TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Karle Nutonen 192258

Tööstusroboti simulatsioonikeskkonna väljaarendus, masinõppe testkeskkonna kasutamise otstarbeks

Magistritöö

Juhendaja: Vladimir Kuts
PhD

Tallinn 2021

Autorideklaratsioon

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

Autor: Karle Nutonen

10.05.2021

Annotatsioon

Magistriõppe lõputöö

Lõpetaja K. Nutonen

Juhendaja V. Kuts

Tallinna Tehnikaülikool 2021

Käesolevas magistriõppe lõputöös antakse ülevaade tööstusroboti simulatsioonikeskkonna arendusest masinõppe rakendamiseks. Töö teostamiseks käsitleti masinõppe kasutamise võimalusi kolmemõõtmelises keskkonnas objekti optimaalse trajektoori leidmiseks punktist A punkti B ja saadud tulemi liidestamist tööstusroboti digitaalse kaksikuga simulatsioonikeskkonnas, millele on rakendatud pöörd kinemaatika funktsionaalsus.

Töö tulemuseks on tööstusroboti liikumise simuleerimine kolme mõõtmelises virtuaalkeskkonnas roboti lõpp-punkti manipuleerimisega, kuhu on rakendatud masinõppe ja pöördkinemaatika koosluse võimekus. Kogu kontseptsioon võimaldab treenida tööstusroboteid virtuaalkeskkonnas tegema uusi toiminguid, nii et see ei sega reaalse roboti väärtusliku tööprotsessi aega. Samas on see võimalik hiljem aja efektiivselt üle kanda päris robotile.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 34 leheküljel, 6 peatükki, 23 joonist, 6 tabelit.

Abstract

Development of environment of an Industrail robot simulation for the use of test environment of a machine learning

Master thesis

Graduater: Karle Nutonen

Supervisor: Vladimir Kuts

Tallinn University of Technology 2021

This master's thesis provides an overview of the development of an industrial robot simulation environment for the implementation of machine learning. To perform the work, the possibilities of using machine learning in a three-dimensional environment to find the optimal trajectory of an object from point A to point B and interfacing the obtained result with an industrial robot digital twin simulation environment with inverse kinematics functionality were discussed.

The result of the work is the simulation of the movement of an industrial robot in a 3-dimensional virtual environment by manipulating the end effort of the robot, to which a machine-trained model and the capability of inverse kinematics combination have been applied. Throughout the concept, train industrial robots in a virtual environment to perform new operations so that it does not interfere with the real work process of a real robot. However, it is possible to efficiently transfer time later to a real robot.

The thesis is in Estonian and contains 34 pages of text, 6 chapters, 23 figures, 6 tables.

Lühendite ja mõistete sõnastik

AR	Augmented Reality, liitreaasus
asset	Unity skriptide, mudelite, stseenide jne kogumpakett
CCD	Cycling Coordinate Descent,
FABRIC	Forward And Backward Reaching Inverse Kinematics
IK	*Inverce kinematics*, pöörd kinemaatika
ML	Machine Learning, masinõpe
OLP	*Off-line programming*, tööstusroboti programmeerimine
RL	Reinforcement Learning, kinnitusega masinõpe
VR	Virtual Reality, virtuaalreaalsus

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

1.1 Taust

Elame ajastul, kus maailm on pidevas kiires muutuvuses, millega peab sammu pidama iga ettevõte, kes soovib olla konkurentsivõimeline. Viimaste aastakümnetega on tootmisettevõtted liikunud robotiseeritud tootmisprotsesside suunas, kuid viimase kümnendiga on see rohkesti hoogu kogunud. Selle ajendiks on suuresti 2014. aastal Saksamaa Liitvabariigi liidukantsleri Angela Merkeli poolt väljakäidud Tööstus 4.0 tööstusrevolutsiooni kontseptsioon, mille peamine eesmärk on tuua tootmine tagasi Euroopasse ja muuta see kuluefektiivsemaks läbi erinevate toomisprotsesside digitaliseerimise ja automatiseerimise [1].

Üheks digitaliseerimise ja automatiseerimise valdkonnaks on eelnevas lõigus välja toodud tootmiseprotsesside robotiseerimine. Robotite rakendamine võimaldab produkti tootmise kulumäära alandada ja samas on robotid rutiinsete tööde teostamisel paremad ning teevad vähem vigu. See annab ettevõttele parema konkurentsieelise ja suurendab tulumäära. Robotite kasutamine tööstuses tähendab spetsiifiliste teadmistega tööjõu olemasolu, kes neid töös hoiavad ja hooldavad. Kuid vastavate laialdaste robottehniliste süsteemide teadmistega inimest osakaal on ettevõttetes madal või puudub üldse.

1.2 Probleem

Suurimaks probleemiks on tootmisettevõtetes eelpool nimetatud peatükis vastava kvalifikatsiooniga inimeste vähesus. Samuti on probleemiks erinevate robotite tootjate kombineerimine ühes tootmisliinis, sest kõigil neil on üldjuhul erinev programmeerimiskeel, mis muudab nende seadistamise keeruliseks. See tähendab, et on vaja rohkem töölisi, kellel on vastava roboti tootja seadistamise ja kasutamise teadmised, sest maailmas on vähe neid kes suudaks töötada kahe või rohkema tootja robotitega. Samas on probleem ka tootmisliini ümber seadistamise ajaline kulu, mis toodab muudatuste tegemise hetkel ettevõttele kahju. Ajaline kulu võib küündida

tundidest nädalatesse, mis oleneb muudatuste mahust, sest iga teostatud seadistus tuleb eelnevalt läbi testida mitmeid kordi, et see vastaks tootmise eesmärkidele.

1.3 Eesmärk

Antud magistritöö eesmärgiks on arendada multifunktsionaalse kasutamiskogemusega tööstusroboti simulatsioonikeskkonna väljaarendus masinõppe testkeskkonna kasutamise otstarbeks. See annab võimaluse simuleerida ja treenida tootmisprotsesse virtuaalkeskkonnas ning mille tulemust on võimalik tulevikus füüsilisele robotile üle kanda. See peaks tagab omakorda ettevõtete jaoks tootmises suurema modulaarsuse, vähendab kulutusi liinide ümber seadistamisel nii ajalise teguri kui tööjõu kulude pealt. Samas võimaldab antud keskkonnas läbimängida erinevaid väljamõelduid tootmise stsenaariumeid nii et see ei mõjuta käimas olevat tootmist. Kui parim liini seadistamise lahendus on saavutatud, on seda võimalik rakendada olenevalt tootmisliini ja muudatuste suurusest mõne tunni või päeva jooksul. See tagab ettevõttele väiksema seadistamise ja aja kulu.

Eesmärgi saavutamiseks, tuleb lahendada alljärgnevaid ülesandeid:

1. Uurida ja analüüsida sarnase teemaga töid;
2. Uurida ja valida vajalikud metoodikad/tööriistad, et püstitatud eesmärk saaks saavutatud;
3. Anda ülevaade teostatud tööst;
4. Analüüsida saavutatud eesmärgi tulemust.

Teema on väga aktuaalne ka sellepärast, et see käsitleb valdkonda, millega tegelevad Ida-Virumaa ja teised Eesti tootmisettevõtted, kus on käsil tootmisliinide pidev robotiseerimine, kuid puuduvad küllaldased robottehnilised teadmised, et rakendada neid oma tootmises. Loomulikult aitaks kaasa arendatav keskkond disainida paljudel ettevõtetel oma tootmisliine kiiremini ja väiksemate kulutustega.

Antud lõputöö uuenduslikus seisneb simulatsioonikeskkonnas kasutatava masinõppe rakendamises, kus ei treenita robotit liikuma läbi kõigi selle liigendite kaudu vaid lõpp-punkti liigutamise läbi, kus liigendite pöördnurkadega seotud muudatuste tegemine jääb

pöördkinemaatika (*inverse kinematic* - IK*)* funktsionaalsuse tööks. Vastavalt autori uurimistele, pole sellist lähenemist robotite masinõppe rakendamisel varem kasutatud.

Maailmas on mitmeid robottehnilisi simulatsioonikeskkondi, kus vähestel on võimalik rakendada masinõppe võimekust, kuid paljud neist eeldavad kohest väljaminekut litsentsi soetamiseks ja on rakendatavad ainult ühel operatsioonisüsteemil. Need rakendused seavad kasutajatele koheselt piirangud, mis omakorda võib muuta ettevõtte konkurentsieelist turul.

1.4 Lõputöö struktuur

Käesolev magistritöö koosneb neljast peatükist. Esimeses peatükis uurib ja analüüsib autor sarnase teemaga teostatuid töid, tuues välja nende teostatud töö lühikese ülevaate. Teises peatükis käsitleb autor magistritööga seotuid metoodikaid ja tööriistu. Kolmandas annab autor ülevaate enda teostatud arendusest. Viimases ehk neljandas peatükis annab autor analüüsiva ülevaate teostatud töö tulemusest ja edasiarendustest.

1.5 Magistritöö tulemus

Antud lõputöö tulemusena sai loodud tööstusroboti simulatsioonikeskkonna prototüüp *Unity* mängumootoril, milles on võimalik rakendada masinõpet, et saadud mudeli abil võimaldada simuleerida tööstusrobotite tööd erinevate stsenaariumite jaoks.

2 Kirjanduse ülevaade

Tööstusrobotid on kahekümnenda sajandi ühed parimad ja edukamad leiutised, mis on muutunud tootva tööstuse ilmet väga palju. Roboteid kasutatakse rutiinsete tööde tegemisel, näiteks toodete komplekteerimisel, asjade tõstmisel ühest kohast teise, keevitamisel, värvimisel jne. Samuti teevad robotid neid toiminguid täites palju vähem vigu kui inimesed. Kuid nende kasutamine tööstuses pole nii lihtne kui võiks. Robotite seadistamiseks on vaja vastava teadmistega spetsialiste, aga vastavat tööjõudu on vähe. Samuti on uue tootmisprotsessi konfigureerimine töömahukas protsess, mis on ettevõtte jaoks nii rahaline kui ka ajaline kulu.[2]

Et parendada eelpool välja toodud probleemi on loodud palju erinevaid simulatsioonikeskkondi, mis baseeruvad roboti 3D digitaalse kaksiku mudelite kasutamisel virtuaalkeskkonnas, kus on võimalik simuleerida roboti tööd ning programmeerida neid täitma mingit ülesannet, mida nimetatakse inglise keeles *off-line programming (OLP)*, tagades sellega protsesside kiirema konfigureerimise ilma suuremahuliste teadmiste olemasoluta. Igal kommerts roboti tootjal on loodud selline simulatsioonikeskkond, näiteks KUKA Sim, Robotstudio, MotoSim jne.[3] Kuid vastavad näited simuleerivad ning võimaldavad programmeerida ainult vastava tootja robotite tööd (Tabel 1).

Tabel 1. Robotite tootjad, off-line programmeerimise tarkvara ja programmeerimiskeel

Roboti tootja	**Tarkvara**	**Programmeerimiskeel**
Comau	RoboSim Pro	PDL2
Fanuc	RoboGuide	TP/Karel
Yaskawa- Motoman	MotoSim	Inform
ABB	ABB RobotStudio	RAPID
Kawasaki	K-Roset	AS
Nachi Fujikoshi	FD On Desk	Slim

Denso	WinCaps III	PAC
Kuka	Kuka.Sim Pro	KRL
Mitsubishi	MELFA WORKS	MELFA BASIC

Leidub ka veel sellised keskkondi, mis võimaldavad kasutada simulatsioonikeskkonnas rohkem, kui ühe tööstusroboti tootja seadmeid, nendeks on Visual components, Emulated3D, VREP, ROS, Unity jne (Tabel 2).

