TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Rasmus Iila, 176813 IAPM

ROS-IL PÕHINEVATE ROBOTITE SIMULEERIMINE GAZEBO, CARLA JA LGSVL SIMULAATORITES

Magistritöö

Juhendaja: Gert Kanter, MSc

Tallinn 2019

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.

Author: Rasmus Iila

07.05.2019

Annotatsioon

Käesoleva lõputöö eesmärgiks oli võrrelda ROS-il põhinevate robotite simuleerimist Gazebos ning mängumootoritel Unreal Engine 4 ja Unity põhinevaid CARLA ja LGSVL simulaatoreid.

Simulaatorite võrdlus jaotati kolme ossa. Esimeses osas võrreldi simulaatorite lihtsasti mõõdetavaid näitajaid, nagu operatsioonisüsteem, litsents või programmeerimiskeeled. Võrdluse teises osas käsitleti sügavamalt simulaatorite funktsionaalsust ja võimekust. Võrdluse kolmas osa hõlmas simulaatorite koormusteste: kirjeldati mitu teststsenaariumit, mida simulaatorite peal jooksutati. Selle vältel mõõdeti meetrikat, mis näitab kui hästi suudab simulaator valitud stsenaariumiga toime tulla.

Töö teiseks eesmärgiks oli luua tööriist, mis lubab konverteerida ühe simulaatori projekti teise simulaatori kujule. Selline tööriist lubab kasutajal ühe simulaatori pealt teisele migreeruda tunduvalt lihtsamini.

Töö tulemusena valmis põhjalik kolmeosaline võrdlus Gazebo, CARLA ja LGSVL simulaatorite vahel. Samuti loodi konverter, mis lubab kolme uuritava simulaatori vahel projekte muundada.

Töö on kirjutatud eesti keeles ning sisaldab teksti 62 leheküljel, 6 peatükki, 9 joonist, 8 tabelit.

Abstract

Simulating ROS-based robots in Gazebo, CARLA and LGSVL simulators

The primary objective of this master's thesis was to create an informative comparison between the Gazebo simulator and the Unreal Engine 4 and Unity game engine based CARLA and LGSVL simulators. More specifically, the three simulators' ability to simulate ROS-based robots is compared.

The comparison of the simulators was divided into three parts. In the first part, easily comparable and measurable metrics were compared. Examples of those metrics would be operating system, license or programming languages. This kind of comparison could make it clear to the user from the start if they are unable to use a certain simulator due to some limitations. The second part of the comparison contained a deeper analysis of the functionality and the capabilities of the simulators. This included using the simulators with ROS2. The third part of the comparison was the execution of performance tests. Several test scenarios were made and executed on the simulators. Metrics were then measured to see how well the simulators perform with the chosen scenario.

The secondary objective of the thesis was to create a tool that allows for conversion of a simulator's project to be usable in another simulator. This tool allows the user to migrate from one simulator to another more easily.

As a result of this thesis, a thorough three-part comparison was created between the Gazebo, CARLA and LGSVL simulators. In addition to this, a converter was created that allows to convert a project of any of these three simulators to a project of another of the listed simulators.

Present thesis is written in Estonian and is 62 pages long, including 6 chapters, 9 figures, 8 tables.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*; programmiliides on arvutiprogrammides alamprogrammi määratluste, protokollide ja tööriistade komplekt rakendustarkvara ehitamiseks
CC-BY	*Creative Commons*; autoriõiguste litsents
CPU	*Central Processing Unit*; kesksead ehk keskprotsessor on arvuti osa, mis täidab arvutiprogrammide juhiseid ning on peamine vahend arvuti ülesannete täitmisel
DEM	*Digital Elevation Map*; digitaalne kõrgusmudel on kõrgusandmetest koosneb maapinna mudel
DART	*Dynamic Animation and Robotics Toolkit*; avatud lähtekoodiga füüsikamootor
FPS	*Frames Per Second*; kuvatavate kaadrite arv sekundis
GB	Gigabait on infohulga mõõtühik
GHz	Gigaherts on protsessi sageduseühik
Git	Versioonihaldustarkvara
GNSS	*Global Navigation Satellite System*; üle maailma asukohamääramist ja navigatsiooni võimaldav sidesatelliitide süsteem
GPS	*Global Positioning System*; üleilmne asukoha määramise satelliitnavigatsiooni süsteem, mille omanik on Ameerika Ühendriikide valitsus
GPU	*Graphics Processing Unit*; fraafikaprotsessor on 3D- ja 2D-graafika visualiseerimiseks ja kiirendamiseks kohandatud mikroprotsessor
IMU	*Inertial Measurement Unit*; inertsiaalandur on mõõtemuundur, mille tajurid mõõdavad joonkiirust ja pöörlemiskiirust
LIDAR	*Light Detection And Ranging*; laserskaneerimisseade
MB	Megabait on infohulga mõõtühik, 1 GB = 1000 MB
MIT	*Massachusetts Institute of Technology*; MIT litsents on vaba tarkvara litsents
ODE	*Open Dynamics Engine*; vabavaraline füüsikamootor
Q&A	*Question & Answer*; küsimus ja vastus, eksisteerivad Q&A veebilehed, mille eesmärk on võimaldada kasutajatel küsida veebilehele teemakohaseid küsimusi, millele võivad vastuseid pakkuda teised veebilehe kasutajad
RADAR	*Radio Detection and Ranging*; raadiolokaator, mis toimib elektromagnetkiirguse põhimõttel ruumis
RAM	*Random Access Memory*; muutmälu, salvestab sagedasti kasutatud käsud nende kiiremaks kätte saamiseks
RFID	*Radio-Frequency Identification*; raadiolaineid kasutav tehnoloogia esemete märgistamiseks ja nende automaatseks jälgimiseks
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
RTF	*Real-Time Factor*; süsteemi kiiruse mõõtmise suurus
UE4	*Unreal Engine 4*; mängumootor
XML	*Extensible Markup Language*; üldotstarbeline märgistuskeel, mille eesmärk on struktureeritud info jagamine infosüsteemide vahel

Sisukord

Joonistе loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust

Robootika on tänapäeval kiiresti arenev eluvaldkond. Robotid on jõudsalt liikumas inimeste igapäeva ellu: isejuhtivad autod, pakirobotid, tolmuimejad, jne.

Üks populaarsemaid robotite juhtimiseks kasutatavaid raamistikke on ROS [1]. ROS ehk Robot Operating System on vahevara, mis võimaldab robotite tarkvara komponente modulaarselt ühendada.

Robotite arendamisel on oluline nende juhtalgoritme perioodiliselt testida. Sõltuvalt robotist võib tekkida raskusi testida juhtalgoritme füüsilise seadme peal.

Üks võimalusi robotite testimiseks on kasutada simulatsiooni. Simulatsiooni keskkondi on erinevaid, igal keskkonnal on omad plussid ja miinused. Näiteks on MarineSIM simulaator mõeldud veesõidukite simuleerimiseks [2]. Samuti tuuakse artiklis välja argument, et väliskatsete sooritamine on kulukas (eriti robotitele mõeldud vees liikumiseks), seega on oluline võimaldada robotite simuleerimist.

Robotite simulatsiooni ei kasutata alati ainult testimise eesmärgil, vaid ka masinõppeks. Näiteks suudeti simuleeritud viie sõrmega robot panna õppima kuubikut pöörama, seejärel kanti õpitud teadmised üle füüsilisele robotile [3]. OpenAI Gym tööriistaga õpetatakse stiimulõppe meetodit kasutades erinevaid roboteid simulatsioonides erinevaid ülesandeid lahendama, näiteks kõndimine, Pongi mängimine, eseme lükkamine jne [4].

Järgnevas robootika blogi sissekandes [5] tuuakse välja veel mitmeid põhjuseid, miks on kasulik roboteid simuleerida. Üks suur eelis on stsenaarium, kus füüsiline robot ei ole veel täielikult valmis, seega on võimalik arendusega pihta hakata simulatsioonis.

1.2 Probleemi püstitus

ROS-il põhinevate robotite testimiseks on üks populaarsemaid simulatsioonikeskkondi Gazebo simulaator. Gazebo simulatsiooni kasutatakse tihti ratastel robotite simuleerimiseks, kuid üks populaarne valik on ka käe- ja kraanalaadsed robotid [6]. Gazebo eeliseks on asjaolu, et tegemist on vabavaraga. Samas esineb Gazebo keskkonnal erinevaid puudujääke, mis muudavad simuleerimise ebatäpseks.

Gazebo keskkond on üldiselt piisav ühe lihtsa roboti simuleerimiseks. Maailma ja/või roboti keerukust tõstes langeb simulatsiooni reaalaja faktor (RTF - real time factor), kuna Gazebo keskkond ei suuda piisavalt kiiresti vajalikke arvutusi sooritada. Samuti tekib Gazebol raskusi mitme roboti korraga simuleerimisel. Lisaks sellele ilmneb Gazebos mitu väiksemat programmiviga, mis raskendavad roboti simuleerimist, näiteks roboti koha peal kõrgsageduslik resonants või probleemid ebatasasel pinnasel sõitmisega. Need probleemid vähendavad roboti juhtimise algoritmide testimise võimalusi, sest tekivad ulatuslikud erinevused päriselu ja simuleeritud keskkondade vahel. Seetõttu langeb testimise kvaliteet. Samuti on Gazebo üks suuremaid puudujääke hajus-andmetöötluse puudumine (on plaanis tulevikus toetada) - see tähendab, et Gazebo ei skaleeru väga hästi suuremahulistema projektide jaoks.

Seetõttu on kasulik uurida alternatiivsete simulatsioonikeskkondade kasutamist. Üks võimalik valik on mängumootorite kasutamine, ideelt nad omavad sama põhimõtet teiste simulatsioonikeskkondadega - simuleerida füüsikalisi protsesse ja nende vahel mõjuvaid jõudusid, sarnaselt päris maailmaga.

Eesmärk on uurida, kui hästi rakendatavad on UE4 ja Unity mängumootorid ROS-il põhinevate robotite simuleerimiseks. Et hiljuti ilmus ROSi uus versioon ROS2, siis uurimise alla läheb ROS-i uuem versioon ROS2.

Selleks, et UE4 ja Unity mootorites kasutada ROS-il põhinevaid roboteid, peab eksisteerima vastav liidestus. Selline liidestus on tehtud projektides CARLA ja LGSVL [7], [8]. CARLA on Unreal Engine 4 mootori peal üles ehitatud simulaator ning LGSVL on Unity peal ehitatud.

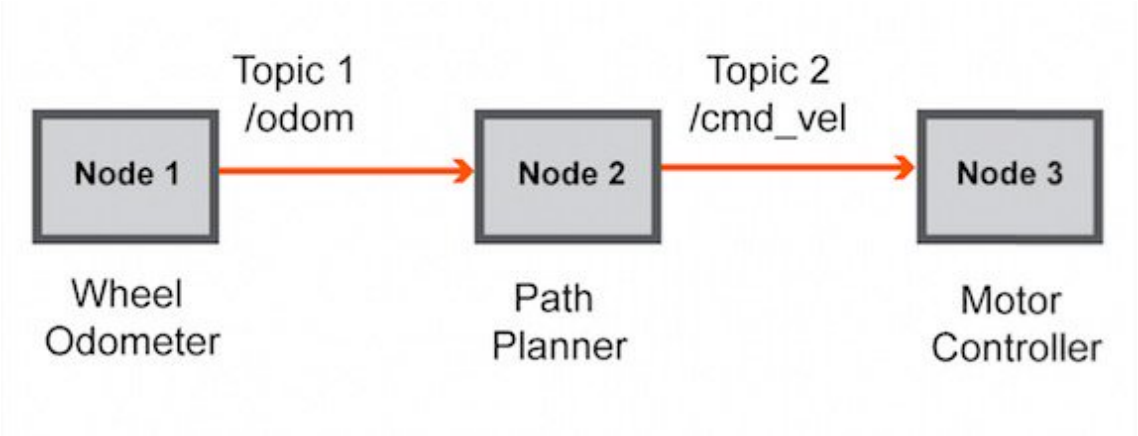
Tänu CARLA ja LGSVL projektidele, eksisteerib nüüd lõputöö jaoks kolm erinevat simulaatorit, mida võrrelda. Võrreldakse simulaatorite võimekust üldiselt ning konkreetselt ROS-i põhiste robotite simuleerimisel ette tulevaid kasutusjuhte. Täpsemalt kuidas suudavad simulaatorid keerukamate ja mitme robotiga toime tulla ning kui täpsed on simulatsiooni elementide kokkupuuted omavahel ning kokkupuuted ebatasase pinnasega.

Lisaks simulaatorite võrdlusele on teine töö eesmärk luua konverter, mis lubab teisendada ühe simulaatori projekti teise simulaatori projektiks - see lubab kasutajal vahetada soovi korral simulaatoreid kergemini.

1.3 Kirjanduse ülevaade

1.3.1 ROS

ROS on tarkvara raamistik robootika tarkvara arendamiseks. ROS sisaldab endas palju funktsionaalsust, kuid lühidalt põhineb ROS ideel, et iga robotisüsteemi komponent moodustab ROS-i sõlme ning ROS-i sõlmed saavad üksteisega suhelda kasutades sõnumite saatmist ning nende kuulamist. See lubab komponentidel tegutseda iseseisvalt, põhineb madala sõltuvuse printsiibil [9]. Järgnevalt joonistel võib näha lihtsat näidet, kuidas ROS-i sõlmed omavahel suhtlevad. Konkreetselt esimene sõlm tegeleb ratta odomeetria arvutamisega ning selle tulemuse välja kuulutamisega, teine sõlm kuulab odomeetria sõnumeid ning sõltuvalt odomeetriale planeerib teekonda.



Joonis 1. Näide ROS sõlmede suhtlusest.

ROS2 on struktuurilt sarnane ROS1-ga, ROS2 põhineb endiselt sõlmede (node) vahelisele sõnumi vahetusele. Kuid leidub palju põhjuseid, mis ajendasid ROS2 loomisele. Näiteks ROS2 suudab multirobotsüsteeme paremini hallata ning halva internetiühenduse korral paremini toime tulla [10]. Samuti kasutab ROS1 nüüdseks vananenud tehnoloogiaid nagu Python 2 ja C++03, samas kui ROS2 rakendab Python 3.5-st hilisemat versiooni ning C++11 versiooni (toetab ka hilisemaid) [11], [12]. Samuti on üks suurimaid muudatusi asjaolu, et kui ROS1 nõudis keskse sõlme (master node) initsialiseerimist enne teiste sõlmede käivitamist, siis ROS2-l seda tingimust ei ole - sõlmed suudavad üksteist leida ilma viiteta [13]. Seega on edaspidi igati kasulik uurida just ROS2 versioonil põhinevaid roboteid.

1.3.2 CARLA ja LGSVL

Artiklis [14] kirjeldatakse, kuidas videomängud on tihedalt seotud simulaatoritega. Mõlemad simuleerivad mingit maailma ning visualiseerivad seda. 21. sajandi alguses

kui arvutusvõimsus kasvas tunduvalt arenesid paralleelselt mängumootorid kui ka simulatsiooni keskkonnad. 2004 aastal tuli välja Gazebo [15] (arenduses aastast 2002 [16] ning UE4 on arenduses 2003-ndast aastast (kuigi tuli välja alles 2012 aastal). Seega on loogiline, et üks võimalus roboteid simuleerida on mängumootoris.

CARLA on autonoomsete sõidukite arenduse jaoks mõeldud simulaator [7]. Simuleeritakse linna tänavaid, sõidukeid ning jalakäijaid. CARLA on üles ehitatud UE4 mängumootori peal, viidatud töös on selle põhjuseks toodud asjaolu, et UE4 pakub realistlikku füüsikat, jalakäijate ja teiste autode käitumise simuleerimist ning võimalust lisada uut funktsionaalsust pistikprogrammide näol.

UE4 ei ole ainus mängumootor, mida on proovitud ROS-ga liidestada. ka Unity mängumootorit on liidestatud ROS-ga [17]. LGSVL on CARLA analoog Unity peal [8]. Unityt kasutatakse robotite simuleerimisel ka selles artiklis [18]. Kirjeldatakse jällegi, kuidas videomängud ja robootika jagavad ühiseid disaini probleeme. Selles artiklis [19] kirjeldatakse multirobot süsteemi arendamist Unityga, põhjuseks asjaolu, et tüüpiliselt kasutatavad simulatsiooni keskkonnad ei vasta autorite poolt välja toodud kriteeriumitele, sama probleem tuleb välja ka selles artiklis [20].

Seega mängumootorite kasutamine ROS-il põhinevate robotite simuleerimiseks võib tõesti olla väga kasulik, eriti kuna mängumootorid pakuvad kasutajale tihti lisafunktsionaalsust, millega simulatsiooni realistlikumaks teha. Nagu näiteks ülalmainitud CARLA jalakäijate kasutamine.

