Kert Karbe

ÕPPEOTSTARBELINE PROGRAMMEERIMISÜLESANNETE AUTOMAATNE KONTROLLI PROGRAMM

LÕPUTÖÖ

Tehnikainstituut
Robotitehnika õppekava
Juhendaja: Villu Lõhmus
Kaasjuhendaja: Kristo Vaher

Tallinn 2024

Autori deklaratsioon ja lihtlitsents

Mina, Kert Karbe, tõendan, et lõputöö on minu kirjutatud. Töö koostamisel kasutatud teiste autorite, sh juhendaja teostele on viidatud õiguspäraselt.

Kõik isiklikud ja varalised autoriõigused käesoleva lõputöö osas kuuluvad autorile ainuisikuliselt ning need on kaitstud autoriõiguse seadusega.

Juhendajad Villu Lõhmus, Kristo Vaher (allkirjastatud digitaalselt)

Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks

Mina, Kert Karbe, sünnikuupäev 15.11.1998, annan Tallinna Tehnikakõrgkoolile (edaspidi kõrgkool) tasuta loa (lihtlitsentsi) enda loodud teose „Õppeotstarbeline programmeerimis ülesannete automaatne kontrolli programm"

1. reprodutseerimiseks paberkandjal kõrgkooli raamatukogus avaldamise ja säilitamise eesmärgil;
2. elektroonseks avaldamiseks kõrgkooli repositooriumi kaudu;
3. kui lõputöö avaldamisele on instituudi direktori korraldusega kehtestatud tähtajaline piirang, lõputöö avaldada pärast piirangu lõppemist.

Olen teadlik, et nimetatud õigused jäävad alles ka autorile ja kinnitan, et

1. lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest tulenevaid ega muid õigusi;
2. PDF-failina esitatud töö vastab täielikult kirjalikult esitatud tööle.

Tallinnas, (allkirjastatud digitaalselt)

Sisukord

LÜHENDITE JA TERMINITE LOETELU

ROS – Roboti operatsioonisüsteem, mis on robotite arendamiseks mõeldud tarkvararaamistike komplekt.

Python – Kõrgtaseme programmeerimiskeel.

OSRF - *Open Source Robotics Foundation*, organisatsioon, kes arendab ROS-i.

RViz – ROS *Visualization*, vahend roboti andmete analüüsimiseks.

Gazebo – ROS- i virtuaalne roboti simulaatori keskkond.

RAM – süsteemi operatiivmälu.

AI – *Artificial intelligence* ehk tehisintellekt.

IoT - *Internet of things* ehk asjade internet.

Backend – Andmetele juurdepääsu kihi tarkvara.

HTTP - *Hypertext Transfer Protocol* on võrguprotokoll teabe edastamiseks.

OpenCV – arvutinägemise ja masinõppe tarkvara teek.

Teek – *library* on komponentide kogu, mis pakub lisatud funktsionaalsust.

IDE - Integreeritud programmeerimiskeskkond.

Dynamic typing – muutujad defineeritakse automaatselt kompilaatoris.

Debug – silumine on protsess, mille käigus parandatakse vigu koodis.

TCP – Transpordikihi võrguprotokoll.

URDF - Roboti kinemaatiline mudel.

SLAM – Roboti samaaegne lokaliseerimine ja kaardistamine.

SBC- *Single-board Computer* ehk plaatarvuti.

I/O – Sisend/väljund, tähistatakse arvuti suhtlust välise maailmaga.

Git – versioonihaldustarkvara.

LMS – *Learning management system* ehk õppehaldussüsteem.

CLI – Operatsioonisüsteemi käsurida.

OS – Operatsioonisüsteem.

URDF – Roboti kinemaatiline mudel.

SISSEJUHATUS

Automatiseerimine on tänapäeva infotehnoloogilises maailmas ülimalt oluline, sest tarkvara programmid muutuvad aina keerulisemaks ja nendega koos suureneb koodide maht. Automaatselt tarkvara koodi kontrolliv programm võimaldab muuta protsessid efektiivsemaks ning mille rakendamisel tõuseb koodi kvaliteet.

Käesoleva lõputöö teema on pärit Tallinna Tehnikakõrgkoolist, kus töö autor omandab bakalaureusekraadi robotitehnika erialal. Teemavaliku ajendiks osutus õppejõudude tagasiside, mille käigus selgus, et koolil puudub ROS-i tarkvara automaatkontrolli programm, mis automatiseeriks õppetööd ning vähendaks töökoormust osapoolte jaoks. Töö on seotud Karl Kingi lõputööga „ROS-i kursuse loomine", mille käigus programmi rakendatakse. Sarnane väljatöötatud süsteem on kasutusel Tallinna Tehnikaülikoolis.

Antud lõputöö eesmärgiks on mobiilrobootika kursuse efektiivsemaks muutmine tänu uue õppevahendi loomisele, mis aitaks tudengitel automaatselt kontrollida tarkvarakoodi sobivust ROS (Roboti operatsioonisüsteemi) platvormi jaoks ning saada kohest tagasisidet moodle keskkonnast sooritusele. Loodud programm kontrollib 8 harjutust, mis muutuvad ajas raskusastmete järgi.

Lõputöö koosneb neljast peatükist. Esimene peatükk annab ülevaate ROS-ist ehk roboti operatsioonisüsteemist, mis see endast kujutab ning milline on otstarve robotis. Teine peatükk räägib programmeerimiskeelest Python. Kolmas peatükk annab ülevaate Tallinna Tehnikakõrgkooli mobiilrobootika kursuse struktuurist ning kirjeldab automaatkontrolli programmi tööpõhimõtet ja kasutatud tehnoloogiaid ning neljas peatükk sisaldab kaheksat harjutust, mida programm kontrollib automaatselt mobiilrobootika kursuse raames.

1. ROS (Roboti operatsioonisüsteem)

1.1 Üldinfo ROS-i kohta

ROS ehk roboti operatsioonisüsteem on avatud lähtekoodiga loodud tarkvararaamistik, mida kasutatakse robotite arendamiseks kasutades Pythoni programmeerimiskeel. See võimaldab mitmeid funktsioone ja tööriistu arendajatele, et luua programme robotite juhtimiseks, kaardistamiseks, navigeerimiseks, pilditöötluseks, simuleerimiseks. [1]

ROS on laialdaselt levinud, kuid peamised kasutusvaldkonnad in järgmised [2]:

- **Tööstuslikud robotid**: ROS-i kasutatakse tööstuslike robotite automatiseerimiseks erinevates tööstusharudes.
- **Teenindusrobotid**: Teenindusrobotite, nagu näiteks koristusrobotide ja hooldusrobotide arendamisel ja juhtimisel on ROS oluline tööriist.
- **Autonoomsed sõidukid**: ROS-i kasutatakse laialdaselt autonoomsete sõidukite, nagu autode, droonide ja allveelaevade juhtimissüsteemide arendamiseks.
- **Meditsiiniline robootika**: Meditsiinis kasutatakse ROS-i näiteks kirurgiliste robotite juhtimiseks ja meditsiiniliste seadmete arendamiseks.
- **Haridus ja teadusuuringud**: ROS on populaarne platvorm robootika alase hariduse omandamise ja teadusuuringute jaoks.
- **Agrorobootika**: Põllumajandussektoris kasutatakse ROS-i näiteks autonoomsete põllumajandusmasinate, droonide ja kasvuhoone robotite arendamiseks.

