TALLINNA TEHNIKAÜLIKOOL
INSENERITEADUSKOND
Virumaa kolledž

Kuue vabadusastmega roboti kaugjuhtimine digitaalse kaksiku abil

Remote control of a robot with six degrees of freedom using a digital twin

TELEMAATIKA JA ARUKAD SÜSTEEMID ÕPPEKAVA LÕPUTÖÖ

Üliõpilane: Dmitry Matveev

Üliõpilaskood: 182713EDTR

Juhendaja: Karle Nutonen,
Doktorant-nooremteadur

Kohtla-Järve, 2022

AUTORIDEKLARATSIOON

Olen koostanud lõputöö iseseisvalt.

Lõputöö alusel ei ole varem kutse- või teaduskraadi või inseneridiplomit taotletud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on viidatud.

"...." ..... 20.....

Autor: .....
/ allkiri /

Töö vastab rakenduskõrgharidusõppe lõputööle/magistritööle esitatud nõuetele
"...." ..... 20.....

Juhendaja: .....
/ allkiri /

Kaitsmisele lubatud
"...." ..... 20.....

Kaitsmiskomisjoni esimees .....
/ nimi ja allkiri /

LIHTLITSENTS LÕPUTÖÖ ÜLDSUSELE KÄTTESAADAVAKS TEGEMISEKS JA REPRODUTSEERIMISEKS

Mina Dmitry Matveev (sünnikuupäev: 22.08.1992)

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose

„Kuue vabadusastmega roboti kaugjuhtimine digitaalse kaksiku abil", mille juhendaja on Karle Nutonen,

1.1. reprodutseerimiseks säilitamise ja elektroonilise avaldamise eesmärgil, sealhulgas 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 punktis 1 nimetatud õigused jäävad alles ka autorile.

3. Kinnitan, et lihtlitsentsi andmisega ei rikuta kolmandate isikute intellektuaalomandi ega isikuandmete kaitse seadusest ja teistest õigusaktidest tulenevaid õigusi.

TalTech Inseneriteaduskond Virumaa kolledž
LÕPUTÖÖ ÜLESANNE

Üliõpilane: Dmitry Matveev, 182713EDTR

Õppekava, peaeriala: EDTR17/18 Telemaatika ja arukad süsteemid, Protsesside automatiseerimine

Juhendaja: Doktorant-nooremteadur, Karle Nutonen, karle.nutonen@taltech.ee

Lõputöö teema:

(eesti keeles) Kuue vabadusastmega roboti kaugjuhtimine digitaalse kaksiku abil

(inglise keeles) Remote control of a robot with six degrees of freedom using a digital twin

Lõputöö põhieesmärgid:

1. Olemasolevate lahenduste analüüs
2. Simulatsioonikeskkonna valimine
3. Roboti komponentide valik
4. Simulatsioonikeskkonna loomine
5. Simulatsioonis esinevate vigade kontrollimine
6. Serveri loomine ja konfigureerimine kontrolleris

Lõputöö etapid ja ajakava:

Nr	Ülesande kirjeldus	Tähtaeg
1.	Erinevate lahenduste uurimine, katsetamine. Robotile varuosade tellimine. 3D mudeli koostamine SolidWorks'is.	01.02.2022-01.03.2022
2.	Lõputöö dokumendi esimene osa – sisukord, sissejuhatus. Programmeerimine Unity's	01.03.2022-01.04.2022
3.	Roboti kokkupanek ja testimine. Rakenduse ja roboti ühendamine üksteisega.	01.04.2022-01.05.2022
4.	Lõputöö dokumendi teine osa – sisulise osa täiendamine (valiku täpsem kirjeldamine, praktilise osa kirjeldamine). Lõputöö viimased parandused.	01.05.2022-01.06.2022

Töö keel: Eesti keel **Lõputöö esitamise tähtaeg:** "....."..... 20.....a

Üliõpilane: ..... "....."..... 20.....a
/allkiri/

Juhendaja: ..... "....."..... 20.....a
/allkiri/

Programmijuht: ..... "....."..... 20.....a
/allkiri/

SISUKORD

EESSÕNA

Selle projekti idee tekkis "Robottehnilised süüsteemid" kursusel õppides lõputöö ajal. Autorile on digikaksikute tehnoloogia huvi pakkunud juba pikka aega, mistõttu otsustati valid lõputööks see teema. Suurtes ettevõtetes arendatakse aktiivselt digitaalset kaksiktehnoloogiat, et hõlbustada füüsiliste toodete loomist. Oskused projekteerimisel, robotsüsteemi konstrueerimisel ja programmeerimisel on abiks kõigile, kes on huvitatud oma robotsüsteemide loomisest. Tänu sellele saab lõputööd kasutada tehnoloogia selgitamiseks ning õpilaste ja kooliõpilaste harimiseks.

Erilist tänu avaldan oma lõputöö juhendajale abi eest lahenduste leidmisel käesoleva töö kirjutamisel.

Võtmesõnad: digitaalne kaksik, kinemaatika, robot, Unity, kaugjuhtimine, diplomitöö.

LÜHENDITE JA TÄHISTE LOETELU

GPIO (general-purpose input/output) – mitmeotstarbeline sisend/väljund

EasyEDA – programm plokkskeemide joonistamiseks, nende simuleerimiseks ja tehtud plaadi tellimiseks

Visual Scripting – programmeerimise võimalus ilma koodi kirjutamiseta

PCB (printed circuit board) – trükkplaat

C# (C Sharp) – programmeerimiskeel

GND (ground) – maandus

IDE (Integrated development environment) – integreeritud programmeerimiskeskkond

STL (STereoLitoraafia) – failivorming stereolitograafia jaoks

FBX (FilmBoX) – failivorming digitaalkontentiks

SISSEJUHATUS

Robootika kiire areng ja tootmise üldine automatiseerimine on viimastel aastatel populaarsust kogumas. Kõik soovivad toota kvaliteetseid tooteid kiiresti, odavalt ning saavutada turukonkurentsis parem koht. Kahjuks läheb suure hulga töötajate värbamine ettevõttele palju maksma, et saavutada parem ja kvaliteetsem tootlikkus. Sel põhjusel püüab enamik suuri ettevõtteid asendada inimesi tehnoloogiaga, mis suudab pidevalt töötada, tehes kvalitatiivselt monotoonset tööd.

Käesoleva lõputöö eesmärgiks on arendada simulatsioonikeskkond füüsilise roboti tööprotsessi simuleerimiseks ja programmeerimiseks, kasutades digitaalset kaksikut. See võimaldab kirjutada konkreetse roboti jaoks juhtimisprogramme ning seejärel need üle kanda reaalsele robotile. Vastav lähenemine võimaldab katsetada erinevaid tööprotsessi lähenemisi, mis ei mõjuta otseselt füüsilise roboti reaalset tööd.

Eesmärgi saavutamiseks tuleb lahendada alljärgnevad ülesanded:

1. olemasolevate digitaalsel kaksikul põhinevate simulatsioonikeskkondade võrdlus ja analüüs;
2. füüsilise roboti ehitus;
3. simulatsioonikeskkonna ja füüsilise roboti vahelise suhtluskanali loomine;
4. digitaalse kaksiku modelleerimine;
5. simulatsioonikeskkonna loomine ja seadistamine.

Antud teema on aktuaalne, sest käsitleb valdkonda, millega puutuvad valusalt kokku nii Ida-Virumaa kui ka üldiselt Eesti tootmisettevõtted, kus rakendatakse robottehnilisi süsteeme. Valukohaks on robottehniliste teadmiste ja vastavate teadmistega spetsialistide puudus. Arendatav keskkonda võimaldab disainida tootmisprotsessi kiiremini, väiksemate kuludega ja mõista lõputöös rakendava roboti võimekust.

Lõputöö uuenduslikkus seisneb füüsilise roboti programmeerimises, mida teostatakse simulatsioonikeskkonnas digitaalse kaksiku baasil. Autori uuringutele tuginedes on tüüpiliste tööstusrobotite jaoks sarnaseid lahendusi turul rohkelt, aga pole rakendatud Stewart Gough platvormi roboti jaoks.

Vastav lõputöö koosneb kolmest peatükist. Esimeses peatükis võrreldakse ja selgitatakse valitud vahendeid. Teises peatükis selgitab autor oma robotivalikut, selle

ehitamise protsessi ja juhtimiskontrolleri loomist. Kolmandas peatükis demonstreerib autor digitaalse kaksiku simulatsiooni loomise protsessi ning probleeme, vigu ja nende lahendusi.

Käesoleva lõputöö tulemusena arendati Stewarti-Gough platvormil põhineva roboti simulatsioonikeskkonna prototüüp, kus on võimalik luua juhtimisprogramm, mida on võimalik üle kanda reaalsele robotile.

1 LÕPUTÖÖ KIRJELDUS JA TÖÖRIISTADE VALIK

Elame ajal, kus elutempo ning tehnoloogia areng on kiire, millega peame igapäevaste kohustuste keskel kohanema. Samade raskustega peavad tegelema ka tootmisettevõtted, et püsida konkurentsivõimelistena oma toodetega turul. Sellest tulenevalt on hakatud asendama liinitootmisel inimesi tehnoloogiaga, et parendada tootlikkust ja suurendada toote kvaliteeti. Eriti rakendatakse masinaid vastavates tööprotsesside, mis on kas rutiinsed või vajavad kiiret ja korrektset teostust. Inimesed võivad teha vigu väsimuse, halva tervise, tähelepanematuse jne tõttu, mida robottehniliste süsteemide rakendamisel ei pea kartma. Kahjuks on kõik masinate või seadmete programmid ja seadistused samuti inimese tehtud. Sellel võib olla programmile kahjulik mõju. Keegi ei tea, kas viga ilmub süsteemi kohe või alles pika aja pärast, kui tarkvara või riistvara parandamine võib võtta palju aega, kulu, raha.

Simulatsioonikeskkonnas on võimalik loodavat tööprotsessi programmi katsetada seadme digitaalsel koopial, kus on võimalik koheselt programmeerimisel tekkivaid vigu parandada, mis võimaldab efektiivsemalt kasutada füüsilise roboti reaalset töö aega.

Suurim projekt, milles kasutatakse digitaalset kaksiksüsteemi, on kosmosejaam „ISS" [1]. Äärmuslike tingimuste ja kosmosejaama kauguse tõttu võib igasugune viga jaama juhtimises maksma minna suure summa raha ja/või inimelusid. Et kõiki vigu saaks vältida, kasutab NASA ühte kõige täpsematest simulaatoritest. Testides tegureid alates temperatuurist ja kosmosejaama kiirusest kuni gravitatsiooni mõjuni lähedalasuvale kosmoseprahile, on võimalik ennetada enamikku võimalikke tulevasi probleeme ja leida lahendusi nende vältimiseks.