1.4 Metodoloogia

Gazebo, CARLA ja LGSVL simulaatorite võrdlemine ei pruugi olla just kõige triviaalsem, arvestades et nad on loodud erinevateks otstarveteks. Võrdluseks on tarvis välja töötada konkreetsed kriteeriumid, mida simulaatorite puhul vaadelda. Leidub paar artiklit, millest võib võrdlusi tehes eeskuju võtta.

Viidatud töös ”Feature and Performance Comparison of the V-REP, Gazebo and ARGoS Robot Simulators” [21] võrreldi kolme simulaatori erinevaid omadusi. Operatsioonisüsteemidega sobitumine, funktsionaalsused, mudelite mitmekesisus, modifitseeritavus ja kasutajamugavus. Samuti sooritati artiklis loetletud simulaatoritega koormusteste. Koormustestideks defineeriti mitu erinevat teststsenaariumi ning loodi igas simulaatoris ligilähedased maailmad, seejärel mõõdeti iga stsenaariumi ajal meetrikat (näiteks arvuti mälu ja protsessori kasutust).

Artiklis ”Comparative Analysis Between Gazebo and V-REP Robotic Simulators” [22] võrreldi Gazebo ja V-REP simulaatoreid. Võrreldi peamiselt simulaatorite

funktsionaalsust, kuid täpsemalt võrreldi ka ROS-iga liidestamist, mis on seotud käesoleva tööga.

Veel üks näide simulaatorite võrdlusest leidub artiklis ”A Survey and Comparison of Commercial and Open-Source Robotic Simulator Software” [23]. Selles artiklis oli võrdluses 11 erinevat simulaatorit. Kuigi see võrdlus oli pigem üldine - loetleti simulaatorite omadused tabeli kujul (näiteks, mis programmeerimis keelt kasutab, kas omab dokumentatsiooni jne.).

Käesolevas töös on plaanis kombineerida ülaltoodud metoodikat. Staranowic’i artikli eeskujul on plaanis alustada lihtsast faktide võrdlusest, kust saab kiire ülevaate.

Sellele järgneb simulaatorite funktsionaalsuste süvaanalüüs. Et lõputöös simuleeritakse ROS-il põhinevaid roboteid ning täpsemalt ROS2 versioonil, siis kindlasti tuleb uurida kui hästi liidestatav on kumbki keskkond ROS2-ga.

Võrdluse kolmas punkt on teststsenaariumite välja töötamine ning nende jooksutamisel meetrika kogumine. Stsenaariumid võivad erineda mitme elemendi poolest:

- robotite arv
- maailma keerukus (suurus, detailsus)
- pinnas (sile, ebatasane)

Neid elemente leidub veel, kuid loetletud on peamised. Samas tuleb stsenaariume jooksutades teha ka visuaalseid hinnanguid, näiteks kui hästi saab sõiduk hakkama ebatasasel pinnasel sõitmisega (kas simuleeritud kehade vastastikmõjud on simuleeritud korrektselt ning robot ei vaju läbi pinnase jne).

Esmaseid järeldusi võib juba niivõrd teha, et CARLA ja LGSVL võimaldavad simuleerida visuaalselt palju realistlikumat maailma. Artiklis ”A Framework for Visually Realistic Multi-robot Simulation in Natural Environment” [24] katsetati ühe drooni jälgimist teise drooniga naturaalses keskkonnas. Simulatsiooni keskkonnaks kasutati UE4, põhjuseks toodi just asjaolu, et UE4 võimaldab simuleerida võimalikult realistlikku visuaalset maailma. Artiklis [25] kiidetakse UE4 fotorealismi võimekust. Samuti võimaldab UE4 simuleerida erinevaid keskkondi - artiklis [26] simuleeriti veealust keskkonda ning robotit.

Simulaatoreid võrreldes tuleb arvesse võtta artiklis [27] välja toodud punktiga, et tarkvara valik tuleks teha vastavalt enda kindlatele vajadustele, mitte vaakumis tehtud võrdluse järgi. Seega lõputöös tehtud võrdlus arvestab ka spetsiifilisi stsenaariume, nagu ebatasasel pinnasel sõitmine. Mida detailsem on võrdlus, seda parem on hiljem teha otsust, millist simulaatorit valida [28].

1.5 Töö ülevaade

Töö jaotub kahte suuremasse ossa. Töö esimene osa on UE4 ja Gazebo keskkondade vahelise võrdluse loomine - seda ülesannet käsitletakse peatükkides 2, 3 ja 4. Võrdluse loomine hõlmab endas mõlemas simulaatoris sarnaste maailmade ning sõidukite loomist ning nende juhtimist, erinevate keskkonna tingimuste simuleerimist, mitme roboti korraga simuleerimist - eesmärk on aru saada kui hästi tulevad nimetatud simulaatorid toime erinevate tingimuste simuleerimisega. Materjale lugedes võib simulaatorite kirjeldustest välja lugeda, mida mingi konkreetne keskkond teha võimaldab, kuid konkreetseid võrdlusi keskkondade vahel leidub vähe. Metodoloogia peatükis viidatud artiklites leidub palju kasulikke näiteid, kuidas simulaatoreid omavahel võrrelda.

Töö teise osana käsitletakse simulatsiooni projekti konverteri ülesannet (peatükk 5), kus eesmärk on muuta ühe simulaatori projekti (see tähendab projektis sisalduvad maailmad ja mudelid) andmed sellisele kujule, et seda sama projekti on võimalik laadida mingis teises simulatsiooni keskkonnas. Näiteks Unreal Engine 4-s on suhteliselt mugav luua maailmasid, Gazebos väga mitte - kui oleks olemas võimalus UE4-s loodud projekt muundada Gazebo maailma kujule, siis oleks ka Gazebos tunduvalt lihtsam maailmasid luua. Sellise vahendi loomise motiiv on asjaolu, et kui rääkida Gazebost ja UE4-st, siis mõlemat keskkonda endiselt aktiivselt arendatakse. See tähendab, et kui täna on optimaalsem simulaator UE4, siis võib-olla tulevikus on selleks Gazebo. Või näiteks muutub kasutaja soovitavad simulatsiooni kriteeriumid. Gazebo simulaatori projekt asendataksegi lähiajal Ignition projekti vastu, mis on Gazebo simulaatori edasiarendus. Sellisel juhul oleks mugav kui saaks lihtsasti simulaatoreid vahetada ilma suurema peavaluta.

1.6 Tulemuste valideerimine

Simulaatorite võrdluse puhul on raske mõelda meetodi peale, mis valideeriks, et võrdluse tulemused olid korrektsed, arvestades, et mingi osa võrdlusest on siiski kogemuse käigus kogunenud teadmised. Jõudlustestid on iseenesest veidi valideeriva iseloomuga, kuna näitavad konkreetselt, kas simulaator tuleb stsenaariumiga toime ning kui palju ressursse selle jaoks kulub.

Konverterit seevastu on võimalik väga hästi testi skriptidega valideerida. Arvestades seda, et tarkvara autor teab, millised maailmad ja mudelid peavad muundamise tulemusena tekkima, siis selle jaoks on lihtne kirjutada skript, mis kontrollib, kas tekkinud maailm sisaldab kõiki elemente, mis vaja.

2 Simulaatorite üldnäitajate analüüs

Järgnevas peatükis võrreldakse Gazebo, CARLA ja LGSVL simulaatorite lihtsasti hinnatavaid omadusi. Esmalt pannakse paika võrreldavad kriteeriumid. Seejärel esitatakse simulaatorite omaduste lihtne võrdlus tabelite kujul ning lõpuks kirjeldatakse täpsemalt simulaatorite omaduste iseärasusi. Selline võrdlus annab kiire ülevaate simulaatorite põhiomadustest.

2.1 Kriteeriumid

Selleks, et sooritada simulaatorite vahel võrdlus, tuleb esmalt paika panna kriteeriumid, mida simulaatorite juures hinnata. Järgnevalt esitatakse tabel kriteeriumitest, kus iga kriteeriumi juures on lühidalt kirjeldatud, mida täpsemalt kriteerium näitab ning milleks ta oluline on.

Tabel 1. Kriteeriumid

Kriteerium	**Kirjeldus**
Saadavus operatsioonisüsteemidele	Milliste operatsioonisüsteemide jaoks simulaator saadaval on? Kasutaja valik võib olla piiratud sõltuvalt tema poolt kasutatavast operatsioonisüsteemist.
Lähtekoodist kompileerimine	Millistel operatsioonisüsteemidel on võimalik simulaatorit lähtekoodist kompileerida? Lähtekoodist kompileerimine võimaldab kasutajal lisada enda või kolmanda isiku poolt loodud funktsionaalsust.
Litsents	Milline litsents tuleb konkreetse simulaatoriga kaasa? Sõltuvalt kasutaja eesmärgist võib litsents piirata kasutaja vabadust.
Süsteeminõuded	Millised on simulaatori poolt süsteemile esitatavad nõuded? Sõltuvalt kasutaja riistvarast võib tema simulaatori valik olla piiratud.
Füüsika mootorid	Millised on simulaatori poolt pakutavad füüsika mootorid? Füüsika mootoreid on erinevaid ning varieeruvad funktsionaalsuse ning jõudluse poolest.
Programmeerimiskeeled	Milliseid programmeerimiskeeli toetab? Kasutajale võib olla isiklikud eelistused, millistes keeltes programmeerida.
Dokumentatsioon	Kui palju leidub simulaatori kohta dokumentatsiooni? Dokumentatsioon lubab kasutajal simulaatoriga paremini tutvuda.

Õpetused	Kui palju leidub simulaatori õpetusi? Õpetused erinevad dokumentatsiooni poolest niivõrd, et näitavad kasutajale, kuidas simulaatoriga midagi konkreetset teha. Õpetuste olemasolu võimaldab kasutajal simulaatorit paremini õppida.
Kasutajabaas	Kui suur on simulaatori kasutajabaas ning kust neid leida võib? Probleemide korral on võimalus küsida abi teiste kasutajate käest. Mida suurem on simulaatori kasutajabaas, seda tõenäolisem on abi saamine.
Sensoorika	Milliseid andureid on võimalik simulaatoriga kasutada. Andureid on võimalik ka kasutaja poolt luua, kuid olemasolevate andurite hulk võib olla kasutajale oluliseks kriteeriumiks, kui on teada simuleeritava roboti poolt kasutatavad andurid.

2.2 Kriteeriumite võrdlus

Kui võrdluskriteeriumid on paika seatud, võib tegeleda võrdlusega. Järgnevas tabelis on igal kriteeriumil iga simulaatori kohta lühidalt kirja pandud simulaatori väärtus antud kriteeriumi kohta.

Tabel 2. Simulaatorite võrdlus

	Gazebo	**CARLA**	**LGSVL**
Saadavus operatsioonisüsteemidele	Linux (Ubuntu, Debian, Fedora, Arch, Gentoo) & Mac	Linux & Windows	Linux & Windows
Lähtekoodist kompileerimine	Linux (Ubuntu) & Mac	Linux & Windows	Linux & Windows
Litsents	Apache License 2.0	MIT litsents, CC-BY litsents, kohandatud UE4 litsents	Kohandatud litsents
Süsteeminõuded	Integreerimata GPU, Nvidia videokaart, Intel I5 CPU või parem, 500MB vaba kõvaketta ruumi, Ubuntu Trusty	Integreerimata GPU, videokaardi tugi OpenGL 4-le	4 GHz Dual core CPU, Nvidia GTX 1080, Windows 10 64-Bit
Füüsika mootorid	ODE, Bullet, DART, Simbody	PhysX 3.3	Unity3D
Programmeerimiskeeled	C++ pistikprogrammid või ROS programmid (C++ või Python)	Python või ROS-programmid (C++ või Python)	Python või ROS-programmid (C++ or Python)

Dokumentatsioon	API dokumentatsioon ja blogi muudatuste logide ja tulevikuplaanide jaoks	API dokumentatsioon	API dokumentatsioon
Õpetused	Suur valik õpetusi eraldi veebilehel	Väike valik õpetusi	Väike valik õpetusi
Kasutajabaas	Eraldiseisev Q&A veebileht ja projekt Bitbucketis	Githubi project ja Discordi server	Githubi projekt
Sensoorika	RGB kaamera, GPS, laserandur, IMU, sügavuskaamera, puuteandur, jõu-momendiandur, multikaamera, RFID andur, ultraheli andur	RGB kaamera, GNSS, LIDAR, sügavuskaamera, puuteandur, semantilise segmentatsiooni kaamera, sõiduraja ületamise andur, takistuste andur	RGB kaamera, GPS, LIDAR, IMU, RADAR sensor

2.3 Võrdluse süvaanalüüs

Lisaks lihtsale võrdlusele tabeli kujul, kirjutatakse täpsemalt, kuidas iga simulaator mingile kriteeriumile vastab.

2.3.1 Saadavus operatsioonisüsteemidele

Gazebo - Saadaval suurematel Linuxi distributsioonidel (Ubuntu, Debian, Fedora, Arch, Gentoo) ja Macil. Windowsi versioon on hetkel arenduses. Installeerimisjuhised saadaval [29].

CARLA - Saadaval Linuxil ja Windowsil. Dokumentatsioon ei täpsusta täpsemalt, millistel distributsioonidel. Käivitamise juhised: [30].

LGSVL - Saadaval Linuxil ja Windowsil. Dokumentatsioon ei täpsusta distributsioone. Käivitamise juhised: [31].

2.3.2 Lähtekoodist kompileerimine

Kui enamik kasutajaid installeerivad tarkvara kasutades ette antud installeerimisskripti, siis vahepeal on kasulik tarkvara kompileerida lähtekoodist. Lähtekoodist kompileerimine lubab kasutajal muuta installeerimissätteid, eemaldada ebavajalike pakette, aga samas muuta olemasolevat tarkvara ja lisada ka enda loodud pakette ja mooduleid.

Gazebo - Võimalik lähtekoodist kompileerida Linuxi (Ubuntu) and Maci peal. Juhised: [32].

CARLA - Võimalik kompileerida Linuxi (Ubuntu) ja Windowsi peal. Linuxi juhised: [33]. Windowsi juhised: [34]. Maci peal kompileerimine võimalik, kuid nõuab lisatööd juhisteta [35]. Oluline märkus on asjaolu, et CARLA lähtekoodist kompileerimiseks on tarvis installeerida Unreal Engine 4.

LGSVL - Võimalik kompileerida Linuxil ja Windowsil, kuigi juhised esinevad vaid Linuxile - [36]. LGSVL-i lähtekoodist kompileerimine nõuab Unity installeerimist.

2.3.3 Litsents

Gazebo - Apache License 2.0. Annab kasutajale kõige enam vabadusi: lubab kasutajal tarkvara kasutada ükskõik, milliseks otstarbeks, ilma et peaks litsentsitasu maksma.

CARLA - CARLA puhul tuleb arvesse võtta mitu erinevat litsentsi. CARLA-spetsiifiline kood on arendatud MIT litsentsi all. MIT litsents lubab samuti tarkvara igaks otstarbeks kasutada, eeldusel et kasutaja lisab originaalse litsentsi tehtud tööle. CARLA-spetsiifilised varad (maailmad, sõidukid) on loodud CC-BY litsentsi all, mis lubab kasutajal loodud varasid kasutada, kuid nende kasutamisele tuleb kindlasta viidata. CARLA on ehitatud UE4 mootori peale, millel on enda eraldi litsents [37]. Litsents lubab UE4 mootorit igaks otstarbeks kasutada, kuid kommertsprojektidel nõutake 5% müügitulust. Samas litsentsitasu ei pea maksma kui müüdav toode ei kasuta UE4 mootorit.

LGSVL - Kasutab erilitsentsi [38]. Litsents lubab kasutada igal eesmärgil, kuid litsentsitasu tuleb maksta igasuguse kommertsprojekti puhul. Muud tüüpi projektide puhul (näiteks hariduslikul eesmärgil) litsentsitasu maksma ei pea.

2.3.4 Süsteeminõuded

Süsteeminõuded on küll iga simulaatori kohta antud, kuid reaalselt ei anna nad aimdust kui hea nende jõudlus on, kuna tegemist on enamasti soovituslike nõuetega. Parema ülevaate simulaatorite jõudlusest leiab töö 4-ndast peatükist.

Gazebo - Soovitatud nõuded süsteemile on integreerimata GPU olemasolu, Nvidia videokaart, vähemalt Intel i5 või parem CPU ning Ubuntu Trusty (14.04) või hilisem. Samuti on nõue omada vähemalt 500 MB vaba ruumi kõvakettal.