ROS-i eelisteks on platvormi avatus, mis tagab kasutusvabaduse ning loodud programmid saab üle kanda teistele robotitele, kui ROS versioonid ühtivad. [1]

Miinusteks on jõudluse ja ressurssi nõuded suuremahuliste andmetöötlusprotsesside või keerukamate simulatsioonide korral ning algajatele mahukas õppimiskõver, sest eeldab programmeerimisoskust ja arusaama süsteemi arhitektuurist. [1]

Välja on töötatud ROS 1 (algne) ja ROS 2 (täiustatud) versioonid, mis erinevad üksteisest kommunikatsiooni protokolli, reaalaja ja operatsioonisüsteemi toetuse, skaleeritavuse ning turvalisuse poolest. 2024 aasta kevade seisuga on kasutusel *Noetic Ninjemys* ROS 1 ja *Iron Irwini* ROS 2 puhul. On olemas veel eraldi tööstusrobootikale spetsialiseerinud ROS *Industrial.* [3]

ROS kui platvormi ise arendab Open Source Robotics Foundation (OSRF). [1]

1.2 Põhiterminid

Kasutatakse järgnevaid termineid [2]:

Tuum (*roscore*) – alati taustal, viib **sõlmi** omavahel kokku.

Sõlm (*node*) – konkreetsele ülesandele pühendatud programm.

Sõnum (*message*) – "muutujatüüp" ehk infokild.

Rubriik (*topic*) – sõne, mis annab **sõnumi** voole tähenduse.

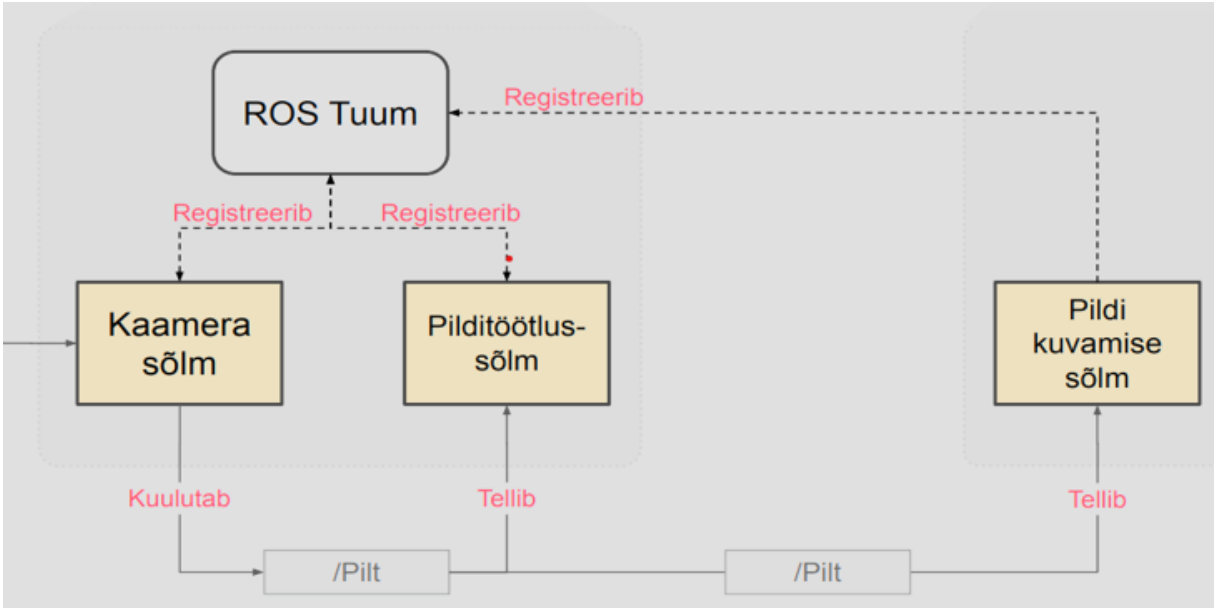
Kuulutaja (*publisher*) – kuulutab **sõnumeid** kindlas **rubriigis.**

Tellija (*subscriber*) – jälgib **sõnumeid** kindlas **rubriigis.**

Kimp (*package*) - funktsionaalne kogum **sõlmi.**

1.3 Infovahetus

Kommunikatsiooni mudel eeldab, et nii kuulutaja (*publisher*) kui ka tellija (*subscriber*) suhtlevad ühe teema kaudu, mis on määratletud konkreetse nimega ning on ühenduses ROS tuumaga, siis ainult sel juhul saavad sõlmed (*nodes*) üksteisega infot jagada. [3]



Joonis 1. Skeem ROS infovahetusest [2]

1.4 Spetsiifilised käsklused ja tööriistad

ROS kasutab oma tarkvaras spetsiifilisi käsklusi, mis võimaldavad kasutajatel hallata ROS sõlmi (*nodes*), teostada tegevusi nagu teemade *(topics*) ja teenuste *(services*) loomist ning käivitada visualisatsioon tööriistu nagu *RViz* (ROS Visualization), mis on visualiseerimisvahend roboti andmete analüüsimiseks ja *Gazebo*, mis on roboti simulaator keeruliste virtuaalsete keskkondade loomiseks reaalajas. Kuvatõmmised *Rviz* ja *Gazebo* keskkondadest on esitatud lisades (Lisa 1 ja 2). [6]

Peamised käsklused [5]**:**

Catkin tööruumi (*workspace*) loomine:

1. **mkdir** – loob kausta, kus hoitakse ROS- i pakette.
   $ mkdir -p ~/catkin_ws/src
2. **catkin_make** – kinnitab valitu kausta ROS -i tööruumina.
   $ cd ~/catkin_ws/
   catkin_make

ROS-i käivitamiseks mõeldud käsklused:

1. **roscore** - käivitab ROS-i põhikomponendi.
   $ roscore
2. **rosrun** - ühe ROS-i sõlme käivitamine.
   $ rosrun kimbu_nimi sõlme_nimi
   $ rosrun näide test.py
3. **roslaunch** - mitme ROS-i sõlme käivitamine.
   $ roslaunch kimbu_nimi käivitusfail
   $ roslaunch robot_test robot.launch

RViz (ROS Visualization) tööriista käivitamine:

1. **rviz** – käivitab simulatsiooni, mis kuvab 3D robotmudeli.
   $ rviz

Gazebo roboti simulaatori keskkonna käivitamine:

1. **gazebo_ros** – käivitab keskkonna
   $ roslaunch gazebo_ros empty_world.launch

2. PROGRAMMEERIMISKEEL PYTHON

2.1 Üldine info

Python on kõrgtaseme programmeerimiskeel, mis on tänapäeval väga populaarne eelkõige oma funktsionaalsuse ja kasutajasõbralikkuse poolest, sest kirjapilt sarnaneb inglise keelele. Pythoni lõi Hollandi programmeerija Guido Van Rossum ning esimene stabiilne versioon tuli välja kui Python 0.9.0 aastal 1991. Aastaks 2024 on kasutusel Python 3 oma erinevate alaversioonidega. [7]

Põhilisemad kasutusvaldkonnad [7]:

- **Andmeteaduses ja andmeanalüüsis**:
  Python võimaldab andmetest arusaamist läbi analüüsi ja visualisatsiooni.
- **Veebiarenduses**:
  Kasutatakse näiteks serveri (backend) poolsete loogikate kirjutamiseks HTTP päringute ja vastuste korral.
- **Masinnägemises:**
  Programmide loomiseks läbi OpenCV tarkvara teegi.
- **AI ehk tehisintellekti ja masinõppes:**
  Kasutatakse masinõppe mudelite programmeerimiseks ja treenimiseks.
- **Mängude arenduses:**
  Luuakse lisa funktsionaalsusi mängudele nagu 3D graafilised elemendid.
- **ROS-is ja robootikas:**
  Põhiline programmeerimiskeel robotites.
- **IoT-s ehk asjade internetis:**
  Kasutatakse Iot seadmete: andurite, täiturite ja mikrokontrollerite programmeerimiseks.
- **Automatiseerimises:**
  Kasulik kliendi poolsete protsesside automatiseerimises nagu vestlusrobotite, emailide, Exceli, maksete ja sotsiaalmeedia puhul.

