TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Mikk Loomets 206110IAIB

ROS2 LIIDESEGA BIOMIMEETILISE ROBOTI TARKVARAPLATVORMI ARENDUS

Bakalaureusetöö

Juhendaja: Jaan Rebane
MSc

Kaasjuhendaja: Gert Kanter
PhD

Tallinn 2025

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Mikk Loomets

01.06.2025

Annotatsioon

Tallinna Tehnikaülikooli Biorobootika keskus kasutab μ-CAT roboteid, et õpetada ROS2 tarkvara. Praegu eksisteerib kaks versiooni robotist. Esimene versioon on disainitud sardsüsteemina ning ei paku ROS2 liidest. Teine versioon robotist lisas uusi andureid ning Raspberry Pi Zero 2W arvuti, aga selle platvormi versiooni jaoks ei ole olemas toimivat koodi.

Kuna viimane roboti platvormi versioon ei ole suuteline suhtlema läbi ROS2 liidese, toimub enamik õppetöödest, kasutades roboti simulaatorit Gazebo.

Käesoleva bakalaureusetöö eesmärgiks oli võimaldada teise μ-CAT roboti versiooni juhtimine üle ROS2 liidese. See on oluline selleks, et robotiplatvormi kasutaja ei peaks teadma süsteemi riistvara iseärasustest.

Selle jaoks arendati süsteemile tarkvara, mis tegeles riistvaraga suhtlemisega ning pakkus ROS2 liidest kasutaja koodile. Lisaks sellele pakkus antud arendus funktsionaalsust andmete saamiseks täiendavatelt anduritelt, mis lisandusid teise μ-CATi riistvara versiooniga.

Loodud tarkvara oli suuteline juhtima kõiki riistvarakomponente ning pakkus kasutajale kõik vajaliku informatsiooni riistvarakomponentidelt. Loodud süsteemi probleemiks oli latentsus, mis tekkis käskude edastamisel kasutajalt platvormile.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 23 leheküljel, 6 peatükki, 5 joonist, 1 tabelit.

Abstract

Development of a Software Platform for a Biomimetic Robot with a ROS2 Interface

Tallinn University of Technology's Center of Biorobotics uses μ-CAT robots to teach the use of ROS2 software. Currently, there exist two versions of the robot. The first version of the robot is designed as an embedded system and doesn't provide a ROS2 interface. The second version of the robot adds new sensors and a Raspberry Pi Zero 2W computer, but this version of the platform doesn't have functional software.

Because the last version of the robot was unable to communicate using the ROS2 bridge, most of the studies utilize the robot simulator Gazebo.

The goal of this bachelor's thesis was to enable control of the second version of the μ-CAT robot via a ROS2 interface. This is important so that the user of the robot platform does not need to be aware of the specific details of the system's hardware.

To achieve this, software was developed for the system that handled communication with the hardware and provided a ROS2 interface to the user's code. In addition, the development included functionality for retrieving data from additional sensors that were introduced with the second hardware version of the μ-CAT.

The developed software was capable of controlling all hardware components and provided the user with all necessary information from the hardware. One issue with the created system was latency occurring when commands were transmitted from the user to the platform.

The thesis is written in Estonian and is 23 pages long, including 6 chapters, 5 figures and 1 table.

Lühendite ja mõistete sõnastik

ADC	Analoog-digitaalmuundur (*Analog-to-Digital Converter*)
COBS	Ühtlase üldkulu baittäitmine (*Consistent Overhead Byte Stuffing*)
CPU	Keskprotsessor (*Central Processing Unit*)
CRC	Tsükliline lisakood (*Cyclic Redundancy Check*)
CSI	Kaamera jadaedastusliides (*Camera Serial Interface*)
DDS	Andmejaotusteenus (*Data Distribution Service*)
EEPROM	Elektriliselt ümberprogrammeeritav püsimälu (*Electrically Erasable Programmable Read-Only Memory*)
FPU	Ujukomaplokk (*Floating-Point Unit*)
GPIO	Üldotstarbeline sisend/väljund(*General Purpose Input/Output*)
I2C	*Inter-Integrated Circuit*
IDE	Integreeritud programmeerimiskeskkond (*Integrated Development Environment*)
IMU	Inertsiaalandur (*Inertial measurement unit*)
LED	Valgusdiood (*Light Emitting Diode*)
PWM	Pulsilaiusmodulatsioon (*Pulse-Width-Modulations*)
QoS	Teenuse kvaliteet (*Quality of Service*)
ROS2	Robotite Operatsioonisüsteem 2 (*Robot Operating System 2*)
RPI	Raspberry Pi Zero 2W
RTOS	Reaalaja-operatsioonisüsteem (*Real-Time Operating System*)
SNTP	Lihtne võrguaja protokoll (*Simple Network Time Protocol*)
SPI	Järjestikune perifeerne liides (*Serial Peripheral Interface*)
SRAM	Staatiline muutmälu (*Static Random Access Memory*)
UART	Universaalne asünkroontransiiver (*Universal Asynchronous Receiver/Transmitter*)

Sisukord

1 Sissejuhatus **9**

2 Taustainfo **10**

2.1 M-CAT versioon 1 10

2.1.1 ATmega328P 10

2.1.2 Inertsiaalandur 11

2.1.3 Mootori kontrollerid 12

2.1.4 Mootori koodrid 12

2.1.5 Rõhuandur 13

2.1.6 Tooniandurid 13

2.2 M-CAT versioon 2 13

2.2.1 Kaamera 14

2.2.2 Raspberry Pi Zero 2W 14

2.2.3 ICM-20608-G 15

2.2.4 MS5837-02BA 15

2.3 Robotite Operatsioonisüsteem 15

2.3.1 ROS1 ja ROS2 võrdlus 15

3 Kasutatud tehnoloogiad **16**

3.1 Programmeerimiskeel 16

3.1.1 RPI koodibaas 16

3.1.2 ATmega328P koodibaas 19

3.2 Sisemine kommunikatsioon 20

3.2.1 Andmepaketid 20

3.2.2 Andmete struktuur ja sõnumid 21

3.3 Väline kommunikatsioon 21

3.3.1 ROS2 distributsioonid 22

4 Lahendus **24**

4.1 Arendus 24

4.1.1 Esimene versioon 24

4.1.2 Teine versioon 25

4.2 Loodud lahendus 25

4.2.1 RPI kood 25

4.2.2 ATmega328P kood 27

5 Test ja katsed **29**
5.1 Latentsuse test 29
5.2 Töökindluse katse 30
5.3 Funktsionaalsuse katse 31

6 Kokkuvõte **32**

Kasutatud kirjandus **33**

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks **35**

Lisa 2 – ATmega328P tsükkel **36**

Lisa 3 – Rust jadaedastuse koodi näide **37**

Jooniste loetelu

Joonis 1. M-CAT versioon 1 10
Joonis 2. M-CAT versioon 2 14

Joonis 3. RPI tarkvara 26
Joonis 4. Andmete jadastamine ja struktureerimine 27
Joonis 5. ATmega328P tarkvara 27

Tabelite loetelu

Tabel 1. *Lõik RPI ajatulemuste mõõtmete tabelist* 30

1. Sissejuhatus

M-CAT (CAT - *Curious Archaeology Turtle*) on robot, mida arendati Tallinna Tehnikaülikooli Biorobootika keskuses. Robotist eksisteerib kaks versiooni. Esimest versiooni juhtis mikrokontroller Atmega328p. Teist versiooni juhtis sama mikrokontroller Atmega328p ning arvuti Raspberry Pi Zero 2W. Teise versiooni puhul oli robotile lisatud kaamera ja tooniandurid.[1]

Kuna teise versiooni jaoks ei olnud olemas tarkvara, mis lihtsustaks süsteemi juhtimist. Seetõttu oleks pidanud iga projekt või tudeng, kes soovis seda roboti platvormi kasutada, arendama uuesti lahenduse riistvara juhtimiseks. Selle lahenduse arendamine oleks olnud ajakulukas ning nõuaks head arusaamist roboti riistvara kohta. Selle asemel arendati selle lõputöö käigus valmis platvormi tarkvara robotile, et lihtsustada roboti kasutust.

Töö käigus arendati välja kaks tarkvarakomponenti: esimene Atmega328P jaoks, mis haldas andurite ja mootoritega suhtlemist, ning teine Raspberry Pi Zero 2W (edaspidi RPI) jaoks, mis vahendas suhtlust kasutaja koodiga ja edastas kaamera pilti.

Tulemuste valideerimiseks sooritati kaks katset ja üks test:

1. Töökindluse katse, mille käigus hoiti süsteemi üleval 12 tundi ning veenduti, et ei teki ühtegi süsteemikriitilist viga.
2. Funktsionaalsuse katse, mille käigus loodi kasutaja kood, mis lahendab ülesande ainest IAS0060. See verifitseeris, et loodud platvormi oli võimalik kasutada ka teiste aine ülesannete lahendamiseks.
3. Latentsuse test, mille käigus mõõdeti aeg, mis kulub informatsioonil ROS2 (*Robot Operating System 2*) liidesest kuni riistvarani jõudmiseks.

See lõputöö koosneb järgnevatest osadest:

- Esmalt uuritakse platvormi arvuteid ja riistvara, mille käigus tutvutakse nende võimekuste ning nõrkustega.
- Seejärel tutvustatakse nõudeid platvormile ja tehakse ülevaade projekti arhitektuurist ning valitud tehnoloogiatest.
- Järgmiseks räägitakse loodud tarkvara lahendustest
- Viimases osas räägitakse testidest, mis viidi läbi, ja nende tulemustest.

2. Taustainfo

μ-CAT on odav, väikeste mõõtmetega, bioloogiast inspireeritud allveerobot. Liikumiseks kasutab μ-CAT 4 individuaalselt juhitavat pehmet uime. Praegu on Biorobootika keskuses 3 sellist robotit, mida kasutatakse uurimistööde testplatvormidena, demonstratsioonirobotitena üritustel ja õpetamiseks erinevatel robootika kursustel.

