TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Rasmus Iila 135041IAPB

LEGO MINDSTORMS ROBOTI SIMULATSIOON GAZEBOS

Bakalaureusetöö

Juhendaja: Gert Kanter
Tehnikateaduse magister

Tallinn 2017

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: Rasmus Iila

01.05.2017

Annotatsioon

Käesoleva lõputöö eesmärgiks on TTÜ informaatika bakalaureuseõppekava uue õppeaine „Robotite programmeerimine“ jaoks testkeskkonna ning Lego Mindstorms roboti simulatsioonimudeli loomine. Tudeng peab saama keskkonna endale arvutisse alla laadida ning seal käivitada simulatsiooni. Tudengi poolt käivitatud kood peab käituma simulatsiooni roboti peal võimalikult sarnaselt füüsilise robotiga. Samuti peavad andurite väljundid simulatsioonis olema vastavuses päris andurite väljunditega.

Tulemusena valmis keskkond, mis lubab tudengi poolt kirjutatud Pythoni koodi käivitada simuleeritud roboti peal. Tudengi kirjutatud kood laetakse alla tema Git-i salvest. Simulatsioonis on realiseeritud roboti mootori peamine funktsionaalsus. Samuti on realiseeritud värvianduri, ultraheliandur, güroanduri ja puuteanduri olulisem funktsionaalsus. Simulatsioonil on visuaalne liides (Gazebo visuaalne liides on simulaatori osa), mis lubab tudengil koodi töötavuses veenduda visuaalsel teel.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 7 peatükki, 8 joonist, 0 tabelit.

Abstract

Lego Mindstorms robot's simulation in Gazebo

The aim of this thesis is to create a test environment for TTÜ's upcoming course „Robots' programming" and to create a simulation model for a Lego Mindstorms robot. The test environment is aimed for the students so that they could test their Python code on the robot outside the class. The students should be able to download the mentioned environment to their personal computers and run the simulation within that environment. The Python code written by the student has to work on the robot in the simulation the same way it works on the actual physical robot. For example, giving a command to the left motor should make the corresponding wheel move in the simulation. Additionally, the simulated sensors' output should be in correlation with the physical sensors' output. The sensors that need to be implemented in the simulation are: color sensor, ultrasonic sensor, gyrosensor and touch sensor.

As a result of this thesis, an environment that allows students to run their Python code on a simulated robot was created. Students can choose which exercise they would like to test and which world instance they would like to try. Students' code will be pulled from their Git repositories. In the simulation, the important functonality of the robot's motor was implemented. Color sensor's, ultrasonic sensor's, gyro sensor's and touch sensor's important functionality was implemented aswell. Because of this, the students are able to use the same commands they would use for the physical robot, on the robot in the simulation. The simulation has a visual user interface (Gazebo simulator) that allows the students to verify how their codes work on the robot.

The thesis is in estonian and contains 33 pages of text, 7 chapters, 8 figures, 0 tables.

Lühendite ja mõistete sõnastik

Git	Versioonihaldustarkvara
RAM	*Random Access Memory*; muutmälu, salvestab sagedasti kasutatud käsud nende kiiremaks kätte saamiseks
RGB	*Red, Green, Blue*; värvide tähistamise viis, mis sisaldab värvi punase, rohelise ja sinise komponendi sisalduse arvu, tavaliselt vahemikus 0 kuni 255
ROS	*Robot Operating System*; tarkvara raamistike kollektsioon robotite arendamiseks; vahevara, mis võimaldab robotite tarkvarakomponente modulaarselt ühendada
USB	*Universal serial bus*; universaalne järjestiksiin; lubab välisseadmeid külge ja lahti ühendada ilma, et arvutit peaks välja lülitama
Wifi	Kaubamärk, millega tähistatakse sertifitseeritud traadita kohtvõrgu klassi kuuluvaid seadmeid; traadita kohtvõrk
XML	*Extensible Markup Language*; märgistuskeel infovahetuseks

Sisukord

Jooniste loetelu

1 Sissejuhatus

2017. aasta sügissemestril lisandub TTÜ esimese aasta informaatika tudengite õppekavasse uus õppeaine nimega „Robotite programmeerimine“. Selles aines saavad tudengid programmeerida Lego Mindstorms roboteid programmeerimiskeeles Python.

1.1 Probleem

Ülalnimetatud õppeaine on kohustuslik kõikidele esimese aasta tudengitele informaatika õppekavas. Seega deklareerib ainet umbes 150 tudengit. Aine on loomult praktiline, mistõttu on tudengitel oluline oma lahendusi sagedasti roboti peal testida. Et roboteid on piiratud arv ning tudengeid suhteliselt palju, siis võivad tekkida ootejärjekorrad roboti kasutamiseks. Kuivõrd ülesanded võivad osutuda tudengitele keerulisteks, on tudengitel soovituslik oma loodud algoritme katsetada simulatsioonis.

Tudengitele on ligipääs robotitele ainult praktikumi ajal, seega lisaks ootejärjekordadele on probleemiks ka asjaolu, et tudeng saab oma koodi ainult teatud ajal testida ning puudub võimalus testida koodi kodus. Ideaalsel juhul saaks koodi kirjutades seda jooksvalt kontrollida.

1.2 Ülesandepüstitus

Et tudeng saaks siiski enda koodi testida praktikumivälisel ajal, on tarvis ehitada keskkond, kus tudeng saab koodi testida simulatsioonis. Keskkonnale kehtivad järgnevad nõuded:

- Tudeng peab saama keskkonna enda arvutisse alla laadida ning seal seda käivitada.
- Keskkond peab minu poolt loodud Git-i (versioonihaldustarkvara) salvest alla laadima simulatsiooni jaoks vajalikud failid.
- Keskkond peab tudengi Git-i salvest alla laadima tema koodi.

- Tudeng peab saama lihtsasti käivitada simulatsiooni koos testitava koodiga.

Samuti tuleb roboti simulatsioonimudel ehitada. Simulatsiooni keskkonnaks kasutatakse Gazebo simulaatorit. Simulatsioonil peab olema järgnev funktsionaalsus:

- Simulatsioonis kujutatav roboti mudel peab vastama piisavalt täpselt füüsilise roboti mõõtmetega. Roboti mudeli piirderisttahukas peab kindlasti vastama füüsilise roboti mõõtmetele.

- Pythonis kirjutatud käsud peavad simulatsioonis robotile samamoodi mõjuma nagu nad mõjuksid päris robotile. Käsud saab ev3dev teegist. [1]

- Füüsilise roboti mootorite mõju ratastele peab olema simuleeritud.

- Simulatsioonimudelis peab robotile olema võimalus lisada värviandur, ultraheliandur, güroandur ning puuteandur.

- Päris roboti andurite olulisem funktsionaalsus peab simulatsioonis olema kaetud.

1.3 Ülevaade

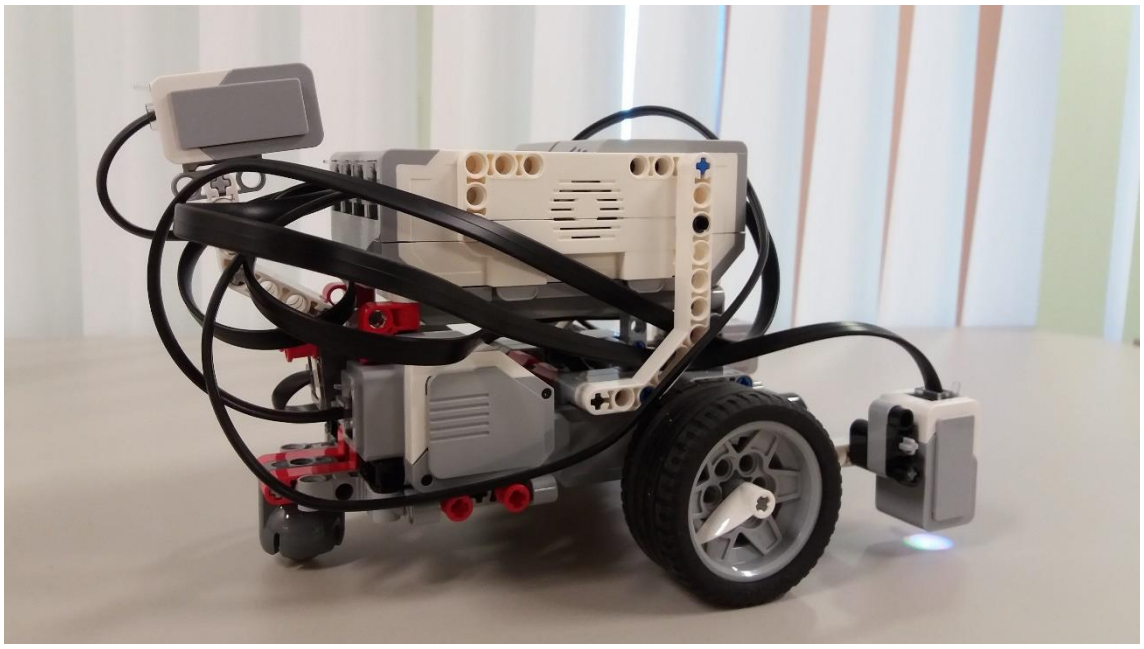
Teises peatükis räägitakse roboti valmis ehitamisest simulatsioonis võttes aluseks füüsiline robot. Kolmas peatükk käsitleb Pythoni koodi ja Gazebo simulatsiooni vahel ühenduse loomist. Neljas peatükk räägib roboti mootori simuleerimisest. Viies peatükk räägib kasutatavate andurite simuleerimisest. Kuuendas peatükis räägitakse tudengile kasutatava keskkonna ülesseadmisest. Kokkuvõttes analüüsitakse tehtud tööd ning mida saaks edasi arendada.