Eelkõige kirjutatakse pythonis kasutades integreeritud programmeerimiskeskkondi ehk *IDE*-sid, mis hõlbustavad tarkvaraarendust oma lisatud funktsionaalsuse poolest. Enamlevinumad on Visual Studio Code ja PyCharm. [8]

2.2 Tugevused ja nõrkused

Pythoni tugevusteks on [7]:

- **Algajate sõbralik:**
  Pythonil on lihtne süntaks ja arusaadav struktuur tänu, millele on koodis vigade ja paranduste leidmine ehk silumine (debug) kergem võrreldes teiste keeltega.
- **Laialdased rakenduse võimalused:**
  Kasutusvaldkonnad andmeteadusest, veebiarenduse, AI kuni robootikani.
- **Avatud lähtekoodiga ja tasuta:**
  Pythoni kasutamisel puuduvad litsentsitasud isegi ärilisteks eesmärkideks.
- **Rikkalik teekide (libraries) tugi:**
  On kasutusel erinevad teegid, mis lisavad koodile funktsionaalsusi ning säästavad programmeerijate aega ja vaeva.
- **Automatiseerimise võimalused:**
  Sobib suurepäraselt lihtsate programmide loomiseks, mis muudavad korduvad ülesanded automaatseteks ja tänu sellele suurendavad tootlikkust.
- **Hästi toetatud keskkond:**
  Pythonil on laiapõhjaline võrgukogukond oma e-kursuste ja foorumiga, kus on võimalik saada vastuseid teemakohastele küsimustele.

Pythoni nõrkusteks on olukorrad [7]:

- **Kus rakenduse kiirus on esmatähtis:**
  Python on interpreteeritav keel, see tähendab koodi tõlgendatakse rida-realt, mis muudab selle aeglasemaks kui kompileeritud keeled, nagu C++ ja Java.
- **Kus mälu on piiratud:**
  Kuigi dünaamiline kirjutamine (*dynamic typing*) - muutujad defineeritakse automaatselt kompilaatoris, muudab programmeerimise lihtsamaks, kuid samal ajal raskeneb suuremahuliste programmide silumine (debug) ja hooldamine.

2.3 *Libraries* ehk teegid

Tavaliselt asub raamatukogus kollektsioon raamatuid või koht, kus hoitakse raamatuid hiljem kasutamiseks. Sarnaselt on programmeerimises, kus *library* ehk teek on eelkompileeritud koodide kogu, mida rakendatakse kindlaks määratud operatsioonide jaoks, mis pakuvad lisatud funktsionaalsust. [9]

Pythoni teek on seotud moodulite kogu. See sisaldab koodi pakette, mida saab erinevates programmides korduvalt kasutada, mis omakorda muudab Pythoni programmeerimise lihtsamaks ja mugavamaks, kuna puudub vajadus programmidele sama koodi kirjutada. Standardteek koosneb enam kui 200 põhimoodulist, mis on kirjutatud hoopis C programmeerimiskeeles. Pythoni teegid mängivad väga olulist rolli masinõppe, andmeteaduse ja andmete visualiseerimise valdkondades. [9]

Enamkasutatavad teegid [9]:

- **TensorFlow:**
  Avatud lähtekoodiga teek, mida kasutatakse peamiselt kõrgetasemelisteks arvutusteks, kuid ka masinõppes ja süvaõppe algoritmides.
- **Matplotlib:**
  Andmeanalüüsis kasutatakse arvandmete graafilise kujutamise eesmärgil, mille abil luuakse erinevaid diagramme.
- **Pandas:**
  Masinõppe teek, mis võimaldab oma tööriistadega andmeid analüüsida.
- **Numpy:**
  Numeric python on teek, mis on loodud andmemassiivide haldamiseks ja töötlemiseks sisseehitatud matemaatiliste funktsioonidega.
- **SciPy:**
  Scientific Python on teek, mis koostöös *numpy*-ga teostab keerukaid arvutusi.
- **Scrapy:**
  Avatud lähtekoodiga teek, mis võimaldab veebisaitidelt andmeid hankida. Samuti kasutatakse andmete kaevandamiseks ja automatiseeritud testimiseks.
- **PyTorch:**
  Suurim masinõppeteek, mida kasutatakse närvivõrkude loomiseks ning optimeerib tensorarvutusi GPU kiirendusega.

Pythoni programmis rakendatakse teeke kasutades käsklust import „teegi nimi". [9] Näitena kutsume välja pandas teegi järgneva käsklusega: **import** pandas **as** pd.

2.4 Kõrgkeele ja madalkeele (masinkoodi) võrdlus

Programmeerimises loetakse kõrgkeelteks neid keeli, mis on inimeste jaoks lihtsasti mõistetavad oma struktuuri poolest nagu näiteks: C, C++, Java, Python ning madalkeeleks loetakse masinkoodi, mis kasutab binaarsüsteemi ehk kahendsüsteemi ja heksadetsimaalsüsteem ehk Kuueteistkümnendsüsteem, seda kasutatakse riistvara spetsiifilistes süsteemides. [10]

Tabel 1. Kõrgkeel vs masinkood [10]

Kõrgkeel	**Masinkood**
Vähese RAM mälu kasutusega.	Suure RAM mälu kasutusega
Kergesti arusaadav.	Raskesti arusaadav.
Vigade kõrvaldamine ehk silumine (*debugging)* on lihtne.	Vigade kõrvaldamine ehk silumine (*debugging*) on keeruline.
Ülalpidamine on lihtne.	Ülalpidamine on keerukas.
On ülekantav (*portable*)	Ei ole ülekantav (*portable*)
Töötab ükskõik, millise operatsioonisüsteemi platvormil.	On sõltuv masina süsteemidest.
Vajab masinale tõlkeks kompilaatorit.	Vajab komplekteerijat (*assembler*) tõlkeks.
Kasutatakse laialdaselt programmeerimiseks.	Ei kasutata tavaliselt programmeerimiseks tänapäeval.

Näited programmist, mis lisab kaks numbrit.

```
print(20 + 50)
```

Python [11]

```
MOV AX, 20
MOV BX, 50
ADD AX, BX
HLT
```

Assemblerkeel [11]

3. AUTOMAATKONTROLLI PROGRAMM

Tänapäeva ühiskond liigub järjest suurema automatiseerimise suunas, mis võimaldab inimestel keskenduda keerulisematele ülesannetele väiksema ajakuluga. Aastaks 2040 prognoositakse keskmise inimese töötundide arvuks päevas 3,8. Ülejäänud ajast töötavad automatiseeritud programmid ning robotid, kes on suudavad paremini ja kauem töötada ilma kvaliteeti langetamata. [18]

Antud lõputöö eesmärgiks on mobiilrobootika kursuse efektiivsemaks muutmine tänu uue õppevahendi loomisele, mis aitaks tudengitel automaatselt kontrollida tarkvarakoodi sobivust ROS (Roboti operatsioonisüsteemi) platvormi jaoks ning saada kohest tagasisidet Github Classroom keskkonnast sooritusele.

3.1 ROS algkursus

Õppeaines tutvutakse mobiilrobootikale omaste positsioneerimis- ning juhtimise vahenditega, kasutades ROS tarkvara. Projekt ülesandena luuakse sõitev autonoomne liikur. Tutvutakse autonoomsete sõidukite olemuse ja rakendusvaldkondadega, erinevate platvormidega ning tehnoloogiliste lahendustega. Vaadeldakse mobiilrobootikast tulenevaid nõudeid taristule erinevates keskkondades: avalik ruum, liiklus, kinnine territoorium, tööstus, sise- ja välistingimused. [12]

Kursus kestab ühe semestri ehk 16 nädalat, mille jooksul on kavas 8 erinevat ROS-iga seotud harjutust, mis on jagatud kahe nädalasteks mooduliteks, kus tudengid tutvuvad ROS-i, Linuxi operatsioonisüsteemi, *micropythoni*, mikrokontrollerite ning erinevate roboti spetsiifiliste ülesannetega, mida kontrollitakse automaatse programmiga.