CARLA - CARLA puhul võib dokumentatsioonist leida ainult vajalikud minimaalnõuded. Nõue on omada arvutit integreerimata GPU-ga. Videokaardile on nõue

toetada OpenGL versiooni 4, kuna hilisemad Unreal Engine 4 versioonid nõuavad OpenGL 4 tuge.

LGSVL - Dokumentatsioonist leitud nõuded on järgmised: kahetuumaline protsessor kiirusega 4 GHz, Nvidia GTX 1080 videokaart ning Windows 10 64-bitise operatsioonisüsteemiga. Neid nõuded vaadates tundub, et dokumentatsiooni on kirja pandud nõuded kõige optimaalsemaks simulaatori jooksutamiseks. Reaalsuses saab kindlasti hakkama ka madalamate näitajatega arvutil. Lisaks sellele on dokumentatsioonis välja toodud, et Linuxi ja Windowsi vahel tuleks eelistada Windowsi, kuna simulaator on paremini optimeeritud Windowsile.

2.3.5 Füüsika mootorid

Gazebo - Lubab kasutada nelja erinevat füüsikamootorit: Open Dynamics Engine (ODE), Bullet, Dynamic Animation and Robotis Toolkit (DART) ja Simbody. Igal füüsikamootoril on oma eelised ning asjaolu, et Gazebol on neid neli, võimaldab kasutajal proovida simulatsiooni mitme erineva mootoriga.

CARLA - Kuna CARLA on ehitatud UE4 mängumootori peal, siis tema poolt kasutatud füüsikamootor on sama, mis UE4-l - PhysX versioon 3.3.

LGSVL - LGSVL on ehitatud Unity mängumootorile, seega kasutab sama mootorit, mis Unity - PhysX (sama, mis UE4), kusjuures samuti versiooniga 3.3, mõlemad mängumootorid uuendavad PhysX versiooni vastavalt siis kui uus versioon välja tuleb (PhysX 3.4 on hetkel eksperimentaalne).

PhysX on põhiliselt mängumootorite jaoks kirjutatud füüsikamootor. Erinevate füüsikamootorite võrdlust võib leida mitmest olemasolevast tööst. 2007-ndal aastal avaldatud artiklis kirjeldatakse seitsme erineva mootori võrdlust, samuti tuuakse välja kuus kriteeriumi, mis määravad füüsikamootori jõudluse [39]. Kolm vaadeldud mootoritest olid PhysX, ODE ja Bullet. Võrdluse tulemusena jõuti järeldusele, et ei eksisteeri füüsikamootorit, mis oleks teistest selgelt üle - iga füüsikamootoril on omad plussid ja omad miinused. Artiklis sooritatud testides sooritas PhysX kõige paremini kehale mõjuvate jõudude arvutamisel. See tähendab, et PhysX mootor suudab kõige täpsemini keha liikumisel tema õige hetke positsiooni arvutada.

2.3.6 Programmeerimiskeeled

Gazebo - Kasutaja saab luua spetsiaalselt Gazebo jaoks loodud pistikprogramme (C++ keeles) või kasutada robotite juhtimiseks ROS-il põhinevaid mooduleid (C++ või Python keeles).

CARLA - Omab Python API-t simulatsiooni elementide kontrollimiseks. Teine võimalus on kasutada ROS-il põhinevaid mooduleid (C++ või Python). Unreal Engine 4 funktsionaalsust saab programmeerida kas C++ keeles või spetsiaalset UE4 sisseehitatud visuaalset programmeerimiskeelt (sarnane Scratch programmeerimiskeelele). Eksisteerib ka Pythoni API uuematele Unreal Engine versioonidele, kuid on hetkel eksperimentaalne [40].

LGSVL - Python API simulatsiooni elementide kontrollimiseks. Sõidukite juhtimiseks saab kasutada ROS-il põhinevaid mooduleid (C++ või Python). Unity funktsionaalsust saab programmeerida C# programmeerimiskeeles.

2.3.7 Dokumentatsioon

Gazebo - Omab API dokumentatsiooni [41] ning blogi uudiste jaoks [42].

CARLA - Dokumentatsioon sisaldab erinevaid juhendeid ning API dokumentatsiooni [43].

LGSVL - Dokumentatsioon, mis sisaldab paari juhendit, kuid neid on väga vähe [44], lisaks API dokumentatsioon.

2.3.8 Õpetused

Gazebo - Suur valik erinevaid õpetusi [45]. Õpetatakse Gazebo installeerimist, käivitamist, kasutajaliidese kasutamist, maailma loomist, robotite loomist, pistikprogrammide loomist ja kasutamist, ROS-ga ühendamist ja veel teisi tegevusi.

CARLA - Õpetused leiduvad samal veebilehel dokumentatsiooniga [43]. Võrreldes Gazeboga on õpetusi vähem: õpetatakse installeerimist, käivitamist, Python API kasutamist, uute maailmade ja robotite lisamist. Kuna CARLA on tihedalt seotud ka UE4-ga, siis tulevad ka kasuks UE4 õpetused, neid võib leida ametlikult Unreal Engine veebilehelt [46] või video kursustelt, mida võivad luua teised UE4 kasutajad [47] - nendest ressurssidest on võimalik väga palju erinevaid tegevusi õppida, nende loetlemine oleks ebamäärane.

LGSVL - Õpetused on kombineeritud dokumentatsiooniga [44]. Kolmest võrreldavast simulaatorist omab LGSVL kõige vähem õpetusi - õpetatakse simulaatori installeerimist, käivitamist ning maailma kaardi loomist. See tähendab, et paljudes tegevustes peab kasutaja rohkelt iseseisvalt uurima. Unity mängumootorile on samas palju õpetusi ametlikul veebilehel [48].

2.3.9 Kasutajabaas

Gazebo - Gazebol eksisteerib Q&A veebileht, kus on võimalik teiste Gazebo kasutajate käest abi küsida kui ka abi pakkuda [49]. Samuti eksiteerib avalik Gazebo projekt Bitbucketis, kus on võimalik suhelda Gazebo arendajatega ning jälgida nende poolt tehtud tööd [50].

CARLA - CARLA projekt githubis võimaldab jälgida arendustööd ning suhelda arendajatega [51]. Samuti eksisteerib CARLA avalik Discordi server, kus on võimalik teiste CARLA kasutajate ja arendajatega suhelda ning abi küsida. Samuti postitakse serverisse uuendusi hetke töö kohta. UE4 jaoks eksisteerib Q&A veebileht [52], üldiselt on UE4 kasutajabaas väga suur, kuna UE4 peal arendatakse populaarseid videomänge (näiteks Fortnite).

LGSVL - LGSVL Githubi projekt on hetkel ainus avalik koht, kus on võimalik LGSVL arendajatega suhelda [53], seega tõenäoliselt kasutajabaas jääb alla teistele simulaatoritele. Unity omab sarnaselt UE4-le Q&A veebilehte, et teiste kasutajatega suhelda [54]. Unity on samuti laialt tuntud mängumootor, seega on ka Unity kasutajabaas väga suur.

2.3.10 Sensoorika

Esmalt selgitatakse lähemalt iga anduri kohta, mis tema eesmärk on, seejärel loetletakse iga simulaatori poolt pakutavad andurid.

Kaamera / RGB Kaamera - Tavaline värvikaamera.

GPS / GNSS - Üleilmne asukoha määramine.

Laserandur / LIDAR - Tuvastab laseritega anduri ees olevaid takistusi.

RADAR - Tuvastab raadiolainetega eesolevaid takistusi.

IMU - Ehk Inertsiaalandur, mõõdab joonkiirust ja pöörlemiskiirust.

Sügavuskaamera - Kaamera, mis tuvastab 3D keskkonda.

Puuteandur - Tuvastab kokkupuudet mingi teise objektiga.

Jõu-momendiandur - Mõõdab rakendatud jõudu ja jõumomenti.

Multikaamera - Gazebo-spetsiifiline andur, mis koosneb mitmest kaamerast.

RFID andur - Raadiolainete abil esemete tuvastus.

Ultraheli andur - Mõõdab kaugusi kasutades ultraheli laineid.

Semantilise segmentatsiooni kaamera - Kaamera, mis jaotab semantikat kasutades kaamera pildi erinevateks segmentideks.

Sõiduraja ületamise andur - CARLA-spetsiifiline andur, mis tuvastab teemärgise ületamist.

Takistuste andur - CARLA-spetsiifiline andur, mis tuvastab teisi objekte.

Gazebo - Sisaldab kõige enam sisseehitatud andureid, andureid esineb mitmest valdkonnast, kuna Gazebo ei spetsialiseeru otseselt mingile alale (nagu CARLA ja LGSVL spetsialiseeriuvad isesõitvatele autodele). Andurid: RGB kaamera, sügavuskaamera, puuteandur, jõu-momendiandur, GPS, IMU, laserandur, multikaamera, RFID andur ja ultraheliandur.

CARLA - Sisseehitatud andurid on üldiselt need andurid, mida on tarvis liikluses liikumiseks: RGB kaamera, sügavuskaamera, semantilise segmentatsiooni kaamera, GNSS, LIDAR, puuteandur, sõiduraja ületamise andur ja takistuste andur.

LGSVL - esineb kõige vähem sisseehitatud andureid: RGB kaamera, IMU, GPS, LIDAR ja RADAR.

3 Simulaatorite süvaanalüüs

Järgmisena võrreldakse simulaatorite võimalusi ja funktsionaalsust, mida ei saa lihtsasti mõõta, vaid vajab sügavamat analüüsi ja katsetusi.

3.1 Installeerimine

Selles alampeatükis võrreldakse simulaatorite installeerimise protsessi: kui hästi on protsess dokumenteeritud, kui lihtne on seda sooritada, millega tuleb arvestada. Installeerimine on esimene kokkupuude tarkvaraga, seega on oluline, et kasutaja saab paigaldamisega hakkama.

Gazebo - Installeerimise jaoks esinevad õpetused igale toetatud operatsioonisüsteemile [29]. Mõlemad installeerimisvalikud (tavapärane paigaldus ja lähtekoodist kompileerimine) on õpetustest kaetud. Samuti pakutakse alternatiivseid installeerimismeetodeid. Ubuntule paigaldamine on väga lihtne - üks rida käsureal.

CARLA - CARLA installeerimisel tuleb arvestada mõningate asjaoludega.

Esiteks, CARLA kasutamiseks peab kasutaja paigaldama CARLA serveri ning CARLA kliendi - server tegeleb maailma oleku arvutamisega ning visualiseerimise laadimisega, klient tegeleb maailma elementide kontrollimisega, näiteks sõiduki juhtimisega. Need kaks komponenti võivad asuda samal masinal, aga ka eraldi masinatel (eraldi masina eelis on see, et kasutaja ei pea oma arvuti peal mõlemat komponenti jooksutama - hoiab arvuti ressursse kokku). Serveri paigaldamine on lihtne: tuleb allalaadida server ning käivitamiseks tuleb käivitada vastav täitmisprogramm (Windowsil .exe ning Linuxil .sh laienditega). Kliendi paigaldamiseks tuleb samuti vastav tarkvara alla laadida, kuid lisaks sellele peab kasutaja paigaldama CARLA Pythoni API - selleks peab kasutaja omama veidi lisapädevust, kuna ametlik CARLA õpetus Pythoni API installeerimist ei õpeta [55].

Teine asjaolu, millega kasutaja peab arvestama, kehtib juhul kui kasutaja soovib CARLA simulaatorile uusi sõidukeid, maailma või uut loogikat lisada. Sellisel juhul peab kasutaja kompileerima CARLA serveri lähtekoodist. Selle tegemiseks eksisteerivad instruktsioonid [43], kuid sellest sõltumata võib installeerimisel esineda palju probleeme, mille lahendamiseks tuleb instruktsioonidest välja poole vaadata ning

võib-olla ka teistelt kasutajatel abi küsida. Lisaks sellele, peab kasutaja enda süsteemile ka UE4 installeerima. Seda on tunduvalt lihtsam teha Windowsi peal, kuna UE4 Windowsi versioon omab automaatset paigaldamisskripti, samas kui Linuxi peal tuleb UE4 lähtekoodist kompileerida [56]. Samuti võtab Linuxi peal UE4 üle 100 GB kõvaketta ruumi, samas kui Windowsile installeerides võtab UE4 ainult 30 GB ruumi.

LGSVL - LGSVL-i installeerimisel võib tõmmata paralleele CARLA installeerimisega: mõlemad simulaatorid vajavad eraldi serveri ning kliendi installeerimist, kui kasutaja soovib teha muudatusi, on tarvis LGSVL lähtekoodist kompileerida. Lähtekoodist kompileerimiseks tuleb oma süsteemile installeerida Unity mängumootor. Lõputöö autor proovis simulaatorit lähtekoodist kompileerida nii Windowsi kui ka Linuxi peal - Linuxi peal esines palju programmivigu, mida tuli ükshaaval uurida ja lahendada, samas Windowsi peal läks kompileerimine probleemideta. Seda toetab ka dokumentatsioon, et simulaatori kasutamine Windowsi peal toodab paremaid tulemusi [44]. Pythoni API paigaldamine on juhendis kirjeldatud ning tunduvalt lihtsam kui CARLA Pythoni API paigaldamine [57].

3.2 Kasutamine ROS2-ga

Simulaatori kasutamine ROS2-ga on töö üks olulisemaid punkte, kuna üks töö eesmärkidest, on uurida, kui hästi kasutatav on uuritavad simulaatorid ROS2-ga.

3.2.1 Simulaatorid ametliku ROS2 toeta

Selleks, et kasutada simulaatorit ROS2-ga, peab eksisteerima mingisugune sild, mis suudab pakkuda suhtlust simulaatori ja ROS2 moodulite vahel. Mõne simulaatori jaoks selline sild juba eksisteerib, mõne jaoks veel mitte. Nendel simulaatoritel, millel puudub sild ROS2-ga, eksisteerib vähemalt sild ROS1-ga. Esialgne plaan oli nendele simulaatoritele sild ROS2-ga ise luua (hõlmas ametliku rosbridge_suite paketti muutmist sobivaks ROS2 jaoks), kuid see osutus pikas perspektiivis ebamõistlikuks, kuna ROS-i esimene versioon asendatakse mõne aasta jooksul täielikult ROS2-ga, mis tähendab, et igal juhul asendatakse eksisteerivad ROS1 sillad uute ROS2 sildadega.

Kuna siiski mõnel simulaatoril silda ROS2-ga ei eksisteeri, siis tuleb olukord lahendada loominguliselt. Eksisteerib ametlik ROS2 tarkvara pakett ”ros1_bridge”, mis loob kahepoolse silla ROS2 ja ROS1 vahel. Käivitamisel näeb olukord välja järgmine:

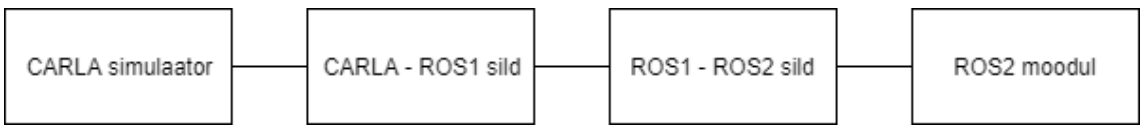
- simulaator käivitatakse
- simulaator ühendatakse sillaga, mis ühendab simulaatori ROS1-ga
- käivitatakse ROS2 ja ROS1 vaheline sild

Sellise lahendusega on võimalik ka ainult ROS1 toetav simulaator ühendada ROS2-ga. Lisaks sellele esineb töö autoril veel üks kitsendus: autori poolt kasutatav arvuti on Windows operatsioonisüsteemiga, kuid soovitatav operatsioonisüsteem ROS-le on Linux. See tähendab, et autor kasutab Linuxi jaoks virtuaalmasinat. Seega tööprotsess näeb välja selline, et virtuaalmasinas Linuxi peal jooksutatakse ROS programme ning ülalmainitud sillad ühendavad virtuaalmasina ROS-i Windowsi peal jooksvate CARLA või LGSVL simulaatoritega.

3.2.2 Võrdlus

Gazebo - ROS2 ja Gazebo vahelise silla loomine on hetkel küll pooleli, kuid kaetud on enamus vajalikust funktsionaalsusest [58]. Dokumentatsioonist võib leida õpetuse, kuidas ühendada ROS2 Gazeboga ning lisaks on toodud paar demo maailma, mida kasutaja proovida saab [59].