2.1 M-CAT versioon 1

Esimene versioon μ-CAT robotist on ehitatud Arduino-UNO kontrolleriga kasutamiseks. Roboti anduriteks on IMU (inertsiaalandur), mootorite koodrid, rõhuandur ja tooniandurid. Vaata joonis 1.

Joonis 1. M-CAT versioon 1

Kuna esimese versiooni μ-CAT robotite peal on ainult Arduino-UNO mikrokontroller, mis ei ole suuteline jooksutama operatsioonisüsteemi, ei ole võimalik lisada sellele ROS2 liidest.

Tarkvara arendus selle versiooni peal tähendas madala taseme koodi kirjutamist C või C++ keeles. Kuna Arduino-UNO kasutab ATmega328P protsessorit, on süsteemi võimekus piiratud.[2] Tarkvara, mis selle roboti jaoks oli arendatud, pidi olema kiire ning pidi vältima ujukoma arve. Kuna antud roboti protsessor on piiratud võimekusega ning tal puudub operatsioonisüsteem, siis muudab see tarkvara loomise keerulisemaks.

2.1.1 ATmega328P

ATmega328P on 8-bitine mikrokontroller, mis kuulub Atmeli (nüüd Microchip Technology) AVR-perekonda ning on laialdaselt tuntud tänu oma kasutamisele näiteks Arduino UNO

platvormil. See sisaldab 32KB välkmälu, 2KB SRAM-i (staatiline muutmälu) ja 1KB EEPROM-i (elektriliselt ümberprogrammeeritav püsimälu), pakkudes piisavalt ruumi väiksemateks ja keskmise keerukusega rakendusteks. Mikrokontroller töötab maksimaalselt 20 MHz sagedusel ning toetab kuni 23 sisendi-/väljundliidest, mis võimaldavad ühendada erinevaid andureid, mootoreid ja muid komponente. Lisaks sisaldab see kolme taimerit, 6-kanalilist 10-bitist ADC-d (analoog-digitaalmuundur) ja mitmeid suhtlusliideseid nagu SPI (järjestikune perifeerne liides), I2C (Inter-Integrated Circuit) ja UART (universaalne asünkroontransiiver), mis muudavad selle sobivaks nii hobi- kui ka professionaalseteks projektideks. ATmega328P töötab tavaliselt 5V loogikapingega, mis tähendab, et selle sisendid ja väljundid loevad ja annavad välja kõrgeid loogikataseme signaale 5-voldise pingega. See on oluline erinevus tavapärasest 3.3V loogikapingest, mida kasutavad paljud tänapäevased andurid ja mikrokontrollerid. [3]

Kuigi ATmega328P on paljudele harrastajatele ja inseneridele tuttav ning usaldusväärne valik, on sellel siiski mitmeid märkimisväärseid puudujääke, mis võivad piirata selle sobivust keerukamateks rakendusteks. Üks silmapaistvamaid kitsaskohti on riistvaralise ujukomaplokki (FPU) puudumine. See tähendab, et kõik ujukomaarvulised tehted tuleb teha tarkvaraliselt, mis on protsessori jaoks tsüklite kulukas ja ebaefektiivne võrreldes FPU-d omavate kiipidega. See võib tekitada kitsaskohti näiteks anduriandmete töötlemisel või signaalitöötluses. Samuti on ATmega328P-l piiratud arv PWM-kanaleid (pulsilaiusmodulatsiooni kanaleid) ja puudub sisseehitatud USB-tugi, mis muudab selle vähem sobilikuks tänapäevasteks kommunikatsiooninõueteks ilma täiendavate kiipide kasutamiseta. Need piirangud ei pruugi olla probleemiks lihtsamates projektides, kuid võivad osutuda probleemiks suurema võimsusega või ressursimahukam süsteem.

2.1.2 Inertsiaalandur

Inertsiaalandur on elektrooniline komponent, mis mõõdab objekti joonkiirendust ja pöörlemiskiirust. Inertsiaalandur on kombinatsioon kiirendustajurist, mis mõõdab joonkiirendust ja güroskoopandurist, mis mõõdab pöörlemiskiirust. Inertsiaalandurid on kasutusel telefonides, sammuloendurites, virtuaalreaalsuse prillides, droonides, helikopterites ja lennukites.

Esimene versioon μ-CAT robotitest kasutas MPU-6050 inertsiaalandureid. MPU-6050 on populaarne inertsiaalandur, mis ühendab ühes kiibis kolmeteljelise güroskoobi ja kolmeteljelise kiirendusanduri. [4]

2.1.3 Mootori kontrollerid

Mootori kontrollerid on seadmed, mis juhivad ja reguleerivad elektrimootorite tööd. Need võimaldavad määrata mootori kiirust ja pöörlemissuunda. Mootori kontrollerid on olulised nii tööstuslikes automaatikaseadmetes kui ka igapäevastes elektroonikaseadmetes, pakkudes täpset juhtimist ja suurendades energiatõhusust. Neid on erinevat tüüpi, näiteks alalisvoolu ja vahelduvvoolu mootorite kontrollerid.

DRV8830 on Texas Instrumentsi loodud madalapinge harjadega alalisvoolumootorite kontroller, mis on mõeldud kasutuseks väikese võimsusega rakendustes, nagu robotid, mänguasjad ja kaasaskantavad seadmed. See kiip sisaldab ühte H-silda, võimaldades juhtida ühte alalisvoolumootorit või ühte samm-mootori mähist. DRV8830 toetab I2C liidest, võimaldades lihtsat ühendamist mikrokontrolleritega. [5]

M-CAT kasutab nelja DRV8830 mootori kontrollerit, et juhtida nelja mootorit, mille küljes on uimed. Mootori kontrollerid on ühendatud trükkplaadil oleva I2C siiniga.

2.1.4 Mootori koodrid

Mootori kooder on elektromehaaniline seade, mis teisendab mootori võlli pöörlemise elektrilisteks signaalideks. See võimaldab täpset tagasisidet mootori asendi, kiiruse ja pöörlemissuuna kohta. See tagasiside on hädavajalik täpse liikumise juhtimiseks automatiseeritud süsteemides.

Koodreid on kahte peamist tüüpi: inkrementaalsed ja absoluutsed. Inkrementaalne kooder saadab iga pöörde kohta kindla arvu impulsse, võimaldades süsteemil järgida liikumist ja kiirust, kuid mitte täpset asendit ilma alguspunkti määramata. Absoluutne kooder määrab igale võlli asendile unikaalse koodi, võimaldades täpset asendi määramist ka pärast toitekatkestust.

AS5040 on kontaktivaba magneetiline pöördkooder, mis võimaldab täpset nurga mõõtmist kogu 360° ulatuses. See kiip ühendab endas Halli anduri, analoogliidese ja digitaalse signaalitöötluse, pakkudes 10-bitist eraldusvõimet ühe pöörde kohta. [6]

Anduri väljundid võimaldavad nii absoluutset nurgaandmete edastust kui ka inkrementaalseid signaale. AS5040 on programmeeritav, võimaldades kasutajal määrata nullasendi ja kohandada väljundrežiime vastavalt vajadustele. Kiip töötab toitepingega vahemikus 3.3V kuni 5V ning toetab pöörlemiskiirusi kuni 30000p/min.

M-CAT kasutab koodrite absoluutse positsiooni andmeid, et määrata iga uime praegust asendit. See võimaldab täpse uimede liigutamise ning aitab kompenseerida riistvara erinevusi uimede mootorite vahel. Andmete edastamiseks kasutavad koodrid SPI liidest, mis on ühendatud ATmega328P kiipi.

2.1.5 Rõhuandur

Rõhuandurid, mida võib leida väikestes robotites, on andurid, sageli kasutavad piesoelektrilisi takisteid ning väljastavad analoog- või digitaalsignaali, mida saab hõlpsasti lugeda mikrokontrolleri abil. Sellised andurid on tavaliselt mõeldud madalate rõhkude mõõtmiseks (alla 1 baari) ja on vastupidavad väikestes süsteemides esinevatele vibratsioonidele ja temperatuurikõikumistele.

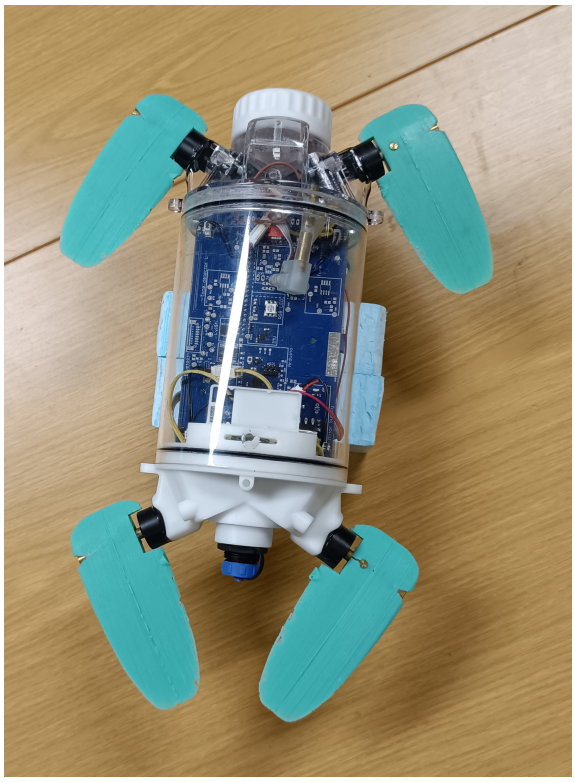
MS5407-AM on väike ja veekindel rõhuandur, mis on mõeldud kasutamiseks kompaktsetes seadmetes, nagu robotid, droonid ja kantavad seadmed. Andur mõõdab rõhku ja temperatuuri, võimaldades näiteks veealusel robotitel määrata sügavust või droonidel hinnata kõrgust. Selle veekindel konstruktsioon ja lihtne ühendamine mikrokontrolleritega teevad MS5407-AM sobilikuks erinevate keskkondade jaoks, kus on vaja usaldusväärset rõhuandurit. [7]

2.1.6 Tooniandurid

Tooniandurid (inglise keeles *tone detector*) koos fotodioodidega on kokkupandud elektrilised süsteemid, mis võimaldavad reageerida kindlale valguse sagedusele. M-CATi peal on neid paigaldatud kaks tükki. Andurid asuvad roboti eesosas, üks vasakul ja teine paremal. See võimaldab anda robotile informatsiooni keskkonnas, kus ei pruugi olla juhtmevaba sidet ning ei ole võimalik robotit lahti teha. Mikrokontrolleri jaoks on tooniandurid lihtsad loogika signaalid, mis on kõrged, kui tooniandur tajub õiget valgusesagedust. Õige sagedus on riistvaraliselt paika pandud analooglülitusega.