Ajakava:

1./2.nädal – Sissejuhatus ROS / Linux paigaldamine /Pythoni ja Linux CLI harjutused

3./4.nädal – Robotplatvorm – andurid / täiturid (MicroPython programmeerimine)

5./6.nädal – ROS installeerimine + ROS Hello World + Turtlesim / Rqt (demo)

7./8.nädal – Mikrokontrolleri ja serveri vahelise suhtluse seadistamine (TCP / Serial)

9./10.nädal – Roboti kinemaatiline mudel (URDF) + andurite / täiturite sidumine ROS-i

11./12.nädal – Navigeerimine (SLAM, obstacle avoidance, navigation)

13./14.nädal – Mitme roboti koostöö

15./16.nädal – Kokkuvõte / lõpuprojektide demo

3.2 Mikrokontroller

Iga roboti keskmes on mikrokontroller. Tavaliselt on see on mikrokiip, milles on peale protsessori veel komponente, näiteks mälu ning see vastutab koodi käitamise eest, et robot saaks oma ülesandeid täita. [13, lk 4]

Enamkasutatavad mikrokontrollerid on Arduino Uno ja Raspberry Pi Pico, mis võivad välja näha sarnased plaatarvutitega (SBC), näiteks Raspberry Pi 4 või BeagleBone, kuid erinevad salvestusruumi, protsessori kiiruse, tarkvara keerukuse ja riistvara poolest. [13, lk 6]

Joonis 2. Raspberry Pi Pico [14]

Joonis 3. Arduino Uno Rev3 [15]

Kuigi Raspberry Pi Pico sobib suurepäraselt näiteks robotite juhtimiseks, kuid ei sobi suure mälumahu või protsessoriga ülesannete jaoks, nagu AI või visuaalne tuvastamine. [13, lk 6]

Tabel 2. Arduino Uno võrdlus Raspberry Pi Pico-ga [13, lk 7]

Spetsifikatsioon	**Arduino Uno**	**Raspberry Pi Pico**
Digitaalne IO sisend/väljund	14	26
Analoog IO sisend/väljund	6	4
Protsessor	Ühe tuumaline ATmega328 16MHz	Kahetuumaline RP2040 133MHz
Välkmälu (Flash)	32 KB (plus 1 KB EPROM)	2 MB
RAM	2 KB	264 KB

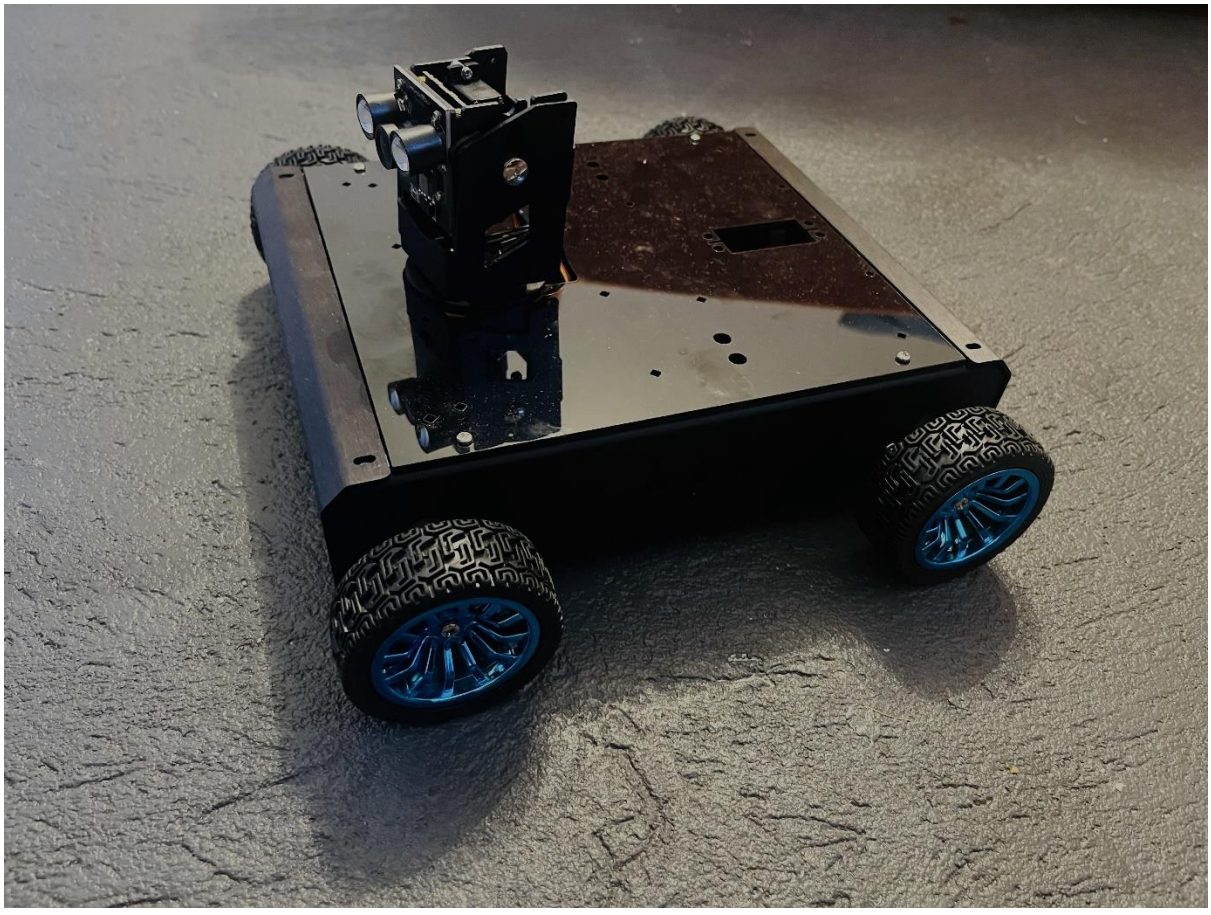
Eelnev tabel näitab, et Raspberry Pi Picol on kiirem mitme tuumaline protsessor koos rohkema salvestusruumi ja digitaalsete IO sisend/väljund kontaktidega. Lisaks on Raspberry Pi Picol ainulaadne programmeeritav IO (PIO) süsteem, mis tagab äärmise paindlikkuse andmete kontaktidesse ja sealt välja suunamisel. Ametlikud pico plaadid on samuti odavamad kui ametlikud Arduino omad. [13, lk 7]

3.3 Kasutatav platvorm

Mobiilrobootika kursuse raames kasutatakse roboteid, mille pardal on Raspberry Pi Pico mikrokontroller, mis juhib robotit ROS tarkvaraga. Koodid kirjutatakse Pythonis ning ROS-iga suheldakse läbi Linux operatsioonisüsteemi käsurea (CLI).

Linuxi nagu Windows, iOS ja Mac OS on operatsioonisüsteem, mis on tarkvara riistvara ressursside haldamiseks. Lihtsamalt öeldes haldab operatsioonisüsteem tarkvara ja riistvara vahelist suhtlust ning ilma operatsioonisüsteemita (OS) tarkvara ei töötaks. [16]

Linuxil on mitmeid erinevaid versioone, mis sobivad igat tüüpi kasutajale. Raspberry Pi kasutab linux operatsioonisüsteemi versiooni Raspbian. ROS-ile soovitatakse Pi jaoks alla laadida Ubuntu versioon. ROS-pakettide installimine ja nende haldamine Raspbianis on üsna keeruline, samas kui Ubuntu puhul töötab samamoodi nagu tavalises arvutis või sülearvutis. [17]



Joonis 4. Roboti prototüüp

3.4 Moodle

Moodle on avatud lähtekoodiga õpikeskkond haridusasutustele, eeskätt üldhariduskoolidele, kutseõppe asutustele ja kõrgkoolidele. Keskkonda on sisse ehitatud hulk erinevaid e-õppe vahendeid, nii õppesisu (õppematerjalid, harjutused, testid) kui ka õppeprotsesside (juhendamine, hindamine, tagasiside, arutelud, kodutööd, rühmatööd) haldamiseks. [20]

Tallinna Tehnikakõrgkooli moodle on leitav veebilehelt: https://moodle.tktk.ee/.