2 Roboti ehitamine

Roboti simuleerimiseks on esmalt vaja kokku panna füüsiline robot. Seejärel ehitame füüsilisele robotile analoogse roboti mudeli simulatsioonis.

2.1 Füüsiline robot

Füüsilise EV3 Lego Mindstorms roboti kokkupanekuks on olemas juhend. Juhend õpetab, kuidas roboti vajalikud osad kokku panna ning sisaldab ka eraldi peatükke lisaosade (näiteks erinevad andurid) juurde panemiseks. Kokkupandud robotit võib näha jooniselt 1.



Joonis 1. Lego Mindstorms robot külgvaates.

Robot koosneb raamist, pardaarvutist, kahest rattast ning veerevast kuulikesest. Lisaks võib robotile külge panna erinevaid andureid. Füüsilise roboti kõige olulisem osa on pardaarvuti. See sisaldab roboti protsessorit, muutmälu, mälukaarti, mälukaardi lugejat, USB (*universal serial bus*) pesa koos USB wifi seadmega ning võimaldab edastada käske tema külge käivatele anduritele ja täituritele ning pärida infot. Pardaarvutiga samas

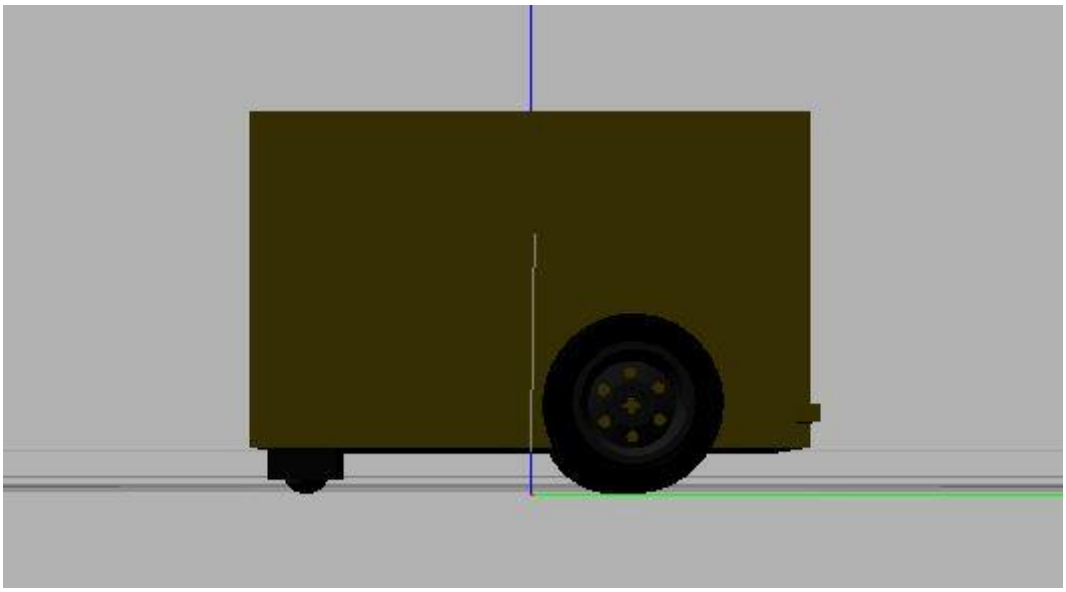
korpuses on ka aku, mis on vooluallikaks nii pardaarvutile kui ka mootoritele ja anduritele. Juhtmed ongi mõeldud andurite ja mootorite ühendamiseks pardaarvutiga. [2]

Robot liigub kasutades tema kahte ratast ning vabalt pöörlevast kuulist. Rattaid ei saa keerata vasakule-paremale nagu näiteks autol, vaid roboti pööramiseks rakendatakse ratastele erinevaid kiirusi. Ratastele antakse kiirused mootorite poolt. Veereva kuuli eesmärk on lubada liikumist igas suunas, samuti pakkuda robotile kolmandat tugipunkti, tagades robotile tasakaalu. Antud konfiguratsiooni nimetatakse diferentsiaalajamiks, mis erinevalt autolaadsetest robotitest võimaldab ka kohapeal pööramist.

Andurite paigutus on suhteliselt paindlik. Andureid saab tõsta vastavalt vajadusele erinevatesse kohtadesse. Samuti ei ole andurid alati kohustuslikud.

2.2 Roboti ehitamine simulatsioonis

Simulatsiooniks kasutatakse Gazebo simulatsiooni keskkonda. Simulatsioonis ei ole vaja robot luua üks-ühele kujuga ja värvidega. Esiteks raiskaks liiga detailse roboti tegemine aega ning teiseks nõuaks see simuleerimisel suuremat arvutusvõimsust. Seega võttis autor robotit simulatsioonis ehitades arvesse füüsilise roboti välimisi mõõtmeid. Niimoodi saab simulatsioonis kindlalt teada, kas teatud manööver on võimalik või mitte. Mõõtmed, mida arvesse võetakse on: roboti maksimaalne füüsiline laius, pikkus ja paksus, kõrgus maapinnast, rataste läbimõõt, nende paksus, veereva kuuli läbimõõt. Samuti on olulised rataste ja kuuli suhtelised asukohad võrreldes kehaga. Lisaks võeti arvesse ka andurite mõõtmeid, kuna need seadistatakse roboti mudelisse roboti korpusest eraldi ning nende lisamisel roboti külge suureneb tema kokkupõrgete tuvastamiseks kasutatava piirderisttahuka (*bounding box*) suurus. Kasutades programmi *Blender* loodi simulatsioonis kasutamiseks roboti, mida võib näha jooniselt 2.



Joonis 2. Lego Mindstorms roboti külgvaade simulatsioonis.

Jooniselt vaadates tunduvad rattad väga detailsed, kuid see on ainult visuaalne osa ning selline visuaalne mudel oli juba loodud ning tasuta saadaval 3D faile pakkuval leheküljel.[1] Kokkupõrgete tuvastamise piirdeobjektina kasutatakse ratta jaoks ikkagi lihtsat silindrit. See vähendab simulatsioonile rakendatud koormust. Rattad on ühenduses roboti kehaga liigendiga, mis lubab liikumist ainult ümber x-telje. See tähendabki, et ratas saab pöörelda ainult edasi või tagasi. Rattale on ka oluline seada hõõrdetegur, kuna ratas puutub kokku väljakuga. Ratta välispind on tehtud kummist. Järgneva allika [3] kohaselt on kummi hõõrdetegur enamus pindadega umbes 0.6.

Veereva kuuli simuleerimiseks kasutatakse lihtsat kasti, mille hõõrdetegur on seatud nulliks. See on jällegi simulatsiooni koormuse vähendamiseks ning tähendab, et ei ole vaja leiutada liigest, mille üks osapool saab liikuda igas suunas. Visuaalse mudelina kasutatatakse siiski kera, mitte kasti. [4]

On oluline määrata roboti osade massid ning nendest tulenevalt inertsimomentide maatriksid. Nende puudumisel käitub robot simulatsioonis väga kummaliselt. Näiteks edasi-tagasi värelemine, maa alla vajumine, või agressiivne iseeneslik põrkamine. Maatriksi saab välja arvutada valemitega, kasutades roboti osade mõõtmeid ning masse.

[1] https://grabcad.com/library/lego-wheels-1

Lõpuks on oluline see, et robot käituks simulatsioonis sarnaselt päris robotiga. Välimus ei ole niivõrd oluline. Peaasi, et tudeng saab peale vaadates ikkagi aru, et tegemist on robotiga. Olulised on just maksimaalmõõtmed.