2.2 M-CAT versioon 2

Teisele versioonile roboti platvormist lisati täiendavaid riistvarakomponente. Peamine täiendus süsteemile on RPI arvuti, mis võimaldab operatsioonisüsteemi peal tarkvara jooksutamist ning võimaldab ROS2 liidese loomist. Lisaks arvutile on uuele robotile lisatud kaamera ja mõned riistvarakomponendid vahetati välja .[1] Vaata joonis 2.



Joonis 2. M-CAT versioon 2

2.2.1 Kaamera

Üks kõige olulisemaid laiendusi roboti riistvarale on kaamera lisamine. Kaamera pakub kasutajale palju informatsiooni teda ümbritsevast maailmast ning võimaldab keerukamat juhtimisloogikat. Kaamera mooduliks μ-CAT robotite peal on Raspberry Pi kaamera moodul versioon 2.[8] See pakub mitmeid erinevaid resolutsioonide ning pildiformaatide valikuid ning on ühendatud Raspberry Pi plaadi külge läbi CSI (kaamera jadaedastusliidese). Tulenevalt roboti piiratud protsessimisvõimekusest kasutab kaamera pildi edastus arvestatavat protsessori ja võrgu ressurssi. See tähendab, et enamasti ei ole soovitatav sarnase arvutusvõimsusega robotites kasutada kaamera maksimaalset pildiformaati ega kiirust.

2.2.2 Raspberry Pi Zero 2W

Raspberry Pi Zero 2W on kompaktne ja odav monoplaatarvuti, mis sobib väikese võimsusega projektideks. RPI pakub märkimisväärset jõudlust oma suuruse kohta, olles kuni viis korda kiirem kui algne Raspberry Pi Zero. Zero 2W sisaldab sisseehitatud Wi-Fi ja Bluetooth ühendust, mini HDMI-videoväljundit, kahte mikro-USB porti ning 40-viigilist GPIO (üldotstarbeline sisend/väljund) liidest, mis võimaldab laiendatavust. Tänu oma väikesele energiatarbimisele ja laiale kasutajaskonnale on see populaarne valik robootika

projektides.[9]

2.2.3 ICM-20608-G

Uute μ-CAT robotite peal on kasutusel ICM-20608-G IMU. ICM-20608-G on TDK IvenSense'i arendatud kuue teljega IMU, mis ühendab endas kolme teljega güroskoobi ja kolme teljega inertsiaalanduri. Mõlemad andurid kasutavad 16-bitiseid ADC-sid, mis annavad täpse digitaalse väljundi. ICM-20608-G toetab I2C ja SPI liideseid. M-CAT robotitel on see ühendatud I2C liiniga, mis jookseb ATmega328P kiipi ja RPI liidesesse. Lisaks tavalistele funktsioonidele, mida pakuvad enamus IMU-sid, on ICM-20608-G peal temperatuuriandur, programmeeritav madalpääsfilter ja ärkamine liikumise tuvastamisel.

2.2.4 MS5837-02BA

Uued μ-CAT robotid kasutavad MS5837-02BA rõhuandureid MS5407-AM andurite asemel. Võrreldes MS5407-AM anduritega on MS5837-02BA andurid väiksema mõõtevahemikuga, madalama toitepingega ning kasutavad digitaalset I2C suhtlust analoogsuhtluse asemel. [7]

2.3 Robotite Operatsioonisüsteem

Robotite Operatsioonisüsteem (*ROS*) on avatud lähtekoodiga tarkvararaamistik, mis on mõeldud robotite arendamiseks ja juhtimiseks. ROS ei ole tüüpiline operatsioonisüsteem, vaid pigem teekide ja tööriistade kogum. See aitab arendajatel luua keerukaid ning modulaarseid süsteeme, kuhu on võimalik lihtsalt lisada eksisteerivaid tarkvaralahendusi, mis on loodud ROS'iga ühildumiseks. ROS võimaldab komponentidel omavahel suhelda, tänu millele on võimalik koodi taaskasutada.

2.3.1 ROS1 ja ROS2 võrdlus

ROS2 erineb ROS1-st mitmete oluliste iseärasuste poolest, millest märkimisväärseimad on hajussüsteemide parem tugi, sisseehitatud reaalaja võimekus ja täiustatud turvafunktsioonid. Erinevalt ROS1-st, mis põhineb tsentraliseeritud sõlmede haldamisel (*rosmaster*), kasutab ROS2 hajutatud andmevahetuseks DDS-i (andmejaotusteenus), võimaldades otsest suhtlust sõlmede vahel ilma keskse juhtsõlmeta. Lisaks on ROS2 kujundatud sobivamaks reaalaja rakendustele, sisaldades paindlikumaid sõnumiedastusprotokolle, mis tagavad kiiremad ja täpsemalt määratletavad sõnumivoogude juhtimise võimalused. Tänu nendele omadustele sobib ROS2 paremini keerukatesse ja tööstuslikesse robotrakendustesse.[10][11]

3. Kasutatud tehnoloogiad

Kuna antud lõputöö tegeleb μ-CAT roboti teise versiooni riistvaraga, siis kogu tarkvara arhitektuur keskendub sellele platvormile. Loodud tarkvara arhitektuur keskendub töökindlusele ja lühikesele reaktsiooniajale. Töökindlus on oluline, kuna robot peab toimima veealuses keskkonnas, kus ei ole võimalik sooritada parandusi ning roboti vigade tuvastamine on keerukas. Kui robotil peaks tekkima süsteemi kriitiline viga, võib robot muutuda toimimisvõimetuks. Roboti lühike reageerimise aeg on vajalik, et võimaldada autonoomset juhtimist.

Loodud tarkvara pidi pakkuma ROS2 liidest RPI peal ning võimaldama ühilduvust riistvarakomponentidega üle selle liidese. Skoobi piiramiseks ei planeeritud sisse riistvarakomponentide programmeerimist isegi kui komponendid võimaldasid seda.

3.1 Programmeerimiskeel

Tähtis osa selle projekti juures on otsus programmeerimiskeele valiku osas, kuna see keel mõjutab arenduse keerukust, ühilduvust erinevate süsteemi komponentidega ja süsteemi jõudlust.

Kuna projektis on kaks koodibaasi: üks, mis jookseb RPI peal ja teine, mis jookseb ATmega328P peal, siis võib kumbki koodibaas kasutada erinevaid programmeerimiskeeli. Kuigi on eelistatud hoida mõlemad projekti osad samas keeles, ei tohiks ühe poole valik piirata teist. Vähendamaks keelest tulenevaid probleeme, on oluline valida keel, mis on olnud kasutuses sarnastes projektides.

3.1.1 RPI koodibaas

RPI koodibaasi valiku piirab suuresti ära ROS2 võimekus. Kuna RPI pealne pool peab looma ROS2 sõlme (*ROS2 node*) on mõistlik valida keel, mille jaoks on olemas projekt, mis pakub võimalikult palju ROS2 funktsionaalsust. Praeguse seisuga on ametlikult toetatud ROS2-tiimi poolt seega nende teegid pakuvad kõiki ROSi funktsionaalsusi. Lisaks sellele on olemas teegid Rust ja JavaScript keeltes. Ülejäänud keelte jaoks ei ole projekti, mis pakuks ROS2 funktsionaalsust, või eksisteerivad ainult väga algelised või toetamata versioonid ROS2 toeks.

Python

Python on kõrgetasemeline, üldotstarbeline programmeerimiskeel, mida iseloomustab lihtne ja loetav süntaks. Seda kasutatakse laialdaselt erinevates valdkondades, sealhulgas veebiarenduses, andmeteaduses, tehisintelekti jaoks, automatiseerimises ja mujal. Python toetab mitmeid programmeerimisparadigmasid, nagu objektorienteeritud, imperatiivne ja funktsionaalne programmeerimine, ning selle suur ja aktiivne kogukond tagab rohkete teekide ja tööriistade olemasolu.

Python oleks hea valik enamiku ROS liidest kasutavate süsteemide puhul tänu kiirele arendustsüklile, kuna see ei vaja kompileerimist. Pythoni koodi ei kompileerita, kuna see on interpreteeritud keel, millel on oma interpretaator. See interpretaator vähendab programmi kiirust ja suurendab mälu kasutust. Eelpool toodud asjaoludest tulenevalt ei ole võimalik nõrgema riistvara puhul saavutada lühikest reaktsiooniaega, välja arvatud juhul, kui on tegemist väga efektiivse tarkvaraga. Kahjuks on oht, et isegi hästi optimeeritud kujul jääb süsteemil võimekust puudu, et suuta saavutada soovitud funktsionaalsus antud riistvara peal. Seetõttu on mõistlikum kasutada teisi keeli, et ei peaks tarkvara ümber kirjutama hilisemas projekti faasis.