3.5 Tööpõhimõte

Õppekeskkond Moodle võimaldab lisada kursustele erinevaid funktsionaalsusi pakkuvaid vahendeid, mis muudavad õppeprotsessi interaktiivsemaks. Üks nendest vahenditest on väline tööriist, mille kaudu ühendadatakse Github Classroom Moodle keskkonnaga eesmärgiga pakkuda tudengitele võimalust tarkvaralise koodi automaatseks kontrolliks antud platvormil.

Pärast ülesande vastuvõtmist Github Classroomis käivitatakse automaatsed testid iga üleslaadimise korral tehtud muudatuse järel. Kasutaja saab näha testide tulemusi ja vajadusel muudatusi teha. Erinevaid testimise võimalusi kasutades saab tagada ülesande nõuetele vastavuse. GitHub Classroomi kaudu on võimalik jälgida osalejate testide läbimist. Roheline märk näitab, et kõik testid on läbitud, punane X viitab probleemidele. Võimalusel kuvatakse ka punktisumma ülesande täitmise eest. [19]

Hindamismeetodid hõlmavad kahte tüüpi teste: sisend/väljunditestid ja käsu käivitamise testid. Sisend/väljundi testid võimaldavad käivitada seadistus käsu ja anda standardse sisendi testikäsu jaoks, mille GitHub Classroom hindab oodatud tulemusega võrreldes. Käsu käivitamise testid aga käivitavad esmalt seadistus käsu ja seejärel testikäsu ning GitHub Classroom kontrollib testikäsu väljumise olekut, kus väljumiskood 0 näitab edukust ja muud koodid viitavad ebaõnnestumisele. [19]

3.6 GitHub classroom

GitHub Classroom on õppetööriist, mis võimaldab õpetajatel ja kooli administraatoritel luua ning hallata digitaalseid klassiruume ja ülesandeid. Antud keskkond võimaldab luua ülesandeid üksikutele õpilastele või õpilaste rühmadele, määrata tähtaegu ja jälgida ülesannete edenemist. Lisaks on GitHub Classroomil palju funktsioone, mis lihtsustavad selliseid toiminguid nagu tagasiside andmine, ülesannete hindamine ja olemasolevate õppevahendite integreerimine. [21]

Pakub erinevaid funktsioone nii õpetamise kui õppimise lihtsustamiseks [21]:

- **Automaatne kontroll**
  GitHub Classroom kaudu saab konfigureerida teste sel viisil, et iga kord kui ülesanne esitatakse toimub automaatne hindamine.
- **Harjutuste kujundused**
  Luuakse ülesandeid koos standardkoodi, dokumentatsiooni ja muude ressurssidega, mis hõlbustavad õppimist.
- **Ühendus LMS ehk õppehaldussüsteemiga**
  Võimalus ühendada virtuaalne klassiruum õpihaldussüsteemide nagu moodle, canvas, sakai, google classroom, et importida õpilaste nimekirjad.
- **Tagasiside päringud**
  Antud funktsioon võimaldab lisada ülesandele spetsiaalse tõmbe päringu, mis võimaldab õppejõududel jätta kommentaare ja tagasisidet koodi kohta.
- **Integreeritud programmeerimiskeskkonnaga (IDE) kombinatsioon**
  Võimaldab konfigureerida ülesannet kasutama integreeritud programmeerimiskeskkonda (IDE). IDE-d nagu Visual Studio Code lubab õpilastel koodi kirjutada, programme käivitada ja omavahel koostööd teha ilma Git allalaadimiseta.

3.7 Seadistamine

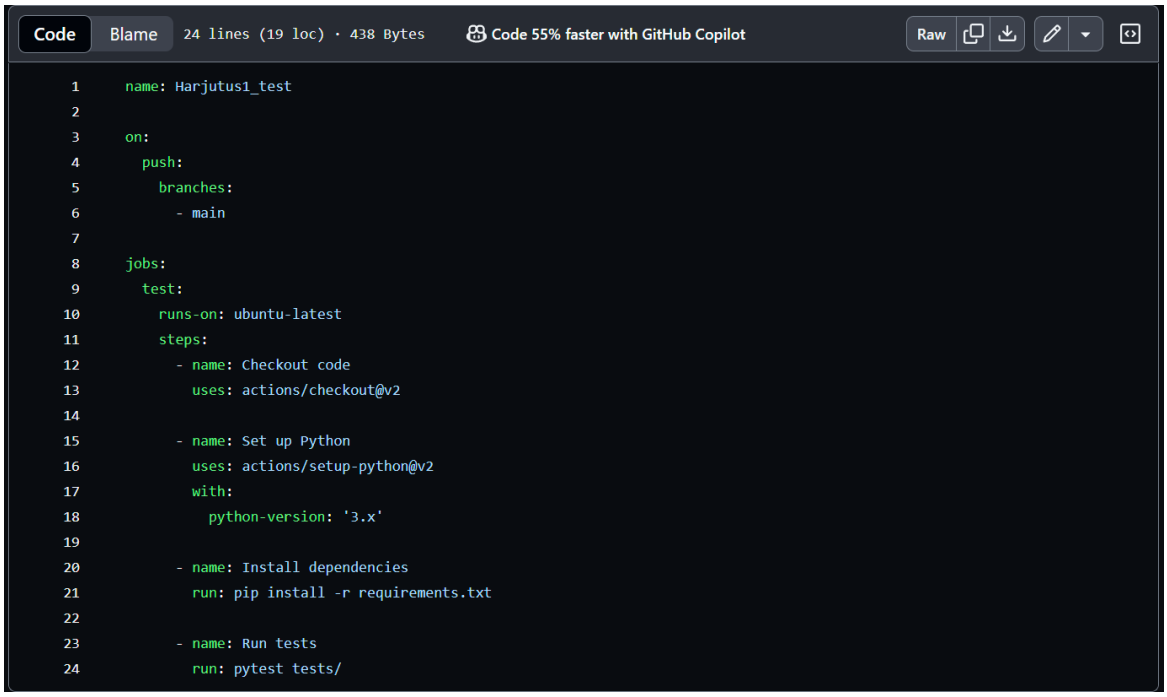
Seadistamise juhend Github Classroom automaatse kontrolli programmi loojale.

GitHub Classroom'i ülesande seadistamine [19]:

1. Mine veebilehele GitHub Classroom ja logi sisse GitHubi kontoga.
2. Loo klassiruum ja ülesanne, valides "New Classroom" järgides juhiseid.
3. Loo uus ülesanne, valides "New Assignment".
   Anna ülesandele nimi ja kirjeldus, ning lisa vajadusel juhised.
4. Lisa vajalikud failid ülesandele, näiteks Pythoni skripti fail ja testimise skriptid.