Tabel 2. Kolmanda osapoolte robotite off-line programmeerimise tarkvarad ja programmeerimiskeel

Arendaja	Tarkvara	Programmeerimiskeel
Visual Components OY	Viusal Components	Python
Rockwell Automation	Emulate3D	C#, JScript, QuickLogic
Coppelia Robotics	V-Rep	LUA
Open Source Robotics Foundation	Gazebo	C++
RobotDK	RobotDK	Python
OpenRAVE Community	OpenRAVE	C++, Python
MathWorks/ Peter Corke	Robotic System Toolbox/ Robotics Toolbox	MATLAB skriptimise keel/ Octave GNU
Unity Technologies	Unity	C#, C++

Paljud uurimistööd sisaldavad uuringuid vastavate keskkondade kasulikkusest tootmisele ja seda, kuidas nende funktsionaalsust veel paremini rakendada, et tulemus annaks tööstusettevõtetele suuremat lisandväärtus. Samuti on teadlased pakkunud välja ideid, ettepanekuid ja arendusi, kuidas antud keskkondi paremaks muuta ja uurida nendes keerulisi tootmisprotsesse. Allpool on esitatud mõningad näited:

1. A. K Bedaka jt vaatlesid 3 mõõtmelise nägemise rakendamist OPL funktsionaalsusega, et luua roboti liikumise trajektoor [3].

2. B. Jakubiec annab laialdase ülevaate erinevate roboti simulatsiooni tarkvarade olemusest ja nende kasutamise valdkondadest [4].

3. Prasanth T analüüsis oma töös ROS-i ehk Robot Operatsioonisüsteemi implementeerist roboti simulatsioonikeskkonnana, kus ta vaatles OPL rakendamist ning loodud programmikoodi edastamist reaalsesse robotisse [5].

4. L. Hong ja teised uurisid OPL võimekust avatud lähtekoodil VTK(visualiseerimise tööriistakomplekt) baasil, mis toetab paljusid 3D mudelite formaate. See uuring näitab väga efektiivselt, kuidas rakendusi on võimalik kasutada robotite simulatsioonikeskkonnana, mis pole algselt mõeldud töötamiseks virtuaalsete robotitega [6].

5. L. Shan jt uurisid oma töös Unity mängumootoris masinõppe *ML Agent* rakendamist tööstusliku roboti liikumissuuna leidmisel objektini [7].

6. M. Lin, L. San, Y. Ding toovad oma uurimustöös välja roboti programmerimise võimaluse virtuaalreaalsuses Unity keskkonnas, mida tänu 3D modelleerimise ja virtuaalreaalsuse lihtsamale rakendamisele on võimalik lihtsalt luua [2].

7. S. Starke ja teised lõid oma uurimise käigus universaalse robotite täis tagasi kinemaatika(*full inverse kinematic)* funktsionaalsuse virtuaalsete simulatsioonikeskkondade tarvis, kasutades ära olemasolevat tagasi kinemaatika võimkust, kuid lisades nende juurde suurel hulgal lisandväärtusi. Antud tagasi kinemaatika sobib nii humanoid robotite kui ka tööstusrobotite tagasi kinemaatika implementeerimiseks. See kannab nime BioIK [8].

8. L. Pitonakova jt. annavad oma uurimistöös ülevaate kolme üldlevinud avatud lähtekoodiga roboti simulatsioonikeskkondade tunnustest ja jõudlusest, andes teistele uurijatele nende kasutamise kohta teadmisi [9].

9. Ciao Souza ja Luiz Velho vaatlevad oma töös masinõppega teostatud animeerimist, mis on roboti simulatsioonikeskkonnas väga vajalik aspekt, püüdes lahendada seda võimalikult lihtsalt [10].

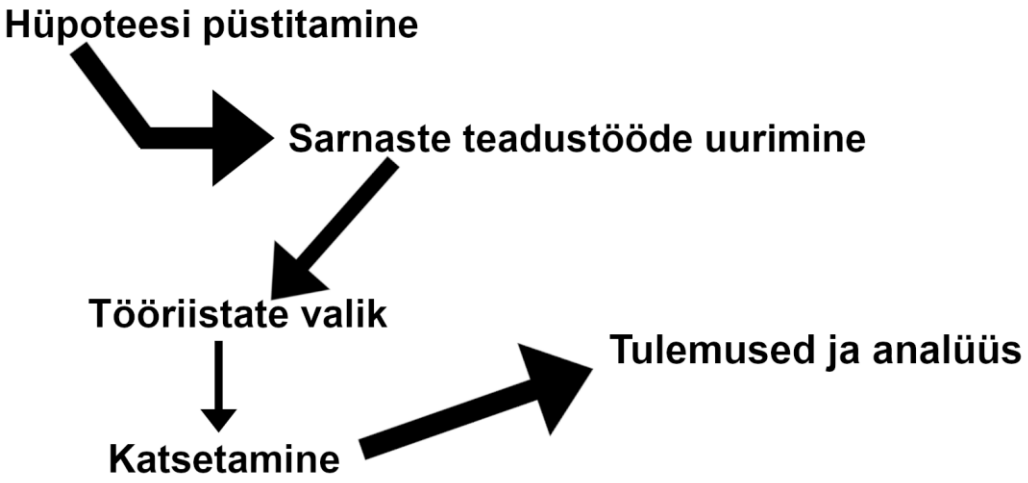
10. W. Zhang ja teised implementeerisid oma uurimuses FABRIK pöörd kinemaatika funktsionaalsust pideval robotkäel (*continuum robot arm*), mis

võimaldab simuleerida väga erinevate robotite liikumist simulatsioonikeskkonnas [11].

Eelpool välja toodud uurimistöödest paistab silma simulatsioonikeskkonna vajalikkus maailmas, sest see annab väga hea ülevaate ja aitab luua täpset ja töökindlat tootmisprotsessi. Nendes näidetes on rakendatud kinnitusega masinõpet, lähtudes tööstusroboti õige asendi leidmist roboti lülide pöördnurkade kaudu. Selline lähenemine on andud väga häid tulemusi, kuid saadud masinõppe mudeleid on võimalik kasutada ainult vastava tööstusrobotiga, millele on mudel loodud. See kõik võimaldab vähendada aega tootmisprotsessi liini disainimise ja seadistamise käigus. Kuid eelpool uuritud töödes pole kokku liidetud masinõppe ja pöördkinemaatika võimekust, kus masinõpet rakendatakse ainult roboti lõpp-punktile ja selle liikumise tulemusel rakendatakse robotile pöörd kinemaatika funktsionaalsus, mis võimaldaks masinõppes saadud mudelit kasutada erinevat tüüpi robotitega.

3 Metoodika

Uurimiseesmärgi täitmiseks töötas lõputöö autor läbi kirjanduse ülevaate, mis on esitatud eelmises punktis (2 – kirjanduse ülevaade). Joonis 1 illustreerib üldist empiirilise uurimise metoodikat



Joonis 1. Empiirilise uurimuse metoodika joonisena

3.1 Töö kirjeldus

Tulenevalt Tööstus 4.0 kontseptsioonist, mille üheks essmärgiks on tööstuse tootmise automatiseeritus ja digitaliseeritus, on paljud teadlased tegelenud selle kontseptsiooni karakteristikute välja töötamisega ja defineerimisega [12].

Bauer (2014) on defineerinud Tööstus 4.0 kui intelligentset reaalajas inimeste ja masinate horisontaalset ja vertikaalset integreerimist koos objektide, informatsiooni

ning kommunikatsiooni süsteemidega, et luua keerukate süsteemide paindlik ja dünaamiline süsteem [1].

Simulatsioonikeskkonna idee on vähendada aega ja optimeerida tööstuse erinevate protsesside disainimist ja väljaarendamist ning simuleerida antud protsesse ja reaalset keskkonda nii täpselt kui võimalik. Simulatsioonikeskkonnad on tänapäeval väga võimekad simuleerima reaalelu füüsikalisi omadusi, näiteks gravitatsioon, kinemaatika, raskuste erinevust jms. Antud keskkondi kasutatakse isesõitvate autode, robotite masinõppe jmt taolise arendamiseks/simuleerimiseks, sest see võimaldab luua stsenaariumeid, mida reaalses keskkonnas pole võimalik või otstarbekas hetke olukorras teostada. Samuti annab see võimaluse rakendada disainimse hetkel veel mitte olemas olevat tehnikat ning teostada selle ostmise/vajalikkuse analüüsi.

Töö autor valis simulatsioonikeskkonna loomiseks *Unity* mängumootori teiste olemasolevate keskkondade seast, mis on välja toodud peatükis 2 tabelites 1 ja 2, sest see võimaldab luua suhteliselt lihtsalt väga erineva vajadustega tootmise simuleerivaid stsenaariumeid. Samuti saab sinna importida kiirelt erinevates formaatides 3D mudeleid ning programm on kasutatav kolmel erineval operatsioonisüsteemil (Windos, Linux ja MacOS), mis annab loodavale keskkonnale laialdasema kasutamise võimekuse.

Algselt on *Unity* arendatud videomängude loomiseks, kuid viimaste aastate jooksul on hakatud antud platvormi kasutama tööstusrobotite, isejuhtivate, tööstusprotsesside jms simuleerimise rakendamiseks/arendamiseks. Sest *Unity*-s on sisseehitatud väga hea füüsika mootor, mis suudab väga hästi simuleerida reaalseelu füüsikalisi omadusi. See annab võimaluse keskkonda kasutada erinevate seadmete/protsesside arendamiseks ja võimaldab aega kokku hoida ning laseb vigu likvideerida enne füüsilise objekti/protsesside loomist.

Simulatsioonikeskkonna loomiseks on vaja luua ja optimeerida kasutatava objekti 3D-mudel ehk digitaalne teisik (joonis 2), mis suudab implementeerida virtuaalses keskkonnas nii täpselt kui võimalik reaalse elu liikumist, füüsikalisi omadusi ning visuaali.

Joonis 2. Tööstusroboti Fanuc LR Mate 200Id/4S päris ja digitaalse teisiku visuaal

3.2 Tööstusroboti kinemaatika

Antud osas käsiteleb autor tööstusrobotite kinemaatika olemust ja vajalikkust virtuaalkeskkonnas ning teeb ülevaate erinevatest hetkel levinumatest algoritmidest/meetoditest.

Roboti kinemaatika on matemaatikaliste arvutustega teostatud geomeetriline väljendus, kus vaadeldakse roboti lülide paigutust/muutust globaalse taustsüsteemi G suhtes ja millele igale lülile määratakse koordinaatistike raamistikud, sest iga lüli teljed võivad olla pööratud üksteise suhtes [13].

Kinemaatika arvutamise/kirjeldamise võimalusi on mitmeid pöörd-, orientatsioon, liikumise, edasine, nurk ja kiirus kinemaatika. Kuid lähtuvalt lõputöö eesmärgist vaatame põhjalikumalt pöördkinemaatika lähenemist.