Digitaalset kaksikut kasutatakse pea kõikides tootmisvaldkondades, kus tuleb vähendada veapotentsiaali. Alates meditsiini ja elektrivõrgu katsetamisest kuni masinate projekteerimise ja katsetamiseni tee pidavuse ja õhuvoolutakistusteni välja.

Digitaalse kaksikuga töötamise programmide põhijooned:

a) simulatsiooni võimalus (liikumine / koormus / välistegurid);
b) võimalus edastada teavet füüsilisele objektile;
c) teabe edastamine füüsiliselt meediumilt simulatsioonile;
d) masinõppe / süvaõppe võimalus;
e) võrguühenduseta programmeerimine;
f) erinevate robotite mudelite ühendamine.

1.1 Simulatsioonikeskkonna valik

Järgnevalt on esitatud erinevate simulatsioonikeskkondade olemus.

1. Tööstusroboti tootja keskne virtuaalkeskkond

   Pea kõigil tööstusrobotite tootjatel on olemas omapoolsed simulatsioonikeskkonnad. Tabelis 1.1 on toodud keskkonnad, mis on loodud simuleerimaks ja programmeerimaks ainult vastava ettevõtte robotit (vt Tabel 1.1).

Tabel 1.1 Simulatsiooniprogrammide võimaluste võrdlus [2]

	Roboti tootja	Programmeerimis keel	Visuaalne programmeerimine	Hind
RobotStudio	ABB	RAPID	+	1500$ / aasta
Kuka.Sim Pro	Kuka	KRL	-	hind pärast päringut
MotoSim	Yaskawa-Motoman	INFORM	-	hind pärast päringut
RoboGuide	Fanuc	KAREL	+	2500$ / aasta
MELFA WORKS	Mitsubishi	MELFA-BASIC	-	2500$

2. Kolmanda osapoolte loodud virtuaalkeskkond

   Turul on olemas samuti kolmanda osapoole simuleerimise keskkonnad, kus on võimalik simuleerida rohkem, kui ühe roboti tootja seadmeid (vt Tabel 1.2).

Tabel 1.2 Simulatsioonimootori võrdlus

	Mathlab/ Simulink [3]	Unreal Engine	Unity	RoboDK
Graafika	2.5D	3D	3D	3D
Programmeerimis keel	MATLAB	C++	C#	Python
visuaalne programmeerimine	+	+	+	-
Õppematerjali kogus	Väike kogus tasuta materjale koolituseks. Enamasti tasulised kursused. Algajatele on arendajatelt tasuta materjal.	Peamiselt tasulised kursused, hea koolitusmaterjal arendajalt, palju infot foorumites.	Suur hulk erinevaid kursusi, koolitusvideod (nii tasulised kui ka tasuta), suur hulk kasutajafoorumeid, arendajate koolitusmaterjale, suur hulk näiteid / valmisprogramme arendajalt.	Arendaja koolitusmaterjal
Hind	1000$ - 13000$	Tasuta, kui teenid vähem, kui 1 000 000$	Tasuta, kui teenid aastal oma programmiga vähem, kui 100 000$	145€ / 2 aastat (üliõpilaste jaoks) 2995€ - pro versioon

Lõputöö olemuse eesmärgiga, st tähendab luua simulatsioonikeskkond roboti tööprotsessi programmeerimine digitaalsel kaksikul, sobib enim Unity mängumootoril baseeruv simulatsiooni rakendus. Selle rakenduse eeliseks on tasuta kasutamise võimalus ja suur kasutajate kommuun, mis võimaldab arendatava keskkonna loomise käigus esinevaid probleeme kiirelt lahendada, ning autoril oli selle rakenduse kasutamise kogemus.

1.2 Simulatsiooni keskkonna programmeerimiskeele valik

Simulatsioonikeskkonnas roboti liikumise simuleerimiseks on vaja luua programmeerituid koode. Vastavaid koode on võimalik luua erinevatel viisidel, kasutades graafilist (Visual Scripting) või siis standardset (C#) programmeerimise lähenemist. Kummagi rakendamisel on nii eelised kui ka miinused (vt Tabel 1.3).

Tabel 1.3 Erinevate programmeerimistüüpide võrdlus

	Visual Scripting	C#
Programmeerimise kiirus	aeglane	kiire
Testimise kiirus	+	-
Õpitavus	Lihtne	Keeruline
Materjali kogus	Vähene	Suur

Ülaltoodud võrdluse alusel on mõlemad lähenemised olemuselt sarnased, st on võimalik teostada vajaminevat funktsionaalsust. Lõputöö autor lähtus programmeerimise keele valikul omapoolsest huvist omandada uusi teadmisi ja oskusi. Sellest tulevalt sai valitud Visual Scripting programmeerimise viis. Lisaks võimaldab vastav programmeerimise keel visuaalselt vaadelda koodi tööd, mis võimaldab kiirelt aru saada tekkinud vigadest ning viia sisse parandused.

1.3 Füüsilise roboti juhtimiskontrolleri valik

Vastava lõputöö üheks osaks on reaalse roboti juhtimine läbi kontrolleri ning sinna programmikoodi edastamine. Kontrolleri valikul lähtuti kättesaadavusest, hinnast ja sobilikkusest. Sellest tulenevalt valiti *single-board* arvuti/kontroller, mis hõlmab endas nii seadmete juhtimise kui ka andmete edastamise/vastu võtmise võimekust ning võimaldab rakendada mitmehäälset töötlust ja multiprotsessorlust, mis on väga tähtsal kohal kontrolleri valimisel.

Miniarvutit ehk *single-board* arvutite tootjate hulk on aastate jooksul kasvanud. Vastavasse kategooriasse panustavad nii suurarvutite kui vähem tuntud tootjad.

Alljärgnevas tabelis on välja toodud ja võrreldud erinevaid nn üheplaadi arvutite tüüpe, mida on võimalik rakendada antud lõputöö raames (vt Tabel 1.4).

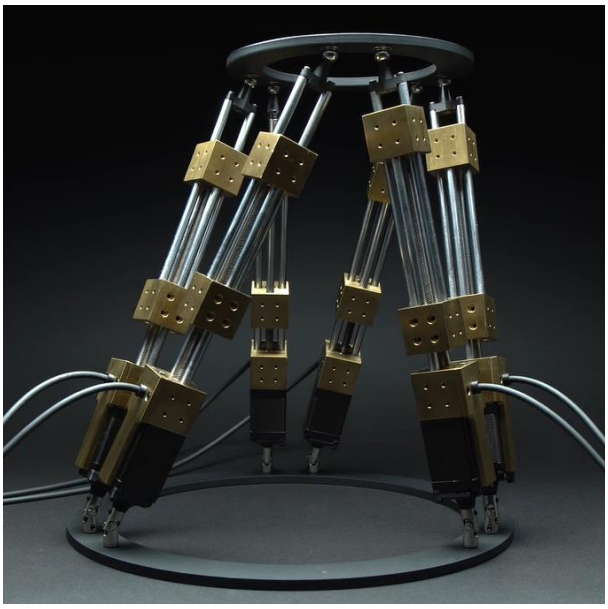
Tabel 1.4 Juhtimiskontrolleri võrdlemine

	Orange Pi 4	Banana Pi Pro	RaspberryPi 3 B+
GPIO arv	26	40	40
CPU	1.8GHz Dual-core	1 GHz Dual-core	1.4GHz Dual-core
RAM	3GB LPDDR4	1GB DDR3	1GB SDDR2
USB väljundid	2USB 2.0, 2USB 3	3 USB 2.0	4 USB 2.0
Wi-Fi	+	+	+
Video väljund	HDMI	HDMI	HDMI
Hind	102€	82€	70€

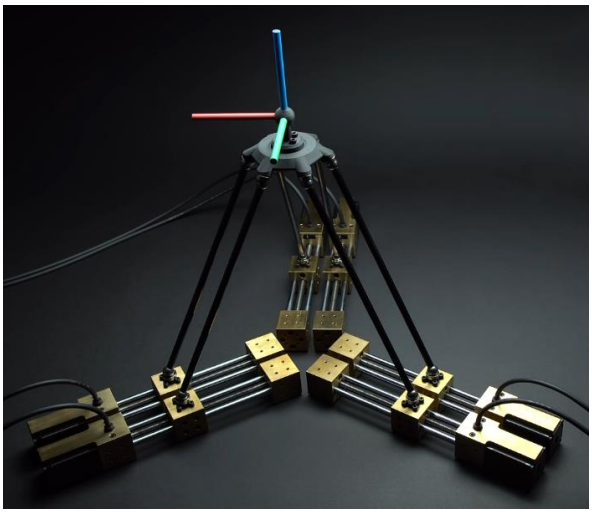
Tuginedes ülaltoodud võrdlusele on näha, et kõiki väljatoodud üheplaadi tüüpi arvuteid on võimalik rakendada lõputöö eesmärgiga. Autori valikuks jäi Raspberry Pi3 B+ *single-boardi* miniarvuti, sest selle kättesaadavus osutus autorile kõige lihtsamaks.

2 ROBOTI VALIK JA LOOMINE

Robotitehnilist süsteemi valides arvestati autori soovi enda kätega ise ehitada ja programmeerida ning mittestandardset tööstusrobotit. Lihtsaid manipulaatoreid või delta-roboteid kasutatakse kõikjal ja autor ei tahtnud teha juhitavat masinat. Idee pärineb insener Oleksandr Stepanenko Youtube'i kanalit, kust autor valis Stewart-Gough platvorm (vt Joonis 2.1), millele on rakendatud horisontaalne liikumine pöörleva kruvi abil (vt Joonis 2.2). Sellel robotil on kuus vabadusastet, mis võimaldab tal liikuda mööda kolme telge ja pöörelda ümber kolme telje.



Joonis 2.1 Stewart-Gough platvorm[4]



Joonis 2.2 Horisontaalne Stewart-Gough platvorm[5]

Autor eelistas antud lahendust, sest sellel oli mitmeid eeliseid laialdaselt leviva tööstusliku roboti ees. Näiteks võimaldab roboti kõigi liigeste ühtlane koormuse jaotus kasutada liikumiseks sammmootoreid ja seega liigutada platvormil palju rohkem kaalu. Võrreldes tavalise Stewart-Gough platvormiga oli selle eeliseks see, et see vähendas mootoritele kanduvat koormust.