Automaatse hindamise seadistamine [19]:

1. Looge testimise skriptid: Kirjutage Pythoni skriptid, mis automaatselt testivad tudengite poolt esitatud lahendusi. Näiteks võiksid testid kontrollida, kas õige tervitussõnum on väljastatud, kui sisestatakse kindel nimi ja vanus.
2. Paigaldage vajalikud sõltuvused: Veenduge, et testimise skriptid saaksid edukalt käivitada kõik vajalikud sõltuvused. See võib hõlmata Pythoni moodulite paigaldamist või muude vajalike tarkvarakomponentide installimist.
3. Seadistage automaatne hindamine: Lisage ülesandele .github/workflows kaust ja looge sinna GitHub Actions'i töövoogudesse vastav YAML-fail, mis käivitab testid iga kord, kui tudengid oma lahendused GitHubi esitavad. Töövoo struktuuri ülesehitus:




```yaml
name: Harjutus1_test

on:
  push:
    branches:
      - main

jobs:
  test:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Set up Python
        uses: actions/setup-python@v2
        with:
          python-version: '3.x'

      - name: Install dependencies
        run: pip install -r requirements.txt

      - name: Run tests
        run: pytest tests/
```

   Joonis 5. Töövoo struktuur
4. Seadistage testid: Looge testid vastavalt ülesande nõuetele.

3.8 Kasutamise juhend

Juhend Github Classroom automaatse kontrolli kasutamiseks.

Ülesande vastuvõtmine:

1. Õppejõud jagab ülesande linki, mille kaudu saab GitHub Classroom klassiruumi.
2. Klikkige lingil ja järgige juhiseid kopeerimaks ülesanne oma GitHubi kontole.

Ülesande lahendamine:

1. Ava ülesanne GitHubi repositooriumis.
2. Leia ülesanne ja selle juhised README.md failis.
3. Looge või muutke vajadusel Pythoni skripti vastavalt ülesandele.
4. Kui vajalik lisage skriptile testid kontrollimaks kas see töötab õigesti.

Esitamine ja automaatne hindamine:

1. Valmis lahenduse esitage GitHubi repositooriumisse.
2. Lahenduse esitamisel käivitatakse automaatselt testid GitHub Actions'i abil.
3. Oodake kuni testid lõpule jõuavad.
4. Kontrollige testide tulemusi ja veenduge, et kõik testid oleksid edukad.
5. Kui kõik testid on edukad, on teie lahendus õige ja vastab nõuetele.
6. Kui testid ebaõnnestuvad vaadake testide väljundit nägemaks, millised vead on ilmnenud.
7. Parandage koodi vastavalt testide väljundile ja esitage uuesti.
   Vajadusel korrake seda protsessi kuni kõik testid on edukad.

Tagasiside saamine:

1. Kui testid on edukad saadakse automaatset tagasisidet testide väljundist.
2. Kui on küsimusi või vajate abi pöörduge õppejõu poole.

4. AUTOMAATKONTROLLI PROGRAMMI HARJUTUSED

4.1 Harjutus 1

Sissejuhatus ROS / Linux paigaldamine /Pythoni ja Linux CLI harjutused.

Programm, mis küsib kasutajalt nime ja vanust ning väljastab tervitussõnumi:

Harjutus1.py

```python
def main():
    # Küsime kasutajalt nime ja vanust
    name = input("Sisestage oma nimi: ")
    age = input("Sisestage oma vanus: ")

    # Konverteerime vanuse stringist arvuks
    age = int(age)

    # Väljastame tervitussõnumi, kasutades nime ja vanust
    print(f"Tere, {name}! Teie olete {age} aastat vana.")

if __name__ == "__main__":
    main()
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\kertk\ROS harjutused> & C:/Users/kertk/anaconda3/python.exe
 "c:/Users/kertk/ROS harjutused/Harjutus1.py"
Sisestage oma nimi: Kert
Sisestage oma vanus: 25
Tere, Kert! Teie olete 25 aastat vana.
PS C:\Users\kertk\ROS harjutused>
```

Joonis 6. Kuvatõmmis Harjutus 1-st

4.2 Harjutus 2

Robotplatvorm – andurid / täiturid (MicroPython programmeerimine)

Programm, mis kasutab ultraheliandurit, et mõõta kaugust ja väljastab selle ekraanile:

Harjutus2.py

```python
import utime  # type: ignore

# Ultraheli anduri pinnid
trigger_pin = Pin(2, Pin.OUT) # type: ignore
echo_pin = Pin(3, Pin.IN) # type: ignore

def measure_distance():
    # Saada lühike triggerimpulss
    trigger_pin.off()
    utime.sleep_us(2)
    trigger_pin.on()
    utime.sleep_us(5)
    trigger_pin.off()

    # Mõõda kui kaua võtab aega ehho pulss tagasi jõuda
    pulse_time = time_pulse_us(echo_pin, 1) # type: ignore

    # Kauguse arvutamine mikrosekundites
    distance = pulse_time * 0.0343 / 2  # Kaugus cm-des

    return distance

while True:
    distance = measure_distance()
    print("Distance: {:.2f} cm".format(distance))
    utime.sleep(1)

```

Joonis 7. Kuvatõmmis Harjutus 2-st

4.3 Harjutus 3

ROS installeerimine + ROS Hello World + Turtlesim / Rqt (demo)

Programm "Hello World" ROS-i Pythonis, mis kasutab rospy (ROS Pythoni teegi) paketti:

Harjutus3.py

```python

import rospy # type: ignore

def main():
    rospy.init_node('hello_node')
    rospy.loginfo("Hello, ROS!")
    rospy.spin()

if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass

```

Joonis 8. Kuvatõmmis Harjutus 3-st

4.4 Harjutus 4

Mikrokontrolleri ja serveri vahelise suhtluse seadistamine (TCP / Serial).

Programm Pythoni kood Raspberry Pi Pico ja serveri vahelise TCP/IP ühenduse loomiseks ja andmete saatmiseks kasutades socket moodulit:

Raspberry Pi Pico kood (MicroPython):

Harjutus4.py

```python
import socket
import network  # type: ignore

# Wifi ühenduse seadistamine
wifi = network.WLAN(network.STA_IF)
wifi.active(True)
wifi.connect('SSID', 'parool')

# TCP/IP ühenduse loomine
server_ip = 'serveri_ip_aadress'
server_port = 12345
sock = socket.socket()
sock.connect((server_ip, server_port))

# Andmete saatmine
data_to_send = b"Tere server!"
sock.send(data_to_send)

# Vastuse saamine serverist
received_data = sock.recv(1024)
print("Serverilt saadud vastus:", received_data)

# Ühenduse sulgemine
sock.close()

```

Joonis 9. Kuvatõmmis Harjutus 4-st

4.5 Harjutus 5

Roboti kinemaatiline mudel (URDF) + andurite / täiturite sidumine ROS-i

Programm ROS-i sõlme jaoks, mis demonstreerib juhtimist ja andurite lugemist URDF-i mudeliga:

Harjutus5.py

```python

import rospy # type: ignore
from sensor_msgs.msg import LaserScan # type: ignore
from geometry_msgs.msg import Twist # type: ignore

class RobotController:
    def __init__(self):
        rospy.init_node('robot_controller', anonymous=True)
        rospy.Subscriber('/scan', LaserScan, self.scan_callback)
        self.pub_cmd_vel = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
        self.rate = rospy.Rate(10)  # 10 Hz

    def scan_callback(self, data):
        # Pöörata robotit vastavalt esimese anduri nägemisele
        min_distance = min(data.ranges)
        if min_distance < 1.0:  # Kui objekt on liiga lähedal
            vel_cmd = Twist()
            vel_cmd.angular.z = 0.5  # Pööra 0.5 rad/s
            self.pub_cmd_vel.publish(vel_cmd)
        else:
            vel_cmd = Twist()
            vel_cmd.linear.x = 0.2  # Liigu edasi 0.2 m/s
            self.pub_cmd_vel.publish(vel_cmd)

    def run(self):
        rospy.spin()

if __name__ == '__main__':
    controller = RobotController()
    controller.run()

```

Joonis 10. Kuvatõmmis Harjutus 5-st

4.6 Harjutus 6

Navigeerimine (SLAM, obstacle avoidance, navigation)

Programm Raspberry Pi-ga ühendatud ultrahelianduri abil takistuste tuvastamiseks ja navigeerimiseks:

Harjutus6.py

```python
import RPi.GPIO as GPIO # type: ignore
import time
import numpy as np

# Määrake ultrahelianduri pinnid
TRIG_PIN = 23
ECHO_PIN = 24

# Funktsioon takistuse kauguse mõõtmiseks ultrahelianduriga
def distance_measurement():
    GPIO.output(TRIG_PIN, True)
    time.sleep(0.00001)
    GPIO.output(TRIG_PIN, False)

    start_time = time.time()
    stop_time = time.time()

    while GPIO.input(ECHO_PIN) == 0:
        start_time = time.time()

    while GPIO.input(ECHO_PIN) == 1:
        stop_time = time.time()

    elapsed_time = stop_time - start_time
    distance = (elapsed_time * 34300) / 2  # Kauguse arvutamine (kiirus * aeg)
    return distance

# Initsialiseerige GPIO pinnid
GPIO.setmode(GPIO.BCM)
GPIO.setup(TRIG_PIN, GPIO.OUT)
GPIO.setup(ECHO_PIN, GPIO.IN)

try:
    while True:
        # Mõõdame kaugust
        dist = distance_measurement()
        print("Distance:", dist, "cm")

        # Simuleerime navigeerimist, kui takistus on lähedal (nt alla 30 cm)
        if dist < 30:
            print("Takistus lähedal, pöörame...")
        else:
            print("Puuduvad takistused, liigume edasi.")

        time.sleep(0.5)  # Ootame enne järgmist mõõtmist

except KeyboardInterrupt:
    GPIO.cleanup()

```

Joonis 11. Kuvatõmmis Harjutus 6-st

4.7 Harjutus 7

Mitme roboti koostöö

Programm, kus luuakse kaks mobiilse platvormi robotit (robot1 ja robot2), mis liiguvad mõlemad ettepoole kiirusega 0,5 m/s ning seejärel peatuvad pärast 5 sekundit:

Harjutus7.py

Harjutus7.py > ...

```python

import rospy # type: ignore
from geometry_msgs.msg import Twist # type: ignore
from nav_msgs.msg import Odometry # type: ignore

class RobotController:
    def __init__(self, robot_name):
        self.robot_name = robot_name
        self.cmd_vel_pub = rospy.Publisher('/' + robot_name + '/cmd_vel', Twist, queue_size=10)
        rospy.Subscriber('/' + robot_name + '/odom', Odometry, self.odom_callback)
        self.velocity = Twist()

    def odom_callback(self, msg):
        # Sõlme reaktsioon odomeetri andmetele
        pass

    def move_forward(self, speed):
        # Liikumine ettepoole
        self.velocity.linear.x = speed
        self.cmd_vel_pub.publish(self.velocity)

    def stop(self):
        # Peatamine
        self.velocity.linear.x = 0
        self.velocity.angular.z = 0
        self.cmd_vel_pub.publish(self.velocity)

def main():
    rospy.init_node('multi_robot_cooperation')

    # Loome kaks robotit
    robot1 = RobotController('robot1')
    robot2 = RobotController('robot2')

    # Mõlemad robotid liiguvad ettepoole
    robot1.move_forward(0.5)
    robot2.move_forward(0.5)

    # Ootame 5 sekundit
    rospy.sleep(5)

    # Peatame mõlemad robotid
    robot1.stop()
    robot2.stop()

    rospy.spin()

if __name__ == '__main__':
    main()

```

Joonis 12. Kuvatõmmis Harjutus 7-st

4.8 Harjutus 8

Kokkuvõte / lõpuprojektide demo.

Programm, mis loob ROS-i sõlme, et kontrollida roboti liikumist edasi-tagasi:

Harjutus8.py

```python

import rospy # type: ignore
from geometry_msgs.msg import Twist # type: ignore

class MovementNode:
    def __init__(self):
        rospy.init_node('movement_node', anonymous=True)
        rospy.on_shutdown(self.shutdown)

        # Väljundpublitseerija, et saata kiiruskäsk roboti liigutamiseks
        self.cmd_vel_pub = rospy.Publisher('cmd_vel', Twist, queue_size=10)

    def move_forward(self, linear_speed):
        # Looge Twist-sõnum ja määrake selle lineaarse kiiruse komponent
        twist = Twist()
        twist.linear.x = linear_speed
        twist.angular.z = 0.0

        # Avaldage kiiruse käsk
        self.cmd_vel_pub.publish(twist)

    def move_backward(self, linear_speed):
        # Tagurpidi liikumine
        twist = Twist()
        twist.linear.x = -linear_speed
        twist.angular.z = 0.0
        self.cmd_vel_pub.publish(twist)

    def shutdown(self):
        # Peatage robot
        rospy.loginfo("Stopping the robot...")
        self.cmd_vel_pub.publish(Twist())

if __name__ == '__main__':
    try:
        movement_node = MovementNode()
        # Liigutage edasi kiirusega 0.2 m/s
        movement_node.move_forward(0.2)
        rospy.sleep(2)  # Oodake 2 sekundit
        # Liigutage tagasi kiirusega 0.1 m/s
        movement_node.move_backward(0.1)
        rospy.sleep(2)  # Oodake 2 sekundit
    except rospy.ROSInterruptException:
        pass

```

Joonis 13. Kuvatõmmis Harjutus 8-st

KOKKUVÕTE

Automatiseerimise olulisus tänapäeva ühiskonnas on vaieldamatu, kuna see võimaldab efektiivsemat tööd, suuremat täpsust ning hoida aega kokku. Kiirelt areneva tehnoloogia ajastul on automatiseerimine saanud peamiseks viisiks optimeerida erinevaid protsesse, sealhulgas haridusvaldkonnas.

Antud lõputöös keskenduti just sellise automaatse kontrolli programmi väljatöötamisele, mis võimaldaks hõlpsasti hinnata tudengite poolt loodud programmeerimisülesandeid mobiilrobootika kursusel, mille aluseks on ROS platvorm. ROS (Roboti operatsioonisüsteem) platvormi kasutamine programmeerimisülesannete alusena pakub mitmekesiseid võimalusi tudengitele praktiliste oskuste arendamiseks ja sügavamaks arusaamiseks roboteid ning autonoomseid süsteeme puudutavatest teemadest. Selline lähenemine mitte ainult ei toeta tudengite õppimist, vaid valmistab neid ka ette tuleviku väljakutseteks, kus robootika ja autonoomsete süsteemide oskuste nõudlus on üha kasvav.

Moodle õppekeskkond võimaldab integreerida erinevaid vahendeid, mis rikastavad õppeprotsessi interaktiivsusega. Üks neist vahenditest on Github Classroom, mis võimaldab kasutajatel tarkvarakoodi automaatset kontrolli. Pärast ülesande esitamist Github Classroomis käivitatakse automaatsed testid iga muudatuse järel, võimaldades kasutajal näha testitulemusi ja vajadusel parandusi teha. Erinevaid testimise võimalusi kasutades saab tagada ülesande nõuetele vastavuse ning GitHub Classroom võimaldab jälgida osalejate testide edukust. Hindamise meetodid hõlmavad sisend/väljunditestide ja käsu käivitamise teste, mis võimaldavad kontrollida ülesande täitmist vastavalt standardsetele kriteeriumitele.

Lõputöö eesmärgiks oli muuta mobiilrobootika kursus efektiivsemaks uue õppevahendi loomise näol ja pakkuda tudengitele võimalust omandada praktilisi oskusi ROS platvormi baasil, tagades samal ajal kiire ja objektiivse tagasiside nende tehtud tööle ning tõsta õppimise kvaliteeti tõsta, pakkudes paremat tuge ja juhendamist nende õppeprotsessis.

SUMMARY

Automated control program for educational programming assignments

The importance of automation in today's society is undeniable, as it enables more efficient work, greater accuracy, and time saving. In the rapidly evolving technological era, automation has become the primary means of optimizing various processes, including in the field of education.