JavaScript/TypeScript

JavaScript on skriptikeel, mida kasutatakse laialdaselt nii veebibrauserites interaktiivsete lehtede loomiseks kui ka serveripoolses arenduses, näiteks Node.js abil. Siin kontekstis on JavaScript ja TypeScript pandud kokku, kuna TypeScript on JavaScriptiga ühilduv laiendus, mis lisab tüübikontrolli ja võimaldab kasutada olemasolevaid JavaScripti koodibaase. See teeb neile pakutud ROS2 liidese ka ühilduvaks mõlema keelega. Edasiminnes kasutan nime JavaScript, aga öeldu kõik kehtib ka TypeScripti kohta. JavaScript ei sobi hästi, kuna sellega tekib märkimisväärne üldkulu muutmälu ja CPU (keskprotsessori) ressursi kasutuses. Seetõttu ei ole JavaScript ideaalne süsteemide jaoks, kus on see ressurss piiratud. JavaScripti võimekus ühilduda RPI riistvaraga on piiratud ning ainsad variandid, mis pakuvad enamuste perifeeriate juhtimist, on ümbrised süsteemi omateekidele. Ümbristel on probleem käskude täpsete ajastamistega ja reaalajas jooksva juhtimisega. Kuna JavaScript ei kasutata robotikas väga sageli, on sellel vähem tuge eksisteerivate draiveritega. See teeb ka süsteemi teenustega ühendamise raskemaks.

C++

C++ on suure jõudlusega, mitmeparadigmaline programmeerimiskeel, mida kasutatakse laialdaselt süsteemitarkvara, mängude, rakenduste ja robotika arenduses. C++ on hea keel ROS2 ühilduvuseks, populaarne süsteemiprogrammeerimises ning robotikas ja tänu oma võimekusele sobib hästi vähema võimekusega süsteemide jaoks. Selle laialdane kasutus

sarnastes projektides tähendab, et abi leidmine internetist on lihtsam. Kahjuks on C++ mõned olulised puudujäägid.

1. Mäluohutus: C++-is on lihtne teha vigu mäluhalduses, kus süntaktiliselt korrektne kood võib programmi loogikas vigasid põhjustada. See tähendab, et on võimalik kirjutada kompileerivat koodi, mis töötamise ajal ei pruugi põhjustada kokku jooksmist, aga vead mälu kasutuses tekitavad oodatust erinevat käitumist või süsteemi vigu.
2. Määramatu käitumine: C++ standardid defineerivad ära, millised koodid on defineeritud. Kõik, mis läheb nendest määratud piiridest välja muudab terve programmi koodi määramatuks. Kuna kompilaatorid ei pea defineerimata käitumisel andma veateateid võib kompilaator koodi ära kompileerida, aga tekkinud programmi käitumine ei ole etteaimatav.
3. Tööriistad: C++ ei määra standardseid tööriistu, et projekte manageerida. Tänapäevaste programmeerimiskeelte puhul on harjutud, et eksisteerib palju tööriistu, mis lihtsustavad arendust. Näiteks paljud keeled pakuvad paketihaldurit (*package manager*), vormindit (*formatter*) ja standardset ehitamise protsessi. Nende tööriistade puudumine teeb projekti manageerimise keerukamaks. Rohkem aega kulub tööriistade üles seadmisele ning vähem koodi arendamisele.
4. Palju kaudset käitumist: C++ on käitumisi, mis on standardis määratud, aga koodis ei pruugi väljenduda. See tähendab, et kompilaator võib teha muudatusi selles, mis on kirjutatud ning mis päriselt juhtub. Näide sellest oleks kaudne tüübi muutmine, mida kompilaator võib teha kui sisend ja väljund tüübid erinevad (määratud piirangutega). See tekitab palju vigu, mida on keerukas leida kuna kood on ametlikult korrektne.
5. Ohtlik konkurrentsus: Kui kirjutada koodi, kus sündmused võivad toimuda samaaegselt on lihtne tekitada vigu mälu kasutuses, mida ei ole võimalik kompilaatoril leida. Need ilmnevad alles jooksmise ajal ning ei pruugi esimesel korral väljenduda.

Rust

Rust on kaasaegne süsteemitaseme programmeerimiskeel, mis keskendub turvalisusele, jõudlusele ja mälulekete vältimisele ilma mälukoristuseta. Rust lahendab kõik eelmises lõigus kirjeldatud probleemipunktid ning pakub võrreldavat võimekust C++ keelega. Kuna Rust on inspireeritud suuresti C++ keelest, on paljud osad nende süntaksist sarnased ning keelte vaheline ühilduvus on väga hea. Tänu tagasiühilduvuse puudumisele varasemate C ja C++ versioonidega oli võimalik lahendada Rustis palju probleeme, mis on leitud aastate jooksul, millal C++ keelt on kasutatud. Võrdluseks, kaasaegsed C++ versioonid on lahendanud ka sarnaseid probleeme, aga kuna iga uus versioon peab olema alati tagasiühilduv varasemate versioonidega, on alati võimalik teha samu vigu uues versioonis. Sellest tulenevalt jääb arendajale suurem vastutus korrektse koodi kirjutamisel ning kompilaator

saab vähem aidata. Rusti kasutusega tekivad omad probleemid, mida ei eksisteeri C++ keeles:

1. Puudulik ROS2 tugi: Rusti kõige populaarsem ROS2 toetuse teek on ikka arenduses ning seetõttu ei toeta kõiki ROS2 funktsionaalsusi. Peamised puudujäägid tulenevad: toimingute, Transform Library 2, näidete ja keerukamate QoS (teenuse kvaliteedi) sätete puudumisest.
2. Ehitussüsteemide ühilduvus ja tööriistade erinevused: Kuna Rusti teegid kasutavad ROS2 C++ funktsionaalsusi, et ehitada ja ühilduda Rust koodiga, nõuab see ühilduvust Rusti tööriistade ja C++ tööriistade vahel. See ühilduvus on loodud, aga võib tekitada probleeme kuna see vajab lisalahendusi.
3. Väiksem arendajate arv ja kogukond: Kuna Rust on uuem keel siis on vähem arendajaid, kes on kokkupuutunud või arendanud Rust keeles. Seetõttu on ka sarnaste projektidega tegelevate arendajate kogukond väiksem. See tähendab, et on vähem tõenäoline, et keegi teine on sarnaste probleemiga tegelenud, mis võivad projektis tekkida.
4. Õppimise keerukus: Kuna Rust on range oma reeglitega nõuab see, et arendaja saab aru, kuidas toimivad mitmed keerukad teemad nagu: omandid (*ownership*), eluajad ja asünkroonsed paradigmad.

Nende hinnangute põhjal sai RPI projektis valitud keeleks Rust, kuna kõrgema taseme platvormil oli arendaja jaoks kasutajakogemus parem kui sama lahenduse loomine C++ keeles. Sellest võib tekkida potentsiaalne probleem hiljem projekti arendajate leidmisega, kes oskaksid Rust keelt.

3.1.2 ATmega328P koodibaas

Kuna ATmega on sardsüsteemi mikrokontroller, eeldab see sardsüsteemide arendust toetava programmeerimiskeele valikut. Nende jaoks kõige populaarsemad keeled on C või C++. Tänapäeval ei ole häid põhjuseid, miks peaks uusi projekte kirjutama C keeles C++ asemel, kuna C++ pakub ühilduvust eksisteerivate C teekidega ning kõiki enamusi funktsionaalsusi, mida C. Lisaks sellele annab C++ lisa funktsionaalsusi, mis teeb sardsüsteemide arendamise lihtsamaks. Seepärast keskendub see võrdlus ainult C++ keelele. Teine keel, mida kasutatakse aina rohkem sardsüsteemides, on Rust. Kuna see sarnaneb C++ keelele paljudes kohtades, on Rust keel kogunud populaarsust uute süsteemide arendamisel. Rusti tähtsad osad sardsüsteemide jaoks on mälu hallatavus, mälukoristuse puudumine ning ühilduvus eksisteerivate C või C++ koodidega. Harva esineb ka MicroPython keel sardsüsteemide loomisel, mis kasutab Pythoni süntaksit, et juhtida madalataseme süsteemi.[12] Selle projekti raames ei ole võimalik MicroPythonit kasutada, kuna kiibil on liiga

vähe muutmälu ja välkmälu. MicroPython vajab vähemalt 256KB välkmälu ning 16KB muutmälu[13]. ATmega328P kiip pakub 32KB välkmälu ja 2KB muutmälu [3]. Keeled nagu GO, Java, Swift või Haskell on ka vahel kasutuses sardsüsteemides, aga nad ei toeta AVR perekonna protsessoreid.

Kuna järgi jäänud keeled on C++ ja Rust ning C++ on laialdasemalt kasutuses, sai C++ valitud arenduskeeleks. Esialgne versioon koodist kirjutati C++ keeles ning saavutas peaaegu täieliku funktsionaalsuse, aga välisest teegist tekkis määramatu käitumine ning kood ei toiminud. Peale mitme nädalast tööd vea leidmiseks, vahetati projekti keel Rusti peale lootuses vältida sama viga. Kogu ATmega328P koodibaas kirjutati ümber Rusti peale.

3.2 Sisemine kommunikatsioon

Enamus sisemisi madalataseme kommunikatsiooni protokolle on määratud elektroonika ülesehitusega. Näiteks andurid kasutavad I2C või SPI siini. Mootorite juhtimine on kasutades I2C liidest. ATmega328P ja RPI vahel toimub suhtlus üle UART liidese. I2C ja SPI suhtlus on määratud vastavalt komponentidele ning seal ei ole valikuvõimalust selle projekti puhul. Loodud tarkvara peab vastama komponentide tootjate kommunikatsiooni kirjeldusele. Suhtlus arvutite vahel on piiratud ainult UART liidesega, mis määrab ainult riistvara tasemelise kirjelduse, kuidas andmeid saadetakse. Selleks, et arvutite peal jooksvad koodid saaksid üksteisest aru, tuleb defineerida, kuidas informatsiooni vahetus toimib, et tõlkida kätte saadud baite.

Loodud lahenduses jadastatakse kogutud andmed või käsud. Seejärel raamitakse andmed pakettideks ning saadetakse teisele osapoolele.