Roboti tarvikud:

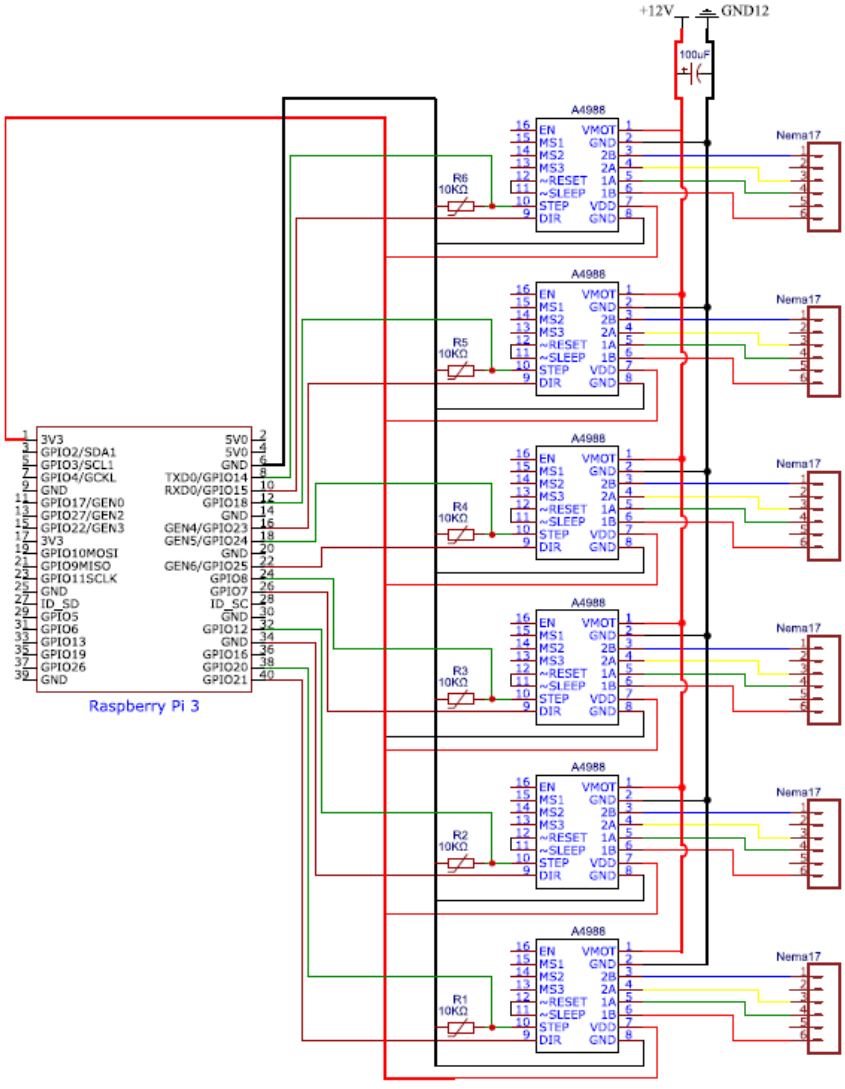
Tarvikute nimetus	tk	Hind / 1tk
Samm-mootor Nema17	6	7,20€
Trapetsikujulised kruvid	6	12,58€
Kruviotsik	6	
Laager koos hoidikuga	6	
Mootori draiverid a4988	6	3,6€
Raspberry Pi 3 B+	1	40,5€
PCB (jootmiseks)	1	6€
Takistid	6	0,5€
Kondensaator	1	0,5€
12 volti toiteallikas	1	20$
Metallist naastud	6	2,8€
Universaalsed ühendused	12	0,39€

Samm-mootori Nema 17 valik põhines selle kaubamärgi populaarsusel 3D-printimise valdkonnas. Selle kohta oli palju koolitusmaterjale, elektriskeeme ja tehtud palju katsetusi. See motor on täpne, seda on lihtne programmeerida ja kiiresti kättesaadav.

Samm-mootorite draiverid valiti kahest kõige populaarsemast kaubamärgist a4988 ja drv8825 (7,2€ / tk). Konkreetse draiveri valikul ei olnud peamine mitte ainult hind, vaid ka madalam energia tarve ja draiveri väiksem soojuseraldus, mis võimaldab kasutamist ilma sundjahutuseta. [6]. See valik tekitas siiski mõningaid probleeme. Sageli kasutatakse neid draivereid mikrokontrolleritega (nt Arduino või ESP32), nii et Raspberry jaoks teekide leidmine ja nende töö seadistamine projektis võttis palju aega. Samuti põhjustavad draiveri ja mikroarvuti vahelised pikad juhtme ühendused palju häireid, mis muudavad juhi korraliku töö võimatuks. Probleemi lahendamiseks vajaliku informatsiooni leidmine oli keeruline ning ei võimaldanud probleemi likvideerida korrektselt. Ainus lahendus leiti selle draiveri mikrokiipide tootja foorumist. Tõrgete likvideerimiseks tuli lisada takistus GND ja draiveri STEP-väljundi vahele [7].

Kruvi ja laagrid valis autor oma vahendite hulgast. Siiski mõned kruvi omadused (nt tüüp (neljakäivitus) ja keerme tüüp) on olulised projekti hilisemal simulatsiooni käigus seadistamisel.

Pärast ühendusskeemide uurimist ja makett-plaadil esialgse süsteemi katsetamist, koostati EasyEDA programmis täielik projektiskeem (vt Joonis 2.3). See programm võimaldab mitte ainult kasutada tuntud kiipide valmismalle, vaid ka kujundada ja tellida omapoolseid plaate. Järgmine samm oli plaadi koostamine, mootorite testimine ja teostatud tulemi kontrollimine lihtsate juhtimis-programmide abil.



Joonis 2.3 Mootori juhtimisprogrammi skeem

Juhtimisprogramm on kirjutatud Pythoni programmeerimiskeeles kasutades Thonny IDE keskkonda.

Mootori juhtimisprogrammi algoritmi loomisel teostati järgmised etapid (vt Joonis 2.4):

1. imporditi vajalikud teekid;
2. RaspberryPi vajalike GPIO-de deklareerimine;
3. juhtimisfunktsiooni loomine;
4. programmi käivitamine soovitud parameetritega.

```
import RPi.GPIO as GPIO

# import the library
from RpiMotorLib import RpiMotorLib

#define GPIO pins
GPIO_pins = (14, 15, 18) # Microstep Resolution MS1-MS3 -> GPIO Pin
direction= 20       # Direction -> GPIO Pin
step = 21      # Step -> GPIO Pin

# Declare an named instance of class pass GPIO pins numbers
mymotortest = RpiMotorLib.A4988Nema(direction, step, GPIO_pins, "A4988")


# call the function, pass the arguments
mymotortest.motor_go(False, "Full" , 100, .01, False, .05)

# good practise to cleanup GPIO at some point before exit
GPIO.cleanup()
```

Joonis 2.4 Mootori juhtimise kood[8]

2.1 Robotkontrolleri serveri funktsionaalsus simulatsioonikeskkonnaga suhtlemiseks

Andmeid töötlev serveriosa asub RaspberryPi kontrolleris. Serveriprogrammina kasutati Bottle web raamistikku. See raamistik valiti selle kasutusmugavuse, väikese mahukuse (167 kb), suurepärase funktsionaalsuse tõttu võrreldes tavapäraste serveriprogrammidega ja võimalusega töötada sellega Pythoni programmeerimiskeeles, mis oli autorile tuttav ja mida on lihtne rakendada.

Raamistiku paigaldamise käsud kontrollerisse (vt Joonis 2.5):

```
$ sudo pip install bottle             # recommended
$ sudo easy_install bottle            # alternative without pip
$ sudo apt-get install python-bottle   # works for debian, ubuntu, ...
```

Joonis 2.5 Serveri raamistiku paigaldamine[9]

Pärast raamistiku installimist pidi serveri käivitamiseks kirjutama minimaalse programmi ja käivitama skripti (vt Joonis 2.6)

```
from bottle import route, run

@route('/hello')
def hello():
    return "Hello World!"

run(host='localhost', port=8080, debug=True)
```

Joonis 2.6 Minimaalse veebilehekülje kood[9]

Veebilehtede kuvamine ei ole selle töö kontekstis oluline (ainult testimiseks), kuid päringute töötlemine ja muude skriptide käitamine on väga oluline.

Peamised funktsioonid, mida server selle töö jaoks vajab, on:

1. vastus kontrollitaotlustele;
2. simulatsiooniprogrammist saadetud failide salvestamine;
3. failide käivitamine.

Kõigi vajalike serverifunktsioonide täitmiseks on vaja järgnevaid teeke (vt Joonis 2.7):

1. bottle – veebiserveri teegi käitamiseks;
2. psutil – teek süsteemi seisundi ja tööprotsesside kohta teabe hankimiseks;
3. write - impordiprogramm loodud failide loendi kirjutamiseks ja lugemiseks;
4. start – importprogramm failide käitamiseks ja mootorite juhtimiseks;
5. os - kaustade ja dokumentidega töötamiseks;
6. multiprocessing - uue programmi käivitamiseks asünkroonsel lõimel.

```
1  from bottle import route, run, get, post, request
2  import psutil
3  import write
4  import start
5  import os
6  from multiprocessing import Process
7  import multiprocessing
8
```

Joonis 2.7 Vajalikud teegid

Serverile päringu tegemisel peamised sissetulevad andmed on (vt Joonis 2.8):