The focus of this thesis was on developing an automatic control program that would easily assess programming assignments created by students in the mobile robotics course, based on the ROS platform. Using ROS (Robot Operating System) as the basis for programming assignments offers students diverse opportunities to develop practical skills and gain a deeper understanding of topics related to robots and autonomous systems. Such an approach not only supports student learning but also prepares them for future challenges, where the demand for robotics and autonomous systems skills is constantly growing.

The Moodle learning environment allows for the integration of various tools that enrich the learning process with interactivity. One of these tools is Github Classroom, which enables students to automatically check software code. After submitting an assignment in Github Classroom, automatic tests are run after each change, allowing users to see the test results and make corrections if necessary. By using various testing options, compliance with assignment requirements can be ensured, and Github Classroom allows tracking of participants' test success. Assessment methods include input/output tests and command execution tests, which allow checking the completion of assignments according to standard criteria.

The aim of the thesis was to make the mobile robotics course more efficient through the creation of a new educational tool and to provide students with the opportunity to acquire practical skills based on the ROS platform, while ensuring quick and objective feedback on their work. Additionally, the goal was to improve the quality of learning by offering better support and guidance throughout their learning process.

VIIDATUD ALLIKAD

[1] ROS ametlik veebileht [Võrgumaterjal]: https://www.ros.org/ [Kasutatud 28.02.2024]

[2] ROS koolitus [Võrgumaterjal]: https://sisu.ut.ee/ros/koolitus?lang=et [Kasutatud 28.02.2024]

[3] Wikipedia ROS [Võrgumaterjal]: https://wiki.ros.org/ [Kasutatud 28.02.2024]

[4] ROS koolituse materjalid [Võrgumaterjal]: https://ut-ims-robotics.github.io/ros_koolitus/html/index.html [Kasutatud 01.03.2024]

[5] ROS *intro* [Võrgumaterjal]: https://web.fs.uni-lj.si/lampa/rosin/ROS%20Summer%20School/Day%201/ros_intro/ [Kasutatud 01.03.2024]

[6] Medium [Võrgumaterjal]: https://medium.com/teamarimac/integrating-sonar-and-ir-sensor-plugin-to-robot-model-in-gazebo-with-ros-656fd9452607 [Kasutatud 03.03.2024]

[7] Programiz [Võrgumaterjal]: https://programiz.pro/resources/everything-you-need-to-know-about-python/ [Kasutatud 05.03.2024]

[8] Wikipedia [Võrgumaterjal]: https://et.wikipedia.org/wiki/Integreeritud_programmeerimiskeskkond [Kasutatud 05.03.2024]

[9] GeeksForGeeks [Võrgumaterjal]: https://www.geeksforgeeks.org/libraries-in-python/ [Kasutatud 06.03.2024]

[10] GeeksForGeeks [Võrgumaterjal]: https://www.geeksforgeeks.org/difference-between-high-level-and-low-level-languages/ [Kasutatud 09.03.2024]

[11] Programiz [Võrgumaterjal]: https://programiz.pro/resources/high-level-vs-low-level/ [Kasutatud 09.03.2024]

[12] Õppeinfosüsteem Tahvel [Võrgumaterjal]: https://tahvel.edu.ee/#/subject/32391 [Kasutatud 14.03.2024]

[13] Danny Staple, „Robotics at Home with Raspberry Pi Pico", Packt Publishing 2023. [Kasutatud 15.03.2024]

[14] Raspberry Pi [Võrgumaterjal]: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html [Kasutatud 15.03.2024]

[15] Arduino e-pood [Võrgumaterjal]: https://store.arduino.cc/products/arduino-uno-rev3 [Kasutatud 15.03.2024]

[16] Linux.com [Võrgumaterjal]: https://www.linux.com/what-is-linux/ [Kasutatud 25.03.2024]

[17] The Robotics Back-End [Võrgumaterjal]: https://roboticsbackend.com/using-ros-on-raspberry-pi-best-practices/ [Kasutatud 18.03.2024]

[18] Impact [Võrgumaterjal]:https://www.impactmybiz.com/blog/advantages-of-process-automation/ [Kasutatud 23.03.2024]

[19] GitHub docs [Võrgumaterjal]: https://docs.github.com/en/education/manage-coursework-with-github-classroom/teach-with-github-classroom/use-autograding [Kasutatud 20.04.2024]

[20] HTM moodle [Võrgumaterjal]: https://moodle.edu.ee/ [Kasutatud 20.03.2024]

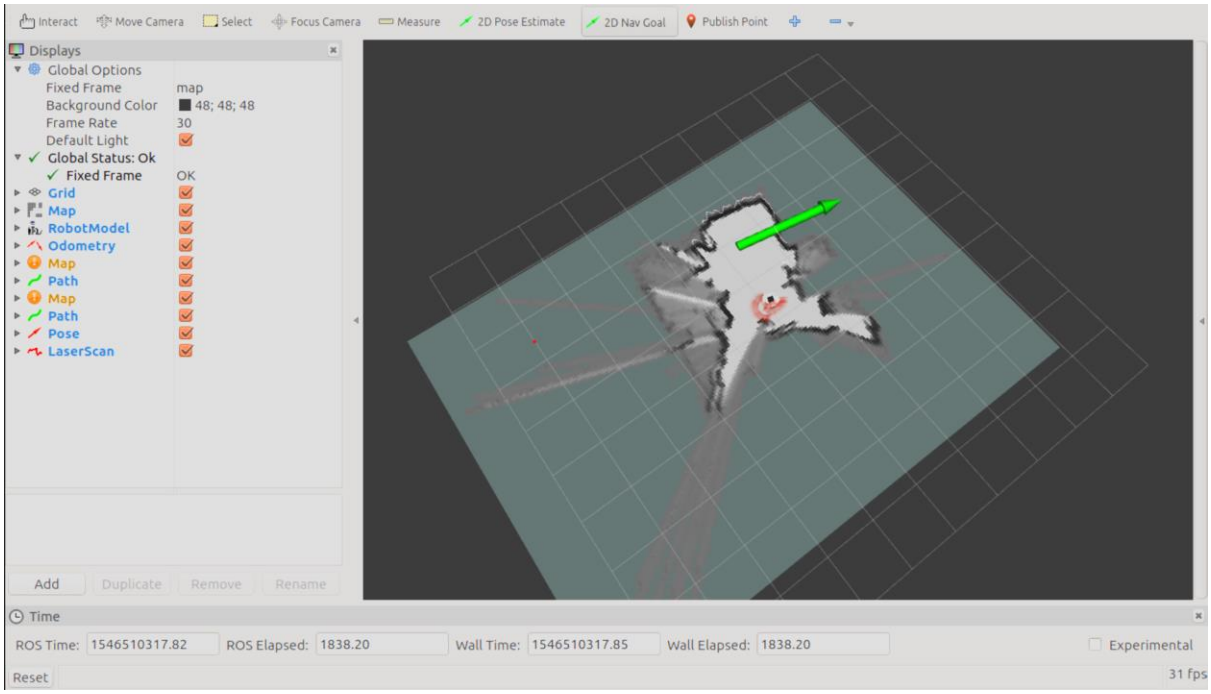
[21] GitHub classroom [Võrgumaterjal]: https://docs.github.com/en/education/manage-coursework-with-github-classroom/get-started-with-github-classroom/about-github-classroom [Kasutatud 21.03.2024]

LISAD

Lisa 1. Kuvatõmmis *Rviz* keskkonnast

Lisa 2. Kuvatõmmis *Gazebo* keskkonnast

Lisa 1. Kuvatõmmis Rviz keskkonnast



Lisa 1. Kuvatõmmis Rviz keskkonnast [5]

Lisa 2. Kuvatõmmis Gazebo keskkonnast

Lisa 2. Kuvatõmmis Gazebo keskkonnast [6]