3.2.1 Andmepaketid

Andmepaketid on andmestruktuurid, mida kasutatakse andmete edastamiseks arvutite vahel. Need paketid jagavad suuremad andmeplokid väiksemateks osadeks, mida saab edastada iseseisvalt. Paketid võimaldavad vea tuvastust ja parandust, kasutades kontrollsummasid, CRC (tsükliline liiasuskontroll) koode ja paarsust. Lisaks teevad paketid andmed väiksemateks ja lihtsamini hallatavateks plokkideks.

Selleks, et tagada võimalikult väikest andmete mahu üldkulu, otsustati projektis kasutada COBS-i (ühtlase üldkulu baittäitmist). COBS on algoritm, mida kasutatakse andmete raamimiseks (pakettideks tegemiseks) viisil, mis tagab, et teatud eribait (tavaliselt nullbait)

ei esine kunagi kodeeritud andmetes, välja arvatud pakettide piiride tähistamiseks. See võimaldab vastuvõtjal lihtsalt ja üheselt tuvastada, kus iga andmeplokk algab ja lõppeb, ilma et oleks vaja teostada keerulisi analüüse sisu osas. Tänu oma väikesele ja prognoositavale lisakoormusele on COBS eriti populaarne sardsüsteemides ja reaalaja rakendustes, kus töökindlus ja efektiivsus on kriitilise tähtsusega. Tänu standardi populaarsusele on olemas mitmeid standardi implementatsioone, mida on võimalik kasutada selles projektis.[14]

3.2.2 Andmete struktuur ja sõnumid

Andmete sisu tuleb ära defineerida sellisel viisil, kus on võimalik eristada erinevaid sõnumeid üksteisest ning lugeda välja erinevat tüüpi andmeid selleks, et andmete vastuvõtja saaks andmete sisust aru. Kasulik oleks, kui saaks andmeid teha eraldi sõnumiteks, et ei peaks kõiki andmeid ühe korraga saatma. Mitme sõnumi kasutus vähendab kahju, kui ühe paketi sisu on rikutud või läheb kaduma.

Eksisteerib mitmeid jadastamise formaate, mis on üpris populaarsed. Näiteks sobiksid selle projekti raames: CBOR, MessagePack, Protobuf, FlatBuffers või Cap'n Proto. Nende standardid on üpris võrdväärsed oma suuruses ja kõik toetavad skeemiga sõnumite defineerimist. CBOR ja MessagePack ei vaja skeemi, aga toetavad seda. Seetõttu võiks võrdväärses projektis valida neist ükskõik millise. Selle projekti raames sai valitud Protobuf, kuna see on hästi dokumenteeritud ning arendajal oli varasem kogemus selle kasutamisega.[15]

Sõnumid ise olid eraldatud komponentide põhjal. Eesmärk oli vältida ujukoma arve ning et sõnumid oleksid võimalikult väikesed. Sõnumi skeemi failid asuvad ATmega328P koodi hoidlas [16] ning RPI koodibaas kasutab viimast versiooni nendest failidest, et hoida mõlema poole sõnumite definitsioonid sünkroniseeritud ning pakkuda ainult ühte definitsiooni.

3.3 Väline kommunikatsioon

Välise kommunikatsiooni alla jääb kõik, mis on platvormi koodist väljapoole ning hõlmab endast kasutaja koodi suhtlust platvormi koodiga. Kuna projekti alguses oli määratud, et platvorm peab pakkuma ROS2 liidest, kasutati ROSi sõnumeid, et suhelda kasutaja koodiga.

3.3.1 ROS2 distributsioonid

ROS2 distributsioonide valik oli piiratud roboti platvormi poolt. Selleks, et saada võimalikult hea toega ROS2 tuge ning vältida vajadust ehitada ROS2 nullist, piirati operatsioonisüsteemide valik Ubuntu peale. Kuna Raspberry Pi Zero 2W toetab ainult Ubuntu 22.04 operatsioonisüsteemi, peab valitud ROS2 distributsioon toetama sama operatsioonisüsteemi. Aastal 2025 ainsad ROS2 distributsioonid, mis saavad veel uuendusi, on Humble Hawksbill ja Jazzy Jalisco.[17] Nendest ainult Humble Hawksbill jookseb Ubuntu 22.04 operatsioonisüsteemi peal. Seepärast kasutab projekt Humble Hawksbill distributsiooni ROS2-st.

Micro-ROS

Micro-ROS on spetsiaalselt väikestele ja piiratud ressursiga seadmetele (nagu mikrokontrollerid ja andurid) mõeldud versioon ROS-ist. See võimaldab väikestel seadmetel suhelda suuremate robotisüsteemide ja arvutitega, kasutades samu tööriistu ja protokolle nagu tavapärane ROS. Micro-ROS aitab ehitada keerukaid, hajutatud robootika süsteeme, kus ka väiksemad komponendid saavad olla omavahel ühendatud.

Esmapilgul võib tunduda micro-ROS ideaalne lahendus μ-CAT roboti ROS liidese jaoks, aga mõned kriitilised detailid teevad selle valiku sobimatuks. Kõik võrdlused tehti ATmega328P võimekustest lähtuvalt, kuna see on mikrokontroller, mida kasutavad mõlemad μ-CAT roboti riistvara versioonid.

1. Muutmälu kasutus - ATmega328P piiratud muutmälu (2KB) tõttu pole micro-ROS-i sellel mikrokontrolleril tõenäoliselt võimalik edukalt kasutada. Micro-ROS-i ametliku dokumentatsiooni järgi on soovitatav vähemalt 10KB muutmälu ning väikseim ametlikult toetatud mikrokontroller sisaldab 32KB, mis ületab oluliselt ATmega328P võimekust.[18][3]
2. Protsessori sõna pikkus - Soovituslik on kasutada 32 bitise sõna pikkusega protsessoreid. ATmega328P kasutab 8 bitiseid sõnu. See ei tähenda, et micro-ROSi ei ole võimalik jooksutada ATmega328P peal, aga see on kindlasti tunduvalt aeglasem [18]
3. Püsimälu - Isegi väiksemad micro-ROSi konfiguratsioonid kasutavad peaaegu 100-500KB püsimälu, mis on oluliselt ATmega328P võimekusest kõrgem. ATmega328P on 32KB püsimälu, millest 2KB kulub *Bootloaderi* jaoks. [19]
4. RTOS (reaalaja-operatsioonisüsteem) - Selle jaoks, et micro-ROS saaks toimida on vaja süsteemile veel paigaldada RTOS, mis omakorda vajab toimimiseks mälu ressurssi. Seetõttu on vajalik mälumaht veel suurem.[20]
5. Vajaduse puudumine - Kuna on olemas riistvara, millel on operatsioonisüsteemiga

arvuti, siis ei ole vaja paigaldada ATmega328P peale micro-ROSi. Operatsioonisüsteemiga arvuti, kus peal on võimalik jooksutada ROS2te, on oluliselt võimekam kui mikrokontrolleri piiratud riistvara.

4. Lahendus

4.1 Arendus

Tarkvara arendus algas 2024 suvel ning kestis kuni 2025 aasta kevadeni. Selle jooksul on eksisteerinud mitu tarkvara versiooni. Esimest versiooni arendati 2024 suve algusest kuni 2024 aasta lõpuni. Järgmises alapeatükis kirjeldatud probleemide tõttu tuli tarkvara ümber kirjutada.

4.1.1 Esimene versioon

Tarkvara esimene versioon kasutas Rust keelt RPI tarkvara arenduses. ATmega328P kood kirjutati C++ keeles, kuna μ-CAT roboti esimene versioon oli kirjutatud C++ keeles ja oli lootust taaskasutada koodi, mis oli varem loodud.

RPI kood oli sünkroonne ning toimis kui ühenduskiht UART suhtluse ja ROS2 liidese vahel. Lisaks pakkus RPI kood võimekust juhtida LEDi (valgusdioodi), mis ei vajanud UART suhtlust. Kood ei kasutanud logimist ning oli võimalikult lihtne.[21]

Selleks, et kood oleks sarnane esimese roboti versiooniga ja lihtsustada uue arendaja leidmist peale lõputöö lõppu, kasutati Arduino IDE-d (integreeritud programmeerimiskeskkond). Esimene versioon μ-CATist kasutas ka Arduino IDE-t, seega ühilduvus koodibaaside vahel oli parem kui lihtsalt C++ kasutamisega. Osa koodist (mootori enkoodrite ja mootori kontrollerite kood) sai taaskasutatud esimesest μ-CATi versiooni koodibaasist ning see vähendas arenduse ajakulu.[22]

Kuna Protobufi standard on keerukas ning on väga oluline, et kood sellele vastab, ei olnud mõttekas hakata ise kirjutama Protobufi käsitlemise teeki. See oleks kindlasti olnud väga ajakulukas ning kuna on olemas avalikke teeke, mis sellega tegelevad, ei loodud uut versiooni olemasolevast. Seepärast arenduse käigus võeti kasutusele avalik Protobufi implementatsiooni teek, nanopb.[23] See teek tekitas koodis defineerimata käitumise ning takistas koodi toimimist. Defineerimata käitumine väljendus koodi taaskäivitumises, kui üritati andmeid jadastada, kasutades valitud teegi funktsioone. Kuna ei eksisteerinud ühtegi sobilikku asendust sellele teegile ning arhitektuuri muutmine oleks osutunud liiga ajakulukaks, otsustati luua uus versioon koodist, mis on kirjutatud Rust keeles. Rust võimaldas kasutada teisi Protobufi implementatsiooni teeke kui C++. Tänu Rusti rangemale tüüpide

süsteemile ja andmete omamise nõuetele on Rust programmeerimiskeeles määramatu käitumine haruldasem.