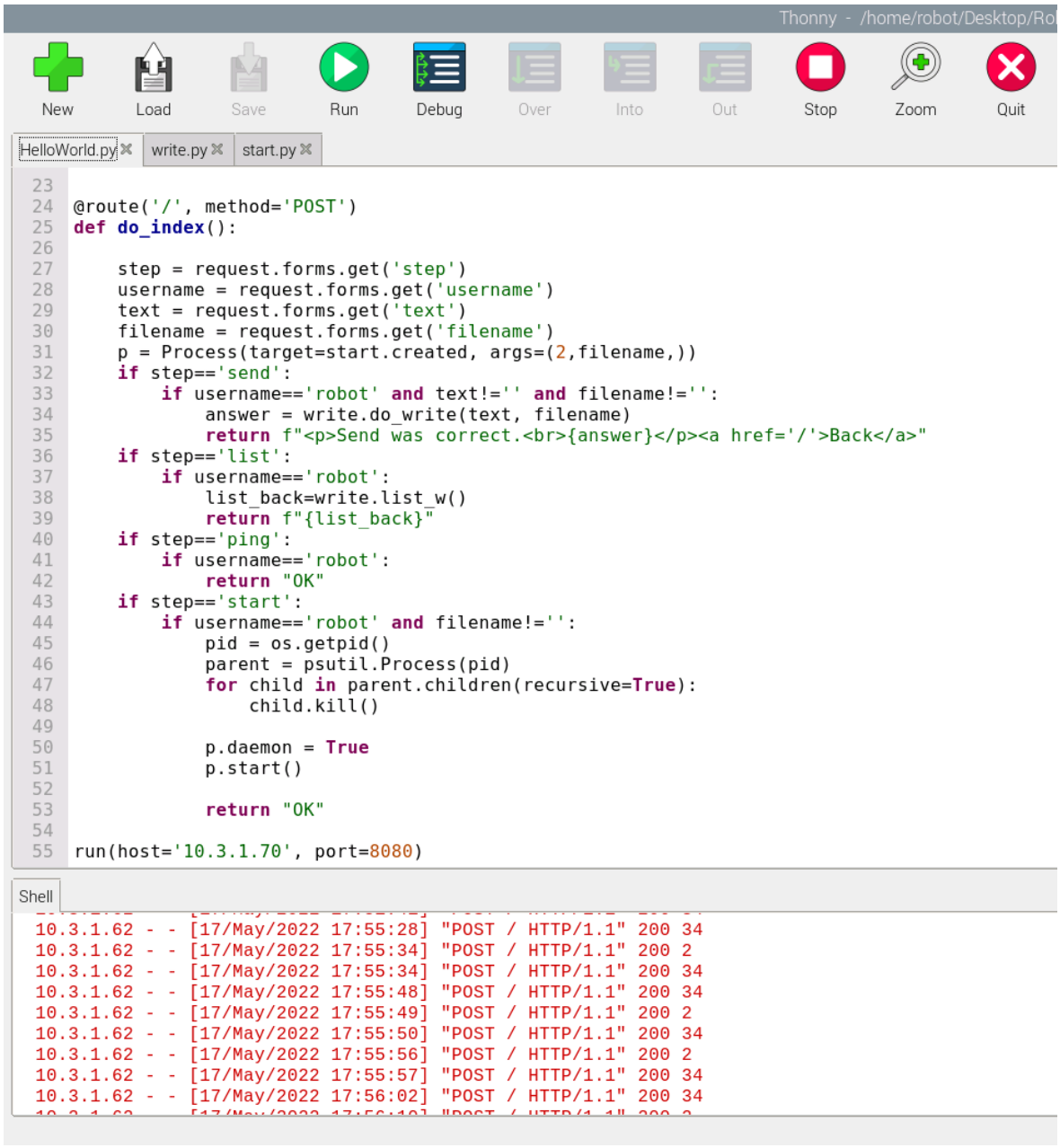
1. samm – näitab, milliseid toiminguid serverilt nõutakse;
2. kasutajanimi – kasutajanimi (valikuline, kuid seda saab kasutada programmi kaitse/paroolina teistelt võrgukasutajatelt);
3. tekst – faili kirjutatav programmi tekst;
4. failinimi – uue või käivitatava faili nimi.

```
27      step = request.forms.get('step')
28      username = request.forms.get('username')
29      text = request.forms.get('text')
30      filename = request.forms.get('filename')
```

Joonis 2.8 Vajalikud andmed töötamiseks

Serveri tegevus vastusena taotlusele:

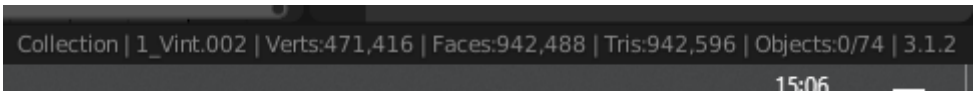
- Kui server peab ainult oma tööd kontrollima, tagastab server vastuseks "OK".
- Kui serverile tuleb päring luua uue faili, kontrollitakse saadetud faili nimi ja faili tekst. Käivitatakse funktsioon "write", mis on imporditud põhikoodi sisse. See funktsioon loob soovitud nimega faili ja kirjutab sinna saadetud teksti. Seejärel saadab server vastuse tehtud töö kohta.
- Kui tuleb päring loodud failide loendi saamiseks, loob server kõigi loodud failide loendi, kasutades funktsiooni "list_back", ja saadab vastuse koos loendiga tagasi.
- Kui on vaja käivitada juba olemasolev fail, võtab server süsteemis selle identifitseerimisnumbri ja hävitab kõik alamprogrammid (et peatada varasemad töötavad failid), misjärel käivitab uue tööfaili, mis juhib mootoreid vastavalt käivitatud server programmile (vt Joonis 2.9).



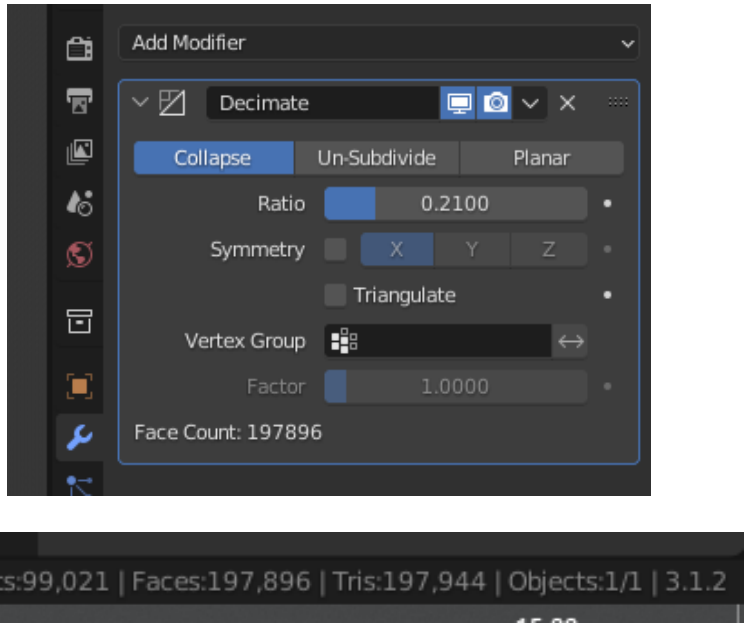
Joonis 2.9 Peaserveri programm, mis jälgib tegevusi.

3 MODELLEERIMINE JA SIMULATSIOONI KOOSTAMINE

Roboti 3D-mudel loodi SolidWorks'is, sest antud tarkvara annab töötada täpse mõõduga, ja autori eelnev kasutamiskogemus. Kahjuks töö käigus selgus, et SolidWorks'st mudeli ülekandmiseks simulatsiooni programmi puudus võimalus eksportida mudelit vajalikus formaadis, mida on pärast võimalik kasutada *Unity*-keskkonnas. Üks sobiv variant oli STL-vorming, mida üldjuhul kasutatakse 3D-printimiseks. Sellesse formaati üleviimisel tükeldati mudeli hulknurgad kolmnurkadeks, millest tulenevalt suurenes 3D-mudeli polügoonide arv, et seda mudelit simulatsioonis kasutada (vt Joonis 3.1). Autor pidi kasutama vaheprogrammi Blender polügoonide vähendamiseks. Importides kõik mudelid Blenderisse ja rakendades modifikaatorit "Decimate".See toiming võimaldab vähendada polügoonide arvu 79 protsendi võrra (optimaalne väärtus antud projekti 3D-mudelit jaoks) ilma mudeli visuaalset kvaliteeti rikkumata (vt Joonis 3.2). Pärast seda oli võimalik mudelit eksportida FBX-vormingusse, et sellega saaks hiljem töötada Unity keskkonnas.



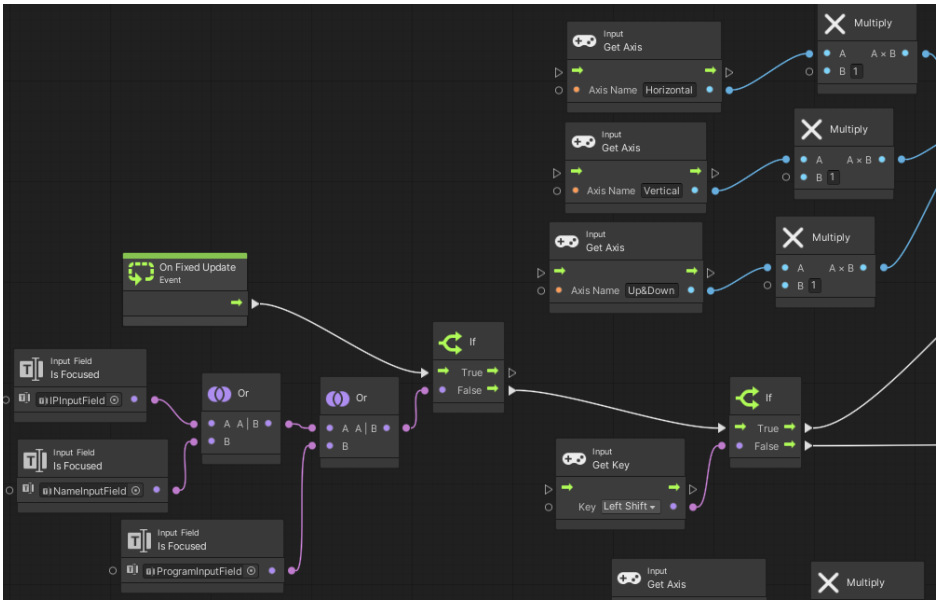
Joonis3.1 Poligoni arv (942 488) enne modifikaatorit.



Joonis 3.2 Modifikaatori tööriist ja poligoni arv (197 944) pärast modifikaatorit.

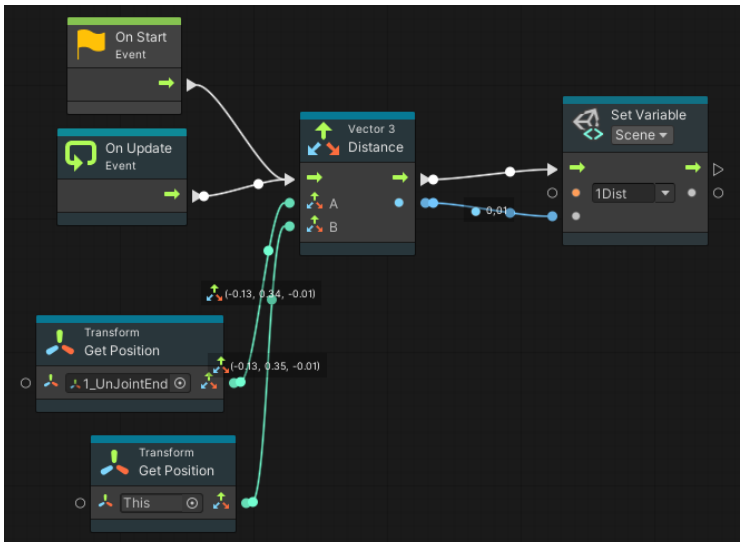
3.1 Visuaalne programmeerimine digitaalse roboti juhtimiseks

Visuaalne programmeerimine võimaldab meil luua programm, ilma programmeerimiskeele oskuseta. Kogu programmeerimine taandub soovitud ploki valimisele ja selle ühendamisele teise plokiga, andmete edastamisele või programmi tegevuse jätkamisele [10]. Sellist lähenemist on rakendatud ka programmeerimiskeeles Scratch (vt Joonis 3.3).



Joonis 3.3 Visuaalprogrammeerimine Unity-s.

Sellisel viisil programmeerimine võimaldab mitte mõelda keele süntaksile ja võimaldab jälgida milliseid andmeid edastatakse plokkide vahel programmi käivitamise. Antud lähenemine on hea programmi kiireks testimiseks ja probleemide tuvastamiseks (vt Joonis 3.4).