3 Robotile käskude andmine Pythonis

Antud aines programmeeritakse roboteid programmeerimiskeeles Python. Tudeng peaks saama täpselt sama koodi jooksutada nii füüsilise robotiga kui ka simulatsiooni robotiga. See tähendab, et robot simulatsioonis peab vastu võtma käske Pythoni koodist. Gazebo simulaatoris selline funktsionaalsus hetkel puudub, mis tähendab, et see funktsionaalsus tuleb realiseerida.

3.1 Pythoni ja Gazebo ühendamine

Eesmärk on võimaldada Pythoni koodi suhtlemist Gazebo simulatsiooniga. Lõputöö algusfaasis kasutas autor teeki nimega *'pygazebo'* [5]. Antud liides pakub tõepoolest võimaluse Pythoni ja Gazebol omavahel suhelda. Pärast liidese proovimist sai aga selgeks, et antud liides ei tööta Pythoni versiooniga 3, mis on hetkel soovituslik Pythoni versioon.

Teiseks võimaluseks on kasutada vahevara. Nii öelda silla ehitamine Pythoni ja Gazebo vahel. Selleks vahevaraks on ROS (*robot operating system*). Ta sisaldab endas väga palju funktsionaalsust. Oluline funktsionaalsus antud töö skoobis on erinevate komponentide omavahelise suhtluse lihtsustamine. Antud töös kasutatakse Gazebo simulatsiooniga andmevahetuse jaoks loodud ROS-i sõlme, mis võimaldab saata käske Gazebo simulatsiooni ja saada tagasisidet simulatsioonis toimuva kohta (mudelite ja liigendite olekute kohta) kasutades ROS-i andmevahetussüsteemi. [6]

3.2 ROS-i kasutamine

ROS kasutab teiste programmidega suhtlemiseks sõnumite süsteemi. ROS ise ei saada teistele sõnumeid, vaid teised programmid saadavad ROS-le sõnumeid. Kui mingi teine programm on huvitatud teatud sõnumitest, läheb ta ise seda konkreetset sõnumiliini kuulama. Seda võib võrrelda teadetetahvli süsteemiga, kus üks osapool paneb teated tahvlile ja teine osapool loeb neid teateid mis talle huvi pakuvad.

Kui osapoolteks on Python ja Gazebo, siis mõlemad nendest saadavad sõnumeid ROS-le ning samas on nad ka teise osapoole sõnumitest huvitatud. Näiteks mootori juhtimine Pythoni skriptiga. Python saadab ROS-le sõnumi, mis sisaldab endas järgmist käsku mootorile. Gazebo samal ajal kuulab mootorile suunatud sõnumiliini. Kui Pythonilt tuleb uus sõnum, saab Gazebo selle kätte ning kui sõnum on õiges formaadis, osatakse seda simulatsioonis rakendada. Sõnumiliine eraldatakse nime järgi. Vastupidises suunas liiguvad andmed värvianduri väärtuse küsimisel. Pythoni programm kuulab värvianduri sõnumiliini. Samal ajal saadab Gazebo ROS-le värvianduri sõnumeid. Kohale jõudes kuuleb seda Pythoni programm ning saab sõnumi kätte.

Arvestada tuleb sellega, et Gazebo sõnumid on kindlas formaadis ning oluline on Pythonis uue sõnumi loomisel ta Gazebo-le tuntavas formaadis teha. Vastasel juhul ei ole Gazebo-l võimalik edukalt sõnumit töödelda. Seda võib võrrelda sellega, et Python räägib kõiki maailma keeli, kuid Gazebo räägib ainult Gazebo keelt. Sõnumi saatmise lihtsustamiseks on Pythonis olemas Gazebo sõnumite teek, mis lubab lihtsasti Gazebo sõnumeid luua. Jooniselt 3 võib näha ühe Gazebo sõnumi näidet.

```
header:
  seq: 341
  stamp:
    secs: 34
    nsecs: 200000000
  frame_id: /sonar_link
radiation_type: 0
field_of_view: 0.0
min_range: 0.0289999991655
max_range: 2.54999995232
range: 1.41587138176
```

Joonis 3. Gazebo sõnumi näide. Antud juhul on tegemist ultraheliandurilt saadud sõnumiga.

4 Mootori simulatsioon

Roboti töö simuleerimise juures on kindlasti kõige olulisem tema liikuma panemine. Füüsilisel robotil on rataste liikuma panemiseks mootorid. Simulatsioonis sellised mootorid puuduvad, seega peame ise rataste liigenditele kiiruse määrama.

4.1 Mootori töörežiimid

EV3 roboti mootorit saab mitmel viisil liikuma panna. Mootor omab sisemisi väärtusi, näiteks vaikimisi kiirus, aeg, kiirendus, mida saab muuta. Kusjuures kiirus võib olla ka negatiivne, sellisel juhul paneb mootor ratta tagurpidi pöörlema. Seejärel mootorile käske andes kasutab ta neid sisemisi väärtusi. Mootori võib panna igavesti pöörlema või mingiks ajaks. Enamikus režiimides ei saa kiirust muuta ilma uut käsku välja kutsumata, kuid otse jooksutamise režiimis on see võimalik. Simulatsioonis on implementeeritud igavesti, ajaline ja otse jooksutamise režiimid.

Mootoril on seisma jäämiseks erinevad režiimid. Neid režiime kasutatakse seismapaneku käskude jaoks. Pidurdusrežiimis jääb robot pidurdades seisma. Vabakäigu režiim ei pidurda, vaid lülitab mootori välja nii, et robot veereb vabalt seiskumiseni. Simulatsioonis kasutame me ainult vabakäigu režiimi, kuna nende kahe režiimi vahe ei ole suur ning ülesannetes ei ole otseselt vaja nii täpselt sõita, et pidurdamine kasulik oleks. Füüsilise roboti rattad on piisavalt jäigad, et vabalt veeredes jõuab ta edasi liikuda suhteliselt vähe.

EV3 mootoril on lisaks veel mõned võimalikud režiimid näiteks tingimuse ootamine, liikumine absoluutsesse või suhtelisse positsiooni, kuid neid me simulatsioonis ei realiseeri.

4.2 Arendus simulatsioonis

Gazebos puudub otsene vajadus mootorit kui sellist simuleerida. Antud hetkel huvitab meid ratta ja liigendi vaheline pöörlemiskiirus, mitte otseselt mootori detailne simulatsioon. Gazebos on võimalik mootorile pöörlemiskiirus anda, mis tähendab, et meil

on võimalik mootori tööd simuleerida. Olemasolevad mootori käsud tuleb teisendada sobivaks Gazebos ratta liigendile rakendamiseks.

Et Pythoni koodiga rattale käske anda, tuleb selleks luua pistikprogramm [7], mis võtab vastu Pythoni koodi käsu ning edastab selle Gazebos rattale. Robotil on kaks ratast ning mõlemat neist peab saama üksteisest sõltumatult juhtida. Selleks seome mõlema ratta eraldi pistikprogrammiga. See tähendab ka seda, et ratastele antud käske tuleb eraldi lõimedes teostada. Ainult nii on võimalik kahele rattale samaaegselt käske anda. Ratas Gazebos võtab sisendiks nurkkiiruse radiaanides. EV3 mootorile antakse sisendiks nurkkiirus kraadides, seetõttu on pistikprogrammis realiseeritud teisendus kraadidest radiaanidesse.

Kui rattale edastatud käsud mootori poolt õnnestuvad, tuleb mootori töörežiime hakata simuleerima. Lihtsaim neist on tõenäoliselt mootori igavesti jooksutamise režiim, kus algoritm on realiseeritud lõpmatu tsüklina, kus iga tsükli sammul edastatakse etteantud sagedusega ettenähtud kiiruse. Sageduse saab ise määrata, empiiriliste katsete tulemusena osutus sobivaks sageduseks 60 Hz. Ajaliselt jooksutamise režiim ei ole väga palju raskem. Tuleb teha analoogne tsükkel igavesti jooksutamise režiimiga, kuid ajalise piiranguga.

Veidi töömahukam on režiim, mis on oma loomult sarnane igavesti jooksutamisega, kuid antud režiimi kiirust saab jooksutamise ajal muuta – otse jooksutamise režiim. Eelmiste töörežiimide puhul peab kiiruse muutmiseks uue käsu välja kutsuma. Seega tuleb tsüklis silma peal hoida ka kiirusel, mida kasutaja muuta võib. Kusjuures kiirus antakse selle režiimi puhul protsendina. See protsent võetakse rataste maksimaalsest võimalikust pöörlemistkiirusest. Katsetamise teel sai selgeks, et ratta maksimaalkiirus varieerub vahemikus 780 kuni 820 kraadi sekundis. Seega seati maksimaalkiiruseks 800 kraadi sekundis ning eelnimetatud protsent võetakse 800-st.