CARLA - CARLA simulaatoril hetkel puudub otsene tugi ROS2-ga. Samuti ei ole ROS2 toe loomist välja toodud ka 2019. aasta CARLA plaanides [51]. See tähendab, et CARLA kasutamiseks ROS2-ga, peab kasutusele võtma ülalmainitud ros1_bridge paketi. Selleks peab kasutaja lisaks CARLA-le ja ROS2-le enda süsteemi ka ROS1 installeerima. Juhend paketi käivitamiseks on väga põhjalik [60]. Üldiselt ei ole väga keeruline sellist lahendust käima saada, kuid asjaolu, et sama süsteemi peal peavad eksisteerima korraga ROS1 ja ROS2 võib kasutajale veidi segadust tekitada. CARLA ühendamine ROS2-ga on nähtav Joonisel 2.



Joonis 2. CARLA simulaatori ühendamine ROS2-ga.

LGSVL - LGSVL toetab ühendamist mõlema ROS-i versiooniga, see tähendab ROS1 ja ROS2-ga. Kuigi ROS2 silla paigaldamisel tekkis konflikt Nodejs tarkvara uuema versiooniga ning lahenduseks oli tarvis Nodejs (8.0) vanem versioon installeerida. ROS1 silla puhul sellist konflikti ei tekkinud, seega alternatiivne võimalus on sarnaselt CARLA-le kasutada ROS2 ja ROS1 vahelist silda. Kuid konflikt sai siiski lahendatud, seega on võimalik kasutada otse LGSVL ja ROS2 vahelist silda. Silla ülesehitust illustreerib Joonis 3.

Joonis 3. LGSVL simulaatori ühendamine ROS2-ga.

3.3 Versioon

Selles peatükis räägitakse, kui oluline on iga simulaatori juures tema versioon: kas ühildub vanemate versioonidega, kas vajalik funktsionaalsus toimib nii nagu peaks ning üldiselt mida peaks silmas pidama versiooni valikul.
Gazebo - Igal ROS-i versioonil on temale vastav Gazebo versioon, mida ametlikult soovitatakse kasutada. Samas on võimalik ka ise valida, millist Gazebo versiooni soovitakse kasutada, sellisel juhul tuleb veidi lisa mooduleid installeerida [61]. Üldiselt on kasulik kasutada Gazebo kõige uuemat versiooni. Maailmad ja robotid on üldiselt ühilduvad kõikide uuemate Gazebo versioonidega.

CARLA - CARLA puhul on oluline, et serveri ja kliendi versioonid ühtivad. See tähendab veidi lisa tööd eriti siis kui kasutaja uuendab CARLA versiooni, sellisel juhul peab uuendama nii serveri kui ka kliendi versiooni. Tuli ette ka intsident, kus CARLA versioon 0.9.4 ei toiminud ROS-ga, kuid sellele eelnev versioon 0.9.3 toimis nii nagu vaja. 0.9.4 oli sel momendil kõige uuem väljalase, seega võib-olla esines seal mingisugune programmiviga.

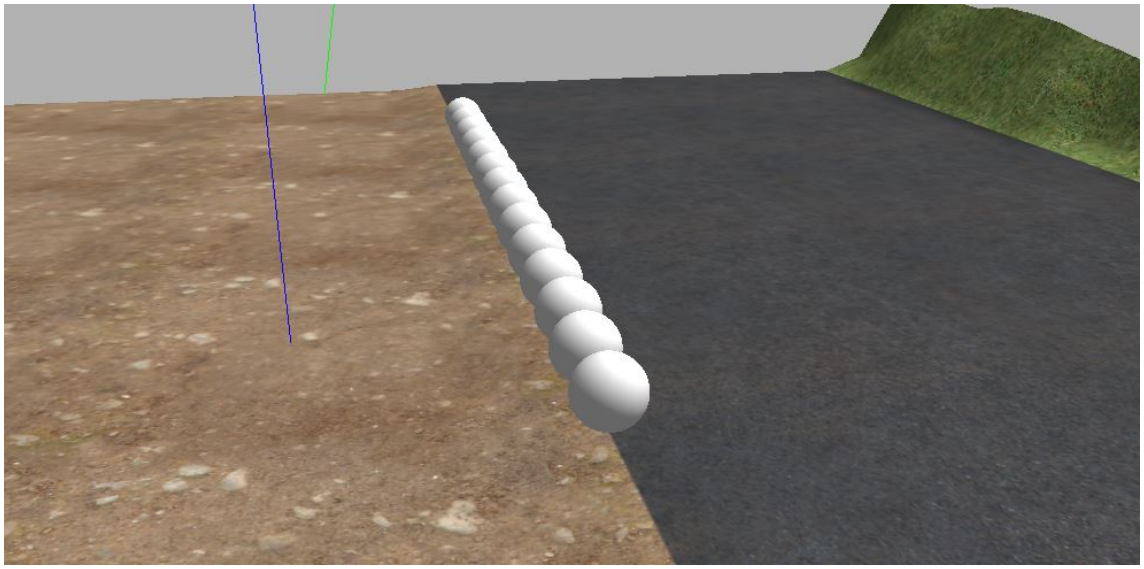
LGSVL - Sarnaselt CARLA-le, on LGSVL-i juures oluline, et serveri ja Pythoni API-t kasutav klient on sama versiooniga, vastasel juhul ei suuda kliend korrektselt ühendust luua serveriga. Lisaks sellel juhtus intsident, kus 2019.01 versioon ei edastanud ROS-le sõnumeid nii nagu vaja. Sellele järgnev versioon 2019.03 aga toimis nii nagu vaja ROS-i sõnumite saatmisel.

3.4 Simulatsiooni füüsika stabiilsus

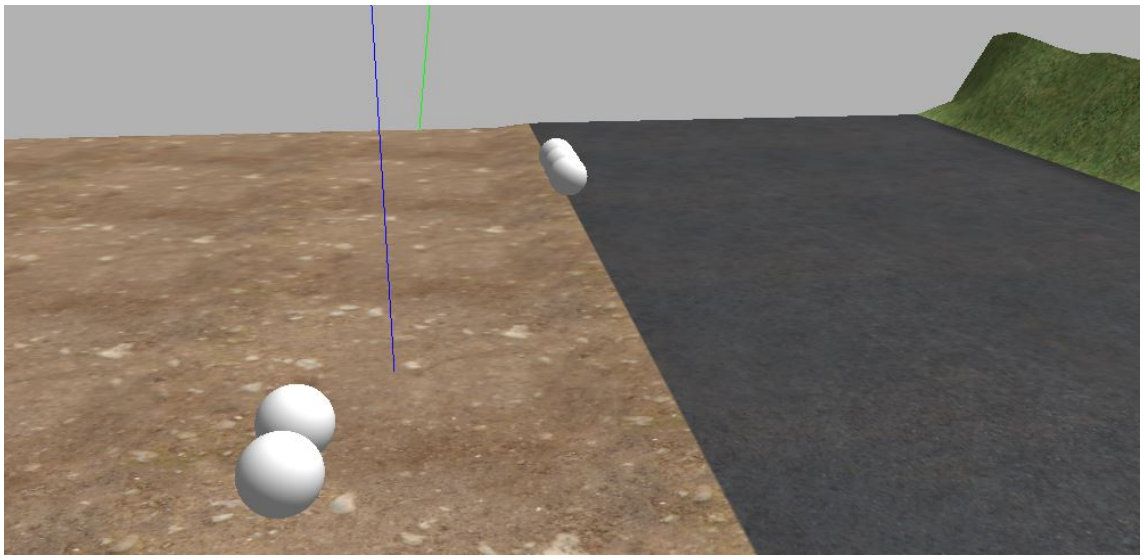
Selles peatükis arutatakse kui stabiilsed ning täpsed on elementide vastastikmõjud simulatsioonis. Simuleerida dünaamilisi objekte, on oluline, et objektide ja maailma vahelised vastastikmõjud toimivad nii nagu peaks. See tähendab pall peaks kallakust alla veerema, konarlikel pindadel peaks visuaalne komponent vastama füüsilisele komponendile.
Gazebo - Tasapinnal liikudes lihtsa robotiga probleeme ei teki. Rattad veerevad sirgel pinnal nii nagu peab. Suur probleem Gazeboga tekib siis kui tegemist on ebatasase pinnasega, see tähendab igasugused künkad, mäed, augud ja muud konarlused.

Üks konkreetne probleem elementide füüsilise komponendiga Gazebos dokumenteeriti lõputöö autori poolt detsembris 2017 [62]. Põhiline probleem oli objektide kukkumine läbi pinnase, kust selgelt ei tohiks läbi kukkuda. Lühidalt demonstreeritakse probleemi kirjelduses maailma, kus esineb maailma keskel üks nõlv, nõlva ülemisele äärele on asetatud 10 kera läbimõõtudega 76 cm. Kerad asetati sirgesse ritta, kus iga kera asetati paar millimeetrit lähemal nõlva äärele. Ideaalsel juhul veereksid osad kerad nõlvast alla ning ülejäänud kerad jääksid nõlva tippu paigale. Tegelikult aga veeresid osad pallid korralikult nõlvast alla, osad pallid jäid nõlva tippu, aga esinesid mõned pallid, mis kukkusid läbi maapinna. See tähendab, et igal nõlval esines kitsas riba, kust oli võimalik objektidel läbi kukkuda - see on eriti ebamugav ratastega sõidukite puhul. Probleem dokumenteeriti Gazebo 8.0 versiooniga, kuid sama probleem esineb ka kõige hiljutisemas versioonis (töö kirjutamise ajal 9.8). Kirjeldatud probleem on seotud ka varasemalt 2012 aastal dokumenteeritud probleemiga, kus maastiku visuaalne komponent ei ühtinud füüsilise komponendiga [63]. Siiani ei ole leitud konkreetset lahendust, mis probleemi lahendaks.

Probleemi illustreerimiseks on lisatud ka kaks joonist: esimeselt joonisel võib näha kerade algpositsiooni, teiselt jooniselt võib näha, kuidas osad kerad on läbi maa kukkunud.



Joonis 4. Kerade algpositsioon.



Joonis 5. Kaks sekundit pärast simulatsiooni algust. t = 2.0 s

Teine suur probleem Gazebos on keerulistema robotite loomisel tekkiv füüsika probleem. Keerulisemal robotil esineb omavahel sõltuvuses olevaid lülisid päris palju, igal lülil on tarvis seada korrektne inertsimaatriks, mis vastaks lüli dimensioonidele ja massile - see määrab, kuidas teatud lüli füüsikamootoris käitub. Inetrsimaatriksite seadmisel on olemas juhendid, kuid praktikas tuleb tihti katseeksitusmeetodiga intertsimaatrikseid muuta. Olukorras, kus lüli on palju (suurusjärgus 10 või rohkem), tekib tihti olukordi, kus iga lüli intertsimaatriks on väikse arvu võrra erinev ideaalist. Seetõttu tekivad olukorrad, kus terve robot hakkab kergelt kõrgsageduslikult vibreerima, mistõttu liigub teatud aja jooksul sõiduk teise kohta, ilma et oleks saanud liikumiskäske. See suurendab lokalisatsiooni viga.

CARLA - Kuna CARLA on ehitatud UE4 mootori peale, siis reaalselt analüüsitakse UE4 elementide vastastikmõjude stabiilsust. Ülal kirjeldatud probleemne stsenaarium kerade langemisest läbi maapinnase üritati luua ka UE4 mootoris. Esialgu prooviti kerasid täpselt samade mõõtmetega ning samade positsioonidega, kuid kindluse mõttes prooviti kerade mõõtmeid suurendada ja vähendada, nende positsioone muudeti. Lisaks keradele prooviti teisi kujundeid (kuupi, tetraeedrit, silindrit). Samuti kasutati UE4 maastiku tööriistasid, mis lubavad luua palju keerukamaid maastiku reljeefe kui Gazebos. Loetletud katsetustega ei suudetud leida ühtegi stsenaariumi, kus ükski ese oleks langenud läbi maapinna.

LGSVL - Järgnev analüüs on väga sarnane CARLA sektsiooniga. Kuna LGSVL on ehitatud Unity mootori peale, siis reaalselt analüüsitakse Unity vastastikmõjude stabiilsust. Ülal kirjeldatud probleemne stsenaarium kerade langemisest läbi maapinnase üritati luua ka Unity mootoris. Esialgu prooviti kerasid täpselt samade mõõtmetega ning samade positsioonidega, kuid kindluse mõttes prooviti kerade mõõtmeid suurendada ja

vähendada, nende positsioone muudeti. Lisaks keradele prooviti teisi kujundeid (kuupi, tetraeedrit, silindrit). Samuti kasutati Unity maastiku tööriistasid, mis lubavad luua palju keerukamaid maastiku reljeefe kui Gazebos. Loetletud katsetustega ei suudetud leida ühtegi stsenaariumi, kus ükski ese oleks langenud läbi maapinna.

3.5 Uute maailmade loomine

Järgnevas peatükis kirjeldatakse iga simulaatori kohta nende võimalusi luua uusi maailmasid. Täpsemalt räägitakse, kui lihtne on maailmaid luua, kas esineb alternatiive ning kas esineb olulist infot, mida järgida. Lisaks sellele uuritakse täpset kasutusjuhtu, kus kasutaja soovib luua ebatasase pinnasega maailma (naturaalne õueala).

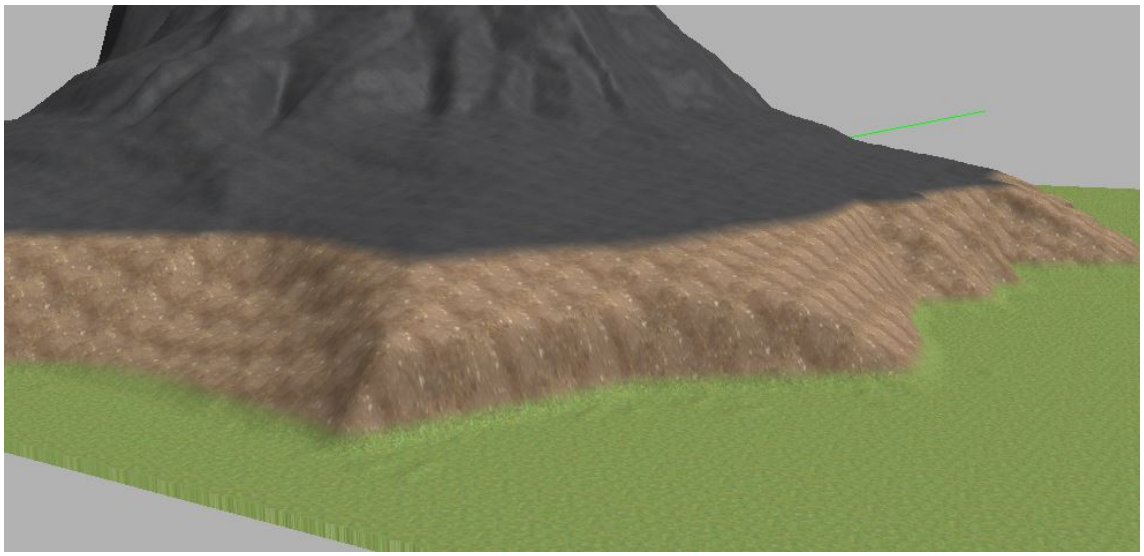
3.5.1 Gazebo

Gazebole on kirjutatud mitmeid õpetusi, kuidas maailma luua, muuta, mis elemente lisada [64]. Gazebo maailmad kirjeldatakse XML-failidena (programmilaiendiga .world). Maailmasid on võimalik luua ka otse Gazebo programmivaates - mudeleid on võimalik maailma vaatesse mudelite nimekirjast sikutada (drag and drop) ja nende positsiooni määrata. Lõpuks salvestatakse maailm siiski XML-failiks. Iga mudeli saab salvestada eraldi mudeli failiks (XML-tüüpi fail, laiendiga .sdf) ning maailma failis viidata vastavatele mudeli failidele, mida parajasti soovitakse kasutada. See lubab kasutajal maailmasid ja mudeleid paremini organiseerida.

Asjaolu, et maailma loomine toimub XML-failis, omab eeliseid ja puuduseid. Üks suur puudusi on see, et mudelite muudatusi ei ole võimalikult koheselt simulaatoris kuvada. Kui kasutaja soovib näiteks teatud kasti liigutada 3 meetrit kõrvale, siis seda on võimalik otse simulaatoris teha. Kui aga kasutaja soovib muuta mingi keerulisema mudeli omadusi nagu näiteks värv, lüli positsioon või hoopis füüsilise komponendi dimensioone, siis selleks peab kasutaja tegema muudatuse mudeli failis, sulgema simulatsiooni ning selle uuesti avama. See aeglustab veidi töö käiku. XML-faili kasutamise eelis on aga võimalus luua skripte, mis genereerivad automaatselt kasutaja soovide järgi mingisugused maailma. See nõuab küll lisatööd kasutaja poolt, kuid võimalus on olemas. Samuti on võimalik XML-faile lihtne testida. Peatükis 5, kus räägitakse maailma konverterist, räägitakse täpsemalt, miks on positiivne, et XML-faile on lihtne testida.