Joonis 3.4 Andmete saatmine blokkide vahel

Peamised simulatsiooni võimalused on järgmised:

1. digitaalse roboti seadistamine simulatsioonikeskkonnas koos liikumisvõimekusega;
2. füüsilise ja digitaalse roboti ühtlustamine simulatsioonikeskkonnas;
3. vigade kontrollimine;
4. andmete edastamine roboti juhtkontrollerisse või vastupidi;
5. kasutajale liidese loomine.

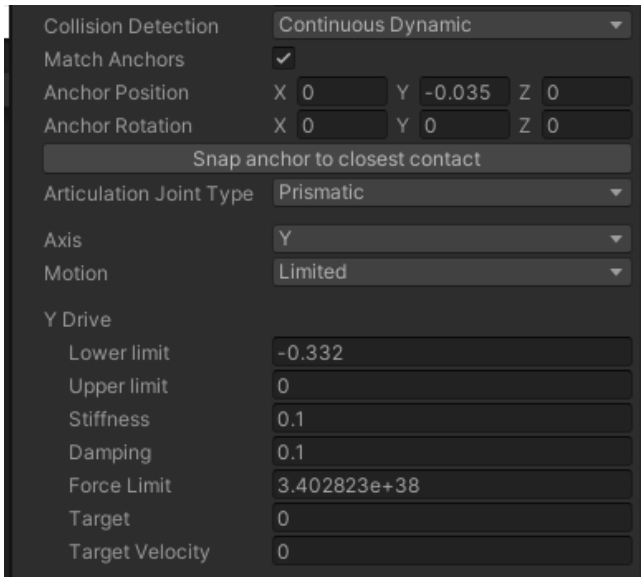
3.2 Digitaalse roboti ettevalmistamine simulatsioonikeskkonnaks

Roboti digitaalse kaksiku konfigureerimine Blenderis ja lisamine *Unity'sse* ei võtnud kaua aega, tuli lihtsalt mudeli üle viia projektikausta ja lisada uue objektina simulatsiooni keskkonda. Järgmine samm oli roboti üksikute komponentide vaheliste ühenduste moodustamine. Esialgu koondati kõik digitaalse roboti 3D-mudeli osad rühmadesse ja loodi hierarhiline struktuur.

Roboti digitaalse kaksiku osade vaheliseks ühendamiseks simulatsioonis kasutati funktsiooni „Articulation Body", mis võimaldab kahte objekti ühendada omavahel lahutamatu ühendusega, millel on kindel liikumistee.

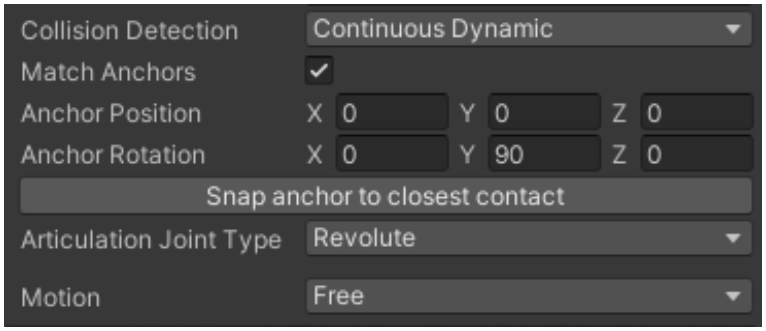
Iga roboti osade vahelise ühenduse jaoks koostati erinevad liikumistrajektoorid, sõltuvalt sellest, kuidas need pidid liikuma. Kõige raskemaks osaks oli tugikuubiku liikumise reguleerimine mööda kruvi. Vajalik oli määrata piiratud „Prismatic" tüüpi

liikumine ja väga täpselt reguleerida ühenduspunkt ülemobjektiga. See tagab, et sõidu ajal ei ületata liikumisepiire (vt Joonis 3.5).



Joonis 3.5 Piiratud ühesuunaline liikumine

Teistel osadel olid „Revolute"-ühendused, mis võimaldasid objektidel pöörelda vanema objektiga ühenduspunkti suhtes (vt Joonis 3.6).



Joonis 3.6 Pööramise liigutuse sätted

Objektide ühendamise meetodit „Articulation Body" kasutatakse tavaliselt "manipulaatori" tüüpi objektide loomiseks. See toob kaasa sellise ühenduse suure puuduse - kõiki ülemobjekte ei saa iseseisvalt liigutada. Nad liiguvad automaatselt allaobjekti taha. Kahjuks valitud roboti jaoks mootorid on peamised ülemobjektid ja platvorm peab liigutama kõiki enda taga olevaid ülemobjekte. Samal ajal platvormil ei saa olla rohkem kui 1 ülemobjekt. Selle probleemi lahendus leiti "Joint" ühenduse abil. Platvormist jaoks loodi 6 tühja alamobjekti, millest igaüks on ühendatud „Fixed" ühendusega viimaste objektidega "Articulation body" struktuuris. Sel viisil saab

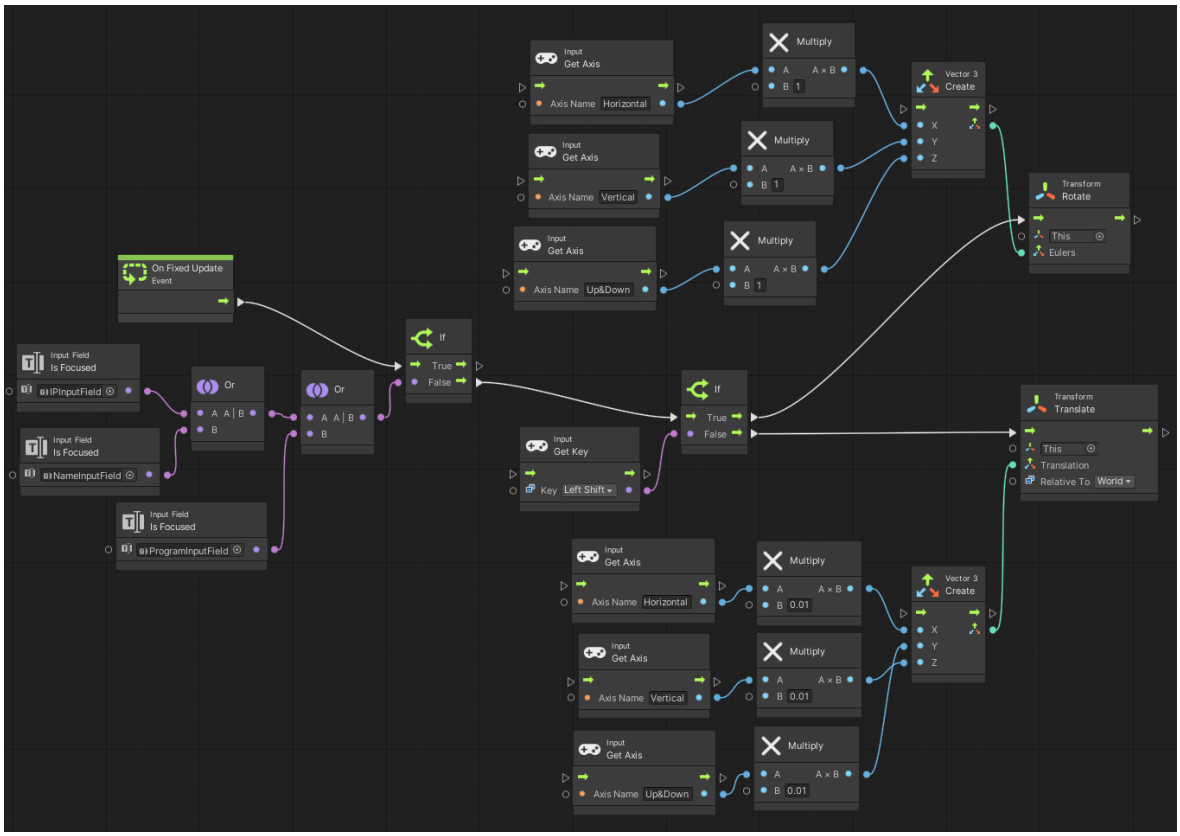
objekte omavahel ühendada, kuid "Joint" tüüpi ühendus võib simulatsiooni käigus hävida. Seda tuleb liikumise kontrollimisel arvesse võtta.

3.3 Liikumine simulatsioonis

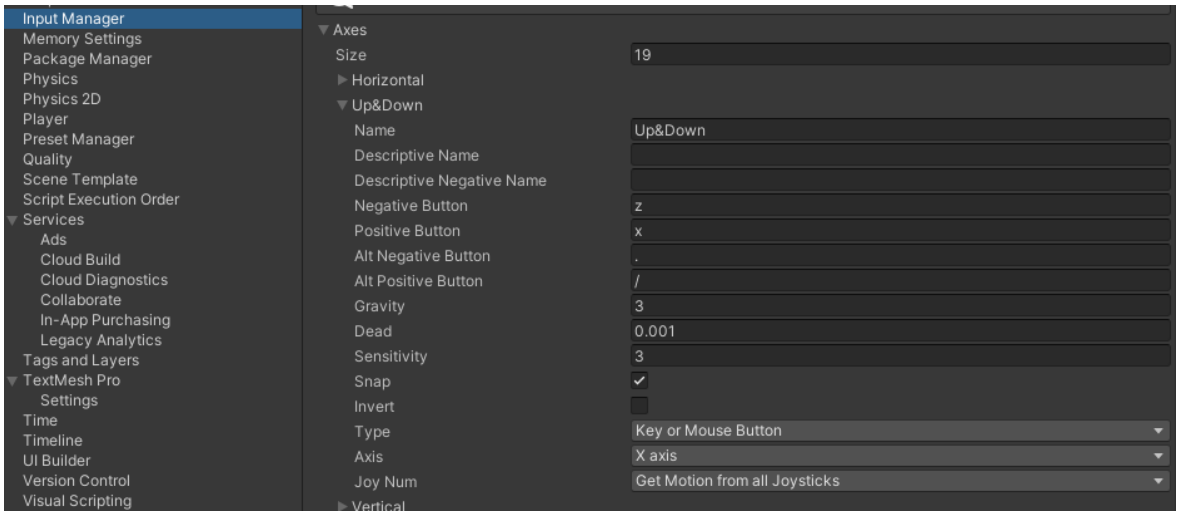
Roboti liikumine viidi ellu põhiplatvormi asukoha juhtimisega (vt Joonis 3.7):

1. iga 0.02 sekundit kontrollitakse, kas juhtklahvid on vajutatud (läbi plokki „On fixed update“);
2. kontrollitakse, kas kursor on mõnes sisestusväljas;
3. kontrollitakse vasakpoolse Shift-klahvi vajutamist;
4. kui vajutatakse Shift-klahvi, kontrollitakse, millise telje suhtes platvormi pööratakse;
5. kui Shift-klahvi ei vajutata, kontrollitakse, millise telje suhtes platvormi liigutatakse;
6. liikumiskiirus korrutatakse 0,01-ga, et muuta liikumise kiirust (vajutatud klahv annab väärtuse 0 kuni 1, ja see on liiga suur väärtus platvormi liikumiskauguse jaoks);
7. objektile saadetakse pöörde- või liikumiseandmed.