4.1.2 Teine versioon

Teises versioonis kirjutati ümber ATmega328P koodi baas Rust keeles. Kuna otsest tõlget ei olnud võimalik luua, pidi võrdväärse funktsionaalsuse uuesti arendama. Lisaks ei olnud võimalik taaskasutada varasema μ-CATi koodi või samu teeke varasema koodi versiooniga. Uus ATmega328P kood elimineeris defineerimata käitumise ning võimaldas koodist elimineerida võimalikke süsteemikriitiliste vigade juhtumeid, mis võisid põhjustada koodi peatumist või protsessori taaskäivitumist.[16]

Lisaks sellele parandati RPI koodibaasi, lisades asünkroonsuse ning optimeerides koodi kiirust ja mälukasutust. Juurde lisandus kaks funktsionaalsust: kaamera ja akutaseme mõõtmine. Kaamera pildi jaoks võeti kasutusele Libcamera teek, mis pakkus madalataseme juhtimist kaamera üle, et efektiivselt kaamera pilti edastada ROS2 teemade peale.[24] Aku taseme jaoks mõõtis ATmega328P analoogsisendit, mis oli ühendatud pingejaguriga, mis viis aku positiivse sisendini. Täiendati RPI koodi logimisvõimekust, lisades rulluv logifail, kuhu salvestati erinevate logitasemetega ajatembeldatud informatsiooni.[16]

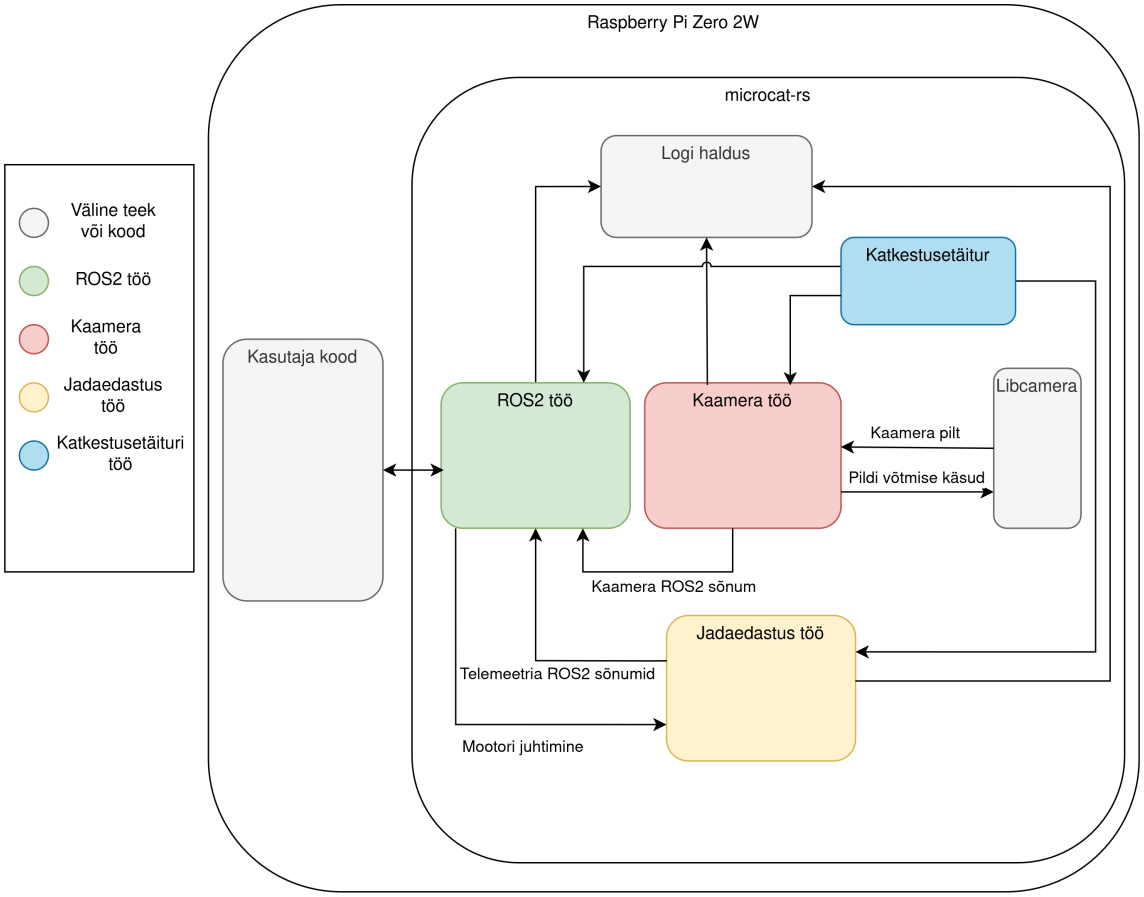
4.2 Loodud lahendus

Lõplik lahendus oli teine versioon, millest räägiti eespool (vaata paragrahv 4.1.2). Loodud tarkvara on andmete edastuskiht kõrgetaseme koodi (kasutajakoodi) ja riistvara vahel.

4.2.1 RPI kood

Kood, mis jooksis Raspberry Pi peal, oli kirjutatud Rust keeles. Projekti nimeks oli microcat-rs ja loodud koodi saab leida Githubist.[21] Kood oli ehitatud asünkroonse täitmiskeskkonna (*Asynchronous Runtime*) peale. See võimaldas tekitada erinevaid koodiplokke (töid), mida jooksutatakse asünkroonselt. Need tööd jagasid koodi ülesannete järgi ning tagasid, et ükski töö ei takistaks teisi jooksmast ootamise hetkel. Loodud koodibaasis oli kolm peamist tööd: ROS2 suhtluse töö, kaamerapildi töö ja jadaedastuse töö . Lisaks nendele kolmele tööle kasutas kood logi halduse tööd, mis oli implementeeritud välise teegi poolt, ning katkestushalduri tööd, mis toimis katkestustäiturina, olukordade jaoks, kus kasutaja edastas katkestuse programmile. Katkestusetäitur hoolitses selle eest, et iga jooksev töö saaks teada katkestusest ning lõpetaks oma töö. Vaata joonis 3. Koodi ehitamise õpetus on leitav salvest. Juhul kui on soov ehitatud tarkvara jooksutada RPI peal,

tuleb ehitamine teha ARM64 protsessoriga arvuti peal.



Joonis 3. RPI tarkvara

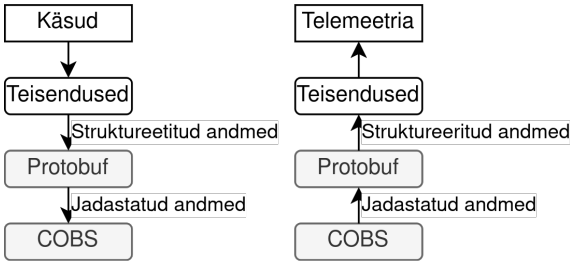
Informatsiooni edastamiseks tööde vahel seadistati eelnevalt mitteblokeerivad sõnumipõhised kanalid. Lisaks pakkusid need kanalid võimekust hoida meeles kindlat arvu sõnumeid enne, kui informatsioon üle kirjutatakse. Kõik kanalid olid üles seatud mitme looja ja üksiku vastuvõtja jaoks. Tänu sellele said näiteks jadaedastuse töö ja kaamerapildi töö saata samal ajal ROS2 liidese tööle sõnumeid, ilma et tekiks probleeme sünkroniseerimisega.

ROS2 suhtluse töö tegeles sõnumite vastuvõtmisega ROS2 liideselt ning sõnumite edastamisega ROS2 liidesele. Kuna ROS2 sõlm (*node*) jooksis eraldi töö sees, tagab see, et kui kasutaja saadab sõnumeid väga kõrge sagedusega, ei hakka muud süsteemi ülesanded selle tõttu kannatama. ROS2 suhtluse töö on esimene, mis luuakse, et süsteem saaks võimalikult varakult hakata vastu võtma sõnumeid kasutajalt. Ainus komponent, mida juhtis ROS2 suhtluse töö otse, oli LEDi juhtimine.

Kaamerapildi töö ülesanne oli seada üles õiged kaamera sätted ning seada üles suhtluskanal, mille kaudu oli võimalik saada kätte kaamera pildi informatsiooni. Kõik suhtlus ja konfigureerimine jooksis üle Libcamera teegi, mis on Raspberry Pi kaasaegne standard kaamera pildi saamiseks üle CSI (kaamera jadaedastusliidese).[25] Kaamera pildi edastus

koosneb kaadritest. Kohe kui võimalik, võtab kood uue kaadri. Pildi informatsioon oli edastatud ROS2 suhtluse tööle, kasutades eraldi kanalit.

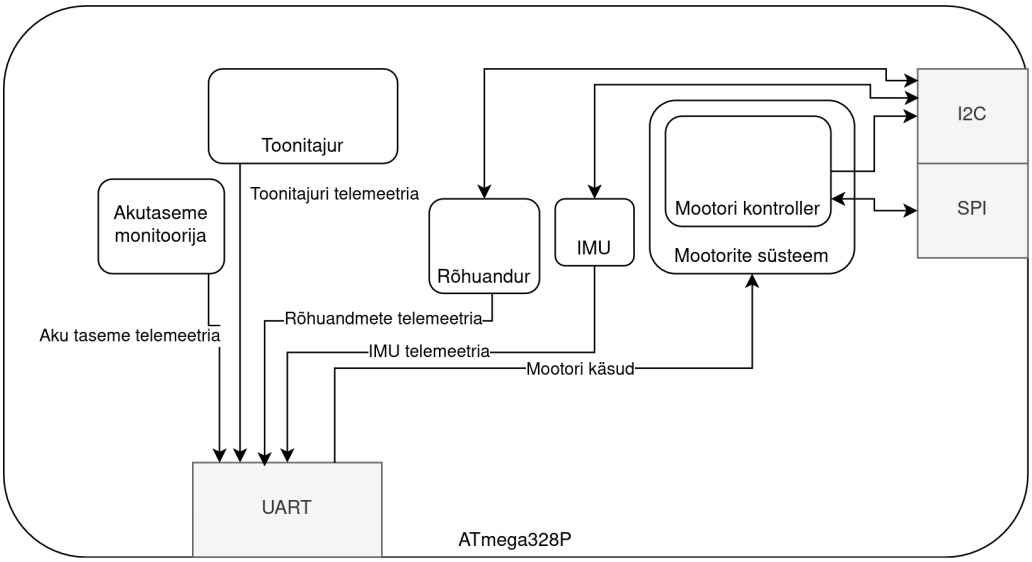
Jadaedastuse töö tegeleb info saatmisega UART liinile ja UART liinilt lugemisega. Lisaks teisendab see töö suhtlust ROS2 sõnumitest Protobufi sõnumiteks ja Protobufi sõnumitest ROS2 sõnumiteks. Olenevalt suunast peab jadaedastuse töö ka raamima või lahti raamima jadastatud sõnumeid vastavalt COBS standardile (joonis 4). Näide Rust koodist jadaedastuse saatmiseks on leitav lisas 3.