Lisaks on vaja simuleerida mootori seisma jäämine ehk siis antud juhul vabakäiguga veeremine. See nõuab katsetamist päris mootoritega. Katsetamise tulemusena jõuti järeldusele, et vabakäigul aeglustub ratas iga sekund umbes 5 radiaani sekundis kuni seisma jäämiseni. Selline väärtus sai simulatsioonis vabakäigu režiimile lisatud. Seismajäämise käsku saab kasutaja kas ise välja kutsuda või mootor teeb seda automaatselt siis kui mootorit jooksutatakse aja režiimis ning määratud aeg on läbi saanud.

4.3 Simulatsiooni testimine

Mootori simulatsiooni juures on palju asju, mida testida. Kas kiirused vastavad füüsilise mootori väärtustele, kas kõik režiimid ja käsud töötavad nii nagu peavad. Kiirust saab testida mingi aja jooksul läbitud vahemaa mõõtmisega nii simulatsioonis kui füüsilise robotiga. Režiimide õigsust saab samuti testida võrreldes füüsilise robotiga.

Kiiruse testimisel on kasulik panna mõlemad rattad sama kiirusega samaaegselt liikuma. Sellisel juhul liigub robot otse ning võimalik on lihtsasti välja arvutada läbitud teepikkus. Kusjuures niimoodi saab testida ka seda, et kas rattaid on võimalik üheaegselt sõitma panna. Näiteks pannakse roboti rattad liikuma kümneks sekundiks kiirusega 360 kraadi sekundis. See tähendab 1 täispööre sekundis. Niimoodi sooritab ratas kümne sekundi jooksul 10 täispööret ning arvestades, et roboti ratta raadius on 2.8 sentimeetrit, siis teoorias peaks robot läbima umbes 175.9 sentimeetrit pluss pidurdusteekond umbes 11 sentimeetrit. Simulatsioonis läbis robot keskmiselt kokku umbes 170 sentimeetrit. Füüsiline robot läbis keskmiselt 180 sentimeetrit. Füüsiline robot läbis sama ajaga 5.5% rohkem. Selline erinevus on vastuvõetav antud aines lahendatavate ülesannete kitsendusi arvesse võttes.. Lisaks võib testida näiteks kui kaua võtab mingil kindlal kiirusel robotil aega teha n arv täispööret. Siin on abiks güroandur, mis mõõdab roboti orientatsioon ning tähendab, et meil ei ole tarvis silma järgi vaadata ja hinnata. Simulatsioonis võttis 5 täispöörde jaoks kiirusel 360 kraadi sekundis aega umbes 25 sekundit, millest võtame 10% maha, kuna simulatsioon jooksis 90 protsendise kiirusega võrreldes reaalajaga. See teeb umbes 22.5 sekundit. Füüsilisel robotil kulus samade parameetritega umbes 20 sekundit. See tähendab, et simulatsioonis kulus katse sooritamiseks 12.5% kauem aega. Ka see erinevus on ülesannete kitsendusi arvesse võttes vastuvõetav. Seega tuli katsetest välja, et robot on simulatsioonis veidi aeglasem kui päris robot.

Testida tuleb veel, et uue käsu sisestamisel lõpetatakse eelmise käsu töö. Seda testitakse iga käsu kohta. Testiti veel, kas mootori sisemised muutujad salvestuvad õigesti. Ajalise jooksutamise režiimi puhul kontrollin, kas mootorit jooksutatakse tõesti nii pikalt. Igavesti jooksutamise režiimi puhul ei saa igavesti jooksmist kontrollida, kuid mõne minuti möödudes võib ilmselt kindel olla, et mootor jookseks ka edasi. Muutuva kiiruse režiimis kontrollin, kas tõepoolest on võimalik kiirust käsu keskel muuta. Kontrollin, kas vabakäiguga peatumine toimub pärast ajalist jooksutamise režiimi ning teda eraldi välja kutsudes.

5 Andurite simulatsioon

Kui mootori simuleerimisel on oluline see, kuidas rattad kasutaja poolt sisestatud käskudele reageerivad, siis andurite simuleerimisel on oluline see info, mida kasutajale simulatsioonist tagastatakse. Simulatsioonis ei ole implementeeritud andurite kogu funktsionaalsust, kuna füüsilisel roboti anduritel esineb funktsionaalsust, mida tudengil ülesannete lahendamiseks vaja ei lähe.

5.1 Värvianduri simulatsioon

EV3 värviandur (*Color sensor*) suudab tuvastada kaheksat erinevat värvi. Samuti suudab tuvastada värvi tooret väärtust RGB (*red, green, blue*) koodis, tagasipeegeldunud valguse intensiivsust ning ümbritseva valguse intensiivsust. Üldiselt on ta suunatud maha, et tuvastada lähedal maas olevaid värve.

5.1.1 Võimalikud väljundid

Värvituvastus töörežiimis annab värviandur väljundiks numbri vahemikus 0-st 8-ni. Numbrid 1 kuni 7 tähistavad vastavalt järgmisi värve: must, sinine, roheline, kollane, punane, valge ja pruun. Number 0 tähistab varianti *No color*, mis tähendab, et andur ei tuvastanud ühtegi eelnevalt loetletud värvi. Minu poolt simuleeritud värviandur tuvastab kõiki loetletud värve, välja arvatud kollane ja pruun.

Värvianduri RGB režiim pakub väljundiks vaadeldava pinna punase, rohelise ja sinise komponendi väärtuse täisarvuna vahemikus 0 kuni 255. Mida heledam on pind, seda suuremad on nende arvude väärtused. Värviandur simulatsioonis katab RGB režiimi kogu funktsionaalsuse.

Peegeldunud valguse intensiivsuse režiim annab väljundiks täisarvulise protsendi vahemikus 0-st 100-ni, mis näitab kui hele või tume on vaadeldav pind. Mida heledam on pind, seda suurem on tagastatav arv. Simuleeritud värviandur katab kogu antud funktsionaalsuse.

Ümbritseva valguse režiim annab samuti väljundiks täisarvulise protsendi 0-st 100-ni. Näitab, kui hele või tume on ümbritsev valgus. Jällegi, mida suurem see arv on, seda heledam on ümbritsev valgus. Simulatsioonis seda töörežiimi ei kasutata.

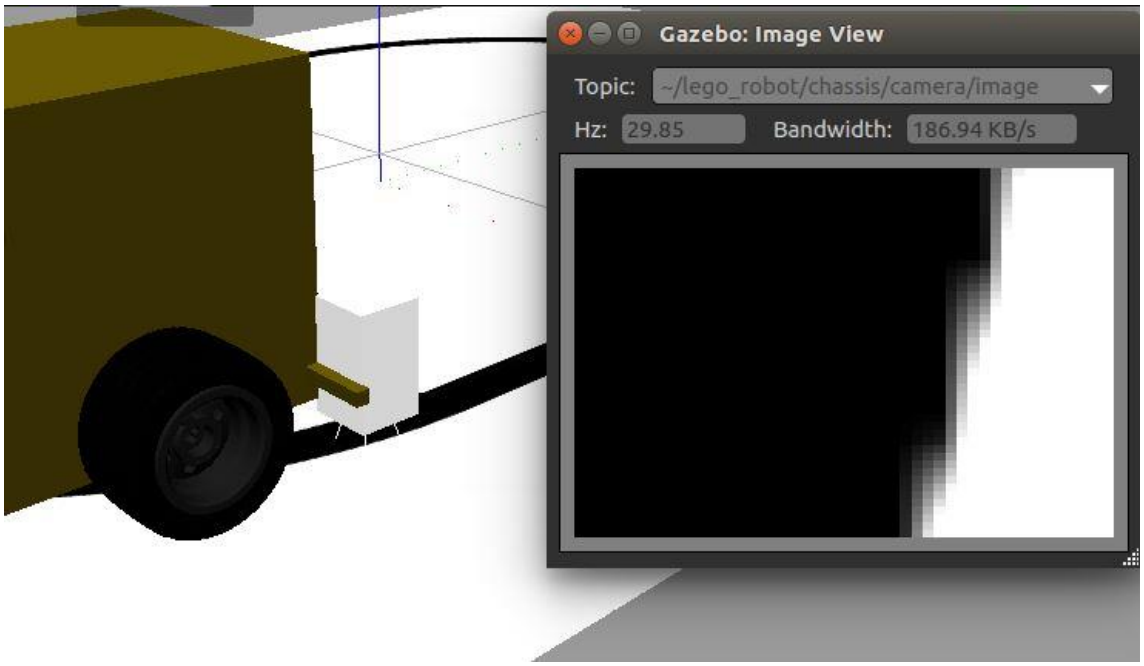
5.1.2 Arendus simulatsioonis

Värviandurit simuleerida Gazebos oli neljast kasutatavast andurist kõige keerulisem. Gazebos ei ole värviandurit, seega loogiliselt tuleb meil see muudest olemasolevatest materjalidest valmistada.