Kui kasutaja soovib luua naturaalset ebatasast maastikku, siis selleks on kõige otstarbekam kasutada Gazebosse sisse ehitatud DEM-i funktsionaalsust (DEM - Digital Elevation Map). DEM lubab kasutajal genereerida maastiku, kasutades selleks halliskaalal pildifaili. Pildi tumedamad pikslid tähendavad maastiku madalamaid kohti, heledamad pikslid tähistavad maastiku kõrgemaid kohti. DEM-il esinevad aga Gazebos

erinevad probleemid. Nagu stabiilsuse peatükis mainitud, siis DEM-i füüsiline komponent ei ole igalpool ideaalne, mõned esemed võivad maastikust läbi kukkuda. Samuti tekib probleeme mitme erineva tekstuuri kasutamisel - kui kasutaja soovib kasutada muru ja kivide tekstuuri, siis tekstuurile saab määrata ainult kõrguse parameetri. See tähendab, et näiteks 10 meetri kõrgusel asub alati kivi tekstuur ning samale kõrgusele mõlemat tekstuuri panna ei saa. Selle illustreerimiseks on Gazebos loodud maailm, mis kujutab mägist pinda. Kui kasutaja soovib lisada mäe ülemisele osale väikese murulapi, siis seda Gazebos võimalik teha ei ole.



Joonis 6. Tekstuurid Gazebos.

3.5.2 CARLA

Oluline CARLA maailmade loomisel on see, et kasutaja peab CARLA lähtekoodist kompileerima. CARLA dokumentatsioonist võib leida õpetusi, kuidas luua uusi maailmasid [65]. Antud õpetus näitab, kuidas luua maailm RoadRunner tarkvaraga, mis sobib hästi kokku CARLA-ga, kuna see on loodud linnaliikluse simuleerimiseks. Mis siis kui kasutaja soovib hoopis luua maailma ilma konkreetsete teedeta, vaid naturaalset looduslikku keskkonda ebatasase pinnasega?

Maailm ise luuakse UE4 mootoris, seega kasutatakse UE4 tööriistu. UE4-s luuakse maailm peamiselt lohistamise meetodil - kasutaja loob esmalt vajalikud mudelid (või laeb need alla internetist: UE4-l on palju vabavaralisi mudeleid, mida võib iga kasutaja alla laadida) ning seejärel lohistab need maailmasse. UE4 eeliseks Gazebo üle on see, et UE4-s on võimalik mudeleid kohe UE4 programmiaknas muuta - programmi ei pea taaskäivitama. Lisaks sellele on UE4-s võimalik luua palju keerukamaid polügone kui Gazebos. Keeruliste polügonide lisamiseks Gazebosse peab mudelid looma välise 3D-modelleerijaga (näiteks Blender), UE4-s on aga modelleerija juba sisse ehitatud.

Kui rääkida konkreetsemalt maastiku loomisest, siis UE4-l on selle jaoks väga kasulikud tööriistad [66]. Maastiku saab genereerida sarnaselt Gazebole halliskaalal pildifaili järgi, aga samas on võimalik maastikku luua või muuta ka käsitsi. Käsitsi muutmise tööriistad lubavad kasutajal luua palju loomulikuma keskkonna. Järgnevalt loetletakse olulisemad tööriistad:

- Värvi tööriist - lubab kasutajal tekstuure pinnasele ”värvida”. Kasutaja saab valida pintsli tugevuse, suuruse läbipaistvuse. Kasulik olukordades, kus tekstuurid võivad esineda läbisegi (murulapi peal on pisike kivihunnik jne.).
- Erosiooni tööriist - lubab kasutajal simuleerida pinnase erosiooni. See tööriist on väga kasulik naturaalse keskkonna lihvimiseks - jääb mulje nagu pinnas on aastaid kulunud.
- Vormimise tööriist - lubab kasutajal määratud punkti maastikul tõsta või langetada.
- Rambi tööriist - lubab kasutajal luua sirge rambi ühest punktist teise. Kasulik kallakuga tee loomisel.

Kui maailm on UE4-s valmis loodud, tuleb see maailm CARLA simulaatoris tööle panna. Üks võimalus on lihtsalt UE4 programmiaknas ”Play” nupu vajutamine, mis käivitab simulatsiooni parajasti muudetavas maailmas. See aga kulutab palju ressursse - optimaalsem oleks loodud maailm käivitada sarnaselt eelkompileeritud versiooniga, kus käima pannakse ainult maailm ise (server), mitte ka UE4 programm. Selleks, et loodud maailma CARLA simulaatoris käima panna, tuleb UE4-s avatud CARLA projekt koos kasutaja poolt lisatud maailmaga avatavaks mänguks kompileerida (inglise keeles kasutatakse terminit ”cook”). Selle jaoks on loodud instruktsioonid [67] (ei leidu ametlikus dokumentatsioonis, seega võib tekkida probleeme leidmisega).

3.5.3 LGSVL

Maailma loomise protsess LGSVL-is on sarnane CARLA-ga. Esitatud juhend maailma loomiseks näitab, kuidas luua linna keskkond teede ja ristmikega [68], samas kui töö eesmärk on analüüsida, kui lihtne on luua pigem looduslik keskkond. LGSVL tuleb Unity mootoris projektina avada ning maailm tuleb luua Unity tööriistasid kasutades.

Unity maailma loomisel lohistatakse aktiivsesse maailmasse mudeleid ning määratakse nende positsioon - sarnane UE4-ga. Maastiku loomise tööriistad on samuti sarnased UE4-ga, kuigi UE4-l on Unityst veidi suurem valik. Unitys saab maastiku luua halliskaalal pildifailist või vabakäe tööriistu kasutades [69]. Kui Unityl esinevad sellised tööriistad

nagu vormimise tööriist ja lihvimise tööriist, siis puuduvad näiteks erosiooni ja rambi tööriist. Seetõttu on UE4 veidi parem loodusliku maastiku loomisel.

Pärast maailma loomist võib selle otse käivitada Unity programmiaknast, kuid eelistatav oleks kompileerida projekt eraldiseisvaks mänguks ning sealt loodud maailm avada. Kahjuks ei ole selle tegevuse jaoks loodud juhendit. Eksisteerib juhend, kuidas Unity projektist eraldiseisev LGSVL versioon kompileerida [70], kuid kuidas uut maailma lisada, kuskil kirjeldatud ei ole, seega kasutaja peab palju vaeva nägema, et enda loodud maailm lisada.

3.6 Uute robotite loomine

Selles peatükis kirjeldatakse uute robotite loomist ja kasutamist. Tihtipeale on mõistlikum alla laadida olemasolev robot ning modifitseerida vastavalt vajadusele.

Gazebo - Gazebos roboti loomiseks eksisteerib juhend, mis näitab kasutajale, kuidas nullist luua väga lihtne kaherattaline sõiduk [71]. Üldiselt luuakse roboteid sarnaselt maailmadega - robot kirjeldatakse XML-failis (.sdf faili formaadis). Robot koosneb tavaliselt mitmest lülist (keha, rattad) ning nende komponentide vahel eksisteerivad liigendid, mis piiravad lülide liikumist (ning rattad tohivad ainult ümber ühe telje pöörelda). Roboti võib luua ka välises 3D-modelleerimisprogrammis, kuid oluline on see, et roboti komponendid tuleb luua eraldiseisvatena (st. üks 3D fail mudel on sõiduki ratas, teine fail on sõiduki keha jne.) ning alles seejärel XML-failis liigenditega ühendada. Selleks, et robotit seejärel ROS-ga kontrollida, tuleb kirjutada pistikprogramm (või kasutada olemasolevat), mis edastaks käske ratastele.

CARLA - CARLA dokumentatsioonis soovitatakse kasutada olemasolevat sõidukit ning modifitseerida vastavalt vajadusele, peamiselt nendes sisalduva skelett-süsteemi tõttu, mis seob sõiduki komponendid kokku [72], [73]. Samuti eksisteerib olemasolevates sõidukites loogika, mis suudab sõiduki ROS-ga suhtlema panna. Kui kasutaja soovib sõidukit nullist luua, siis selleks on tarvis veidi rohkem tööd teha, esiteks tuleb sõiduki mudel luua (kas välises 3D-modelleerimisprogrammis või UE4-s endas), seejärel tuleb sõiduki lülid (rattad) liigenditega siduda [74]. UE4 eelis on asjaolu, et sõiduki komponendid ei pea olema eraldi failides. Lõpuks tuleb CARLA juhendi järgi lisada sõidukile loogika, mis lubaks ROS-ga suhelda.

LGSVL - LGSVL-i video kanalilt võib leida juhendi, mis näitab kuidas lisada uus sõiduk LGSVL-i ning mis seadistused on tarvis sõidukile teha. Sama juhend leidub ka dokumentatsioonis - see juhend lisati 25-ndal aprillil 2019. (töö kirjutamise ajal) [75]. Sarnaselt teiste simulaatoritega, soovitatakse kasutada olemasoleva sõiduki seadeid, ning

seejärel soovitud komponente muuta. Loomulikult on võimalik ka Unitys luua sõiduk täiesti nullist [76]. Sellisel juhul on mõistlik siiski vaadata olemasolevate sõidukite ülesehitust, et aru saada, kuidas sõiduk simulatsiooni jaoks seadistada tuleb.

4 Jõudlustestimine

Selles peatükis kirjeldatakse kolme simulaatori peal jooksutatud koormustestidest ja nende tulemustest. Iga simulaatori peal jooksutati mitu stsenaariumit ning mõõdeti meetrikat. CARLA ning LGSVL puhul jooksutati teste kahte moodi: simulatsioon käivitati otse UE4 või Unity programmiaknas või mänguks kompileeritud versioonis. Mõned simulaatorid võimaldavad käivitada simulatsiooni ilma kasutajaliideseta või võimaldavad simulatsiooni poole pealt peatada. Et koormustestide võrdlus oleks võrdväärne, siis jooksutatakse kõik stsenaariumid simulaatorites kasutajaliidesega ning maailm peab olema käivitatud olekus (mitte pausi peal).

Lisaks sellele käsitleti kahte tüüpi maailmasid: tühi maailm ning suuremahuline maailm. Tühjas maailmas ei eksisteeri midagi muud peale simulatsiooniks vajalikud komponendid. Suuremahulise maailma all mõeldaks mitmeid elemente koosnevat maailma - et võrdlus oleks õiglane, peab igas simulaatoris testitav suuremahuline maailm olema võrdväärne teistega mingi kriteeriumi alusel. Seda on kõige lihtsam teha maastikuga. Iga simulaatori jaoks loodi samasugune maastik suurusega 2048 x 2048 meetrit. Maastikud genereeriti sama halliskaala pildi järgi, et polügonide arv oleks sama.

4.1 Süsteem

Testide osad mõõdetud tulemused sõltuvad süsteemi võimekusest, loogiliselt mida võimekam on süsteem, seda paremad osad tulemused, nagu kaadrisagedus on.

Testid jooksutati Windows 10 operatsioonisüsteemiga arvutil. VMWare tarkvara kasutades jooksutatakse Ubuntu 18.04 LTS operatsioonisüsteemi virtuaalmasinas. Seetõttu on süsteemi ressursid efektiivselt kaheks jaotatud: pooled ressursid kasutab Windows 10, pooled ressursid kasutab Ubuntu virtuaalmasinas.

Süsteemi spetsifikatsioon:

- Protsessor: Intel Core i7-4712MQ, 2.30 GHz
- RAM (vahemälu): 16.0 GB
- Videokaart: Nvidia GeForce 840M
- Videokaardi RAM: 2.0 GB

- Operatsioonisüsteem: Windows 10 64-bit

VMWare virtuaalmasina sätted (võetud ingliskeelsena):

- Processor cores: 4
- Memory: 8192 MB
- OS: 18.04 LTS

Meetrika, mida mõõta:

- FPS - kaadrisagedus, mugav kasutajale; muudetakse ainult juhul kui tegemist on programmiaknas jooksutatud simulatsiooniga, see tähendab Gazebos või UE4 või Unity programmiaknas jooksutatud projektina. See väärtus võib samuti tunduvalt varieeruda sõltuvalt sellest, kui sisse zuumitud simulaatori vaade on.
- RAM / Vahemälu kasutus
- CPU kasutus
- RTF - Real Time Factor ehk pärisaja ning simulatsiooniaja suhe. Näitab kui kiiresti simulatsioon suudab arvutusi sooritada. Tavaliselt on see väärtus 1, kuid kui simulaator ei ole võimeline piisavalt kiiresti vajalikke arvutusi sooritama, siis see väärtus langeb.

Loetletud meetrika on mõõdetud momendil, kus arvuti fookus on simulaatori peal - kui fookus on mõne teise programmi peal, siis ressursside kokkuhoiu eesmärgil vähendab arvuti simulaatori poolt kasutatud ressursse.

4.2 Tühi maailm robotita

See stsenaarium on väga lihtne: simulatsiooni maailm on täiesti tühi. Eesmärk on näha, kuidas simulaator jookseb siis kui midagi ei toimu. Maailmas eksisteerib siiski sirge põrand ning tavaline valgustus.

Tabel 3. Stsenaarium 1 - Tühi maailm robotita

	FPS (fps)	**RAM (MB)**	**CPU (%)**	**RTF**
Gazebo	20	475	53	1
CARLA	60	256.4	11.5	1
CARLA UE4-s	60	1332.4	11.8	1
LGSVL	60	811.5	11.8	1
LGSVL Unitys	500	1030.7	35.5	1

4.3 Suuremahuline maailm robotita

See stsenaarium mõõdab kui hästi suudab simulaator maailma visualiseerimisega toime tulla.

Tabel 4. Stsenaarium 2 - Suur maailm ilma robotita

	FPS (fps)	**RAM (MB)**	**CPU (%)**	**RTF**
Gazebo	3.5	548	35	1
CARLA	60	269	13.2	1
CARLA UE4-s	60	1259.9	14.8	1
LGSVL	60	875.6	17.1	1
LGSVL Unitys	440	983.5	42.4	1

4.4 Üks ROS-il põhinev robot sõitmas tühjas maailmas

See stsenaarium mõõdab kui hästi suudab simulaator ühe sõiduki sõitmise protsessiga toime tulla.

Tabel 5. Stsenaarium 3 - Üks robot tühjas maailmas

	FPS (fps)	**RAM (MB)**	**CPU (%)**	**RTF**
Gazebo	8.2	614.4	43.3	1
CARLA	60	258.3	14.7	1
CARLA UE4-s	60	1374.6	13.3	1
LGSVL	60	1013.9	21.2	1
LGSVL Unitys	172	1042.1	34.1	1

4.5 Üks ROS-il põhinev robot sõitmas suuremahulises maailmas

See stsenaarium mõõdab kui hästi suudab simulaator ühe sõiduki sõitmise protsessiga ning suuremahulise maailma visualiseerimisega toime tulla.

Tabel 6. Stsenaarium 4 - Üks robot suures maailmas

	FPS (fps)	**RAM (MB)**	**CPU (%)**	**RTF**
Gazebo	7.53	663.5	43.6	0.98
CARLA	60	272.5	15.0	1
CARLA UE4-s	60	1264.9	16.1	1
LGSVL	60	1022.1	22.8	1
LGSVL Unitys	105	1056.1	25.8	1

4.6 10 ROS-il põhinevat robot sõitmas tühjas maailmas

See stsenaarium mõõdab kui hästi skaleerub simulatsiooni ressursikasutus mitme roboti korral.

Tabel 7. Stsenaarium 5 - 10 robotit tühjas maailmas

	FPS (fps)	**RAM (MB)**	**CPU (%)**	**RTF**
Gazebo	2.44	925.7	36.7	0.85
CARLA	60	279.1	21.7	1
CARLA UE4-s	60	1479	20.9	1
LGSVL	60	1190	17.0	1
LGSVL Unitys	63.5	1259	32.1	1

4.7 10 ROS-il põhinevat robot sõitmas suuremahulises maailmas

See stsenaarium mõõdab kui hästi skaleerub simulatsiooni ressursikasutus mitme roboti korral suuremahulises maailmas.