Liikumiseks saab kasutada vaikimisi X ja Z telge. *Unity*-l on juba sisseehitatud funktsioon klahvivajutuste kontrollimiseks, et juhtida vastavaid telgesid. Kuid projekti jaoks on vaja kasutada liikumiseks ka Y-telge. Antud telg tuleb lisada käsitsi projekti parameetrisse. (vt Joonis 3.8).

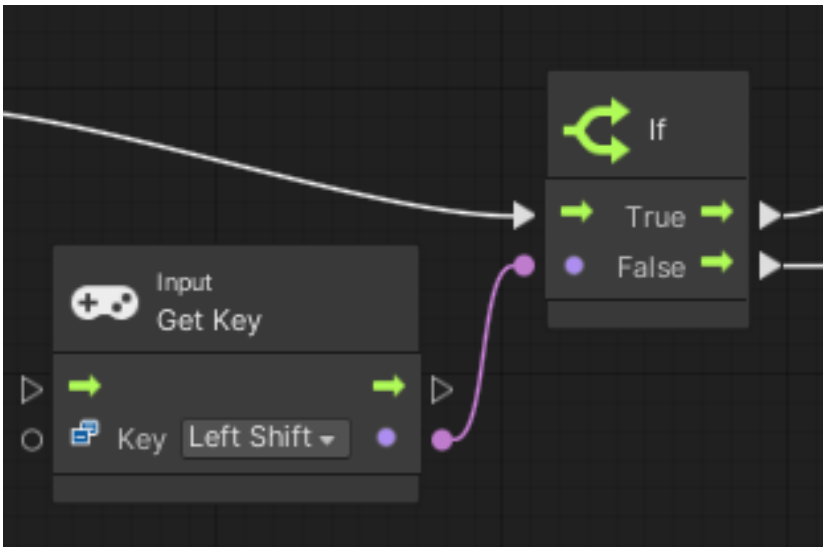


Joonis 3.7 Platvormi juhtimise kood



Joonis 3.8 Uus teljeseade

Platvormi kallutamine tehti, kontrollides, kas vajutati klahvi "Left Shift" (vt Joonis 3.9).



Joonis 3.9 Klahvivajutuse kontrollimine

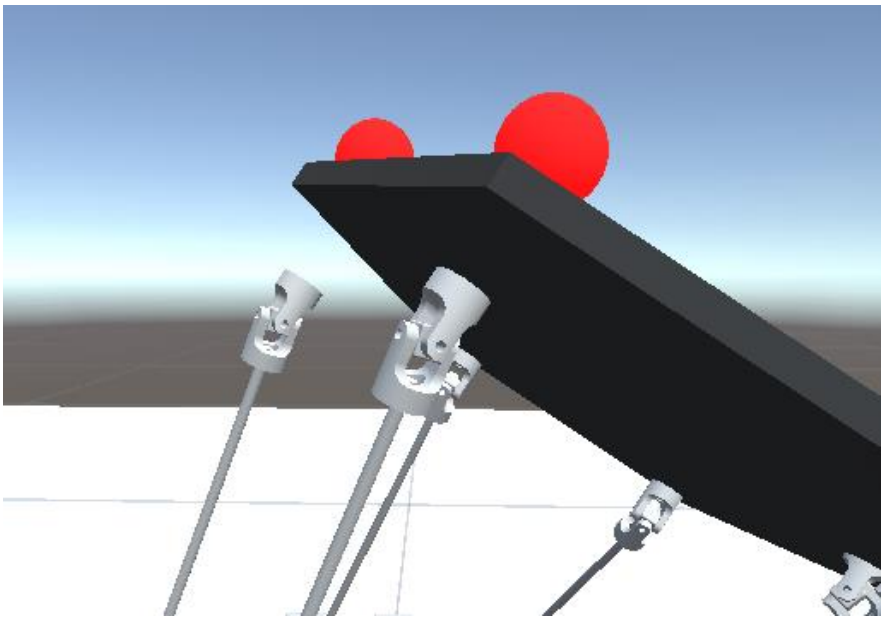
Selline klahvivajutusega kontrollimine toimub iga 0.02 sekundi järel, mille jaoks kasutati plokki "On Fixed Update". See plokk käivitub 50 korda sekundis, mis võimaldab paremat aega füüsikaliste parameetrite arvutamiseks. Seevastu OnUpdate-blokis käivitub kood igal kaadril ja tal ei pruugi olla aega füüsika arvutamiseks. Eriti kui kaadrisagedus on 60 või rohkem kaadrit sekundis.

Roboti jalgade liikumisdistantsi arvutamine on üsna lihtne, sest esialgu oli mudel skaalal 1:1 ning Unity keskkond töötab meetermõõdustikuga. Kuna esialgu on kõik mõõtmised meetrites, siis kuvati objekti seadetes kohe kaugus põhipunktist lõpp-punktini ja see sealt saaks andmeid võtta. Parema kasutamismugavuse huvides asub iga mootori juures tekstiplokk, mis kontrollib roboti jala positsioon muutust, teisendab kauguse mootorist kandva kuubikuni millimeetriteks ja paneb andmed tekstiväljale ning kirjutab need andmed ka muutujatesse, et saaks nende andmetega töötada vastavalt vajadusele (vt Joonis 3.10).

Joonis 3.10 Tekstiväli andmetega

3.4 Esinevate liikumisprobleemi kõrvaldamine

Peamine viga, mis simulatsiooni ajal tekkida võib, on platvormi ja kinnituse vahelise ühenduse katkemine. Seda viga on raske kõrvaldada Unity füüsikatöö ebatäiuslikkuse tõttu, kuid võib välistada uue sammu lisamise võimaluse ja näidata kasutajale, kus objekti rikutakse (vt Joonis 3.11).



Joonis 3.11 Programm näitab, et objekt on rikutud

Probleemi lahendamiseks loodi platvormi jaoks alamobjektina sfäärid ja seati need deaktiveeritud olekusse. Samuti kirjutati kood, mis iga 0.02 sekundi järel arvutas platvormiobjektis asuva tühja objekti ja universaalse ühendusmudeli vahelise kauguse, mille vahel loodi "Fixed" ühendus. See kaugus arvutati, lahutades ühe objekti vektorpunkt teisest. Nii saadi vajalik distants ja pandi paika distantsi piirangud.

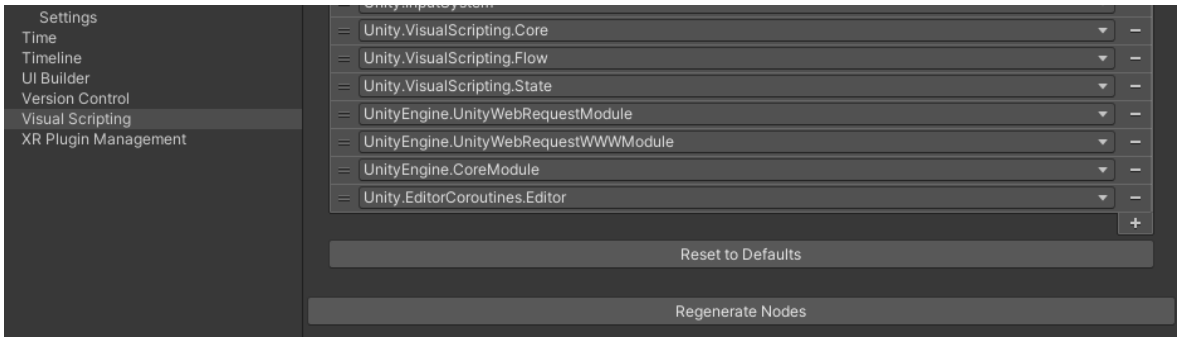
Kui kaugus ühest punktist teise oli üle või alla seatud piiri, siis need sfäärid muutuvad nähtavaks, et anda märku kasutajale vea tekkest.

Simulatsioonikeskkonna füüsikaliste arvutuste ajal võivad objektid kokkupõrgete tõttu veidi visuaalne pilt hüpelda, kuni nad leiavad tee soovitud punkti. Seetõttu on soovitatav pärast platvormi liikumisest oodata, kuni programm arvutab täielikult ümber objektide füüsilised andmed ja nende õige asukoha.

3.5 Simulatsioonikeskkonna ja robotikontrolleri kommunikatsioon läbi võrgu

Andmete edastamine teostati POST-i päringute kaudu kontrollerile. Kahjuks pole võrguga suhtlemisefunktsioone visuaalses programmeerimises vaikimisi lisatud, mistõttu tuleb need sinna käsitsi lisada [11]. Selleks lisati projekti sätete jaotises Visual Scripting-le käsitsi järgmised teegid (vt Joonis 3.12):

- UnityEngine.UnityWebRequestModule;
- UnityEngine.UnityWebRequestWWWModule;
- UnityEngine.CoreModule;
- Unity.EditorCoroutines.Editor.



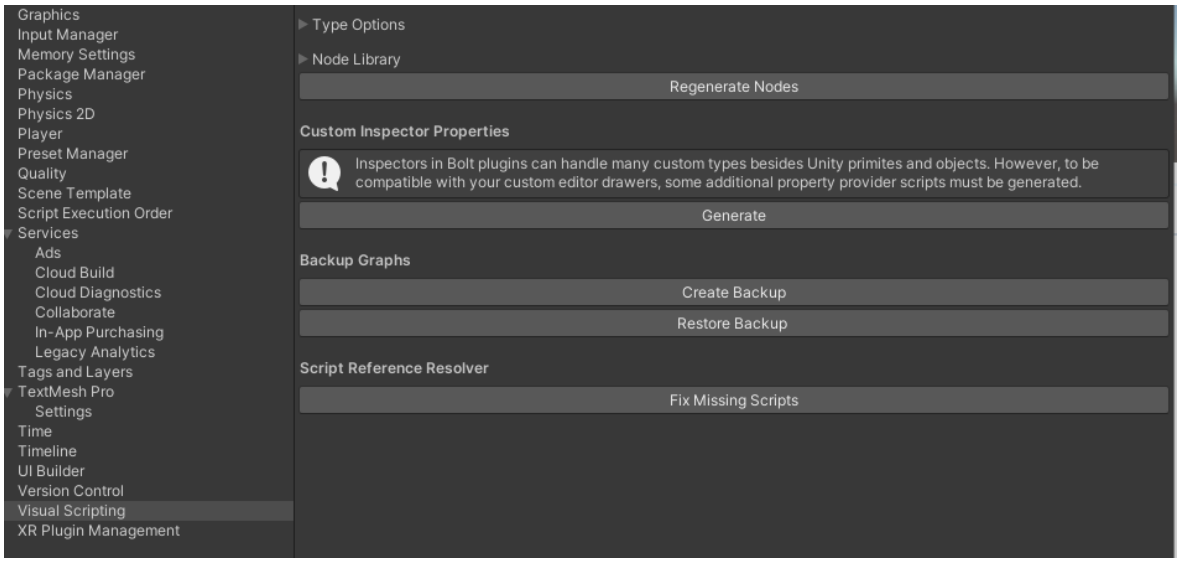
Joonis 3.12 Teekide lisamine simulatsiooni projektile

Samuti on vaja lisada veel sätete tüübid:

- WWW Form;
- Unity Web Request;
- Download Handler.