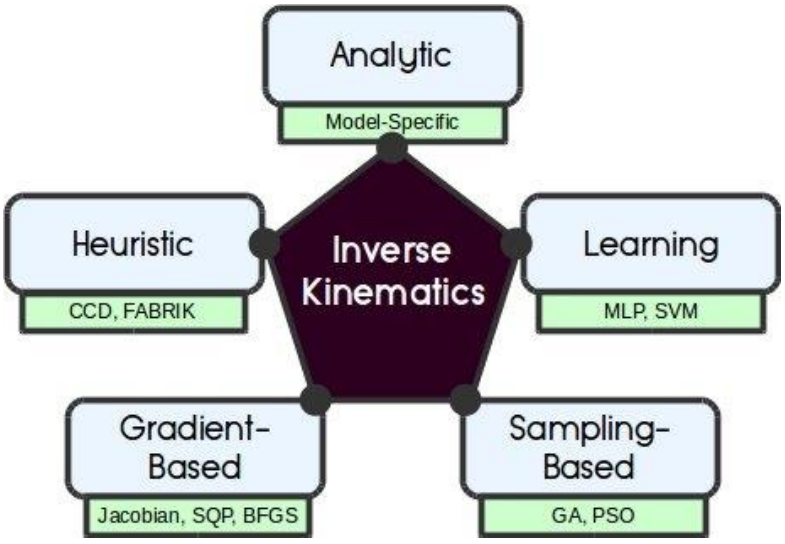
3.2.1 Pöörd kinemaatika

Pöördkinemaatika kirjeldab n-vabadusastmega tööstusroboti lülide asukohta globaalses koordinaatsüsteemis lõpp punkti/asukoha järgi. Selleks leitakse nurk väärtus θ igale lülile segmentide teisenduste $X_{1,\dots,k}$ abil järgneva üldistava valemi (valem 1) alusel [8].

$$\theta = f^{-1}(\chi_{1,\dots,k})$$

Võrrand 1. N-dimensiooni roboti lülide nurga leidmine pöördkinemaatikaga

Pöördkinemaatikat järjestikustes kinemaatilistes ahelate osades on väga hästi uuritud. Siiski on paljudel meetoditel keeruline lahendada kogu keha asendite leidmine keeruliste geomeetrial või mitme piirangu lisamisel. Erinevad pöördkinemaatika lähenemise alogritmid pärinevad paljudest uurimusvaldkondadest ning neid jagatakse järgnevalt nagu näidatud joonisel 3 [14].



Joonis 3. Pöörd kinemaatika algoritmilised metoodikad [14]

Kõigil kinemaatika algoritmidel on omad plussid ja miinused. Järgnevalt on toodud välja üldlevinumate pöördkinemaatika algoritmide olemused.

A) Analüütiline

Analüütiline pöördkinemaatikaga on võimalik luua roboti lülide asetus väga täpselt ja kiiresti, kasutades algebralisi võrrandeid. Kuid antud algoritm toimib hästi väikeste vabadusastmete robotite/mehhanismide juures, sest iga vabadusastme lisamisega suureneb arvutuse keerukus [14].

B) Heuristiline

Heuristilise pöördkinemaatika eesmärgiks on hinnata interaktiivseid värskendusi geomeetriliste arvutuste kaudu. Selleks kasutatakse kahte erinevat lähenemist, milleks on CCD (*Cyclic Coordinate Descent*) ja FABRIK (*Forward and Backward Reaching Inverse Kinematics).*

CCD on üks populaarseim pöördkinemaatika interaktiivne algoritm. Seda on implementeeritud paljudesse graafika ja roboti tarkvaradesse. Antud algoritm on väga lihtne ja kiire [15].

FABRIK on uuem algoritm, kui CCD, ning on kiiresti muutunud oluliseks pöördkinemaatika arvutamiseks animatsioonide puhul. Samuti suudab see algoritm tegeleda rohkemate kui ühe lõpp-punkti lahendamisega [14].

C) Gradiendipõhine

Gradiendipõhine pöördkinemaatika alogitmiga arvutatakse esimese või teise järgu tuletisi, mis muudab arvutamise keerukamaks ja ressurssi rohkemaks, kui eelpool nimetatud heuristilised meetodid. Kuid lisapiirangute lisamisele on see paindlikum. Populaarseim lähenemine on Jacobian maatriksi, mis sisaldab vektorfunktsiooni esimese järgu osalist tuletist, et leida lõpp-punkti positsioon karteesia ruumis [14].

Alljärgnevas tabelis võrreldakse erinevaid pöördkinemaatika lähenemise implementatsioone, mida on võimalik kasutada *Unity* mängumootoril

Tabel 3. Unity mängumootoril implementeeritud pöörd kinemaatika võrdlus

Tüüp	BioIK	Hybride IK	Fast IK	Final IK
IK algoritm	Gradiendipõhine	Heuristiline	Heuristiline	Heuristiline
***Unity* versiooni sobivus**	2017.1.0 või uuem	2018.4.24 või uuem	2019.1.5 või uuem	2017.4.28 või uuem
Viimane uuendus	6. september 2018	5. oktoober 2020	12. august 2019	15. oktoober 2020
Maksumus eurodes(€)	17.87	17.86	0	80.40

Autor valis oma lõputöö jaoks välja Bio IK pöörd kinemaatika implementatsiooni, sest see põhineb gradiendipõhisel algoritmil, mida on võimalik rakendada suurtel vabadusastmetega robotitel/seadmetel. See muudab antud lõputöö tulemust kasutamise laiapõhjalisemaks.

3.3 Masinõpe

Masinõpe võimaldab arvutitel õppida protsesse ilma selgesõnalist programmikoodi kirjutamata. Kuigi see määratlus on üsna ebamäärane, sest see ei järgi eelnevalt programmeeritavaid „reegleid". See funktsionaalsus loodi 1959. aastal Arthur Samueli poolt, kes määratles selle õppevaldkonnana. Üldiselt on masinõpe automatiseeritud protsess, mis võimaldab masinatel analüüsida tohutult andmekogumeid, tuvastada mustreid ja õppida andmetest, et pakkuda tuge ennustuste ja otsustuste langetamiseks. Masinõpe võib tugevdada ka inimeste eelarvamusi, mis võivad andmetes sisalduda, kui kasutada suurandmeid koolitusteks. Masinõpet võib pidada kognitiivsete funktsioonide või teadustööde automatiseerimiseks. Isejuhtivad autod on masinõppe parimad näited. Samas masinõppe puuduseks on õppe keerulised konstruktsioonid. See võib raskendada nende treenitud mudelite usaldamist, kus ühe andmerea üks side kahe neuroni vahel ei anna vaatlejale mingit seletust. Inimesed usaldavad süsteemi siis kui on võimalik arvutussprotsessi tõlgendada. Samas kasutatakse seda tehnoloogiat paljudes eluvaldkondades, näiteks meditsiinis, isesõitvad autod, tootvas tööstuses jne [16].

Masinõppe võimekust on uuritud palju, et automatiseerida tööstusrobotite *off-line* programmeerimist simulatsioonikeskkonnas. Arvestades Tööstus 4.0 kontseptsiooni, kus toote tootmine on pidevas arenemises, peab tootmis protsessi pidevalt täiendama/muutma, mille käsitsi konfigureerimine oleks ajakulukas ning eeldaks pidavalt tootmisliini seiskamist. Masinõppe tehnoloogia rakendamine võimaldab antud situatsiooni parendada ja lühikese ajaga kanda saadud konfiguratsioon üle tootmisprotsessi. Samuti võimaldab masinõpe uurida uute liinide tootmisprotsesside disaine simulatsioonikeskkonna.

Masinõppe teostamiseks on mitmeid võimalusi [17]:

- Juhitud õpe (*supervised learning*): sellel õppeviisil kasutatakse sisendiks treening ja test andmestikke, mille tulemusel luuakse mudel, mida on õpetatud tulemi põhjal õigeid otsuseid tegema;

- Juhtimata õpe (*unsupervised learning*): õppeviiskiks on andmetest sarnaste struktuuride leidmine või looma üldistavaid reegleid;

- Pooleldi juhitud õpe (*semi-supervised learning*): sellel õppeliigil kasutatakse nii märgistatud kui ka märgistamata andmeid, millest mudel peab leidma sarnaseid struktuure ja tegema ennustusi;

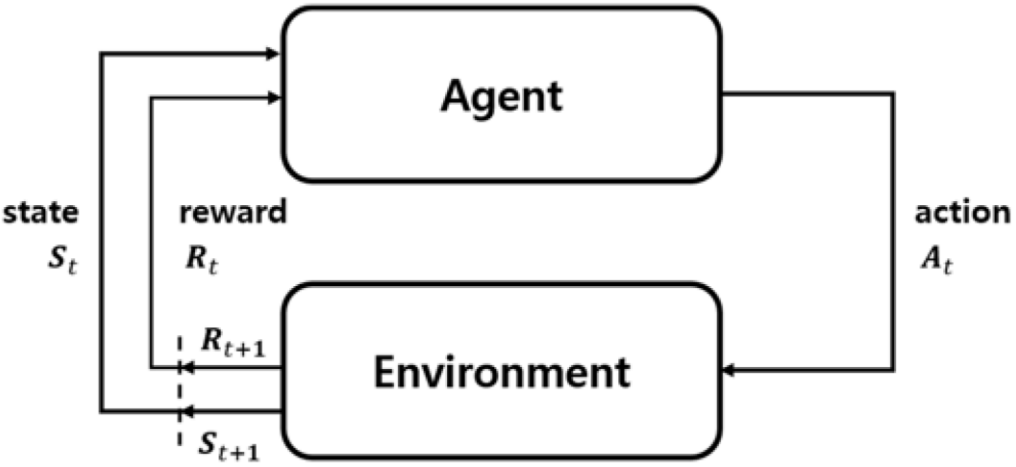
- Kinnitusega õpe (*reinforcement learning*): mudel õpib katse-eksituse meetodil, millised tegevused on parima kaalukusega, kus komponentideks on õppija, keskkond ja tegevused.

Autori pool valitud simulatsioonikeskkonnaga sobib kinnitusega masinõppe võimalus. Sest töö autor soovib, et masinõppe toestamine toimuks iseõppimise teel, kus ainsaks sisendiks oleks keskkonnas olevad objektid ja jõudmine ühest punktist teise võimalikult kiirelt. Selleks on *Unity*-s võimalik kasutada *Unity* arendajate endi poolt arendatav ML (*Machine Learning*) agent, mida on võimalik kasutada nii mängu kui ka simulatsioonikeskkonnas [18]. ML agenti pistikprogrammi on implementeeritud *PyTorch*-il põhinevad tipptehnoloogiliste algoritmide juurutamisevõimalusi, et mängude arendajad ja harrastajad saaksid 2D-, 3D ja VR/AR- mängude jaoks tehisintellekte luua. Samuti on see mõeldud teadlastele, kellel on võimalik kasutada *Python*-i API-sid kinnitatud, imiteerimise, närvivõrke või muid masinõppe meetodeid [19].

3.3.1 Masinõppe kasutamine simulatsioonikeskkonnas

Masinõppe kasutamisega simulatsioonikeskkonnas luuakse täisautomaatne õppija, mis suudab interakteeruda virtuaalkeskkonnaga õppimaks optimaalset käitumist läbi õnnestumiste ja eksimuste. Kuid see on pikaajaline protseduur. Matemaatika printsiipide raamistikul põhinevat kogemuste põhist autonoomset õppimist nimetatakse kinnitusega masinõppeks (RL). RL kasutamisel oli varasemalt probleeme, nagu kõigil teistel masinõppe algoritmidel, kuid sügav närvivõrkude pidevale arendamisele on muutnud RL võimekust paremaks [20].

Kinnitusega masinõppe kasutamiseks simulatsioonikeskkonnas tuleb seadistada autonoomne agent(õppija), mida kontrollitakse masinõppe(ML) algoritmiga, olekute vaatlemist S_t keskkonnas ajahetkel t. Kui agent teostab uue tegevuse, liigub keskkond ja agent uude olekusse S_{t+1}. Nendest olekute kogumist luuakse olekute jada, millest parimale olekule kinnistatakse auhind r_{t+1}, mis on tagasisideks agendile, kas ta käitus õigesti või mitte. Vastavat korduvat tsüklit illustreerib joonis 4.



Joonis 4. Kinnitusega masinõppe tsükkel [21]

4 Tööstusroboti masinõppega programmeerimise simulatsioonikeskkonna arendusprototüübi loomine

Selles peatükis annab töö autor ülevaate tööstusroboti masinõppe rakendamise simulatsioonikeskkonna arendusprototüübi olemusest ja realiseerimisest.

4.1 Masinõppega programmeerimise simulatsioonikeskkonna olemus

Simulatsioonikeskkond arendusprototüüp koosneb virtuaalsest keskkonnast, kus on võimalik kasutada erinevaid seadmete/toodete 3D-mudeli formaate ja testida ML mudeleid, ning eraldiseisvast masinõppe teostamise keskkonnast. See muudab arendatava keskkonna mitmekülgsemaks, milles on võimalik luua nii lihtsaid, ühest/kahest, kui ka keerulisi, paljudest seadmetest, tootmisliinide digitaalseid simuleerivaid stsenaariumeid.

