softwaru robota BlackMetal*. 2019.
9. *Robot Operating System (ROS) on Windows through Windows Subsystem for Linux (WSL)* [online] [cit. 2023-05-14]. Dostupné z : <https://medium.com/@reflectrobotics/robot-operating-system-onwindows-just-got-better-35ca7a55230e>.
10. *ChatGPT*. Dostupné tiež z: <https://chat.openai.com/chat>.
11. *Linux security* [online] [cit. 2023-05-20]. Dostupné z : <https://www.emertxe.com/embedded/why-linux-is-more-secure-than-other-operating-systems/>.
12. 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.

13. *ROS2 documentation* [online] [cit. 2022-12-23]. Dostupné z : <https://docs.ros.org/en/humble/index.html>.
14. *ROS History* [online] [cit. 2022-12-23]. Dostupné z : <https://www.theconstructsim.com/history-ros/>.
15. *Changes between ROS 1 and ROS 2* [online] [cit. 2023-01-08]. Dostupné z : <http://design.ros2.org/articles/changes.html>.
16. *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>.
17. *ElectronicDesign* [online] [cit. 2023-05-07]. Dostupné z : <https://www.electronicdesign.com/markets/automation/article/21258792/realtme-innovations-rti-ros-and-dds-making-the-most-out-of-your-software-framework>.
18. *Trend* [online] [cit. 2023-05-07]. Dostupné z : <https://www.trendmicro.com/vinfo/us/security/news/cybercrime-and-digital-threats/analyzing-the-data-distribution-service-dds-protocol-for-critical-industries>.
19. *ros2 control* [online] [cit. 2023-05-03]. Dostupné z : https://control.ros.org/master/doc/differences_to_ros1/differences_to_ros1.html.
20. MAXON. *EPOS Command Library: Document ID: rel6806*. 2019.
21. *ros2 teleop twist joy* [online] [cit. 2023-05-01]. Dostupné z : https://github.com/ros2/teleop_twist_joy.

Prílohy

A	Dokumentácia ku kódu	47
B	Kód ovládača	48
C	Videá ovládania robota	49

A Dokumentácia ku kódu

Dokumentácia ku kódu je priložená v samostatnom pdf súbore.

B Kód ovládača

Kód na ovládanie robota je prístupný na stránke
<https://github.com/Fildo7525/Bakalarsky-projekt-kod>, alebo v priloženom *.zip* súbore.

C Videá ovládania robota

Videá ovládania robota sú priložené v osobitnom *.zip* súbore.