Joonis 4. Andmete jadastamine ja struktureerimine

4.2.2 ATmega328P kood

Sarnaselt RPI koodile oli ka ATmega328P kood, projektis nimega microcat-avr-rs, kirjutatud Rust keeles.[16] Tarkvara ehitamise ja jooksutamise õpetuse leiab salve *README.md* failist. Erinevalt RPI koodist oli ATmega328P kood sünkroonne. Kuna selle mikrokontrolleri võimsus oli piiratud ja kuna sellel puudusid lisa tuumad, ei oleks asünkroonsus saavutanud arvestatavat eelist. Registrite juhtimine oli koodi baasist välja abstrakteeritud ning perifeeriat juhiti läbi riistvara abstraktsiooni kihi, mida pakkus väline teek. Kood oli üles ehitatud kahes osas: ülesseadmine ja lõputu tsükkel (joonis 5). Vaata ka lisa 2.



Joonis 5. ATmega328P tarkvara

Ülesseadmine koosnes koodi objektide loomisest, mida hakati lõputu tsükli osas kasutama. Iga objekt seadis üles endale vastava komponendi ning algatas sellega suhtluse. Üks tähtsamaid osasid ülesseadmises oli mikrokontrolleri sisemiste registrite seadistamine, et seada üles perifeeria ja taimerid. Perifeeria ja taimerid, mis said üles seatud, olid UART, SPI, I2C, taimer 0, taimer 1, GPIO ja ADC. Taimer 0 oli üles seatud, et võimaldada koodis kellaaja arvestamist. Taimer 0 loendab millisekundeid alates süsteemi üles tulemisest. Kuna taimeri väärtust hoitakse 32-bitises märgita täisarvu tüüpi muutjas, toimub ületäitumine 49 päeva pärast. See oli piisavalt aega enamike kasutusjuhtude jaoks, kus μ-CATi võib vaja minna. Taimer 1 oli üles seatud süsteemi lõputu tsükli ühtlase sageduse hoidmiseks. Taimer 1 katkestus seadis globaalse lipu, mis lubab järgmisel tsükli iteratsioonil joosta.

Riistvarakomponentidele saadeti ülesseadmisel vajalikud seadistused. Ühendus nende komponentidega toimus üle suhtluskanalite (SPI või I2C). Need seadistused tagasid, et komponendid käitusid oodatavalt iga süsteemi taaskäivituse järel. Riistvarakomponendid, mis ei kasutanud suhtluskanaleid (akutaseme monitoorija, tooniandur ja staatuse LED) ei vajanud üles seadmist.

5. Test ja katsed

Selleks, et tagada loodud süsteemi toimivust, sooritati latentsuse test, töökindluse katse ja funktsionaalsuse verifitseerimine. Need kontrollid koos tõestavad süsteemi sobilikkust kasutuseks. Kõik testid sooritati viimase koodi versiooniga, et vältida muutustest tekkivaid erinevusi.

5.1 Latentsuse test

Latentsuse testi käigus mõõdeti aega, mis kulus andmetel jõudmiseks ROS2 liidesest kuni riistvarakomponentideni ja vastupidi. Kõige pikem tee selle jaoks oli ROS2 liidesest kuni ATmega328P arvutini.

Esimene plaan aja mõõtmiseks oli ajatemplite võrdlemine ROS2 liideselt sõnumi saatmisest kuni ATmega328P peal välja saatmiseni. See plaan ei toiminud, kuna RPI poolne jadaedastus liidese koodis tekkis probleem. Selle lõputöö raames ei jõutud seda probleemi lahendada, seega latentsuse testid sooritati teisiti.

Teine viis latentsuse mõõtmiseks oli aja mõõtmine RPI peal. Sõnumi laekudes ROS2 liidesesse võeti ajatempel, mida võrreldi ajatempli vastu, mis võeti enne jadaedastus liidesesse väljasaatmise vastu. Teine pool mõõtmisest toimus ATmega328P peal. ATmega328P peal võeti ajatempel hetkest, millal ATmega328P tõlkis lahti sõnumi jadaedastus liidese pealt. Järgmine ajatempel võeti enne informatsiooni rakendamist riistvara juhtimiseks. Omades neid ajatempleid, oli võimalik arvutada aeg, mis informatsioonil kulus liikumiseks läbi RPI koodi, ning aeg, mis kulus informatsioonil liikumiseks läbi ATmega328P koodi. Lõpliku hinnangu saatmiseks need ajad liideti kokku ning sellele lisati arvutatud ajakulu jadaedastus liidese peal liikumiseks.

ATmega328P mõõdetud ajakulud millisekundites olid leitavad *microcat-avr-rs* hoidlas. [16] Ajakulu asus *manual-latency-test* harus *output.csv* failis. Seal failis oli iga rea peal kirjas ühe katse tulemus, mis kulus sõnumi lahti tõlkimisest kuni juhtimiseks rakendamiseni. Need ajad on piisavalt ühtlased, et tulemused oleksid usutavad. Nende väärtuste mediaan ja mood olid 11ms. Minimaalne ajakulu oli 3ms ja maksimaalne ajakulu oli 19ms.

RPI mõõdetud ajakulud mikrosekundites olid leitavad *microcat-rs* hoidlas *manual-latency-test* harus. [21] Failis *timing_data.csv* olid esimeses tulbas mõõtmise indeksid, teises

tulbas olid kättesaamise ajatemplid, kolmandas tulbas vastavad väljasaatmise ajatemplid ja neljandas tulbas kokku arvutatud ajakulud.(tabel 1) Mediaan nende ajakulude peale oli 435.6μs. Mood nendes ajakuludes oli 442μs ning miinimum ja maksimum olid 354μs ja 3390μs.

Tabel 1. *Lõik RPI ajatulemuste mõõtmete tabelist*

Nr	**ROS2 sõnumi saabumine** (μs)	**Jadaedastusliideselt välja saatmine** (μs)	**Ajakulu** (μs)
0	1747510653569657	1747510653570157	500
1	1747510654622564	1747510654622996	432
2	1747510655673959	1747510655674418	459

Nende aegadele lisandub aeg, mis kulub vastava suurusega andmete edastamiseks üle jadaedastusliidese. Ühe käsu sõnumi pikkus jadastatult oli 7 baiti. Sellele lisandus juurde 2 baiti, et pakkida informatsioon pakettideks. Veel lisandub igale baitile 1 bit, et tähistada baidi algust ja 1 bit, et tähistada baidi lõppu. Kokku on iga sõnumi pikkus $(7+2)*(8+2) = 990$ biti. Kuna süsteem kasutas sümbolikiirust 115 200 sümbolit sekundis, tähendab see, et valem arvutamiseks aeg, mis kulub kontroll sõnumi edastamiseks, on järgnev: $990/115200 \approx 0.00859$ sekundit. See on 8.59 millisekundit.

Kokku arvutatult on keskmine latentsus kontroll sõnumi jaoks 20 millisekundit. See on väga pikk aeg kiirete reaktsioonide jaoks, kuid see ei tohiks takistada roboti juhtimist.

5.2 Töökindluse katse

Töökindluse näitamiseks sooritati katse, mille käigus pidi süsteem püsima üleval 12 tundi. Kuna roboti aku ei pruugi kesta 12 tundi toimus test toiteplokiga. 12 tunnine katse andis kindlama tõestuse süsteemi tarkvara vastupidavusest kui aku pealt oleks olnud võimalik.

Rqt on graafiline tööriistakomplekt, mis tuleb kaasa ROS2-ga ja võimaldab visualiseerida, jälgida ning hallata ROS2 süsteemi komponente. Selleks, et tekitada päris kasutusele vastav olukord, kasutati rqt programmi, et perioodiliselt saata mootori käske. Lisaks kasutati sama programmi, et kuulata teemasid, mida saatis välja μ-CAT robot.

Selle testi käigus ei tekkinud ühtegi süsteemi kriitilist viga (viga, mis oleks põhjustanud süsteemi seiskumist või taaskäivitumist). Töötamise käigus esines vigu sõnumite edastamisega jadaedastus tööle. Tõenäoliselt põhjustas selle vea sõnumite kogunemine ROS2 liideses. Kohe peale sõnumite kättesaamist üritas ROS2 töö edastada sõnumeid jadaedastus

tööle, mis ei jõudnud piisavalt sõnumeid, nii lühikese aja jooksul, välja saata.

5.3 Funktsionaalsuse katse

Funktsionaalsuse kontrollimiseks loodi ajutine kood, mis lahendas aine IAS0060 *Robotics* ülesande. See näitab, et süsteem andis kogu vajaliku informatsiooni ja võimaldas juhtimise, et lahendada ülesandeid seal aines. Kuigi tulemus ei taga täielikku funktsionaalsust, mis võib olla vajalik aine ülesannete jaoks, annab see kõrgema kindluse kui testimata süsteem. Kõikide aine ülesannete lahendamine lõputöö raames oleks olnud liiga mahukas.