Töö eesmärgist on arendada tööstusrobotite ja -masinate masinõppega simulatsioonikeskkond, mis muudaks robotite liikumise teekonna leidmise kiiremaks, lihtsamaks, arvestades selle juures tootmisliini eripärasid ning millest saaks tulevikus koostada roboti programmikoodi.

Simulatsioonikeskkonna arendamiseks pidi autor:

- modelleerima tööstusroboti Fanuc LR Mate 200ID/4S digitaalse kaksiku 3D-mudeli;
- uurima erinevaid sobivaid simulatsioonikeskkondi;
- uurima tööstusroboti pöördkinemaatikat implementatsioone, mida on võimalik rakendada valitus simulatsioonikeskkonnas;
- uurima erinevaid masinõppe kasutamise võimalusi valitud simulatsioonikeskkonnas.

4.2 Üldvaade

4.2.1 Taust

Tööstusroboti masinõppega simulatsioonikeskkonna arendamine on teostatud *Unity* mängumootoril, mis võimaldab luua nii lihtsa kui ka keerulise olemusega robotite töö- või tootmisprotsesside stsenaariumeid.

4.2.2 Simulatsioonikeskkonna eesmärk

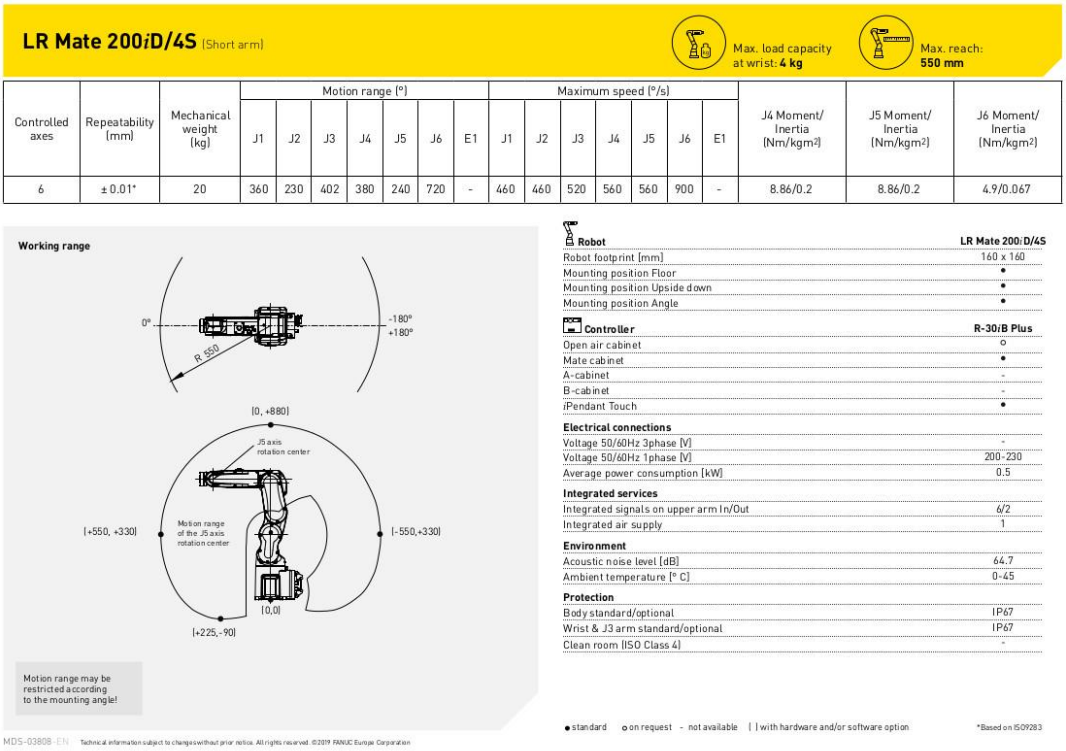
Eesmärgiks on luua digitaalsel kasksikul põhinev virtuaalne simulatsioonikeskkond rakendamaks masinõpet tööstusrobotite liikumise trajektoori kiiremaks leidmiseks.

4.3 Tööstusroboti valik ja modelleerimine

Lõputöös kasutas autor tööstusrobotit Fanuc LR Mate 200Id/4S (joonis 5). Magistritöö teostamiseks roboti valimisel lähtuti Taltechi Virumaa kolledžis olevatest tööstusrobotitest, Fanuc LR Mate 200Id/4S ja Kuka KR3 Agilus. Roboti valimise juures lähtuti veel üle kohtvõrgu juhtimise võimalusest, mille võimekust soovib töö autor edasises arenduses simulatsioonikeskkonnaga liidestada.

Joonis 5. Fanuc LR Mate 200iD/4S

Roboti digitaalse teisiku loomise juures pidi töö autor arvestada tööstusroboti tootespetsifikatsiooniga, et 3D-mudel oleks võimalikult täpne reaalsele objektile nagu on näidatud joonisel 6.



LR Mate 200*i*D/4S (Short arm)

Max. load capacity at wrist: **4 kg**

Max. reach: **550 mm**

Controlled axes	Repeatability (mm)	Mechanical weight (kg)	Motion range (°)							Maximum speed (°/s)							J4 Moment/ Inertia (Nm/kgm²)	J5 Moment/ Inertia (Nm/kgm²)	J6 Moment/ Inertia (Nm/kgm²)
			J1	J2	J3	J4	J5	J6	E1	J1	J2	J3	J4	J5	J6	E1			
6	± 0.01*	20	360	230	402	380	240	720	-	460	460	520	560	560	900	-	8.86/0.2	8.86/0.2	4.9/0.067

Working range

0° -180° +180° R-550

(0, +880)

J5 axis rotation center

(+550, +330) (-550,+330)

Motion range of the J5 axis rotation center

(0,0)

(+225,-90)

Motion range may be restricted according to the mounting angle!

Robot	LR Mate 200*i*D/4S
Robot footprint (mm)	160 x 160
Mounting position Floor	•
Mounting position Upside down	•
Mounting position Angle	•
Controller	**R-30*i*B Plus**
Open air cabinet	o
Mate cabinet	•
A-cabinet	-
B-cabinet	-
iPendant Touch	•
Electrical connections	
Voltage 50/60Hz 3phase [V]	-
Voltage 50/60Hz 1phase [V]	200-230
Average power consumption [kW]	0.5
Integrated services	
Integrated signals on upper arm In/Out	6/2
Integrated air supply	1
Environment	
Acoustic noise level [dB]	64.7
Ambient temperature [° C]	0-45
Protection	
Body standard/optional	IP67
Wrist & J3 arm standard/optional	IP67
Clean room (ISO Class 4)	-

● standard o on request - not available ( ) with hardware and/or software option *Based on ISO9283

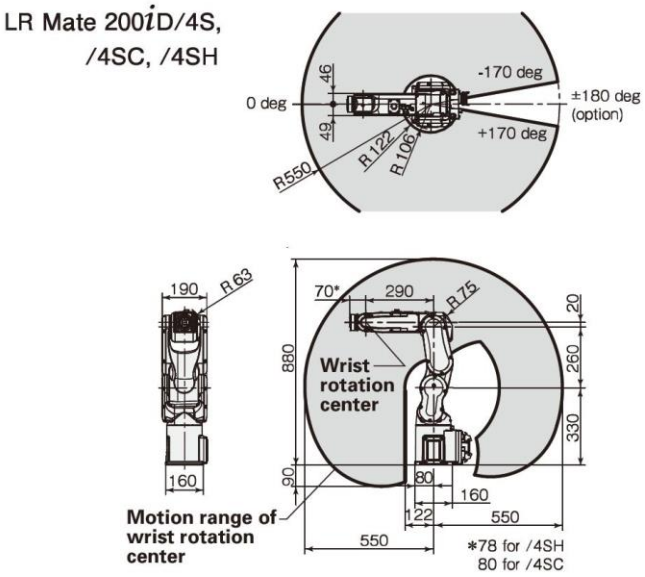
MDS-03808-EN Technical information subject to changes without prior notice. All rights reserved. ©2019 FANUC Europe Corporation

Joonis 6. Fanuc LR Mate 200iD/4s spetsifikatsioon [22]

4.3.1 Modelleerimine

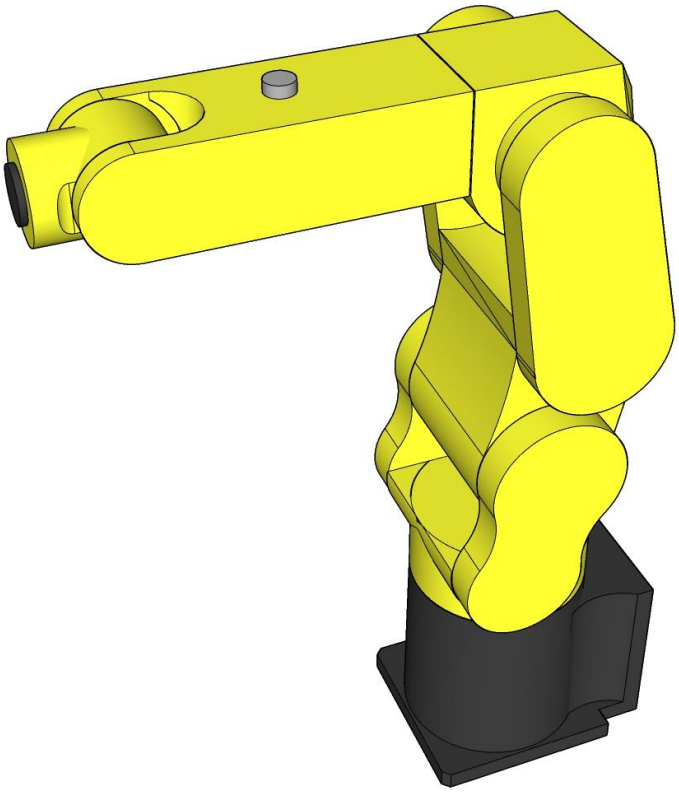
Autor pidi oma lõputöö eesmärgi täitmiseks modelleerima nullist tööstusroboti Fanuc LR Mate 200Id/4S digitaalse kasksiku, sest vastava seeria 3D-mudelit polnud võimalik leida internetist ja kontakteerumine Fanuc-i tugiteenusega ei andnud tulemust, et saada nende poolt roboti 3D-mudel.

Tööstusroboti modelleerimiseks teostas töö autor füüsilise roboti mõõdistamise, milleks kulus aega *ca* 1.5 tundi. Digitaalse kaksiku loomise etapile aitas kaasa tööstusroboti tööalajoonis (joonis 7). Antud joonis oli väga kasulik ja informatiivne 3D-mudeli loomise juures, mis sisaldas täpseid roboti linkide tsentrite vahelisi kaugusi. Selline informatsioon võimaldab luua digitaalse kaksiku, mis töötab virtuaalses keskkonnas sarnaselt päriselule.



Joonis 7. Fanuc LR Mate 200Id/4S tööalajoonis [23]

Roboti digitaalse kakasiku loomiseks kasutas autor *SketchUp* 3D-modelleerimise tarkvara. Tarkvara valimisel lähtuti autori modelleerimisprogrammi kasutamise oskustest. Kuna autoril on suur kogemus *SketchUp* tarkvaraga modelleerimisel, mis vähendas 3D-mudeli loomise aega märkimisväärselt. Tulemus on välja toodud joonisel 8.



Joonis 8. Fanuc LR Mate 200Id/4S digitaalse teisiku 3D mudel SketchUp tarkvaraga

4.3.2 Digitaalse kaksiku ettevalmistamine simulatsioonikeskkonnaks

Tööstusroboti digitaalse teisiku kasutamiseks simulatsioonikeskkonnas, peab antud 3D-mudelit konfigureerima, et selle kasutamine oleks arvutiressurssi vaene, mis omakorda tagab sujuvama kasutamise kogemuse, ja roboti ühendusliigendid oleks pööratavad z-telje suhtes.