Kõige sobilikum olemasolev vahend, millega oleks võimalik värviandurit simuleerida, on kaamera. Kaamera Gazebos annab väljundiks tuvastatud pildi pikslite info. See tähendab, pildi iga piksli kohta on antud tema RGB väärtus. Seda infot kasutades ongi meil võimalik uuritava pinna värv ja tagasi peegelduva valguse protsent määrata.

Gazebo kaameralt saadud info töötlemine on kindlasti võrreldes teiste sensoritega kõige aeganõudvam. Et simuleeritud andur päris värvianduriga võimalikult sarnaselt töötaks, on oluline optimeerida töödeldavat andmehulka. Andmehulga suurus peaks olema võimalikult väike (kuid nii palju kui vaja, et tulemused oleksid piisavalt täpsed) ja et seda andmehulka võimalikult kiiresti töödeldaks.

Gazebo roboti mudelifailis on võimalik määrata kaamera resolutsioon. Pärast katsetamist sai selgeks, et sobilik resolutsioon on 48x48 pikslit (vaata ka joonist 4). Eesmärk oli leida võimalikult väike resolutsioon, mis kuvaks pilti veel üsna täpselt. Väiksemad resolutsioonid hakkasid kaamerapilti tugevalt moonutama, näiteks mõned valged pikslit kuvati punaste, roheliste ja siniste pikslitega.



Joonis 4. Värvianduri väljund Gazebo simulatsioonis resolutsioonil 48x48 pikslit.

Samuti võib resolutsioonist ning jooniselt tähele panna, et kaamerapilt on oma kujult ristkülik. Füüsiline värviandur aga mõõdab pilti, mis on ringikujuline. Geomeetria valemeid rakendades saab kõrvale jätta need pikslid, mis ringi sisse ei kuulu. Kindlasti tuleb kaamerapildi töötlemist teha eraldi lõimes, et ülejäänud programmi töö ei segaks. Töötlemine sai siis selliselt lahendatud, et kaamerapildi infot töödeldakse pidevalt ning salvestatakse eraldi lõimes. See tähendab, et konkreetse töörežiimi väärtuse saab klassimuutujast ilma viiteta kätte..

Samas ei ole ka värvianduri töörežiimide simuleerimine kõige triviaalsem. Värvituvastus režiim on nendest kõige raskem. Uuritava pildi värvi määramiseks on mitu võimalust. Näiteks võib pildi kõikide pikslite keskmise võtta ja selle ligikaudne värv määrata. Samas võib igat pikslit eraldi uurida ning seejärel vaadata, millist värvi piksleid kõige enam esines. Valiti viimane variant.

Piksli värvi määramine on raske, kuna meil on vaja tuvastada 5 erinevat värvi ning värviskaalal ei ole selgelt ära defineeritud, mis osa loetakse mustaks, mis valgeks jne. Kuna sõnades on suhteliselt raske seletada, kuidas värve jagati, siis esitab autor enda poolt kirjutatud koodi koos seletavate kommentaaridega joonisel 5.

```
if red_pixel + green_pixel + blue_pixel < 160 and red_pixel < 80 and
       green_pixel < 80 and blue_pixel < 80:
            # tingimus musta värvi tuvastamiseks
             color_dict['black'] += 1
elif red_pixel + green_pixel + blue_pixel > 600:
            # tingimus valge värvi tuvastamiseks
             color_dict['white'] += 1
elif red_pixel > green_pixel + blue_pixel or (red_pixel + green_pixel +
       blue_pixel > 500 and red_pixel > green_pixel and red_pixel >
       blue_pixel):
            # tingimus punase värvi tuvastamiseks
             color_dict['red'] += 1
elif green_pixel > red_pixel + blue_pixel or (red_pixel + green_pixel +
       blue_pixel > 500 and green_pixel > red_pixel and green_pixel >
       blue_pixel):
            # tingimus rohelise värvi tuvastamiseks
             color_dict['green'] += 1
elif blue_pixel > green_pixel + red_pixel or (red_pixel + green_pixel +
       blue_pixel > 500 and blue_pixel > red_pixel and blue_pixel >
       green_pixel):
            # tingimus sinise värvi tuvastamiseks
             color_dict['blue'] += 1
else:
            # mitte ühtegi värvi ei tuvastatud
             color_dict['none'] += 1
```

Joonis 5. Pythoni kood pikslile värvi määramiseks. Piksel koosneb punasest, rohelisest ja sinisest värvist, mille väärtused on vahemikus 0 kuni 255.

Nagu jooniselt näha, siis umbes sellised on vastavate värvide piirkonnad. Välja jäänud piirkond tuvastatakse värviks '*No color*'. *No color*-i osakaal on suhteliselt suur, kuid see sobib meie kontekstis, kuna värviandurit kasutatavates ülesannetes on värvid väga konkreetsed ning sellist halli ala ei tohiks tekkida.

Ülejäänud režiime saab õnneks lihtsate arvutustega simuleerida. Arvutades välja pildi punase, rohelise ja sinise sisalduste keskmised, saama tagastada pildi RGB väärtuse RGB režiimi jaoks. Samast väärtusest saame ka välja arvutada peegeldunud valguse intensiivsuse. Antud juhul on eeldatud, et RGB on peegeldunud valguse intensiivsusega lineaarses seoses. See tähendab, et kui RGB väärtus on 0, 0, 0 (must värv), siis peegeldunud valgus on samuti 0%. Kui RGB väärtus on 255, 255, 255 (valge), siis peegeldunud valgus on 100%.

5.1.3 Simulatsiooni testimine

Värvianduri testimisel on abiks erinevat värvi plaadid, mida saab Gazebos värvianduri alla liigutada. Plaadi all peab autor silmas väga õhukest eset, mis mahuks värvianduri alla

ning millest robot saaks lihtsasti üle sõita. Punase, rohelise, sinise, valge ja musta plaadi peab värviandur ära tundma. Iga loetletud värvi kohta on vaja ka vaadata veidi erinevaid varjundeid. Ülejäänud värvid peab andur tunnistama *No color*-iks.

RGB režiimi testimine käib samamoodi plaatidega. Kui varem on teada loodud plaadi värv RGB väärtusena, siis on seda hea võrrelda värvianduri poolt tuvastatud RGB väärtusega. Need väärtused peavad kokku langema. Autor eemaldas Gazebos esemete poolt tekitatud varjud, et andur ei tekitaks kaamerapildis varju. Füüsilisel robotil on selle jaoks tuluke. Samuti tuleb testida tulemust sellisel juhul kui värvianduri vaateväljas on kaks värvi. Kui näiteks värvianduri pildis on täpselt pooleks must ja valge värv, siis väljundiks peab värviandur andma keskmise hallitooni. RGB-s on see 127, 127, 127.

Peegeldunud valguse režiimi on hea testida halliskaala värvi olevate plaatidega. Võib ka muude värvidega, kuid kõige lihtsam on halliskaala värvidega testimine. Täiesti valge pinna tuvastamisel peab tulemus olema 100% ning täiesti musta tuvastamisel 0%. Pind RGB väärtusega 127, 127, 127 annaks loogiliselt väljundiks 50%. Samamoodi kui pool pinda on täiesti valge ja pool pinda täiest must, siis peab väljund olema 50%. Kusjuures need protsendid peavad olema täisarvud.

Töö käigus tekkis mul võimalus testida peegeldunud valguse režiimi sügavamalt tänu ühe tudengi poolt loodud koodile aine esimese ülesande jaoks. Ülesanne näeb ette musta joone järgimist valgel taustal. Tulemuse illustreerimiseks tegi autor video.[1]

5.2 Ultraheliandurи simulatsioon

Ultraheliandur (*Ultrasonic sensor/sonar*) tuvastab vahemikus 3.0 cm kuni 255.0 cm asuvat objekti. Ultraheliandur kasutab ultraheli laineid kauguse mõõtmiseks. Lained saadetakse välja koonuseliselt täisnurkse tippnurgaga. Tuvastab ainult selle objekti, mis asub andurile kõige lähemal. Andurit kasutatakse seinte ja erinevate objektide tuvastamiseks.