Uue sätte lisamine võimaldab kasutada programmi visuaalses redaktoris olevaid funktsioone.

Need teegid võimaldavad kasutada funktsionaalsusi, mis vastutavad andmete kontrollerisse saatmise ja sealt päringu vastusena tulevate andmete eest. Pärast nende teekide ja nende täpsustavate juhiste lisamist on vaja Visual Scripting funktsioonilinkide värskendamiseks klõpsama nuppu "Regenerate Nodes" (vt Joonis 3.13).

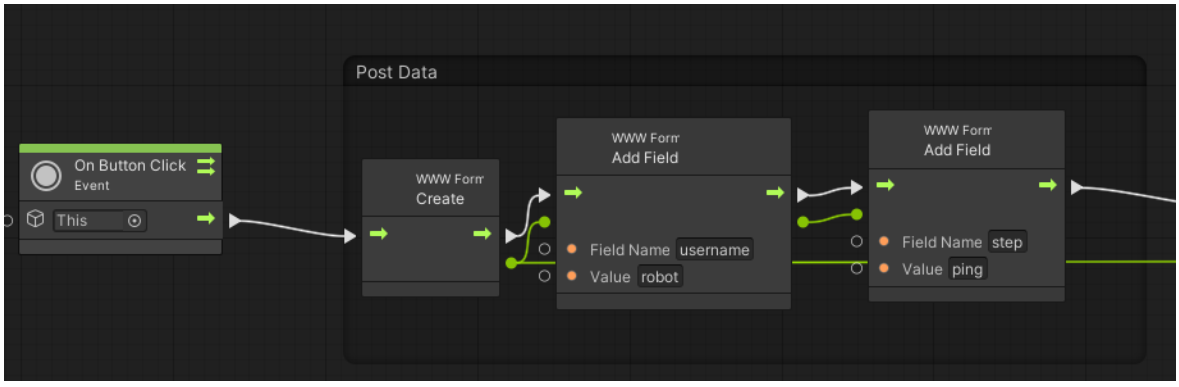


Joonis 3.13 Visuaal Scripti uue funktsiooni lisamise menüü

Töö andmeedastusega toimub 3 etapis:

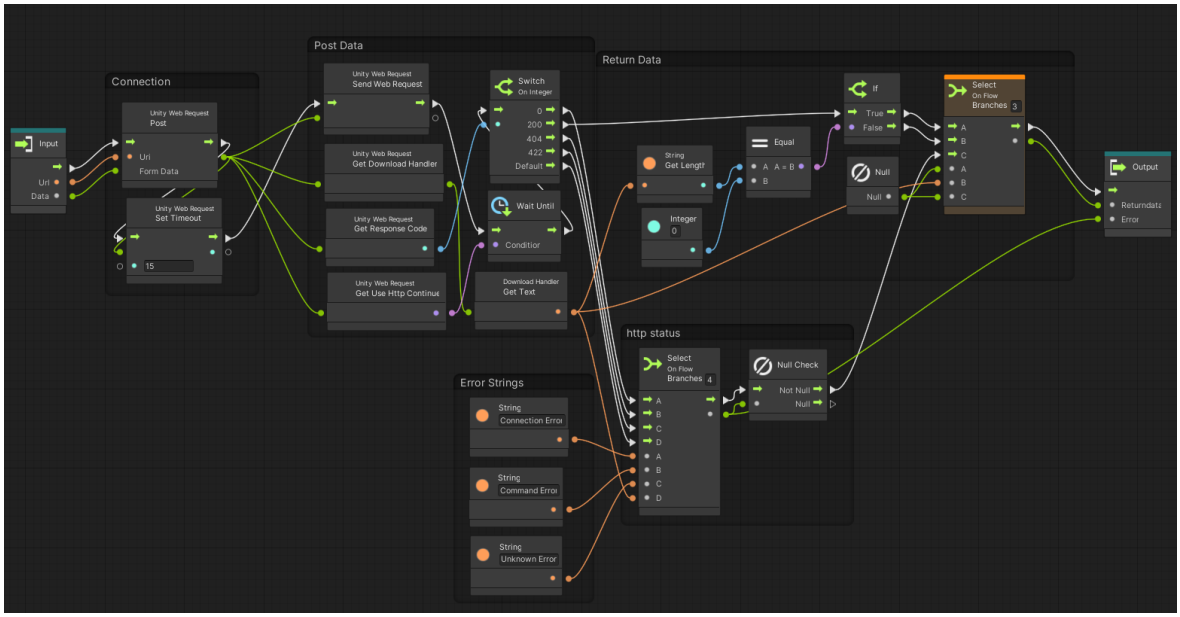
1. etapp - taotlusvormi koostamine;
2. etapp –andmete kontrollerisse saatmine;
3. etapp – serverilt vastuse saamine;
4. etapp - andmete töötlemine serverist (vajadusel).

Päring koosneb mitmest komponendist. Antud töös on selleks päringuvälja loomine ja sellele vajalike väärtuste muutmine, kuhu kasutaja saab sisestusväljalt panna või käsitsi sisestada (vt Joonis 3.14). Lisaks edastatakse andmed sisepäringu funktsioonile. Sellise funktsiooni loomiseks tuleb luua "Super unit" plokk ja määrata selle sees, milliseid andmeid soovitakse saata ja milliseid andmeid antud funktsiooni lõpus tagastatakse.



Joonis 3.14 Andmeteedastamise vormi koostamine

Andmete edastamise funktsiooni alguses esitatakse päring andmetega vormi saatmiseks kontrolleri, seatakse taimer ja oodatakse serverilt vastust. Server tagastab ühendusoleku koodi (vt Joonis 3.15).



Joonis 3.15 Vormi saatmine ja serveri vastuse ootamine

Peamised serveri vastuse koodid on:

- 0 - ühenduse viga;
- 200 – taotlus võeti edukalt vastu;
- 404 – lehekülge ei leitud;
- 422 - tundmatu viga.

Kui programm saab vastuseks koodi „200", siis käivitatakse vastuse saamise plokk ja saadetakse vastuse edasi. Kui on mõni muu vastusekood, väljastab programm veateate. Kogu informatsioon salvestatakse simulatsioonis logi failina, et vaadelda minevikus toimuvat.

Ainus, mida sellise päringu loomisel meeles pidada, on see, et sellist päringut ei saa iga kaadri puhul tööle panna. Taotluse töötlemiseks pole lihtsalt aega enne uue päringu esitamist. Selleks esitatakse sellised päringud ainult nupu vajutamisel või taimerile kohustusliku märguandega, ja kirjutakse, et tegemist on *Coroutine*-ga.

Coroutine on alamprogramm, mis on pikem kui üks kaader ja mida ei töödelda iga kaadriga ning mida saab käivitada kuni selle lõpuni ilma põhiprogrammi segamata[12].

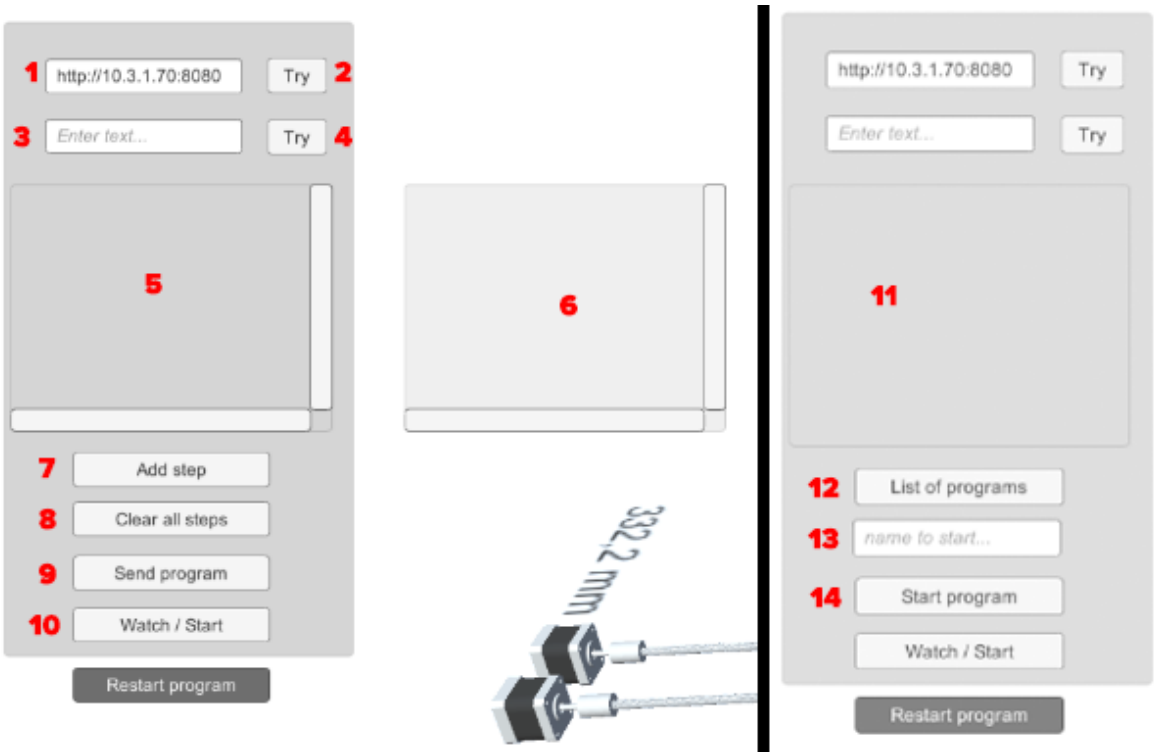
Pärast serveri vastust andmeid töödeldakse. Andmed kirjutatakse andmeväljale või muudetakse liidese nuppude värvi.

3.6 Kasutajale liidese loomine

Kasutajaliidese loomine toimus sisseehitatud kasutajaliidese objektidega. Need objektid on nähtavad ainult programmi käivitusrežiimis ja ei sega kuidagi simulatsiooni seadistamisel ajal. Kasutajaliidese visuaalne struktuur on autori enda poolt koostatud.