Selleks kasutas töö autor vabavaralist tarkvara *Blender* ja läbis järgmises mudeli konfigureerimise etapid:

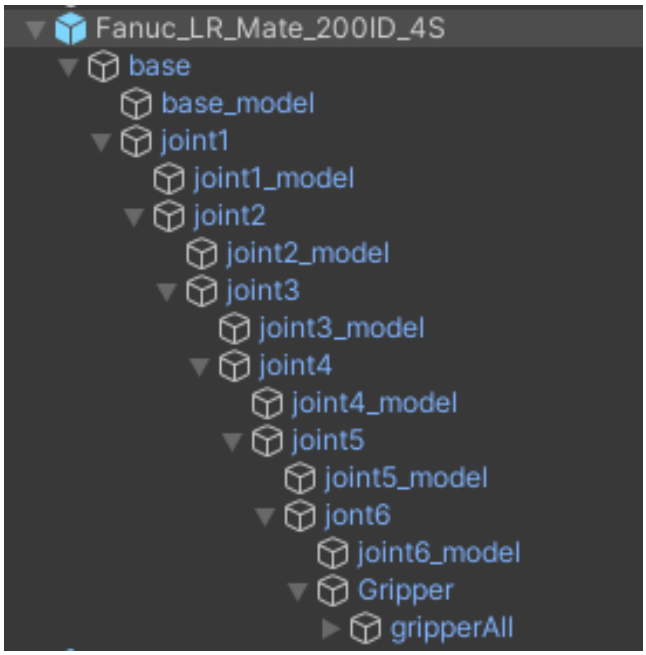
1 Mudeli punktide vähendamine – tagab mudeli faili väiksema suuruse, kuid ei muuda 3D mudeli visuaali;

2 Mudeli lokaalse arvutuspunkti muutmine – antud konfiguratsioon teostamine tagab mudeli korrektse pööramise täpselt õigest punktist;

3 Üleliigse informatsiooni eemaldamine mudelist – *SketchUp*-st eksporditud fail sisaldab endaga ebavajalikke objekte (kaamera ja valgustuse), mida ei ole vaja *Unity* keskkonnas.

4 Mudeli hierarhiapuu loomine – selles osas lõi autor digitaalse kaksiku hierarhiapuu, mis lubab liigutada alamobjekte vastavalt vanemobjekti pööramisel (joonis 9).

5 Mudeli eksportimine – roboti mudel eksporditi .dae formaati, sest antud formaat on väikese kaalukusega ning seda kasutatakse laialdaselt ka teistes roboti simulatsioonikeskkondades.



Joonis 9. Digitaalse kaksiku hierarhia välja nägemine *Unity* tarkvara näitel

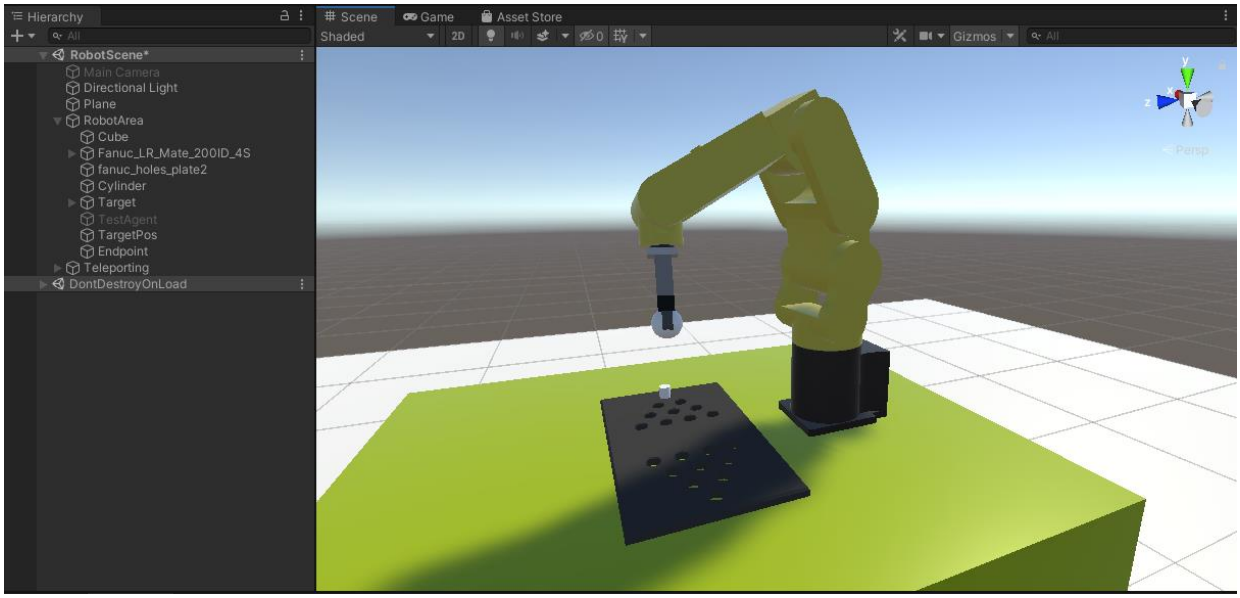
4.4 Simulatsioonikeskkonna teostus Unity-s

Alljärgnevas alampeatükis anna annab autor ülevaate simulatsioonikeskkonna loomisest *Unity*-s.

4.4.1 Simulatsiooni stseeni loomine

Simulatsioonikeskkonna stseen on virtuaalsete objektide kogum, kuhu on lisatud tööstusroboti digitaalne kaksik, virtuaalreaalsuse *Steam VR* plugin. Virtuaalreaalsuse on võimalik tööstusroboti asendit manipuleerida lõpp-punkti liigutades kontrolleriga. Samas on võimalik selles stseenis testida treenitud mudelit ning vaadelda simulatsiooni nii arvuti ekraani kui ka virtuaalreaalsuse vaates.

Järgnev joonis (joonis 10) illustreerib *Unity* programmis loodud roboti simulatsioonikeskkonna stseeni.



Joonis 10. Simulatsioonikeskkonna stseeni visuaal *Unity*-s

4.4.2 Pöörd kinemaatika funktsionaalsuse lisamine tööstusroboti digitaalsele teisikule simulatsioonikeskkonnas.

Töö autor valis välja pöördkinemaatika implementeerimiseks *Unity*-s loodud simulatsioonikeskkonnas Bio IK *asset*-i, mis on loodud Sebastian Starke doktoritöö raames. Antud *asset* võimaldab rakendada pöördkinemaatikat suure arvuliste vabadusastmetega robotitega/seadmetega.

Bio IK rakendamiseks magistritöös pidid autor uurima võimlaikult palju erinevaid uurimustöid, kus on kasutatud Bio IK-d, sest selle kohta puudub konkreetne dokumentatsioon.

Pöördkinemaatika rakendamiseks lisas töö autor tööstusroboti Fanuc LR Mate 200iD/4S digitaalse teisiku mudeli külge Bio IK skripti, kus tuleb määrata (joonis 11):

- iga liikuva liigendi pöörde telg, milleks igal liigendil on *z*-telg;
- maksimaalne ja minimaalne pöördenurk;
- lisada lõpp-punkt, mille asukoha manipuleerimisel, roboti digitaalne teisik muudab oma asendit;
- konfigureerida algoritmilisi parameetreid:
    - Generation – 3

        Ühe kaadri erinevate võimalike asendtite kogum. Võimalik rakendada ka kokkupõrgeteta trajektoori genereerimiseks.[8]

    - Individuals – 150

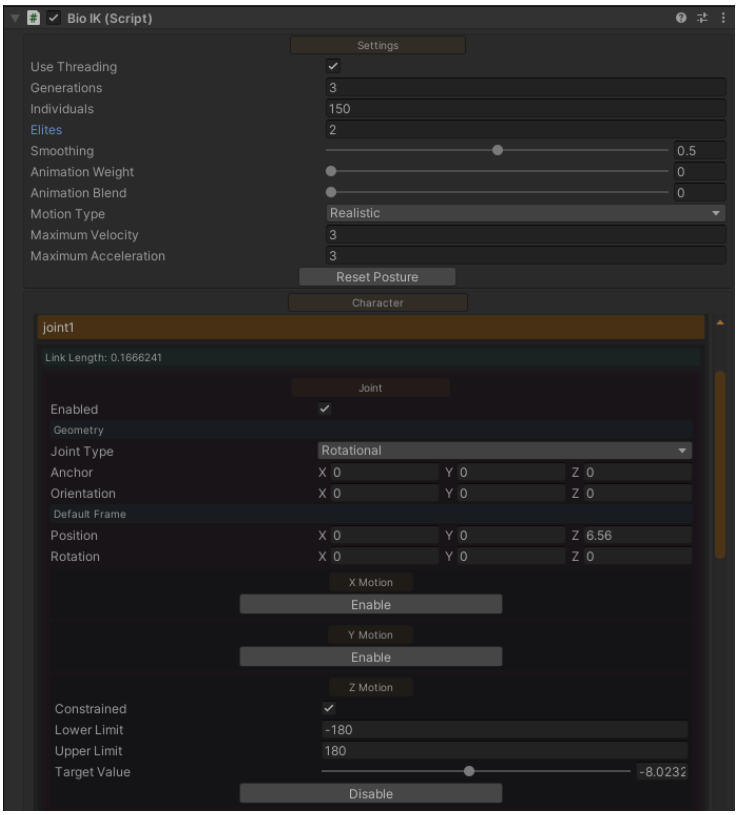
        Eelmiste generatsioonide võimalike asendite kogum.

    - Elites – 2

        Eelnevate edukate asendite omavahelise kvaliteedi kontrollimisarvutuste suurus.

    - Motion type - Realistlic

        Määratleb milline peab olema simuleerimise tulem.



Joonis 11. Bio IK skripti konfiguratsiooni vaade

Järgnevas tabelis on välja toodud iga liigendi pöördetelje valik, maksimaalne ja minimaalne pöördenurk

Tabel 4. Digitaalse teisiku pöörde telg, max/min pöördenurk

Liigendi nimetus	**Pööredtelg**	**Maksimum pöördenurk (kraadides)**	**Miinimum pöördenurk (kraadides)**
Joint 1	Z	180	-180
Joint 2	Z	120	-110
Joint 3	Z	50	-180
Joint 4	Z	180	-180
Joint 5	Z	90	-90
Joint 6	Z	360	-360

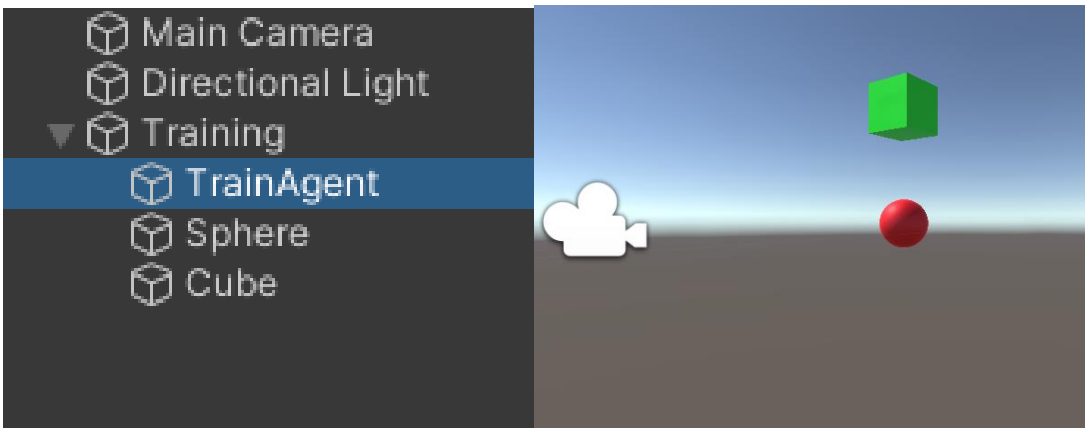
4.5 Masinõppe stseeni ja ML agendi loomine

Alljärgnevas alampeatükis anna annab autor ülevaate masinõppe rakendamisest *Unity* tarkvaras loodud simulatsioonikeskkonnas. Töös on kasutatud *Unity* ML-agent versiooni 1.7.