[1] https://youtu.be/W3xonkiS2Z4

5.2.1 Võimalikud väljundid

Anduril on kaks kauguse mõõtmise režiimi. Esimene neist on sentimeetrites mõõtmise režiim. Kuigi nime järgi võib oletada, et mõõtmise väljundiks antakse tulemus sentimeetrites, siis tegelikult antakse tulemus millimeetrites ümardatud täisarvuni. Simulatsioonis on selle režiimi funktsionaalsus implementeeritud.

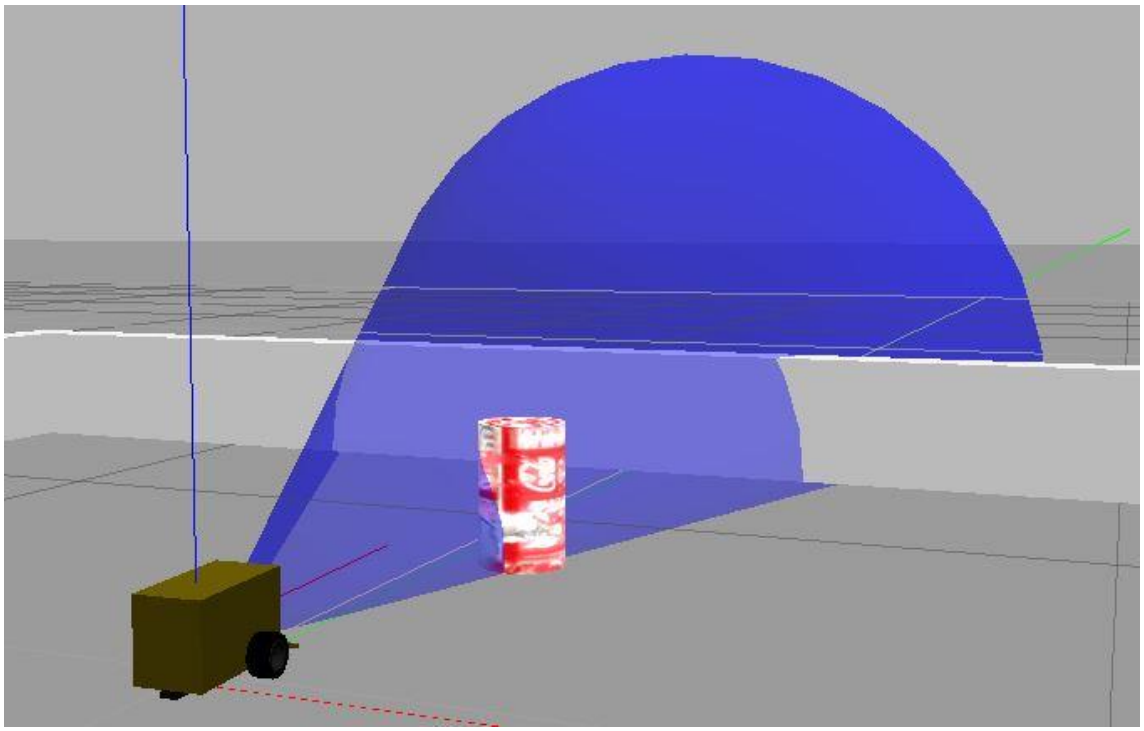
Teine kauguse mõõtmise režiim on tollides mõõtmise režiim. See režiim on analoogne eelmise režiimiga. Väljundiks antakse mõõtmise tulemus tollides, mis on korrutatud kümnega, et vastuseks oleks alati täisarv. Ka see režiim on simulatsioonis implementeeritud.

Lisaks on anduril teise ultraheliandurituvastamise režiim. Väljundiks on tõeväärtus 0 või 1. Kui väljundiks on 1, siis andur tuvastab läheduses mõne teise ultraheliandurituri. Kui väljundiks on 0, siis teisi ultraheliandureid ei tuvastatud. Seda funktsionaalsust simulatsioonis ei implementeerinud, kuna selle infoga ei ole tudengil midagi peale hakata.

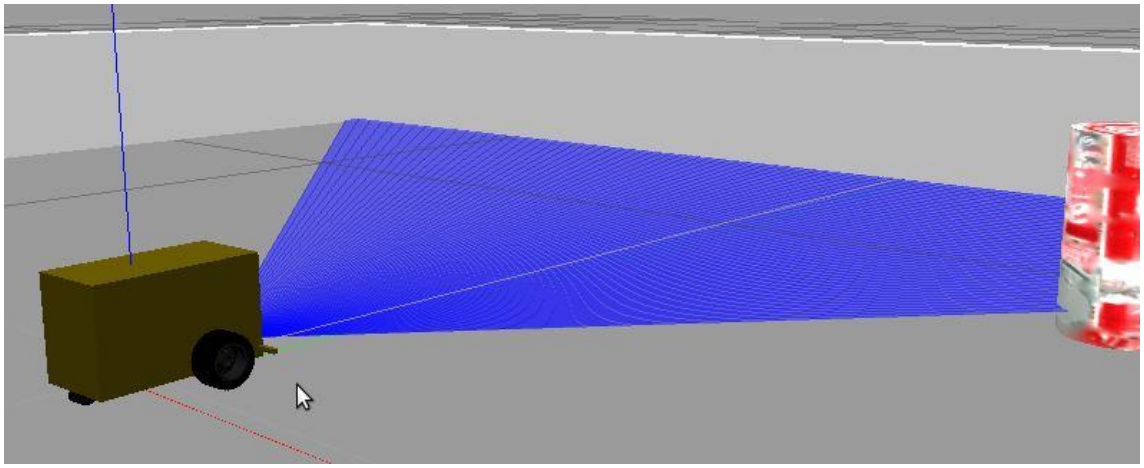
5.2.2 Arendus simulatsioonis

Gazebo simulaatoris on olema ultraheliandur, kuid see ei ole väga usaldusväärne ning kuvas tihti valet infot. Alternatiivselt on võimalik kasutada laserandurit, mis saadab välja soovitud suunda lasereid ning need mõõdavad kaugusi. Esialgu kuvab laserandur tulemuseks iga individuaalse kiire mõõdetud kaugust, kuid kasutades Hectori meeskonna ultrahelisensori pistikprogrammi [8], saame teada kõige väiksema mõõdetud tulemuse. Antud pistikprogramm vaatab läbi kõikide laserite poolt mõõdetud tulemused ning leiab nendest väikseima. Niimoodi saabki simuleerida ultraheliandurituri tööd.

Erinevus füüsilise ultraheliandurituri ja simulatsioonis ehitatud anduri vahel on see, et kui füüsiline andur tuvastab esemeid tema ees olevas koonuses (vaata joonis 6), mille tippnurgaks on 90 kraadi, siis simulatsiooni andur tuvastab ainult neid esemeid, mis on temaga sama kõrgel, kuna andur saadab lasereid välja paralleelselt maapinnaga (vaata joonis 7). See tähendab, et simulatsiooni andur ei tuvasta esemeid, mis on temast kõrgemal, näiteks midagi laes rippuvat. Aga kuna antud olukorras ei ole ühtegi taolist eset vaja tuvastada, siis sobib andurit selliselt simuleerida.



Joonis 6. Füüsilise roboti ultraheliandur saadab lained välja koonuseliselt.



Joonis 7. Ultraheliandur simulatsioonis. Lained saadetakse välja ainult samal kõrgusel.

5.2.3 Simulatsiooni testimine

Ultraheliandurite testimise simulatsioonis juures on olulised just piirjuhud. Kui ühtegi eset ei ole anduri poolt tuvastatud, peab väljundiks olema anduri maksimaalväärtus, milleks on 2550 millimeetrit. Tollides on see 100.4, kuid EV3 robot kuvab väljundiks 1004, et tegemist oleks täisarvuga. Samamoodi peab kuvama ka andur simulatsioonis. Kui andurile on ese lähemal kui 3 sentimeetrit, kuvab andur samuti tulemuseks

maksimaalkauguse. Neid juhte on Gazebo simulatsioonis keskkonnas lihtne testida, kuna simulatsiooni elemente saab lihtsasti liigutada erinevatesse positsioonidesse.

Muidugi tuleb ka veenduda, et kui andur tuvastab vaateväljas mitu eset, siis tulemuseks kuvatakse kõige lähema eseme kaugus. Kui selleks juhtub olema ese, mis on lähemal kui 3 sentimeetrit, siis kuvatakse maksimumväärtus. Mitme esemega testimine on samuti lihtne, kuna Gazebos saab lihtsaid esemeid simulatsioonikeskkonda tekitada.