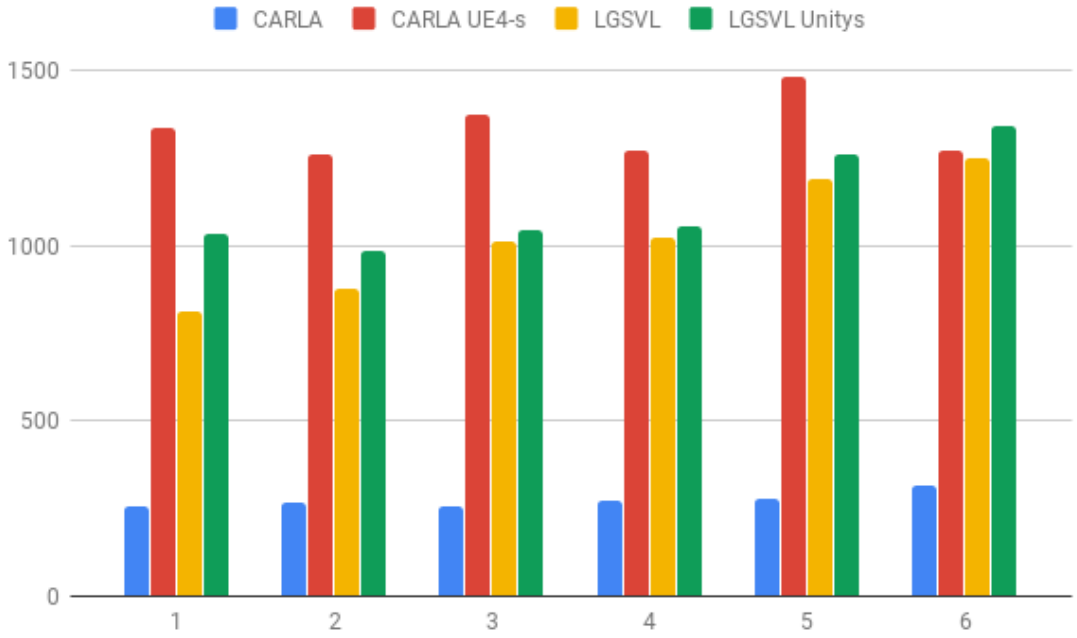
Tabel 8. Stsenaarium 6 - 10 robotit suures maailmas

	FPS (fps)	**RAM (MB)**	**CPU (%)**	**RTF**
Gazebo	1.87	1007.6	39.7	0.81
CARLA	60	317.6	24.3	1
CARLA UE4-s	60	1269.2	18.2	1
LGSVL	60	1248.8	17.2	1
LGSVL Unitys	35	1340.2	24.0	1

4.8 Tulemuste analüüs

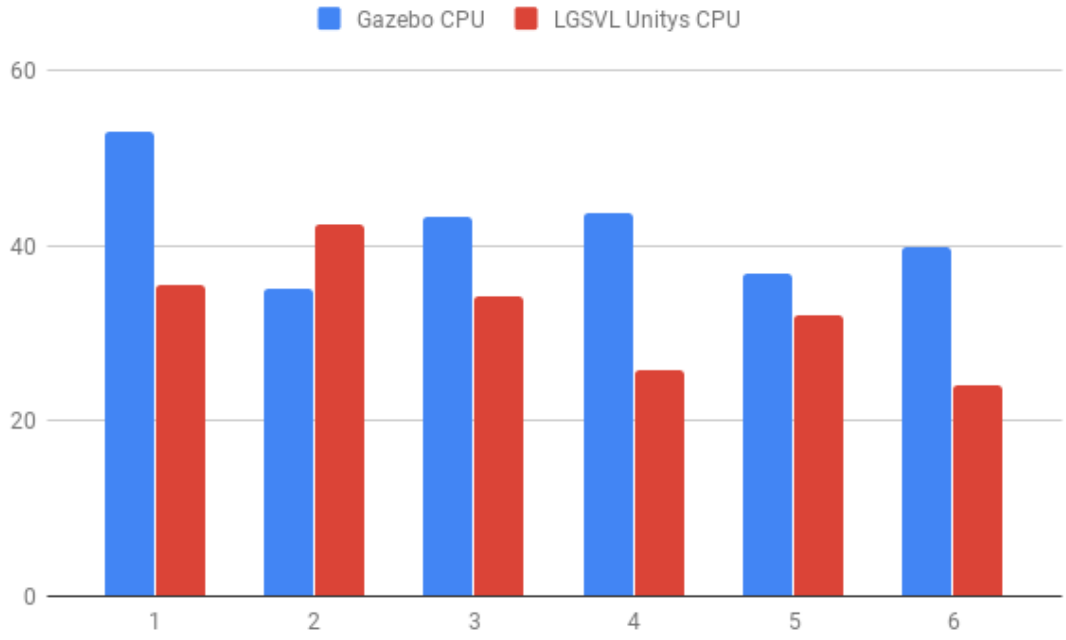
UE4-l põhinev CARLA piirab kaadrisageduse 60 peal, mis on mõistlik, kuna simulatsioonis ei ole üldiselt kõrgemaid kaadrisagedusi tarvis. Nagu näha siis Unity programm ise kaadrisagedust ei piira ning uuendab maailma visuaalset komponenti nii kiiresti kui võimalik. Gazebo kaadrisagedus kannatab kõige enam, langedes kuni paari kaadrini sekundis, mis muudab kasutajal simulatsiooni maailmas ringi liikumise väga ebamugavaks.

Vahemälu kasutused on kõrvutatud joonisel 7. Konkreetselt vaadeldakse kompileeritud versiooni kasutamist võrreldes UE4 või Unity projektina. Nagu näha, siis kompileeritud CARLA versioon hoiab stsenaariumites tunduvalt rohkem vahemälu kokku kui UE4 programmiaknas kasutatud CARLA versioon. LGSVL-i puhul hoiab samuti eelkompileeritud versioon vahemälu kokku, kuid mitte üldse nii palju kui CARLA. Kui üldiselt vahemälu kõikides simulaatorites võrrelda, siis CARLA kompileeritud versioon on teistest simulaatoritest oluliselt parem. Samuti võib jooniselt näha, et vahemälu kasutus on intuitiivselt loogiline, st. hilisemates stsenaariumites kasutatakse rohkem vahemälu, kuna stsenaarium sisaldab rohkem elemente.



Joonis 7. Stsenaariumite vahemälu kasutus.

Huvitav tähelepanek Gazebo ja LGSVL Unity projektina tekkis CPU juures. Kui CARLA puhul oli CPU kasutus stsenaariumite vältel kasvav, siis Gazebo ja LGSVL Unitys puhul oli CPU pigem langev. Seda võib näha jooniselt 8. Üks sellise fenomeni põhjuseks võib-olla simulaatorite kaadrisagedus. LGSVL Unitys kaadrisagedust ei piira ning seetõttu uuendatakse kaadreid nii kiiresti kui võimalik, seetõttu on CPU kasutus suurem. Sama võib täheldada Gazebo puhul, kuigi Gazebo kaadrisagedus on tunduvalt madalam.



Joonis 8. Stsenaariumite CPU kasutus Gazebos ja LGSVL-s.

5 Maailma konverteerija

Eelnevates peatükkides sooritatud võrdluse põhjal võib kasutaja otsuse teha, milline simulaator temale kõige enam sobib. Kuid tarkvara areneb kiiresti ning kasutaja võib soovida ühe projekti juures simulaatorit vahetada, samas on tülikas luua kõik vajalikud komponendid uuesti teise simulaatori jaoks. Seetõttu võib tekkida vajadus tööriista järele, mis suudaks võimalikult palju projekti simulaatori varad üle kanda teise simulaatorisse, kasvõi osaliselt. Sellest tekkis idee luua projekti konverteri tööriist. Märkus, et kuna CARLA ja LGSVL on ehitatud vastavalt UE4 ja Unity mootoritel, siis reaalselt muundatakse elemente Gazebo, UE4 ja Unity vahel.

Lisaks võib kasutaja motiiviks olla projekti jooksutada mitme erineva simulaatori peal korraga. Mitme simulaatori kasutamine suurendab arendatava tarkvara töökindlust, kuna simulaatorite iseärasused taanduvad mitut simulaatorit kasutades välja. Samuti võib mitme simulaatori kasutamine olla kasulik olukordades, kus simuleeritav robot omab väga palju erinevat funktsionaalsust ning osa funktsionaalsust on optimaalsem simuleerida ühes simulaatoris ning osa funktsionaalsust teises simulaatoris. Näiteks ühes simulaatoris on lihtsam simuleerida täpseid füüsika komponente, teises simulaatoris on lihtsam simuleerida visuaalseid komponente kaamerate jaoks.

5.1 Muundatavad elemendid

Enne töö alustamist tuleb paika panna konkreetsed elemendid, mida muundada, mida üldse saab muundada, ning mis kujult mis kujule selle käigus elemendid vaja viia on. Samuti tuleb arvestada, et tegemist on suuremahulise tööga, seega kõik võimalikku funktsionaalsust ei saa kindlasti täidetud. Keskendutakse pigem funktsionaalsusele, mida jõutakse luua lõputöö käigus.

Esimene suur olem, mida võiks kindlasti muundada, on maailm. Simulatsiooni puhul tähendab maailm keskkonda, kus tegevus toimub. Maailm koosneb erinevatest elementidest nagu maapind, objektid, valgustus, füüsika seaded, jne. Objektitel on omakorda erinevad atribuudid nagu suurus, positsioon, värvus, jne. Seega esimeseks ülesandeks on muundada maailm koos elementidega selle sees. Antud juhul eeldatakse, et tegemist on lihtsate kujunditega (kuup, kera, silinder) - keerulistema kujunditega tegeletakse hiljem.

Teine suurem olem, mida muundada, on mitmest elemendist koosnevad objektid, näiteks sõidukid - üldiselt klassifitseeritakse sellist objekti mudelina. Mudelid koosnevad tihti mitmest lülist, kus lülid on liigenditega seotud teiste lülidega. Eesmärk on muundada mudel niimoodi, et säiliksid originaalsed mudeli mõõtmed ning iga mudeli lüli individuaalsed mõõtmed ning nendega seotud liigendite kitsendused. Jällegi käsitletakse hetkel ainult lihtsamaid kujundeid.

Järgmine olem, mida muundada, on maapind. Maapinda on võimalik UE4 ja Unity mootorites luua käsitsi, seega sama käsitsi loodud maapinna uuesti loomine teises simulaatoris võib osutuda väga tülikaks, eriti kui teine simulaator ei oma täpselt samasuguseid tööriistu. Ideaalis võiks olla ka võimalik korrektsete tekstuuride määramine, kuid näiteks Gazebo piirangute tõttu ei ole see kindlasti identselt võimalik.

Viimane olem, mida muundada, on keerulisemad mudelid. Kui Gazebos kasutatavad põhilised kujundid on kuup, kera ja silinder, siis UE4-s ja Unitys on võimalik kasutada lihtsasti palju keerulisemaid käsitsi modelleeritud kujundeid. Kõikides uuritavates simulaatorites on võimalik kasutada 3D-modelleerimisprogrammis loodud mudeleid, seega toetada tuleb ka nende kasutamist.

Kõik loendatud olemid koosnevad atribuutidest ja teistest alamelementidest, seega on mõistlik läheneda lahendusele objektorienteeritult.

5.2 Mittemuundatavad elemendid

Simulaatorite iseärasuste tõttu leidub elemente, mida ei ole võimalik täiesti õigele kujule muundada:

- Maastiku tekstuuride paigutus - Gazebos on limiteeritud, kuidas tekstuure maastikule sättida
- Andurid, mis eksisteerivad ühes simulaatoris, kuid puuduvad teises
- UE4-s ja Unitys kasutatavad visuaalsed komponendid, näiteks valguse peegeldumine, tolmu simuleerimine jne.
- UE4-s ja Unitys kasutatavad spetsiaalsed objektitüübid, mida kasutatakse ainult mängude loomisel (erinevad kontrollerid, mis juhivad mängu käiku)

Lisaks sellele loetletakse elemendid, mida esimese iteratsiooni käigus ei käsitleta, kuid on plaanis tulevikus arendada:

- Simulaatorile iseärased sätted - näiteks kõik UE4 elemendid ei ole teisendatavad Gazebo elementideks

- Mudelitele või maailmadele kirjutatud pistikprogrammide kujul funktsionaalsus
- Mitme maailma või mudeli korraga muundamine, ehk terve projekti muundamine

5.3 Simulaatorid

Selles jaotises kirjeldatakse iga simulaatori puhul, mis moodi on simulaatori elemendid salvestatud. See on oluline selleks, et teada täpselt, mis kujult mis kujule on tarvis elemente viia.

5.3.1 Gazebo

Gazebo mudelid ja maailmad kirjeldatakse XML-failidena - maailma failid laiendiga .world ja mudeli failid laiendiga .sdf. Mudeli failidel on lisaks .sdf failile kaasas veel .config fail (samuti XML-fail), mis määrab mõned administratiivsed väärtused, kõige olulisem neist mudeli faili asukoht. Lisaks sellele võib mudelile kaasa anda materjali faile ja tekstuuri faile, samuti 3D-mudeli faile. Materjali fail on JSON-tüüpi fail, mis määrab materjali tekstuurid ning nende individuaalsed omadused (läbipaistvuse, RGB väärtused, jne.) ning tekstuurid on tavaliseld pildi failid.

Gazebo XML-failidel on oma konkreetne formaat, mille süntaksit võib uurida siit [77]. Sellises failis võib määrata maailma atribuudid ja maailmas esinevad elemendid. Kuna tegemist on XML failiga, siis on lihtne teda lugeda ükskõik millise programmeerimiskeelega - see lubab lihtsasti muundada faili vajalikuks andmetüübiks, mis mingis teises simulaatoris toimiks.

5.3.2 CARLA

Unreal Engine 4 maailmad ja mudelid on binaarsed andmetüübid, see tähendab, et tekstiredaktorid selliseid faile lugeda ei oska ning kasutaja ei ole võimeline käsitsi sellised faile looma. Ainuke viis, kuidas sellistest failidest infot välja lugeda, on Unreal Engine 4 mootoriga. UE4 maailm koosneb maailma sätetest ning maailmasse lisatud objektidest. Selleks, et seda infot lugeda, tuleb luua UE4 siseselt skript programmeerimiskeeles C++, mis suudab vajaliku API-ga vastavad elemendid tuvastada ning nende kohta infot lugeda (näiteks positsiooon, suurus, jne.) [78]. Samamoodi tuleb käituda maailma loomisel: kasutaja peab kas käsitsi lohistama soovitud elemendid maailmasse või automatiseerimiseks peab kasutaja looma skripti UE4 siseselt, mis soovitud elemendid loob ning maailmasse lisab. Kokkuvõtvalt tähendab see seda, et UE4 maailmadega toimetamisel peab alati UE4 mootor käima.
Lisaks UE4 maailmadele ja mudelitele, talletatakse CARLA projektides ka iga maailma ja mudelite kohta navigeerimisega seotud infot. Näiteks maailmadele luuakse 2D kaardid,

mille järgi sõidukid võivad navigeerida. Kuid see funktsionaalsus on hetkel teisejärguline, mida konveteri esimeses iteratsioonis ei käsitleta.

5.3.3 LGSVL

Unity maailmad ja elemendid on samuti binaarsed andmetüübid, ehk ainult Unity mootoriga on võimalik maailmade või mudelite sisu lugeda ning luua. Erinevus Unity ja UE4 vahel on see, et Unity API kasutab programmeerimiskeelt C#, mis on tunduvalt lihtsam õppida kui C++. Muus infos on Unity sarnane UE4-ga.
Ka LGSVL projektides talletatakse lisainfot maailmade ja mudelite jaoks. Seda funktsionaalsust esimeses iteratsioonis ei käsitleta.

5.4 Simulaatorite muundamine

Järgnevalt kirjeldatakse konkreetsete simulaatorite vahelisi seoseid, kuidas viia vajalikud elemendid teise simulaatori kujule.

5.4.1 UE4/Unity - Gazebo

Kuna UE4 ja Unity kasutavad sarnast maailma struktuuri loogikat (lihtsalt omavad erinevaid API-sid), siis kehtib järgnev alampeatükk mõlema UE4 ja Unity kohta.
Kui muundada UE4 või Unity maailm Gazebo maailmaks, siis selleks peab looma skripti vastavas mängumootoris järgmise funktsionaalsusega:

- UE4/Unity maailmast sätete ning elementide lugemine
- Iga individuaalse elemendi atribuutide lugemine
- Muundamine kogutud info XML-faili kujule (laiendiga .world, järgides korrektset süntaksit [77])

Kui muundada UE4 mudel Gazebo mudeliks, siis selleks peab looma skripti järgmise funktsionaalsusega:

- UE4/Unity mudelist alamelementide ja atribuutide lugemine
- Muundamine kogutud info XML-faili kujule (laiendiga .sdf, järgides korrektset süntaksit [77])
- Lisa failide loomine materjalide ja tekstuuride jaoks

Selleks, et muundada Gazebo maailm UE4 või Unity maailmaks, tuleb luua skript vastavas mängumootoris järgneva funktsionaalsusega:

- Maailma XML-faili lugemine
- Iga individuaalse elemendi atribuutide lugemine
- Kogutud info XML-failist muundamine UE4/Unity kujule, see tähendab iga individuaalne objekt tuleb luua, maailmasse paigutada ning seejärel tema vastavad atribuudid määrata

Lisaks tuleb tähele panna teatud iseärasusi, näiteks UE4 kasutab peamise ühikuna sentimeetreid (teised kasutavad meetreid). Unity Y- ja Z-teljed on omavahel vahetuses.