4.5.1 Masinõppe rakendamiseks stseeni loomine

Masinõppeks loodav treenimise stseen on lihtsustatud ülesehitusega, kus on (joonis 12):

1. Kuup(*cube*) suurusega x:20, y:20, z:20mm, mis imiteerib tööstusroboti haaratsit;
2. Sfäär(*sphere*) suurusega x:20, y:20, z:20mm, mis imiteerib lõpp-punkti, kuhu soovitakse tööstusrobotit liigutada või objekti mida tahetakse haarata/manipuleerida;
3. Tühi mänguobjekt(*TrainingAgent*) mille külge on liidetud loodud agendi/õppija skript.



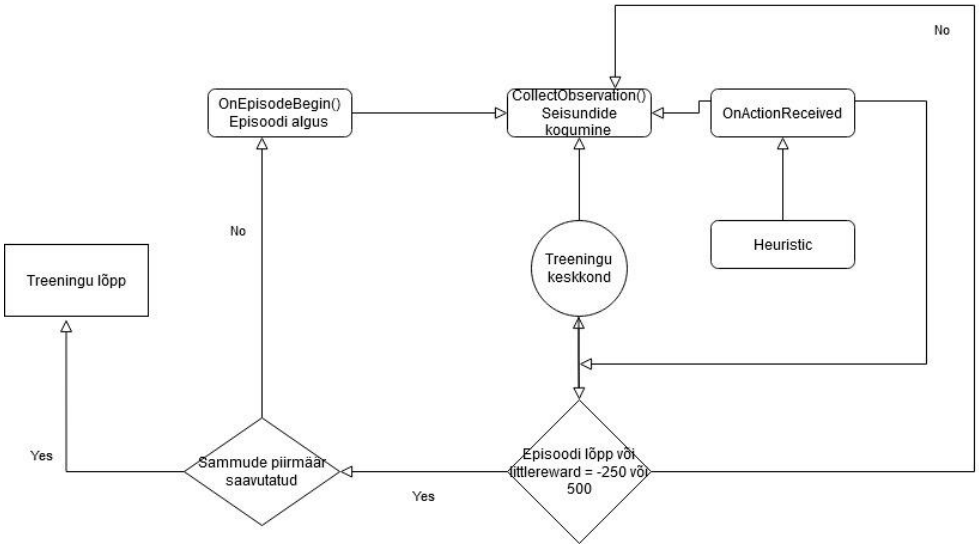
Joonis 12. Masinõppe stseen ja objektide hierarhia Unity-s

Eraldiseisev masinõppe treening stseen loodi, et toestada rohkema kui ühe treeningu samaaegset õpet. See võimaldab kordades kiiremini teostada treeningut. Autor kasutas ühe treenigu väljaga situatsiooni agendi skripti loomiseks ja 36 treeningu väljaga masinõppe teostamiseks.

4.5.2 Agendi skripti loomine

Agendi skript koosneb erinvatest funktsioonidest joonis 13:

- OnEpisodeBegin, kus defineeritakse ära milline peab olema keskkond iga episoodi alguses;
- CollectObservation, mis tegeleb keskkonnas olevate objektide asukoha vaatlemisega;
- Heuristic, mis sisaldab keskkonnas oleva õpetatava mänguobjekti juhtimise võimalusi;
- OnActionReceived, kus teostatakse episoodi ajal toiminguid- liikumine ja autasustamine.



Joonis 13. Agendi skripti töövoo UML diagramm

Töö autor alustas ML agendi/õppija skripti loomist lihtsa ülesande täitmisega, selleks lõi ühel teljel põhineva liikumise, et masinõpe suudaks õppida liikumist etteantud punktini. Seda selleks, et mõista täpsemalt ML agendi pistikprogrammi tööd. Ühe telje masinõppe agendi kood Lisa 1.

Järgmises etapis täiendas autor agendi koodi nii, et see suudaks liikuda kahest teljest koosnevas keskkonnas lisades juurde erinvaid vektoritel põhinevaid liikumisi. Agendi programmi kood Lisa 2.

Kolmandas etapis lisas autor juurde kolmanda telje põhise liikumise, mis muutis vektori põhise liikumise ruumis väga keeruliseks, et tagada kogu ruumiline liikumine kolmel teljel.

Iga lisanduva dimensiooniga täiendas autor autasustamise funktsionaalsust, sest masinõppe mudeli arenemise treeningu visuaalsel hindamisel ja graafikute vaatlemisel selgus, et ühe telje õppe mudeli jaoks loodud auhinna süsteem ei sobi järgenvatele etappidele (Lisa 3).

4.5.3 ML agendi treenimise faili konfigureerimine

Unity ML agent kasutab masinõppe teostamiseks *Python*-i API-t *mlagent*, mis koosneb kahest komponendist:

- Madala taseme API, mis võimaldab mõju avaldada koheselt *Unity* simulatsioonikeskkonnaga.
- Sisenemispunkt (*entry point)*, mis võimaldab toestada masinõpped *Unity* simulatsioonikeskkonnas, kasutades RL õpet. Antud komponent kasutab õppimiseks veel treening konfiguratsiooni faili *yaml* [24]

Lõputöö autor kasutas PPO treenimise tüüpi, mis kasutab närvivõrke, et leida parima funktsioon, mis antud vaatleja ja seisundiga on võimalik kasutada.

Yaml konfiguratsiooni fail sisaldab endas treenimise käitumuslikke iseärasusi (joonis 14).

```
behaviors:
  3DBall:
    trainer_type: ppo
    hyperparameters:
      batch_size: 32
      buffer_size: 2048
      learning_rate: 0.0003
      beta: 0.001
      epsilon: 0.2
      lambd: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 5
    max_steps: 500000
    time_horizon: 1000
    summary_freq: 12000
    threaded: true
```

Joonis 14. Konfiguratsiooni faili ülevaade

5 Tulemused ja analüüs

Järgnevalt toob töö autor välja teostatud tööstusroboti simulatsioonikeskkonnas masinõppega saadud tulemused ja nende hindamine.

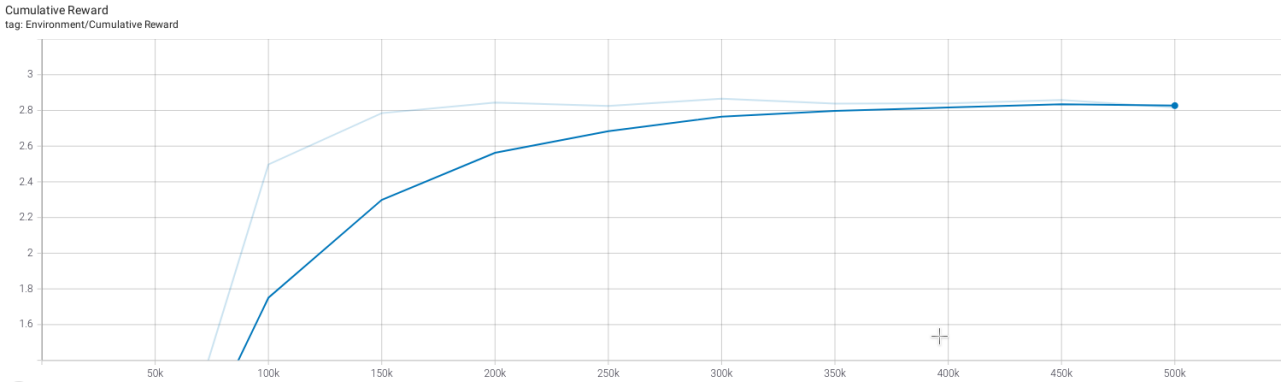
5.1 Masinõppe erinevate etappide tulemused

Masinõppe erinevate etappide läbimiseks kasutas töö autor ühte ja sama *yaml* konfiguratsioone, muutes dimensioonide juurde võtmisega ainult õppe sammude arvu. Tulemustena on välja toodud parimad mudelid.

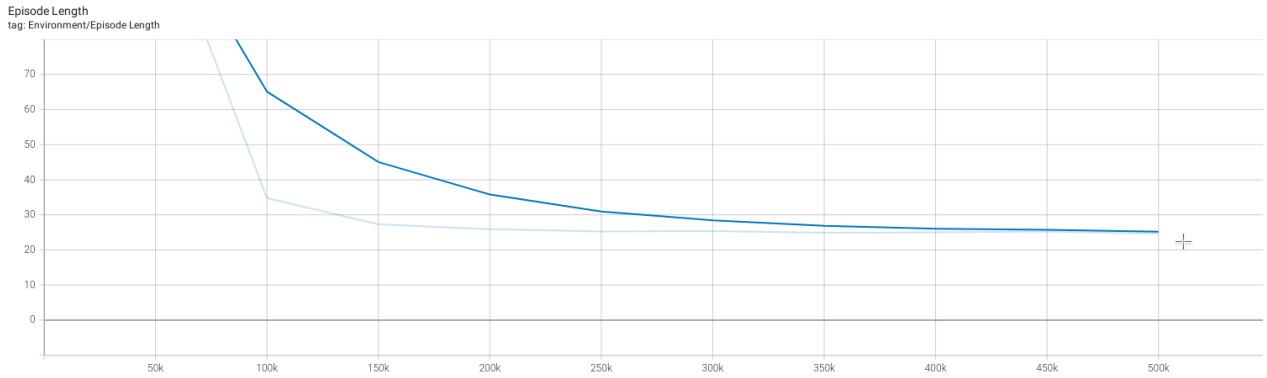
5.1.1 Ühel teljel suhtes liikumine masinõppe mudeli treenimine

Ühel teljel asukoha masinõppega mudeli treenimise eksperiment ei osutunud keeruliseks ülesandeks, kuigi tulemusena on välja toodud 8 katsetus. Mudeli treenimiseks teostati 500 000 sammu, mille tulemusel saavutati väga hea tulemusega mudel.

Graafikult kulminatiivne autasustamine on võimalik välja lugeda pidevalt parenevat masinõppe mudelit (joonis 15) ja samuti ka episoodi pikkuse graafik kinnitab seda, mis on kahanemise trendis (joonis 16).