5.3 Güroanduri simulatsioon

Güroanduri (*Gyro sensor*) eesmärk on hoida meeles roboti orientatsioon. Andur tuvastab roboti poolt keeratud nurka ning tema nurkkiirust mõõdetaval hetkel. Sellist andurit on kasulik roboti suuna hoidmisel või leidmisel kasutada. Näiteks labürindis liikudes on kasulik teha täisnurkseid manöövreid.

5.3.1 Võimalikud väljundid

EV3 güroanduril on kaks põhilist töörežiimi. Nurga mõõtmise ning nurkkiiruse mõõtmise režiim. Nurga mõõtmise režiimis annab andur väljundiks nurga täisarvuna kraadides, mille võrra ta mõõtmise alghetkest pööranud on. See tähendab, et väljundiks võib olla nii positiivne arv kui ka negatiivne arv. Kui andur pöörab päripäeva, suureneb väljastatav arv ning pöörates vastupäeva, see arv väheneb. Simulatsioonis on nurga mõõtmine implementeeritud.

Nurkkiiruse mõõtmisel väljastab andur nurkkiiruse, millega andur end parajasti pöörab. Jällegi, võib see arv olla nii positiivne kui negatiivne. Ühikuks on kraadi sekundis ning väljastatakse täisarv. Ka nurkkiiruse mõõtmine on simulatsioonis implementeeritud.

Anduril esineb ka kolmas režiim, kuid see on esimese kahe kombinatsioon. Väljundiks on nii läbitud nurk kui ka nurkkiirus. Oluline on teada, et ühelt töörežiimi pealt teisele minnes nullitakse läbitud nurk.

5.3.2 Arendus simulatsioonis

Gazebo simulaatoris ei ole olemas güroandurit, mistõttu tuleb see ise teha. Tänu sellele, et Gazebo simulaator hoiab meeles kõikide elementide hetke koordinaadid ning orientatsioonid, siis ei ole väga keeruline seda infot kasutades güroandurit simuleerida.

Ainus info, mida peame Gazebolt küsima, ongi anduri orientatsioon. Vastuseks saame kvaterniooni, millest saame teisenduste abil kätte anduri nurga kraadides. Gazebos suureneb nurk vastupäeva liikudes, mis on vastupidi meie soovitud tulemusele. Seetõttu on tarvis veel mõned arvutused teha, et anduri simulatsioon töötaks õigesti. Tuleb tähele panna, et Gazebo annab nurga pärimisel tulemuseks arvu 0-st 360-ni, ehk siis täispöörde saavutamisel hakkab nurk uuesti nullist loenduma. Güroandur aga ei loe täispöördeid. Samuti on oluline, et Gazebost saadud tulemus on nurk Gazebo keskkonna nulltelje suhtes. Füüsiline andur seab nulliks nurga, mis hetkest mõõtmisi alustati.

Nurkkiiruse arvutamise saab samade andmete pärimisega välja arvutada. Lisaks on tarvis meeles pidada mõõtmiste sooritamise aegu, et siis viimase ja sellele eelneva mõõtmiskorra aegade vahe kasutada viimase etapi nurkkiiruse arvutamiseks.

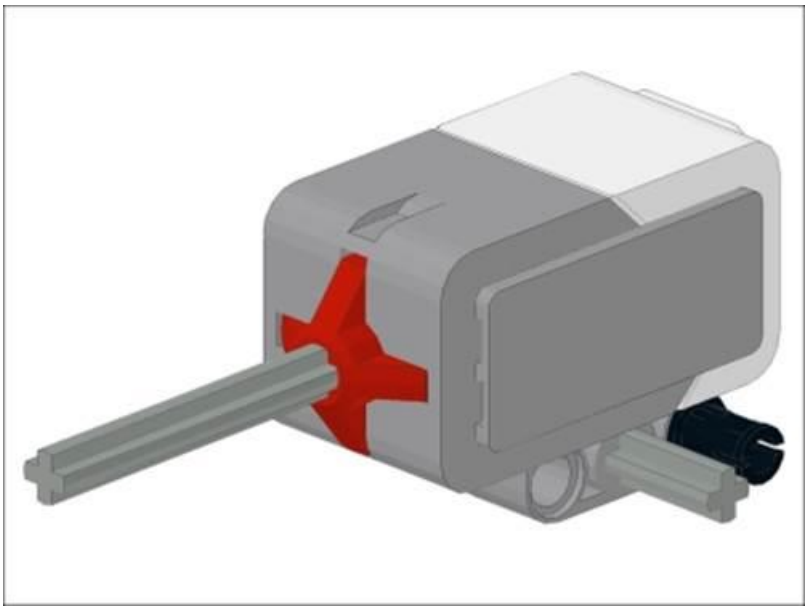
5.3.3 Simulatsiooni testimine

Güroanduri simuleerimisel tuleb testida väljundite õigsust ja täpsust. Kõik väljundid peavad olema täisarvud. Anduri orientatsiooni saab lihtsasti testida näiteks täispöörde saavutamisel seisma jäädes, kontrollimaks et roboti orientatsioon langeb kokku tema algorientatsiooniga. Seda saab teha nii päripäeva kui ka vastupäeva pöörates.

Nurkkiirust saab testida sättides roboti liikuma konstantse nurkkiirusega. Seda saab näiteks saavutada pannes pöörlema ainult ühe ratta. Seejärel nurkkiirust erinevate intervallide järel pärides peab väljund olema läbivalt sama. Lubatud on paari kraadised kõrvalekalded, kuna programmi töös võib tekkida väikseid ajalisi kõikumisi.

5.4 Puuteanduri simulatsioon

Puuteandur (*Touch sensor*) on loomult lihtne: ta tuvastab, kas tema küljes asuv nupp on sisse lülitatud või mitte. Antud nupu normaalseisund on välja lülitatud. Nupu sisse lülitamiseks peab nuppu hoidma, nupu lahti lastes läheb nupp tagasi normaalasendisse. Nupu külge võib panna erinevate pikkustega lego klotse, mis pikendavad nupu haaret (vaata joonis 8). Puutesensorit saab kasutada näiteks seinte ja muude suuremate esemete tuvastamisel. Robot ei tohi olla suuteline neid esemeid eemale lükkama.



Joonis 8. Puuteandur. Anduri punane osa on liikuv. Nupu külge saab panna klotsi haarde pikendamiseks.

5.4.1 Võimalikud väljundid

Puuteanduri funktsionaalsus on elementaarne. Tema ainus töörežiim tuvastabki nupu vajutust. Väljundiks on tõeväärtus 0 või 1. Kui nuppu ei ole hetkel sisse vajutatud, on väljundiks 0. Kui nupp vajutada sisse, muutub väljund 1-ks. Simulatsioonis on see funktsionaalsus realiseeritud.

5.4.2 Arendus simulatsioonis

Gazebo simulatsioonikeskkonnas ei ole puuteandurit, seega on tarvis see ise ehitada ning simuleerida selle tööd. Abiks oleks midagi nupulaadset, kuid kahjuks ei ole Gazebos ka sellist varianti.

Gazebos on olemas selline liigenditüüp nagu *prismatic joint*. Selline liigenditüüp lubab ühel lülil liikuda mööda ühte telge edasi-tagasi, sarnaselt nupule. Mudeli XML failis [9](XML – *extensible markup language*) on võimalik määrata lüli liikumissuund (telg) ja liikumisruum. Tehti mõõtmised päris roboti peal ning tuli välja, et nuppu saab 4 millimeetrit sisse lükata. Vastavalt kirjutati ka mudelifaili. Füüsilise puutesensori küljes on nupp, mis on tegelik liikuv komponent ning sinna nupu külge saab panna lego klotsi nupu pikendamiseks. Simulatsioonis jäeti välja nupu komponent ning hoopis klots tehti liikuvaks komponendiks, kuna puutesensorit kasutades on tal alati küljes pikendav klots.

Kõige olulisem koht puutesensori juures on ära tunda, millal nupp on sisse vajutatud ja millal mitte. Füüsilise roboti peal tehti katseid ning mõõtmisi. Ilmnes, et robot annab väljundiks tõeväärtuse 1 siis kui nupp on vajutatud sisse vähemalt 2 millimeetrit tema liikumise suunas. See on pool nupu kogu liikumise ulatusest. See tähendab, et simulatsioonis on nupu vajutuse tingimuseks see, et liikuv klots peab olema vähemalt 2 millimeetrit sisse lükatud. Seda tingimust saab lihtsasti kontrollida kasutades liikuva klotsi ja anduri enda koordinaate.