Selleks, et muundada Gazebo mudel UE4 või Unity mudel, tuleb luua skript vastavas mängumootoris järgneva funktsionaalsusega:

- Mudeli XML-faili lugemine, atribuutide kogumine
- UE4-s/Unitys vastava mudeli loomine ning atribuutide seadmine
- Kui mudel sisaldab kasutaja poolt loodud materjale, siis nende failide lugemine ja vastava materjali loomine UE4-s/Unitys. Kui mudel sisaldab Gazebosse sisse ehitatud materjale, siis on võimalus luua vastavus UE4-s/Unitys olemasolevatele materjalidele (näiteks igasse simulaatorisse on sisse ehitatud lihtne valge plastmassi materjal).

5.4.2 UE4 - Unity

Kuna mõlemad UE4 ja Unity maailmad ja mudelid on binaarsed failid ning ei eksisteeri sellist mootorit, mis suudaks neid faile mõlemat lugeda, siis on kõige mõistlikum muundada ühe simulaatori maailm Gazebo maailma kujule ning seejärel Gazebo maailm teise simulaatori kujule. Seega on Gazebo XML-kujul failid vahendajaks UE4 ja Unity vahel. Selline vahendaja peab eksisteerima nii või teisiti, kuna ükski redaktor ei suuda lugeda mõlemat UE4 ja Unity faile. Sellega võib muidugi tekkida olukordi, kus tekib teatud info kadu, kuna Gazebo maailm või mudel ei pruugi võimeline olla talletada sama detailset infot, mis UE4 või Unity maailm või mudelid suudavad. Samuti tekib kadu olukorras, kus kasutaja muundab UE4 maailma Gazebo maailmaks ning seejärel tagasi UE4 maailmaks.

5.4.3 Funktsionaalsus

Eesmärk on luua järgmine funktsionaalsus:

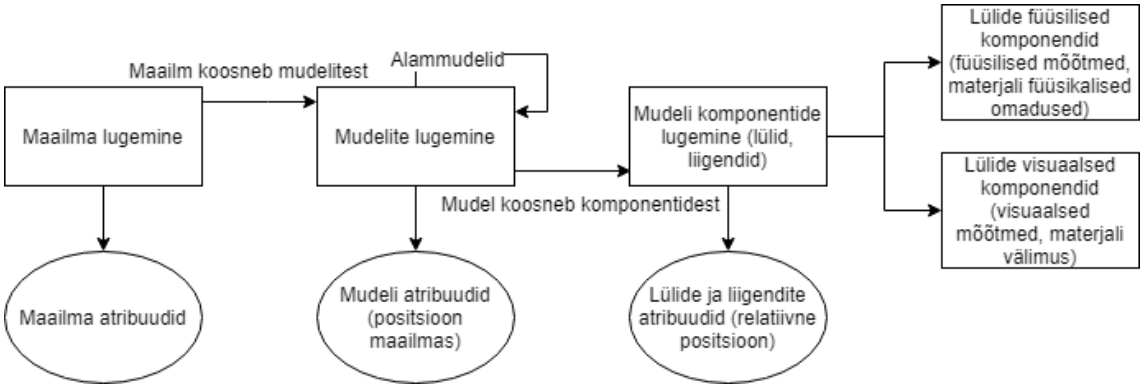
- Maailma muundamine
  - Korrektse maailma faili loomine, mis on avatav vastava simulaatoriga

  - Maailmas sisalduvate objektide lisamine õigetele positsioonidele
  - Maailma sätete õige ülekandmine (näiteks raskuskiirendus, varjude visaliseerimine ja palju muud visuaalseid sätteid)
  - Lisa elementide muundamine, näiteks valgustus

- Mudeli muundamine
  - Lihtsamate kujundite muundamine (kuup, silinder, kera) õigete dimensioonidega
  - Mudelitele tekstuuride ja materjalide lisamine
  - Maastiku konverteerimine
  - Keerulistemate mudelite muundamine

Loetletud funktsionaalsus tuleb luua nii UE4 ja Gazebo vahelisele konverterile ning Unity ja Gazebo vahelisele konverterile. Lisaks tuleb arvestada sellega, et teatud simulatsiooni elemente on võimalik teha mitut moodi, (näiteks UE4-s on võimalik maastikku kahte moodi luua) seega esialgu prioritiseeritakse võimalikult laia funktsionaalsuse katmist, mitte iga funktsionaalsuse perfektne töötamine iga võimaliku süntaksiga.

Maailma lugemise visuaalseks esitamiseks loodi ka diagramm. Jooniselt võib näha, kuidas maailmal on omad atribuudid ning koosneb mudelidest, mis sinna maailmasse sisestatud on. Mudelid võivad omakorda koosneda alammudelitest ning esimese taseme mudelid koosnevad lülidest ja liigenditest, mis koosnevad füüsilistest ja visuaalsetest komponentidest. Kõikidel nimetatud elementidel on oma atribuudid, mis muundatakse teise simulaatori kujule. Alustada tuleb mudelite asukohast ja dimensioonidest ning edasi tuleb liikuda täpsemate detailidega nagu materjali välimus.



Joonis 9. Koverteri maailma lugemine.

5.5 Käivitamine

Esimeses iteratsioonis loodud tarkvaras tuleb kasutajal sooritada teatud määral manuaalset tööd. Esiteks on peatükis 5.4.1 kirjeldatud stsenaariumite kohta iga ühe jaoks kaks skripti (iga stsenaariumi jaoks üks skript maailma muundamiseks, teine skript mudeli muundamiseks), seega kokku 8 erinevat skripti - kasutaja peab kasutama õiget skripti mõeldud konkreetse stsenaariumi jaoks.

Kui kasutaja soovib muundada UE4 või Unity maailma Gazebo maailmaks, siis selleks peab kasutaja käituma järgnevalt:

- Soovitud maailma avamine UE4 või Unity programmiaknas
- Tühja objekti lisamine maailma (UE4-s *Empty Actor*, Unitys *Empty Object*)
- Õige skripti (tuleb kasutada vastavat skripti kirjeldatud stsenaariumites peatükis 5.4.1) kinnitamine äsjaloodud objekti külge, esmakordsel kasutamisel on tarvis skript projekti importida
- Parameetrite sisestamine: salvestatava maailma faili soovitav asukoht failisüsteemis
- ”Play” nupu vajutamine - see käivitab simulatsiooni ning loeb vajalikud andmed maailmast ning seal sisalduvatest elementidest, seejärel genereeritakse kasutaja poolt määratud asukohta .world laiendiga Gazebo maailma fail

Kui kasutaja soovib muundada Gazebo maailma UE4 või Unity maailmaks, tuleb kasutajal käituda järgnevalt:

- Tühja maailma avamine UE4-s või Unitys
- Tühja objekti lisamine maailma (UE4-s *Empty Actor*, Unitys *Empty Object*)
- Õige skripti (tuleb kasutada vastavat skripti kirjeldatud stsenaariumites peatükis 5.4.1) kinnitamine äsjaloodud objekti külge, esmakordsel kasutamisel on tarvis skript projekti importida
- Parameetrite sisestamine: Gazebo maailma asukoht failisüsteemis
- ”Play” nupu vajutamine - see käivitab simulatsiooni, loeb vajalikud andmed viidatud Gazebo maailmast ning seal sisalduvatest elementidest, seejärel genereeritakse olemasolevasse tühja maailma kõik Gazebo maailmas leiduvad elemendid

- Et genereeritud elemendid säiliksid, peab kasutaja selekteerima programmiaknas kõik genereeritud elemendid need lõikelauale (*clipboard)*, simulatsioon tuleb peatada ning seejärel tuleb kleepida lõikelaualt elemendid - UE4 ja Unity maailmadesse on võimalik elemente lisada ainult simulatsiooni jooksutamise ajal, seetõttu on tarvis kirjeldatud lisa samm

Kui kasutaja soovib muundada UE4 või Unity mudelit Gazebo mudeliks, siis selleks peab kasutaja käituma järgnevalt:

- Soovitud mudeli lisamine ükskõik millisesse UE4 või Unity maailmasse avatud programmiaknas
- Tühja objekti lisamine maailma (UE4-s *Empty Actor*, Unitys *Empty Object*)
- Õige skripti (tuleb kasutada vastavat skripti kirjeldatud stsenaariumites peatükis 5.4.1) kinnitamine äsjaloodud objekti külge, esmakordsel kasutamisel on tarvis skript projekti importida
- Parameetrite sisestamine: muundatava mudeli nimi, salvestatava mudeli ja lisafailide soovitav asukoht failisüsteemis
- ”Play” nupu vajutamine - see käivitab simulatsiooni ning loeb vajalikud andmed mudelist, seejärel genereeritakse kasutaja poolt määratud asukohta vastavalt Gazebo nõuetele vajalikud failid

Kui kasutaja soovib muundada Gazebo mudelit UE4 või Unity mudeliks, tuleb kasutajal käituda järgnevalt:

- Ükskõik millise maailma avamine UE4-s või Unitys
- Tühja objekti lisamine maailma (UE4-s *Empty Actor*, Unitys *Empty Object*)
- Õige skripti (tuleb kasutada vastavat skripti kirjeldatud stsenaariumites peatükis 5.4.1) kinnitamine äsjaloodud objekti külge, esmakordsel kasutamisel on tarvis skript projekti importida
- Parameetrite sisestamine: Gazebo mudeli asukoht failisüsteemis
- ”Play” nupu vajutamine - see käivitab simulatsiooni, loeb vajalikud andmed viidatud Gazebo mudelist, seejärel genereeritakse olemasolevasse maailma uus mudel, mis salvestatakse hetkel aktiivsesse projekti

Kui kasutaja soovib muundada UE4 maailmast Unity maailma või vastupidi, siis selleks tuleb kasutada kahte skripti, üks selleks et muundada esialgne komponent Gazebo komponendiks (XML kujul) ning teise skriptiga muundatakse Gazebo komponent soovitud simulaatori komponendiks. Sellisel juhul tuleb järgida kahte eelnevalt loetletud juhiseid sõltuvalt sellest kas muundatakse maailmat või mudelit.

5.6 Tuleviku väljavaated

Selles peatükis arutatakse konverteri funktsionaalsust, mida hetkel loodud ei ole, kuid on mõistlik luua tulevikus.

5.6.1 Vahendaja

Tarkvara järgmistes iteratsioonides tuleb mõelda selle peale, et hetkel kasutatav vahendaja ei ole piisav kõikide UE4 ja Unity funktsionaalsuse katmiseks, st. hetkel tekib infokadu teatud elementidega, mida Gazebo simulaatoris ei kasutata. Esialgu on võimalus lisada olemsaolevasse XML faili lisavälju ainult konverterile kasutamiseks, mida Gazebo simulaatoris ei loeta, kuid konverter arvestab ning kannab üle vastavalt UE4 või Unity mootorisse. Hiljem võib olla mõistlik luua uus märgistuskeel, mis arvestab just konkreetselt uuritavate simulaatorite iseärasusi.

Uue märgistukeele kasutamine võimaldab teoreetiliselt ka kasutajal kasutada kõiki kolme simulaatorit paralleelselt. Sellisel juhul peab kasutaja looma vastavalt loodud märgistuskeelele maailmade ja mudelite kirjeldused ning kasutades konvertereid muundada vastavalt soovitud simulaatorisse.

5.6.2 Ebakindla andmestiku täitmine

Elementide muundamisel võib ette tulla olukordi, kus konverter ei suuda täpselt teatud mudeli väärtusi üle kanda. Üks näide sellisest olukorrast on maastikule tekstuuride määramine. UE4-s ja Unitys on seda võimalik vabakäe tööriistadega teha, Gazebos mitte. See tähendab, et ei leidu triviaalset meetodid, mis suudab määrata, millised maastiku tekstuuri sätted tuleb Gazebo maastikule määrata. Sellisel juhul tuleb kasutajale sellisest ebakindlusest teada anda ning lasta kasutajal käsitsi antud väli täita.

5.6.3 Automatiseerimine

Esimeses iteratsioonis loodud tarkvara kasutades peab kasutaja kõik projekti maailmad ja mudelid ükshaaval muundama ning määrama uute loodud failide asukohad failisüsteemis. Lisaks peab kasutaja UE4 ja Unity maailmade puhul avama vastavad programmi ning seal sees skripti jooksutama. Selline kasutusviis on paari elemendi muundamisel veel lihtne, aga kui kasutaja projekt koosneb väga paljudest elementidest, siis muutub tegevus tüütuks ning võib põhjustada kasutajale segadust.

Kasutaja aja kokkuhoiu huvides on mõistlik luua tööriist, mis automatiseerib projekti konverteerimise protsessi. See tähendab kasutaja saab määrata mitu projekti elementi korraga, mida soovitakse muundada, mis simulaatori projektiks soovtakse muundada ning kuhu tekkinud failid salvestada. Lisaks on kasulik kuvada kasutajale kasutajalogi, mis kuvab kasutajale, mis elemendid said edukalt muundatud ning kas tekkis probleeme.

Ideaalis oleks sellisel tööriistal ka lihtne kasutajaliides, mis lubab kasutajal sirvida läbi failisüsteemi ning valida vastavad elemendid, mida muundada. Käivitades kuvatakse kasutajale progressilogi ning tegevuste logi, seega kasutaja teab, mis elemendid on muundatud ning kus maal protsessiga ollakse.

5.6.4 Robotite funktsionaalsus

Robotite funktsionaalsuse saab luua pistikprogrammide abil. Pistikprogrammides määratakse loogika, kuidas roboti komponendid erinevate käskude puhul käituvad. Simulaatori pistikprogrammid kasutavad kasutatava simulaatori API-t. Selle loogika muundamine on kindlasti kasutajale väga kasulik, kuna see tähendab, et kasutaja ei pea sama funktsionaalsust teises keeles või teise API-ga kirjutama.

Muidugi tekib siinkohal probleemikoht, kuna ühest API-st teise muundamine võib olla suur töö. Samuti ei pruugi ühe programmeerimiskeele mooduli teise keelde muundamine olla täiesti töökindel. Eksisteerib näiteks C++ ja C# vaheline konverter [79], kuid kui hästi see rakendub simulaatori API-de muundamisele, ei ole teada. Võib osutuda, et kõike pistikprogrammides kirjeldatud funktsionaalsust ei ole võimalik muundada, seega saaks kasutada sama loogikat, mida kirjeldati ”Ebakindla andmestiku täitmine” alampeatükis.

5.7 Valideerimine

Konverteri tulemuste valideerimiseks tuleb kirjutada skriptid, mis kontrollivad vastava simulaatori maailmas võid mudelis, kas vajalikud elemendid eksisteerivad ning kas nende elementide atribuudid on korrektsed. Testimise protsessis tuleb esmalt luua maailm erinevate objektidega, millel igaühel on erinevad atribuudid (oluline on kasutada kõiki elemente eelmises alampeatükis loetletud funktsionaalsusest). Seejärel tuleb luua kontrollskript, mis kontrollib, kas muundatud maailmas eksisteerivad vastavad elemendid vastavate atribuutidega, mis kasutaja varasemalt valmis ehitas. Sama protsessi korratakse mudelite puhul.

Kõige lihtsam on kontrollskripte kirjutada Gazebo maailmadele ja mudelitele, kuna tegemist on XML-failiga. Seetõttu võib skripti kirjutada ükskõik, millises programmeerimiskeeles, mis suudab XML faile lugeda (ning vajadusel json faile ja failisüsteemi lugeda). Töö autor valis selleks keeleks Pythoni isikliku eelistuse tõttu.

UE4 ja Unity jaoks tuleb luua kontrollskript vastavalt C++ või C# keeles ning see skript peab olema käivitatav vastava mängumootori sees, kuna see on ainuke viis, kuidas nendes mootorites maailmasid ja mudeleid lugeda.

Kontrollskriptid luuakse paralleelselt funktsionaalsusega. Esimeses iteratsioonis kirjutatakse ainult kontrollskriptid, mis kontrollivad, et muundatud maailmad ja mudelid

oleksid korrektsed. Järgmistes iteratsioonides on plaanis ka kirjutada skriptidele ühiktestid, mis kontrollivad skriptide individuaalseid komponente.