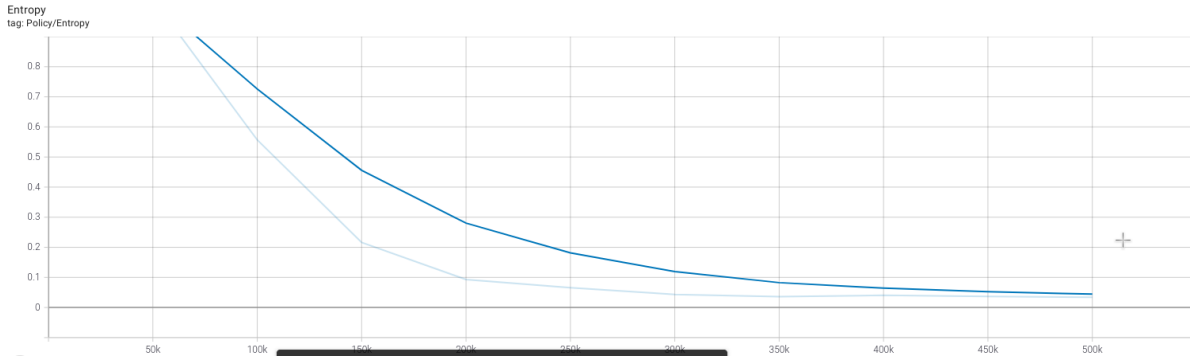


Joonis 15. Ühe telje masinõppe mudeli kulminatiivne autasustamise graafik



Joonis 16. Ühe telje masinõppe mudeli episoodi pikkuse graafik

Mudeli headust tõestab ka entroopia, kus graafik on pidevas languses ja mille väärtus on 500 0000 sammu juures alla 0.1 (joonis 17)



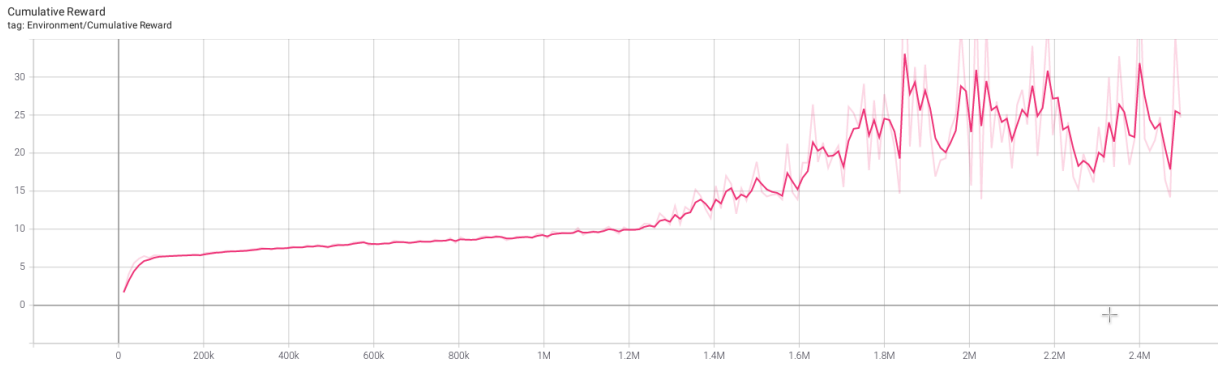
Joonis 17. Ühe telje masinõppe mudeli entroopia graafik

5.1.2 Kahe telje suhtes liikumise masinõppe mudeli treenimine

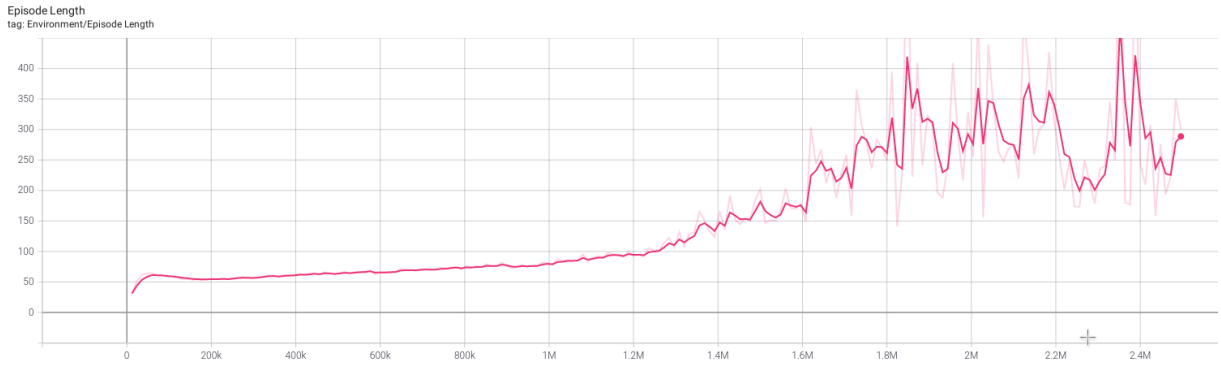
Kahel teljel objekti optimaalse teekonna leidmise treenimine lõpp-punktini oli keerulisem ülesanne, sest see eeldas liikumist nii öelda XY teljel nii positiivses kui ka negatiivses suunas. Täisauhinna +1 punkt määrati kui liikuva ja seisva objektide tsentrite vahelise kauguse eksimus oli >= 2mm. Mudeli treenimise pikkuseks oli 2,5M sammu ja esitletud on 10. katsetus.

Kulminaativse autasustamise graafikult on võimalik näha, et masinõppe mudel muutub järjest paremaks alates 1,4M sammu juures, kuid edasine õppe on hüplik, kuigi positiivse suunaga (joonis 18). Episoodi pikkuse graafik näitab samuti mudeli õppimist hüplikult (joonis 19). Antud tulemuseni võis viia, kiiruse muutmine keset õppe

episoodi. Alustamisel oli kiirus 50mm/s ja lõpp-punktile liginedes >= 50mm muudeti see 20mm/s peale, et ühes kaadris teostamine oleks täpsem, sest +1 autasu anti agendile siis kui lõpp-punkti ja manipuleeritava objektide vaheine kaugus on >= 2mm.



Joonis 18. Kahe teljega masinõppe mudeli kulminatiivne autasustamise graafik



Joonis 19. Kahe teljega masinõppe mudeli episoodi pikkuse graafik

Järgnevalt on esitatud entroopia graafik (joonis 20).

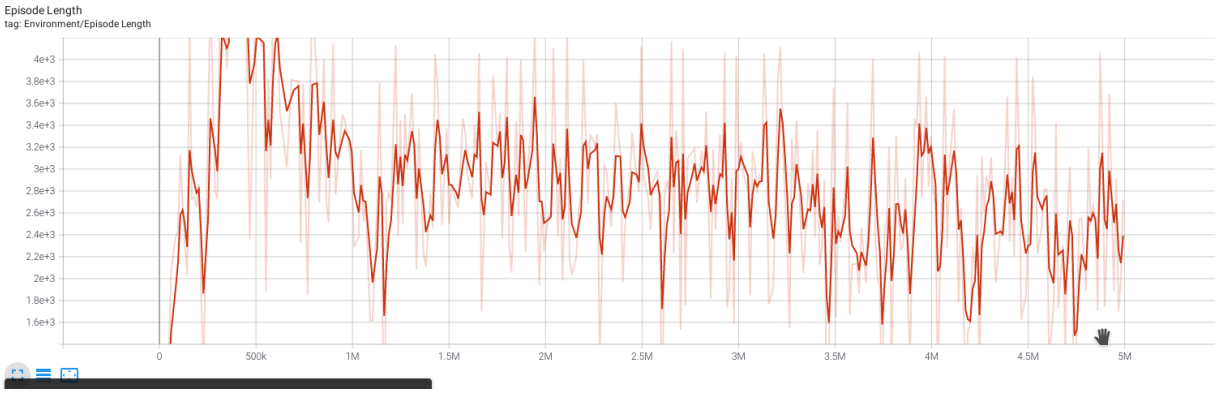
Joonis 20. Kahe teljega masinõppe mudeli entroopia graafik

5.1.3 Kolme telje suhtes liikumise masinõppe mudeli treenimine

Kolme telje suhtes objekti lõpp-punti liikumise treenimine, nagu oli arvata, osutus kõige keerulisemaks. Liikumine on võimalik XYZ telgedel, mis eeldab 26 erinevas suunas liikumise võimalust. Samuti suurenes selle eksperimendis vaatlus vektorite arv ja ümber oli vaja kujundada treenimise autasustamine.

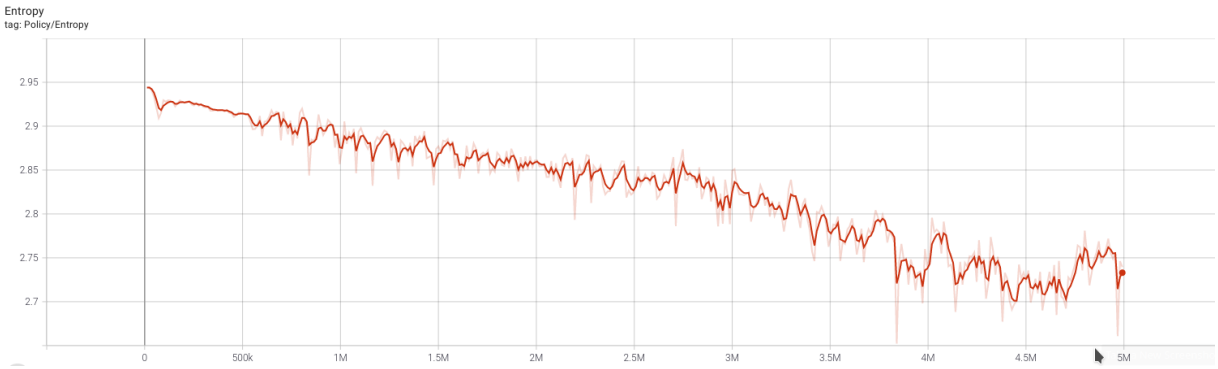
Kulminaativse autasustamise graafikult on võimalik näha, et masinõppe mudeli täpsus kasvab kuni 500 000 sammuni, kuid edasine õppe on väga ülese alla käiv. (joonis 21). Sama pilti näitab ka episoodi pikkuse graafik (joonis 22). Sarnast tendentsi oli märgata ka kahe telje masinõppe juures.

Joonis 21. Kolme teljega masinõppe mudeli kulminatiivne autasustamise graafik



Joonis 22. Kolme teljega masinõppe mudeli episoodi pikkuse graafik

Kui vaadata entroopia graafikut, see sellest tuleb välja, et masinõppe agent pole suutnud väga hästi treeningut teostada (joonis 23). Sellest võib järeldada kahte asja, kas suurendad treenigu sammude arvu või vaadata üle agendi autasustamise väärtused, et parandada treenimise tulemust.



Joonis 23. Kahe teljega masinõppe mudeli entroopia graafik

5.2 Hinnang tulemustele

Eelpool välja toodud tulemustest saab autor järeldada, et sellisel lähenemisel on täitsa oma koht maailmas olemas. See peaks andma simulatsioonikeskkondadele parema kasutamise võimalused, näiteks roboti trajektoori disainimise juures läbi paljude kasutaja poolt paika pandud punktide läbi. Selline lähenemine võimaldab robotite seadistamise pealt aega kokku hoida. Simulatsioonikeskkonnas saab väga lihtsal viisil uusi asju integreerida, et õpetada robotit tegema uusi toiminguid ja testida, mis on kuluefektiivsem kui selle teostamine reaalsel robotil.

Lõputöö autori arvates saaks kolme telje suhtes liikumise masinõppe mudeli treenimist parendada, et tulem oleks veel parem. Selleks oleks vaja veel täpsemalt uurida ja analüüsida, mis mudeli treenimisel valesti läks.

Autori arvates halb tulem on tingitud mitmest aspektist:

- autori vähesed oskused sellise masinõppe rakendamise kohta;
- Mitte täielikkude teadmiste olemasolu nii robottehnilistes süsteemides.

5.3 Alternatiivsed lahendused

Lõputöö autor on välja toonud peatükis 2, et antud teema on väga aktuaalne hetkelises digitaliseerimise ja automatiseerimise ühiskonnas. Loodud on palju sarnase teemalisi uuringuid, mis käsitlevad tööstusrobotite liikumise teekondade automatiseerimist läbi masinõppe mudelite. Kuid antud uuringud on keskendunud tööstusroboti liigendite pöördnurkade automaatsele leidmisele. Samas on selline lähenemine väga head tulemusi andnud, kuid siin seisneb autori arvates kitskoht. Sellise suunitlusega masinõppe mudeleid saab kasutada ainult neil tööstusrobotitel, mille digitaalse kaksikuga on need treenitud.

5.4 Majanduslik tasuvus

Hindamaks antud töö majandusliku aspekti, tuleb välja tuua projektiga seonduvad ressursid (tabel 5 ja 6).