Ülesanne, mida otsustati lahendada, oli Tagasisidega sügavuse juhtimine (*Closed Loop Depth Control*). See kasutas informatsiooni roboti tajuritelt, et juhtida mootoreid. Lahendus arendati Python keeles, kuna see on populaarne valik eelnevalt mainitud aines ning näitab süsteemi toimivust olenemata kasutaja koodi programmeerimiskeelest. Lahenduse koodi saab leida *depth_control* projekti hoidlas. [26]

6. Kokkuvõte

Biorobootika keskuse μ-CAT roboti uuele riistvaraversioonile oli vajalik luua ROS2-liidesega ühilduv tarkvara, mis võimaldaks platvormi kasutamist õppe- ja teadusprojektides. Lõputöö käigus loodi vastav tarkvara. Tarkvara arendamisel jagati platvorm kaheks eraldi kihiks: RPI (*microcat-rs*) ja ATmega328P (*microcat-avr-rs*), mis suhtlevad omavahel jadaühenduse kaudu, kasutades Protobufi andmete jadastamiseks ja COBSi pakettide raamimiseks.

Loodud lahenduse sobivust hinnati latentsuse, töökindluse ja funktsionaalsuse testidega. Latentsuse mõõtmised näitasid, et süsteemi viide oli liiga suur, peamiselt ATmega328P tarkvara piirangute ja optimeerimata koodi tõttu. Töökindluse katse (12h) läbis süsteem edukalt, kuigi kiire järjestikune sõnumite saabumine põhjustas üksikuid tõrkeid. Funktsionaalsuse katse, mille käigus loodi Pythonis näidiskood sügavuse hoidmiseks, oli edukas ning süsteem töötas stabiilselt.

Süsteemi edasiarenduses tasub keskenduda viite välja optimeerimisele. Siis, kui viite probleemid saavad lahendatud, oleks mõistlik lisada jadaedastuse andmepakettidele tsükkelkoodkontroll, eelistatult CRC8. Riistvara paranduste korral oleks kasulik vahetada välja ATmega328P mikrokontroller mõne modernsema mikrokontrolleri vastu, mis oleks vähemalt 32-bitine.

Kasutatud kirjandus

[1] Kilian Hubertus Ochs. “Solid Modelling and Testable Electronic Design of a Small Underwater Robot”. Magistritöö. Tallinn University of Technology, 2020.

[2] Arduino. Arduino. URL: https://docs.arduino.cc/hardware/uno-rev3/#tech-specs (vaadatud 17.05.2025).

[3] *ATmega328P Automotive - Complete Datasheet*. Atmel. URL: https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf (vaadatud 10.03.2025).

[4] *MPU-6000 and MPU-6050 Product Specification Revision 3.4*. InvenSense Inc. URL: https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6000-Datasheet1.pdf (vaadatud 18.05.2025).

[5] *DRV8830 Low-Voltage Motor Driver With Serial Interface*. Texas Instruments. URL: https://www.ti.com/lit/ds/symlink/drv8830.pdf?ts=1747531879595 (vaadatud 18.05.2025).

[6] *AS5040 10-Bit 360º Programmable Magnetic Rotary Encoder*. ams-OSRAM AG. URL: https://look.ams-osram.com/m/1b628d2f3c20b9a4/original/AS5040-DS000374.pdf (vaadatud 10.03.2025).

[7] *MS54XX Miniature SMD Pressure Sensor*. TE Connectivity. URL: https://eu.mouser.com/datasheet/2/418/8/ENG_DS_MS54XX_B4-1134913.pdf (vaadatud 18.05.2025).

[8] Raspberry Pi Foundation. *Raspberry Pi Camera Module 2*. https://www.raspberrypi.com/products/camera-module-v2/. Accessed: 2025-05-18. 2016.

[9] Raspberry Pi Foundation. *Raspberry Pi Zero 2 W*. https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/. Accessed: 2025-05-18. 2021.

[10] Steven Macenski *et al.* “Robot Operating System 2: Design, architecture, and uses in the wild”. *Science Robotics* 7.66 (2022), eabm6074. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[11] Morgan Quigley *et al.* “ROS: an open-source Robot Operating System”. Teoses: *ICRA workshop on open source software*. Köide 3. 3.2. Kobe, Japan. 2009, lk. 5.

[12] Damien George. *MicroPython*. https://micropython.org. Accessed: 2025-05-19. 2013.

[13] Damien George. MicroPython README.md. George Robotics Limited. URL: https://github.com/micropython/micropython/blob/master/README.md (vaadatud 12.04.2025).

[14] Stuart Cheshire ja Mary Baker. "Consistent Overhead Byte Stuffing". Teoses: *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*. ACM. 1997, lk. 211–222.

[15] Google. *Protocol Buffers*. https://developers.google.com/protocol-buffers. Accessed: 2025-05-18.

[16] Mikk Loomets. *microcat-avr-rs*. Versioon v1.0.0. GitHub, 17. mai 2025. URL: https://github.com/miloom/microcat-avr-rs.

[17] Open Robotics. Open Robotics. URL: https://docs.ros.org/en/humble/Releases.html.

[18] micro-ROS Project. *Supported Hardware*. ePorisma, The Eclipse Foundation, 2024. URL: https://micro.ros.org/docs/overview/hardware/ (vaadatud 12.04.2025).

[19] micro-ROS Project. *Benchmarking*. ePorisma, The Eclipse Foundation, 2024. URL: https://micro.ros.org/docs/concepts/benchmarking/benchmarking/ (vaadatud 12.04.2025).

[20] micro-ROS Project. *Supported RTOSes*. ePorisma, The Eclipse Foundation, 2024. URL: https://micro.ros.org/docs/overview/rtos/ (vaadatud 12.04.2025).

[21] Mikk Loomets. *microcat-rs*. Versioon v0.1.0. GitHub, 17. mai 2025. URL: https://github.com/miloom/microcat-rs.

[22] Mikk Loomets. *microcat-avr*. GitHub, 17. mai 2025. URL: https://github.com/miloom/microcat-avr.

[23] *nanopb*. Versioon v0.4.8. GitHub, 17. mai 2025. URL: https://github.com/nanopb/nanopb/tree/nanopb-0.4.8.

[24] *libcamera*. Versioon v0.3.2. GitHub, 17. mai 2025. URL: https://github.com/raspberrypi/libcamera/tree/v0.3.2.

[25] Raspberry Pi Ltd. URL: https://www.raspberrypi.com/documentation/computers/camera_software.html (vaadatud 20.04.2025).

[26] Mikk Loomets. *depth_control*. Versioon v1.0. GitHub, 17. mai 2025. URL: https://github.com/miloom/depth_control.

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

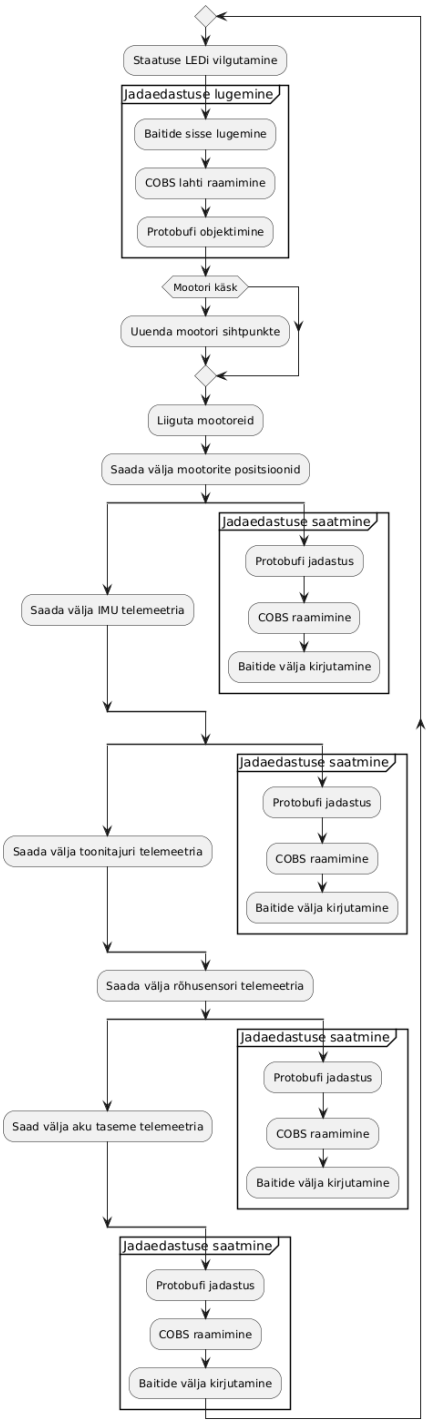
Mina, Mikk Loomets

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose "ROS2 liidesega biomimeetilise roboti tarkvaraplatvormi arendus", mille juhendaja on Jaan Rebane and Gert Kanter
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

01.06.2025

[1] Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - ATmega328P tsükkel



Lisa 3 - Rust jadaedastuse koodi näide

```rust
let message = match command {
    Command::MotorTarget(pos) => {
        let data = motor::MotorTarget {
            target_position: pos.target_position,
            amplitude: pos.amplitude,
            frequency: (pos.frequency * 1000.0) as u32,
            location: match pos.location {
                MotorLocation::FrontLeft =>
                    motor::Location::FrontLeft.into(),
                MotorLocation::FrontRight =>
                    motor::Location::FrontRight.into(),
                MotorLocation::RearRight =>
                    motor::Location::BackRight.into(),
                MotorLocation::RearLeft =>
                    motor::Location::BackLeft.into(),},
        };
        message::Message {
            data: Some(message::message::Data::MotorTarget(data))
        }
    }
};
let mut buf = BytesMut::new();
message.encode(&mut buf);
let mut dest = [0u8; 128];
let len = cobs::encode(buf.iter().as_slice(), &mut dest);
dest[len] = 0;
for byte in dest[..=len].iter() {
    match serial.write_all(&[*byte]).await {
        Ok(_) => {
            tokio::time::sleep(
                tokio::time::Duration::from_micros(2500)).await;
        }
        Err(e) => {
            error!("Failed to write to serial: {}", e);
        }
    }
}
```