Teine oluline nüanss puutesensori juures on see, mis tüüpi nupp ta on. Nupu lahti lastes läheb ta tagasi algasendisse. Selle simuleerimiseks on tarvis lisada liikuvale klotsile teatud suurusega jõud. See jõud peab olema piisav, et roboti liikumisel püsib ta paigal. Samas peab olema see jõud piisavalt väike, et kui sensori liikuv klots jõuab seinani, siis see klots peab sisse liikuma.

5.4.3 Simulatsiooni testimine

Puutesensori juures on oluline testida, et tema poolt väljastatav väärtus oleks alati kas 0 või 1. 0-se väärtuse korral peab klots olema sisse lükatud vähem kui 2 millimeetrit. 1-se väärtuse korral peab klots olema sisse lükatud vähemalt 2 millimeetrit. Seda saab Gazebos testida klotsi liigutamisega ning vastava väljundi võrdlemisega.

Tuleb ka jälgida, et roboti järsu edasi liikumise peale ei lülituks nupp sisse. Seda kontrollitakse andes roboti ratastele maksimaalse lubatud kiiruse. Vastu seina või muud eset sõites peab nupp sisse lülituma, ka väiksematel kiirustel. Seda saab lihtsasti kas seinaga või mingi suurema esemega testida.

6 Töötav keskkond

Lisaks roboti ehitamisele ja simuleerimisele on tarvis valmis teha keskkond, kus tudeng saab oma koodi lihtsasti käima panna. Arvestama peab sellega, et tegemist on esmakursuslastega, seega on Linuxi keskkond neile võõras ning nende poolt nõutavad tehingud peaksid olema minimaalsed ning võimalikult lihtsad. Samuti tuleb neile luua juhend, kuidas keskkonnas toimida.

6.1 Keskkonna ülesseadmine

Keskkond on arendatud Linuxi keskkonnas *VirtualBox*-i virtuaalmasinas. See tähendab, et tudeng peab keskkonnas töötamiseks endale alla laadima virtuaalmasinate halduri. Näiteks *VirtualBox* või *VMware*. Samuti peab tudeng endale alla laadima autori poolt loodud .ova laiendiga faili, mis sisaldab endas virtuaalmasina infot ning seal loodud keskkonda.

Arvestada tuleb sellega, et kõikidel tudengitel ei ole väga võimsad arvutid, seega on loodud keskkonna algsätted suhteliselt madalad. Madalate sätete juures tuleb siiski arvestada, et simulatsioon jookseb suhteliselt aeglaselt. Näiteks proovis autor 2 gigabaidise mäluga (RAM – random access memory) ning ühe protsessoriga. Simulatsioon jooksis 50%-se kiirusega võrreldes reaalajaga ning kasutajaliides kuvas umbes 4 kaadrit sekundis. Võimsamate arvutitega tudengitel on võimalus neid sätteid muuta kõrgemaks ning tekib ka võimalus simulatsiooni jooksutada ülikooli arvutiklassides.

Virtuaalmasina käivitades peab toimima käsk, mis tõmbab minu loodud Git-i salve värskemad failid. See salv sisaldab endas Gazebo mudeleid, pistikprogramme, Gazebo käivitamisfaile ning vajalikke Pythoni ja käsukeele skripte. Ühesõnaga kõike, mida antud keskkonnas simulatsiooni jooksutamiseks tarvis on.

Seejärel saab tudeng virtuaalmasinas loodud kasutajasse nimega ’Student’ sisse logida. Sellel kasutajal on valmis seatud keskkonna muutujad (*environment values*), et kõik Gazebo ja ROS-i käsud töötaksid.

6.2 Tudengi koodi testimine

Tudengile kasutamiseks on loodud käsk nimega *'robot_test'*, et ta saaks lihtsasti oma koodi simulatsioonis katsetada. Antud käsk võtab lisaks veel kolm kohustuslikku argumenti. Mõne argumendi puudumisel kuvatakse veateade.

Esimene argument on tudengi Uni-ID. See on vajalik selleks, et antud käsu jooksutamisel tõmmatakse tudengi vastava aine Git-i salvest alla tema kood. Tudeng peab programmi küsimisel autentimiseks enda Git-i parooli sisestama.

Teine argument on ülesande number. Hetke seisuga tuleb selle aines neli ülesannet. Argument määrab, millist ülesannet tudeng soovib simulatsioonis testida. Vastavalt sellele laetakse simulatsioonis kindel robot vajalike anduritega ning käivitatakse vastav tudengi kood. Kui tudengil puudub kood selle ülesande jaoks, kuvatakse veateade.

Kolmas argument määrab, millist ülesande maailmat tudeng testida soovib. Näiteks labürindi ülesande puhul saab valida mitme labürindi testimise vahel (suurem, väiksem, jne.). Kui antud numbriga maailma olemas ei ole, kuvatakse jällegi veateade. Korrektse maailma numbri sisestamisel laetakse Gazebos vastav maailm ning pannakse käima ka tudengi loodud kood. Hetkel on iga ülesande kohta olemas vaid üks maailm. Neid tehakse juurde siis kui saab selgeks, milliseid konkreetseid maailmu vaja on.

7 Kokkuvõte

2017. aasta sügissemestril bakalaureuse informaatika õppekavasse lisanduv „Robotite programmeerimine“ aine pakub tudengitele võimalust programmeerida Lego Mindstorms roboteid. Et tudengitel oleks võimalus kirjutatud koodi testida ka väljaspool praktikumi, tuleb luua simulatsioonikeskkond, kus kood käivitub simuleeritud roboti peal.

Antud bakalaureusetöö käigus sai loodud keskkond tudengi koodi testimiseks ning lego roboti, mootori, värvianduri, ultraheliianduri, güroanduri ning puuteanduri simulatsioon. Arvestatud on asjaoluga, et keskkonda hakkavad kasutama esmakursuslased, mistõttu on keskkonna kasutamine tehtud võimalikult lihtsaks. Keskkond on lihtsasti tudengitele laiali jagatav ning koheselt kasutatav.

Töö simulatsiooniosa on küll kirjutatud spetsiaalselt tulevase õppeaine jaoks, kuid lego simulatsiooni võivad kasutada kõik, kes soovivad mingil põhjusel lego roboti käitumist simulatsioonis katsetada. Kirjutatud kood on kättesaadav avalikus Git-i salves (vaata Lisa 1).

Õppeaine jaoks on veel tarvis ehitada ülesannetele rohkem maailmaid. Seda saab teha siis kui saab täpselt selgeks, millised maailmad iga ülesande kohta tulevad. Lisaks sellele kui rääkida keskkonna edasi arendamise võimalustest, siis üks võimalus on lisada anduritele ja mootorile simulatsioonis kogu füüsilise roboti funktsionaalsus. Lisaks on võimalik simulatsioonile kirjutada automaatteste, mis tähendab, et tudeng ei peaks alati simulatsiooni käima panema, et enda koodi testida. Kuid need on juba sellest lõputöö skoobist väljaspool.

Kasutatud kirjandus

[1] Pure python bindings for ev3dev [WWW] https://github.com/rhempel/ev3dev-lang-python (18.03.2017)

[2] 31313 Mindstorms Lego [WWW] https://www.lego.com/en-us/mindstorms/products/mindstorms-ev3-31313 (08.03.2017)

[3] Friction and Friction Coeficents [WWW] http://www.engineeringtoolbox.com/friction-coefficients-d_778.html (12.03.2017)

[4] Gazebo: Tutorial: Make a mobile robot [WWW] http://gazebosim.org/tutorials?tut=build_robot&cat=build_robot (10.03.2017)

[5] Pygazebo 2.2.1-2014.1 documentation [WWW] http://pygazebo.readthedocs.io/en/latest/ (28.02.2017)

[6] Documentation - ROS Wiki [WWW] http://wiki.ros.org/ (15.03.2017)

[7] Writing and using a simple plugin [WWW] http://wiki.ros.org/pluginlib/Tutorials/Writing%20and%20Using%20a%20Simple%20Plugin (02.04.2017)

[8] hector_gazebo_plugins [WWW] http://wiki.ros.org/hector_gazebo_plugins (15.04.2017)

[9] SDF [WWW] http://sdformat.org/spec (10.03.2017)

Lisa 1 – Viide tehtud töö Git-i salvele

Simulatsiooniks vajalikud failid on üleslaaditud avalikku GitHubi salve. Salve aaddress on https://github.com/rasmusiila/Lego-Gazebo.

Git-i tarkvara olemasolul on võimalik salv endale kloonida käsuga:

```
git clone https://github.com/rasmusiila/Lego-Gazebo.git
```

Samuti võib salve endale tõmmata .zip failina.