6 Kokkuvõte

Magistritöö käigus sai loodud võrdlus Gazebo, CARLA ja LGSVL simulaatorite vahel. Eesmärk oli uurida, kuivõrd hästi tulevad mängumootoreid kasutavad simulaatorid toime ROS-il põhinevate robotite simuleerimisel. Võrdlus koosnes simulaatorite faktide võrdlusest, kus võrreldi simulaatorite lihtsasti määratavaid kriteeriume, sügavamast funktsionaalsuse analüüsist, kus võrreldi sealhulgas simulaatorite võimekust töötada koos ROS2-ga ning viimaks sooritati koormusteste, kus kirjeldati mitu stsenaariumit, mida igas simulaatoris sooritati ning mõõdeti meetrikat.

Gazebo eeliseks teiste simulaatorite ees on asjaolu, et on ametlik ROS-ga kasutatav simulaator, seega omab palju õpetusi ja dokumentatsiooni. Samuti on Gazebo kasutajabaas teistest suurem, mis tähendab, et abi probleemidele on lihtsam leida. Samas jääb teatud funktsionaalsuses Gazebo teistest simulaatoridest taha poole, näiteks töötades ebatasase pinnasega võib Gazebo kasutajale palju probleeme tekitada. Seega Gazebo on soovitatav pigem lihtsamate simulatsioonide jaoks, kus maapind on sirge ning robotid ei ole väga keerulise ehitusega.

CARLA ja LGSVL võimaldavad kasutajal luua Gazebost palju realistlikumaid ning keerulisema pinnavormidega maailmasid. Samuti on palju mugavam luua keerulisi mudeleid kasutades vastavalt UE4 ja Unity mängumootoreid. CARLA ja LGSVL miinuseks on asjaolu, et ei sisalda palju dokumentatsiooni ja õpetusi, seega kasutaja peab tekkivate probleemidega ise hakkama saama. Samuti on nende kasutamine tunduvalt keerulisem kui Gazebo. Seega CARLA ja LGSVL kasutamine on mõeldud pigem nendele kasutajatele, kes on simulatsiooni alas juba asjatundjad ning soovivad luua keskkondi, mis ulatuvad Gazebo võimekusest väljapoole.

Kui võrrelda CARLA-t ja LGSVL-i omavahel, siis nad üksteisega suhteliselt sarnased, kuid mõned erinevused neil siiski on. Hetkel LGSVL toetab ROS2 moodulite kasutamist, samas kui CARLA seda ei toeta. CARLA jaoks peab kasutama lisa moodulit, mis ühendab ROS2 eelmise ROS-i versiooniga. CARLA jaoks eksisteerib veidi rohkem dokumentatsiooni ja õpetusi ning CARLA Discordi serveris on võimalik teiste kasutajatega ja arendajatega suhelda. Lisaks sellele tulevad mängu UE4 ja Unity erinevused näiteks maailmade ja mudelite loomisel. Üldiselt võimaldavad need mängumootorid saavutada samu tulemusid, kuid UE4 maastiku tööriistad on pisut detailsemad ja edasijõudnumad Unity tööriistadest. Kui sooritada valik CARLA ja

LGSVL-i vahel, siis neil ei ole niivõrd suuri erinevusi, kus üks oleks teisest kõvasti üle. Pigem võib kasutaja toetuda isikliku eelistuse peale. See kehtib siis kui ignoreerida koormusteste.

Kui toetuda koormustestidele, siis kompileeritud CARLA versiooni kasutamine on kõige enam ressursse kokkuhoidvam. Eriti just vahemälu kasutuse puhul, mis on kohati 4 korda väiksem teistest simulaatori variantidest. Samuti on CARLA kõige stabiilsem, hoides stabiilset kaadrisagedust 60. Gazebo kannatab kaadrisageduse poolest kõige enam, kus kohati langeb kaadrisagedus kõigest paarile kaadrile sekundis.

Töö teise osana loodi konverter, mis suudab ühe simulaatori projekti muundada teise simulaatori projektiks. Kuna simulaatori projektil eksisteerib väga palju elemente, mida reaalselt on võimalik muundada, siis on võimalik kindlasti projekti tulevikus edasi arendada.

Kasutatud kirjandus

[1] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[2] P. G. C. Namal Senarathne, W. S. Wijesoma, B. Kalyan, M. D. P. Moratuwage, N. M. Patrikalakis, and F. S. Hover. Marinesim: Robot simulation for marine environments. In *OCEANS'10 IEEE SYDNEY*, pages 1–5, 5 2010. doi: 10.1109/OCEANSSYD.2010.5603839.

[3] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub W. Pachocki, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL `http://arxiv.org/abs/1808.00177`.

[4] Gym openai - robotics. `https://gym.openai.com/envs/#robotics`. Accessed: 2019-03-19.

[5] Ricardo Tellez. Why robotics companies must use simulators. `http://www.theconstructsim.com/why-robotics-companies-must-use-simulators/`, 2015.

[6] W. Qian, Z. Xia, J. Xiong, Y. Gan, Y. Guo, S. Weng, H. Deng, Y. Hu, and J. Zhang. Manipulation task simulation using ros and gazebo. In *2014 IEEE International Conference on Robotics and Biomimetics (ROBIO 2014)*, pages 2594–2598, 12 2014. doi: 10.1109/ROBIO.2014.7090732.

[7] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.

[8] Lgsvl simulator. `https://www.lgsvlsimulator.com/about/`, . Accessed: 2019-03-19.

[9] Wikipedia - loose coupling. `https://en.wikipedia.org/wiki/Loose_coupling`. Accessed: 2019-03-19.

[10] Inc Open Source Robotics Foundation. Why ros 2.0? https://design.ros2.org/articles/why_ros2.html, 2015.

[11] Inc Open Source Robotics Foundation. Changes between ros 1 and ros 2. https://design.ros2.org/articles/changes.html, 2015.

[12] Esteve Fernandez, Tully Foote, Dirk Thomas, and William Woodall. Why you want to use ros 2. https://www.osrfoundation.org/wordpress2/wp-content/uploads/2015/04/ROSCON-2014-Why-you-want-to-use-ROS-2.pdf, 2014.

[13] Inc Open Source Robotics Foundation. Stories driving the development. https://design.ros2.org/articles/stories.html, 2016.

[14] Michael Reckhaus, Nico Hochgeschwender, Jan Paulus, Azamat Shakhimardanov, and Gerhard Kraetzschmar. An overview about simulation and emulation in robotics. 03 2019.

[15] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, 9 2004. doi: 10.1109/IROS.2004.1389727.

[16] Gazebo - robot simulation made easy. http://gazebosim.org/, . Accessed: 2019-03-19.

[17] Ahmed Hussein, Fernando García, and Cristina Olaverri-Monreal. Ros and unity based framework for intelligent vehicles control and simulation. *2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, pages 1–6, 2018.

[18] W. A. Mattingly, D. Chang, R. Paris, N. Smith, J. Blevins, and M. Ouyang. Robot design using unity for computer games and robotic simulations. In *2012 17th International Conference on Computer Games (CGAMES)*, pages 56–59, 7 2012. doi: 10.1109/CGames.2012.6314552.

[19] Binh Son Le, Vy-Long Dang, and Trong-Tu Bui. Swarm robotics simulation using unity. 11 2014.

[20] W. Meng, Y. Hu, J. Lin, F. Lin, and R. Teo. Ros+unity: An efficient high-fidelity 3d multi-uav navigation and control simulator in gps-denied environments. In *IECON 2015 - 41st Annual Conference of the IEEE Industrial Electronics Society*, pages 002562–002567, 11 2015. doi: 10.1109/IECON.2015.7392488.

[21] Lenka Pitonakova, Manuel Giuliani, Anthony Pipe, and Alan Winfield. Feature and performance comparison of the v-rep, gazebo and argos robot simulators. pages 357–368, 7 2018. doi: 10.1007/978-3-319-96728-8_30.

[22] Lucas Souza-Carmo Nogueira. Comparative analysis between gazebo and v-rep robotic simulators. 2014.

[23] Aaron Staranowicz and Gian Luca Mariottini. A survey and comparison of commercial and open-source robotic simulator software. In *PETRA*, 2011.

[24] Ori Ganoni and Ramakrishnan Mukundan. A framework for visually realistic multi-robot simulation in natural environment. *CoRR*, abs/1708.01938, 2017.

[25] Pablo Martinez-Gonzalez, Sergiu Oprea, Alberto Garcia-Garcia, Alvaro Jover-Alvarez, Sergio Orts, and José García Rodríguez. Unrealrox: An extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation. *CoRR*, abs/1810.06936, 2018.

[26] Ori Ganoni, Ramakrishnan Mukundan, and Richard Green. A generalized simulation framework for tethered remotely operated vehicles in realistic underwater environments. *Drones*, 3:1, 12 2018. doi: 10.3390/drones3010001.

[27] Chris Doig. Comparing enterprise software is like apples and oranges. `https://www.cio.com/article/2917796/why-comparing-enterprise-software-products-with-each-other-doesnt-identify-best-fit-software.html`. Accessed: 2019-03-19.

[28] TEC Team. 7 important things you need to know about software comparison reports. `https://www3.technologyevaluation.com/research/article/7-important-things-you-need-to-know-about-software-comparison-reports.html`, 6 2017.

[29] Gazebo tutorials - installation. `http://gazebosim.org/tutorials?cat=install`, . Accessed: 2019-03-19.

[30] Carla simulator - getting started with carla. `https://carla.readthedocs.io/en/latest/getting_started/`, . Accessed: 2019-03-19.

[31] Lgsvl simulator - getting started. `https://www.lgsvlsimulator.com/docs/getting-started/`, . Accessed: 2019-03-19.

[32] Gazebo tutorials - from source (ubuntu and mac). `http://gazebosim.org/tutorials?tut=install_from_source&cat=install`, . Accessed: 2019-03-19.

[33] Carla simulator - how to build carla on linux. https://carla.readthedocs.io/en/latest/how_to_build_on_linux/, . Accessed: 2019-03-19.

[34] Carla simulator - how to build carla on windows. https://carla.readthedocs.io/en/latest/how_to_build_on_windows/, . Accessed: 2019-03-19.

[35] Github - carla simulator - building carla on mac osx. https://github.com/carla-simulator/carla/issues/150, . Accessed: 2019-03-19.

[36] Lgsvl simulator - instructions to build standalone executable. https://www.lgsvlsimulator.com/docs/build-instructions/, . Accessed: 2019-03-19.

[37] Unreal engine end user license agreement. https://www.unrealengine.com/en-US/eula, . Accessed: 2019-03-19.

[38] Lgsvl simulator - license. https://github.com/lgsvl/simulator/blob/master/LICENSE, . Accessed: 2019-03-19.

[39] Adrian Boeing and Thomas Bräunl. Evaluation of real-time physics simulation systems. pages 281–288, 01 2007. doi: 10.1145/1321261.1321312.

[40] Unreal engine - unreal python api introduction. https://api.unrealengine.com/INT/PythonAPI/introduction.html, . Accessed: 2019-03-19.

[41] Gazebo apis. http://gazebosim.org/api.html, . Accessed: 2019-03-19.

[42] Gazebo blog. http://gazebosim.org/blog.html, . Accessed: 2019-03-19.

[43] Carla documentation. https://carla.readthedocs.io/en/latest/, . Accessed: 2019-03-19.

[44] Lgsvl simulator - documentation. https://www.lgsvlsimulator.com/docs/, . Accessed: 2019-03-19.

[45] Gazebo tutorials. http://gazebosim.org/tutorials, . Accessed: 2019-03-19.

[46] Unreal engine 4 documentation. https://docs.unrealengine.com/en-us, . Accessed: 2019-03-19.

[47] Unreal engine online learning. https://academy.unrealengine.com/, . Accessed: 2019-03-19.

[48] Unity documentation. https://docs.unity3d.com/Manual/index.html, . Accessed: 2019-03-19.

[49] Gazebo answers. http://answers.gazebosim.org/questions/, . Accessed: 2019-03-19.

[50] Gazebo - open source robotics simulator. https://bitbucket.org/osrf/gazebo, . Accessed: 2019-03-19.

[51] Carla simulator. https://github.com/carla-simulator/carla, . Accessed: 2019-03-19.

[52] Ue4 answerhub. https://answers.unrealengine.com/index.html, . Accessed: 2019-03-19.

[53] Lgsvl simulator: An autonomous vehicle simulator. https://github.com/lgsvl/simulator, . Accessed: 2019-03-19.

[54] Unity answers. https://answers.unity.com/index.html, . Accessed: 2019-03-19.

[55] Getting started with carla. https://carla.readthedocs.io/en/latest/getting_started/, . Accessed: 2019-03-19.

[56] Unreal engine - linux quick start. https://docs.unrealengine.com/en-us/Platforms/Linux/BeginnerLinuxDeveloper/SettingUpAnUnrealWorkflow, . Accessed: 2019-03-19.

[57] Lgsvl simulator - python api. https://www.lgsvlsimulator.com/docs/python-api/, . Accessed: 2019-03-19.

[58] Gazebo - ros 2 integration overview. http://gazebosim.org/tutorials?tut=ros2_overview, . Accessed: 2019-03-19.

[59] Gazebo - installing gazebo_ros_pkgs (ros 2). http://gazebosim.org/tutorials?tut=ros2_installing&cat=connect_ros, . Accessed: 2019-03-19.

[60] Bridge communication between ros 1 and ros 2. https://github.com/ros2/ros1_bridge. Accessed: 2019-03-19.

[61] Which combination of ros/gazebo versions to use. http://gazebosim.org/tutorials/?tut=ros_wrapper_versions, . Accessed: 2019-03-19.

[62] Which combination of ros/gazebo versions to use. https://bitbucket.org/osrf/gazebo/issues/2388/gazebo-8-heightmap-objects-clipping, . Accessed: 2019-03-19.

[63] Gazebo - heightmap visual is offset from collision object. https://bitbucket.org/osrf/gazebo/issues/245/heightmap-visual-is-offset-from-collision, . Accessed: 2019-03-19.

[64] Gazebo tutorials - category: Build a world. http://gazebosim.org/tutorials?cat=build_world, . Accessed: 2019-03-19.

[65] Carla simulator - how to make a new map with roadrunner. https://carla.readthedocs.io/en/latest/how_to_make_a_new_map/, . Accessed: 2019-03-19.

[66] Unreal engine - landscape outdoor terrain. https://docs.unrealengine.com/en-us/Engine/Landscape, . Accessed: 2019-03-19.

[67] Carla simulator - howto create a dlc for carla. https://github.com/carla-simulator/carla/issues/460, . Accessed: 2019-03-19.

[68] Lgsvl simulator - map annotation. https://www.lgsvlsimulator.com/docs/map-annotation/, . Accessed: 2019-03-19.

[69] Unity documentation - terrain engine. https://docs.unity3d.com/Manual/script-Terrain.html, . Accessed: 2019-03-19.

[70] Unity documentation - instructions to build standalone executable. https://www.lgsvlsimulator.com/docs/build-instructions/, . Accessed: 2019-03-19.

[71] Gazebo - make a mobile robot. http://gazebosim.org/tutorials?tut=build_robot, . Accessed: 2019-03-19.

[72] Carla simulator - how to add assets. https://carla.readthedocs.io/en/latest/how_to_add_assets/, . Accessed: 2019-03-19.

[73] Carla simulator - how to model vehicles. https://carla.readthedocs.io/en/latest/how_to_model_vehicles/, . Accessed: 2019-03-19.

[74] Unreal engine - vehicle user guide. https://docs.unrealengine.com/en-us/Engine/Physics/Vehicles/VehicleUserGuide, . Accessed: 2019-03-19.

[75] Lgsvl simulator - how to add a new ego vehicle. https://www.lgsvlsimulator.com/docs/add-new-ego-vehicle/, . Accessed: 2019-03-19.

[76] Unity documentation - wheel collider tutorial. https://docs.unity3d.com/Manual/WheelColliderTutorial.html, . Accessed: 2019-03-19.

[77] Sdformat. http://sdformat.org/spec. Accessed: 2019-03-19.

[78] Unreal engine api - uworld. http://api.unrealengine.com/INT/API/Runtime/Engine/Engine/UWorld/, . Accessed: 2019-03-19.

[79] Tangible software solutions - c++ to c converter. https://www.tangiblesoftwaresolutions.com/product_details/cplusplus_to_csharp_converter_details.html. Accessed: 2019-03-19.

Lisa 1 - Viide tehtud töö Giti salvele

Järgnevalt viidatud Giti salves leidub magistritöö käigus loodud töö (lisaks 2017. aasta detsembris autori poolt loodud Gazebo maailm). Peamiselt koosneb salv töö käigus loodud maailmadest erinevates simulaatorites ning konverteerija tarkvara skriptidest (lisaks testid).

https://gitlab.cs.ttu.ee/Rasmus.Iila/masters-thesis