Peamised kasutajaliidese objektid on (vt Joonis 3.16):

1. väli serveri IP-aadressi sisestamiseks;
2. sisestatud IP-aadressi kontrollimise nupp;
3. väli uue programmi nime sisestamiseks;
4. nupp sisestatud nime kordumise kontrollimiseks;
5. väli programmi sammude loendiga;
6. väli mootori sammude loendiga, mis saadetakse serverisse valmis programmina (programmi algusest on peitud, aga võimalik avada testimiseks);
7. lisa sammu nupp;
8. programmi tühjendamise nupp;
9. programmi saatmise nupp;
10. nupp saate salvestamise ja käivitamise vahel vahetamiseks;
11. väli programmide loendiga;
12. nupp serveris olevate programmide loendi taotlemiseks;
13. väli käivitatava programmi nime sisestamiseks;
14. programmi käivitamise nupp;
15. tööriistavihje tekst platvormi haldamiseks.



Joonis 3.16 Kasutaja juhtimise interfeis

Kõik nupud kontrollivad kasutaja sisestatud andmeid. Kui tekib ühenduse tõrge kontrolleriga, siis värvitakse vajalikud nupud punaseks. Kui kõik on korras, muutuvad nupud roheliseks.

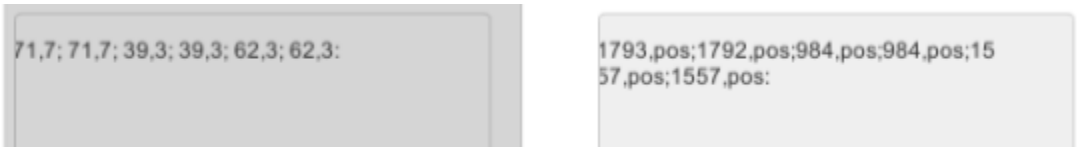
Kui klõpsata sammu lisamise nupul, lisatakse sammude sisestamise väljale muutujates olevad vahemaa andmed. Andmed lisatakse ainult siis, kui platvorm on õiges asendis ja kasutajale vigu ei näidata. Väljale lisatakse ka andmed, mis saadetakse serverisse.

Enne kontrollerisse andmete saatmist simulatsioonikeskkonnast on vaja muuta robotijala positsioon null-punktist füüsilise mootori pöördesammudeks. Selleks on vaja vana astme kaugus lahutada uue astme kaugusest. Kui arv on positiivne, siis kirjutame väärtuse "pos" (ja „neg“ kui on vastupidi). Järgmisena arvutakse läbitud vahemaa põhjal mootori sammude arv. Siin on vaja teadmisi kruvi keermeastme kohta. Antud projekti puhul on kruvi käik (kaugus, milleni kruvi liigub 360 kraadise pöörde korral) 8 millimeetrit ja mootori 360 kraadi pööramiseks vajalik sammude arv on 200 sammu. Arvutustabeli põhjal peame 1 mm liikumiseks tegema 25 sammu (vt Tabel 3.1).

Tabel 3.1 Sammu arvutustabel

Vahemaa	Sammude arv	Pöörlemisaste
1 mm	25	45°
2 mm	50	90°
4 mm	100	180°
8 mm	200	360°

Pärast ühe mootori sammude arvu arvutamist toimuvad need toimingud kõigi mootoritega. Iga mootori sammu andmete eraldamiseks kasutatakse ";" ja kõigi mootorite kirje lõpus kasutatakse ":". Vajame neid märke nende hilisemaks eraldamiseks serveri massiivist (vt Joonis 3.17).



Joonis 3.17 Simulatsiooni ja kontrolleri mootorite kirje esitlemise erinevus

Pärast programmi saatmise nupu vajutamist kontrollitakse sisestatud IP-aadressi, nime unikaalsust ja kui kõik on korras, siis saadetakse andmed (kasutajanimi, toimingu nimi, failinimi, tekst) serverisse. IP-adressi või faili nime sisestamise ajal on digitaalse roboti liigutamine keelatud. Kui kõik on korras, ilmub konsoolile kiri, mis ütleb, et fail on edukalt lisatud (vt Joonis 3.18).

Joonis 3.18 Serveri vastus, kui programm on salvestatud

Programmide loendi vaatamise nupule klõpsamine looks serverile päring kõigi kasutaja poolt juba loodud programmide kohta. Programmi väljundväli sisaldab kõiki programme, mille server meile vastuseks meie päringule tagastas.

„Start program“ Nnupu vajutamise ajal kontrollitakse sisestatud programmi nime ja saadab seejärel serverile käsu, et programm tuleb käivitada.

KOKKUVÕTE

Tootmise ülemaailmne automatiseerimine ja robootika kiire areng sunnib ettevõtteid leidma uusi lahendusi, et konkureerida omavahel. Kvaliteetse ja kiire tootearenduse tagamiseks vaatab üha rohkem ettevõtteid digitaalse kaksiku simulatsiooni poole, mida saab kasutada kõigi võimalike vigade jälgimiseks tootmise ajal või isegi vigade leidmiseks enne toote valmistamist.

Käesoleva töö eesmärk on luua süsteem roboti kaugjuhtimiseks digitaalse kaksiku abil simulatsioonikeskkonnas.

Töö on jagatud mitmesse ossa. Esimeses osas analüüsitakse olemasolevaid digitaalse kaksiku simulatsioonisüsteeme, samuti süsteeme, mis suudavad iseseisvalt luua simulatsioonikeskkonna. Autor kirjeldab digitaalse kaksiku simulatsioonikeskkonna valikut, kuidas seda simulatsiooni luua ja selgitab, miks ta valis mikroarvuti Raspberry Pi 3B+ roboti juhtimise põhikomponendiks.

Teises osas teeb autor valiku loodava roboti tüübi kohta ja kirjeldab, kuidas seda luua, mille alusel luuakse digitaalne kaksik simulatsioonikeskkonnas. Samuti kirjeldatakse, kuidas roboti kontroller suhtleb koos simulatsioonikeskkonnaga.

Kolmandas osas kirjeldab autor digitaalse kaksiku loomist, selle seadistusi, digitaalse kaksiku juhtimissüsteemi, tööruumi kasutaja loomist ja andmete edastamist simulatsioonikeskkonnast kontrollerisse. Kirjeldatakse ka autori tekkinud probleeme ja nende lahendusi.

Selle töö tulemus on täielikult kooskõlas kõigi eesmärkidega, mille autor püstitanud oli projekti alguses. Digitaalse kaksikuga loodud simulatsiooni keskkonnast saab saata ja vastu võtta andmeid füüsilise roboti kontrollerilt, samuti luua uusi roboti liikumise programme ja kontrollida kõiki digitaalse kaksiku liikumisi võimalike vigade suhtes.

Seda projekti saab mitmel viisil täiustada:

1. trükkplaadi saab pärast testimist EasyEDA programmis täielikult kokku panna ja valmis kujul tellida;
2. roboti jaoks võimalik teha lisaosi, mis pole kasulikud, kuid võimaldavad sellel välja näha tehaseversioon;
3. simulatsioonikeskkonnas saab võimalikud vead parandada ja kõik funktsiooniplokid paremini paigutada. See võimaldab neil, kes sellise simulatsiooniga töötavad, kiiremini aru saada simulatsioonikeskkonna loogikast.

Seda lõputööd on võimalik kasutada koolitusmaterjalina nii õpilased kui ka erinevate

eluvaldkondade inimesed, keda huvitavad robottehnilised süsteemid.

SUMMARY

The main goal of the project is to create a simulation of a digital twin in order to be able to control the robot remotely. A digital twin is a technology for checking a digital object for various factors before the object is produced in production or before commands about the actions to be taken are sent to the real object. This method allows you to reduce the likelihood of risks during direct work with the object or during its production.

Existing systems are difficult to learn, have limited capabilities, or are expensive, making them less suitable for individuals, small businesses, or learning.

This work makes it possible for everyone to assemble a robot, program a controller for communication with simulation and create a simulation environment.

This project can be improved in several ways. Such as:

the printed circuit board after its testing can be completely assembled in the EasyEDA program and ordered in finished form;

1. for the robot, you can make additional parts that will not be useful, but will allow it to look like a factory version;
2. several times going through the simulation schemes, you can correct any errors and more beautifully place all blocks of functions. This will allow those who will work with such a simulation to more quickly understand the logic of the simulation environment.

This work allows you to better study the work of data transmission in the network, start programming in a visual programming environment, and also manually assemble the robot.

KASUTATUD KIRJANDUSE LOETELU

1. Carlos, M. What is the real value of digital twin simulations for operations in space. [*Online*] https://www.challenge.org/insights/digital-twin-for-space-operations/ (22.05.2022).

2. RoboDK. Off-line Programming. [*Online*] https://robodk.com/blog/off-line-programming/ (22.05.2022).

3. The MathWorks. Pricing and Licensing. [*Online*] https://www.mathworks.com/support/pricing/2022/usd-us.html (22.05.2022).

4. Stepanenko, O. KINEMATICS | Hexapod (Gough-Stewart platform) 6-axis parallel robot (This is not CGI). [*Online*] https://www.youtube.com/watch?v=xiECumcaEx0&ab_channel=OleksandrStepanenko (22.05.2022).

5. Stepanenko, O. KINEMATICS | Serial robot vs. Parallel robot (This is not CGI). [*Online*] https://www.youtube.com/watch?v=3fbmguBgVPA&ab_channel=OleksandrStepanenko (22.05.2022).

6. A4988 vs DRV8825 Chinese Stepper Driver Boards. [*Online*] https://reprap.org/wiki/A4988_vs_DRV8825_Chinese_Stepper_Driver_Boards (22.05.2022).

7. A4988 with raspberry: floating step during boot. [*Online*] https://forum.pololu.com/t/a4988-with-raspberry-floating-step-during-boot/8645 (22.05.2022).

8. Bipolar Nema11 Stepper motor with A4988 Driver Carrier. [*Online*] https://github.com/gavinlyonsrepo/RpiMotorLib/blob/master/Documentation/Nema11A4988.md (22.05.2022).

9. Hellkamp, M. Tutorial. [Online] http://bottlepy.org/docs/dev/tutorial.html#quickstart-hello-world (22.05.2022).

10. About visual scripting. [*Online*] https://docs.unity3d.com/Packages/com.unity.visualscripting@1.5/manual/index.html (22.05.2022).

11. Inan, L. [Unity/Bolt] Visual Script - UnityWebRequest Tutorial. [*Online*] https://www.youtube.com/watch?v=8GtiOct3Fkk&ab_channel=LeventInan (22.05.2022).

12. Coroutines. [*Online*] https://docs.unity3d.com/Manual/Coroutines.html (22.05.2022).