Tabel 5. Töös kasutatud riistvara

Toode	**Nimetus**	**Hind eurodes**
Arvuti	HP Z1 tower Workstation	800
Graafikakaart	Nvidia RTX 2070	~700
VR peakomplekt	HTC Vive	~650
	Kokku	**~2150**

Tabel 6. Töös kasutatud tarkvara

Tarkvara	**Nimetus**	**Hind eurodes**
Simulatsioonikeskkond	Unity mängmootor	0
Modelleerimise tarkvara	SketchUp Make 2017	0
3D- mudeli konfigureerimise tarkvara	Blender	0
Lähtekoodiredaktor	Atom	0
VR rakendus	SteamVR pistikprogramm	0
Masinõpe	Unity ML-agent	0
Pöördkinemaatika	Bio IK	17.87
	Kokku	**17.87**

Eelpool välja toodud tabelites (tabel 5 ja 6) on väljatoodud projekti teostamiseks vajalikude komponentide maksumused, kuid antud magistritöö teostamiseks sai autor kasutada Taltech Virumaa kolledži seadmeid ja Bio IK *asset*-i kasutamise võimaldas töö

juhendaja. Sams nende tabelite põhjal on võimalik näha kui palju läheb maksma taolise projekti teostamine.

Võrreldes turul olevate masinõppe võimekusega simulatsioonikeskkondadega, mille litsentsi maksumused ainuüksi on 1500+ €, kõrval on magistritöö raames loodud kontseptsioon väga mõistliku hinnaga.

5.5 Töö edasiarendus

Antud tööl on oma kindel potentsiaal nii praegu kui ka edasises tulevikus, sest selline lähenemine võimalda treenida uusi füüsilise tootmisrobotite seadistusi, ilma et peaks kulutama väärtusliku tööaega põhiprotsessis. Simulatsiooni keskkonnas saadud tulemusi on võimalik kanda üle reaalsele robotile. Seega pakub lõputöö autor välja võimalikud edasised arendused.

Töö käigus selgus, et sellisel tööstusroboti masinõppega treenimisel on olemas tulemus, millega võiks edasi liikuda tulevikus. Selleks peaks arendama treeningu agenti nii, et see suudaks ka arvestada lõpp-punkti orientatsiooniga. See annaks loodavale mudelile veel suurema valdkondliku kasutamise võimekuse. Samuti võiks lisada juurde takistuste vältimise funktsionaalsuse. Seda teostada kas kahe mudeli loomisega või lisada see arendatavasse mudelisse. See võimaldaks efektiivselt kasutada tööstusroboti tööaega, rakendades uusi tootmisseadistusi, mis on saadud simulatsioonikeskkonnast.

Samas peaks leidma võimaluse kogu simulatsioonikeskkond ühenda reaalsete füüsiliste tööstusrobotitega, et anda enda panus Tööstus 4.0 kontseptsiooniga kokku käivasse digitaliseerimise ja automatiseerimise edasisse arenduse protsessi.

6 Kokkuvõte

Elame ajahetkel, kus ühiskond on pidevas kiires muutuses ja millega peab sammu pidama samuti tootev tööstus, seda läbi robotiseerimis, automatiseerimise ja kõikide protsesside digitaliseerimise. Kuid hetkel esinevad selles valdkonnas probleemid:

1. spetsiifilise haridusega tööjõu vähene olemasolu ettevõtetes, kes suudaks tööstusroboteid seadistada ja hooldada;

2. robotite seadistamisega ning testimega suur ajaline kulu, mis on ettevõttele suur rahaline väljaminek;

Loodud magistritöö eesmärgiks on arendadavälja tööstusroboti simulatsioonikeskkond masinõppe rakendamise keskkonnana, kus peaks olema võimalik simuleerida roboti tööd, mida saab vaadelda/uurida nii arvutiekraanilt kui ka virtuaalreaalsuses.

Selleks uuris ja analüüsi töö autor tööstusrobotite *off-line* programmeerimise keskkondi ning teadustöid, mis sarnaste teemadega on tegelenud. Samuti valis välja tööriistad, millega püstitatud eesmärki saavutada.

Lõputöö tulemuseks loodi tööstusroboti simulatsiooni- ja masinõppe keskkond. Samuti loodi erinevaid masinõppe mudeleid, mida liidestati virtuaalse tööstusrobotiga integreeritud pöördkinemaatika funktsionaalsusega. See omakorda võimaldab roboti asendit muuta manipuleerides lõpp-punkti asukohta 3 mõõtmelises keskkonnas.

Saadud masinõppe mudeliga koos pöördkinemaatikaga on võimalik tööstusrobotit liikumist manipuleerida, mis oli autori uurimuse eesmärk. Kuid loodud ML mudel ei ole väga täpne püstitatud eesmärgi ootusest. Samas leidis töö autor, et sellise kombinatsiooni rakendamisega võiks tulevikus edasi tegeleda. See võimaldab ühte treenitud mudelit kasutada erinevate sarnaste seadmete juures, et mitte treenida iga seadme jaoks uut masinõppe mudelit. Samas võimaldab selline lahendus hoida kokku uute süsteemide arendamise ajalist ja rahalist kulu.

Kasutatud kirjandus

[1] W. Bauer, S. Schlund, T. Hornung, ja S. Schuler, „Digitalization of industrial value chains-a review and evaluation of existing use cases of Industry 4.0 in Germany LogForum DIGITALIZATION OF INDUSTRIAL VALUE CHAINS-A REVIEW AND EVALUATION OF EXISTING USE CASES OF INDUSTRY 4.0 IN GERMANY“, *LogForum*, kd 14, nr 3, lk 331–340, 2018, doi: 10.17270/J.LOG.2018.288.

[2] M. Lin, L. San, ja Y. Ding, „Construction of Robotic Virtual Laboratory System Based on Unity3D“, *IOP Conf. Ser. Mater. Sci. Eng.*, kd 768, nr 7, 2020, doi: 10.1088/1757-899X/768/7/072084.

[3] A. Kumar Bedaka, J. Vidal, ja C.-Y. Lin, „Automatic robot path integration using three-dimensional vision and offline programming“, *Int. J. Adv. Manuf. Technol.*, kd 102, lk 1935–1950, 2019, doi: 10.1007/s00170-018-03282-w.

[4] B. Jakubiec, „APPLICATION OF SIMULATION MODELS FOR PROGRAMMING OF ROBOTS“, *Soc. Integr. Educ. Proc. Int. Sci. Conf.*, kd 5, lk 283, mai 2018, doi: 10.17770/sie2018vol1.3214.

[5] Prasanth T, „Implementation of ROS-I on industrial robot simulation environment“, 2019. Vaadatud: apr 26, 2021. [Online]. Available at: www.IJARIIT.com.

[6] L. Hong, B. Wang, X. Yang, Y. Wang, ja Z. Lin, „Offline programming method and implementation of industrial robot grinding based on VTK“, doi: 10.1108/IR-04-2019-0093.

[7] M. Lin, L. Shan, ja Y. Zhang, „Research on robot arm control based on Unity3D machine learning“, *J. Phys. Conf. Ser.*, kd 1633, nr 1, 2020, doi: 10.1088/1742-6596/1633/1/012007.

[8] S. Starke, „Bio IK: A Memetic Evolutionary Algorithm for Generic Multi-

Objective Inverse Kinematics“.

[9] L. Pitonakova, M. Giuliani, A. Pipe, ja A. Winfield, „Feature and performance comparison of the V-REP, Gazebo and ARGoS robot simulators“, *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, kd 10965 LNAI, nr February, lk 357–368, 2018, doi: 10.1007/978-3-319-96728-8_30.

[10] C. Souza ja L. Velho, „Plug&Play Task selection Hierarchical Agent High level Learned tasks Low level Animation-Heuristic“.

[11] W. Zhang, Z. Yang, T. Dong, ja K. Xu, „FABRIKc: An efficient iterative inverse kinematics solver for continuum robots“, *IEEE/ASME Int. Conf. Adv. Intell. Mechatronics, AIM*, kd 2018-July, lk 346–352, 2018, doi: 10.1109/AIM.2018.8452693.

[12] E. Sujova, H. Cierna, ja R. Bambura, „Simulation model of production as tool for industry 4.0 implementation into practice“, *Engineering for Rural Development*, 2019, kd 18, lk 1192–1197, doi: 10.22616/ERDev2019.18.N279.

[13] *Theory of Applied*. .

[14] S. Starke, N. Hendrich, ja J. Zhang, „Memetic Evolution for Generic Full-Body Inverse Kinematics in Robotics and Animation“, *IEEE Trans. Evol. Comput.*, kd 23, nr 3, lk 406–420, juuni 2019, doi: 10.1109/TEVC.2018.2867601.

[15] A. Aristidou ja J. Lasenby, „Inverse Kinematics: a review of existing techniques and introduction of a new fast iterative solver“, *Lancet*, kd 158, nr 4071, lk 644, 2009, [Online]. Available at: http://linkinghub.elsevier.com/retrieve/pii/S0140673601720557.

[16] W. Wang ja K. Siau, „Artificial intelligence, machine learning, automation, robotics, future of work and future of humanity: A review and research agenda“, *J. Database Manag.*, kd 30, nr 1, lk 61–79, 2019, doi: 10.4018/JDM.2019010104.

[17] „Masinõppimine - Masinõpe“. https://masinope.ee/masinoppimine/ (vaadatud apr

27, 2021).

[18] J. LAI, X. CHEN, ja X. ZHANG, „Training an Agent for Third-person Shooter Game Using Unity ML-Agents“, *DEStech Trans. Comput. Sci. Eng.*, nr icaic, lk 305–310, 2019, doi: 10.12783/dtcse/icaic2019/29442.

[19] „GitHub - Unity-Technologies/ml-agents: Unity Machine Learning Agents Toolkit“. https://github.com/Unity-Technologies/ml-agents (vaadatud mai 04, 2021).

[20] A. Plasencia, Y. Shichkina, I. Suárez, ja Z. Ruiz, „Open source robotic simulators platforms for teaching deep reinforcement learning algorithms“, *Procedia Comput. Sci.*, kd 150, lk 162–170, 2019, doi: 10.1016/j.procs.2019.02.031.

[21] „The agent{environment i n teraction in reinforcement learning. | Download Scientific Diagram“. https://www.researchgate.net/figure/The-agentenvironment-i-n-teraction-in-reinforcement-learning_fig1_270960086 (vaadatud mai 07, 2021).

[22] „LR Mate 200iD/4S industrial robot“. https://www.fanuc.eu/ro/en/robots/robot-filter-page/lrmate-series/lrmate-200id-4s (vaadatud mai 07, 2021).

[23] „FANUC LR Mate robots“. http://www.americanrobotsales.com/LRMATE/lrmate200id4s.htm (vaadatud mai 07, 2021).

[24] „ml-agents/Python-API.md at main · Unity-Technologies/ml-agents · GitHub“. https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Python-API.md (vaadatud mai 05, 2021).

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

Mina, Karle Nutonen

1. Annan Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „TÖÖSTUSROBOTI SIMULATSIOONIKESKKONNA VÄLJAARENDUS, MASINÕPPE TESTKESKKONNA KASUTAMISE OTSTARBEKS“, mille juhendaja on Vladimir Kuts
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Olen teadlik, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitan, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

10.05.2021

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

Lisa 2. Ühe telje treeningu agendi programmi kood

```
public class SingelVectorLearn : Agent
{
    public Transform cube;
    public Transform sphere;
    public float currentLocationX;
    public float targetLocationX;
    public int solve = 0;
    public float movementLenght = 0.2f;
    public float lastDifferenceX = 0f;


    public override void OnEpisodeBegin()
    {
        targetLocationX = UnityEngine.Random.RandomRange(-1f, 1f);
        sphere.localPosition = new Vector3(targetLocationX,0,0);


        currentLocationX = 0f;
        cube.localPosition = new Vector3(currentLocationX,0,0);

        lastDifferenceX =0f;

    }

    public override void OnActionReceived(float[] vectorAction)
    {

        AddReward(-0.005f);
        var actionY = (int)vectorAction[0];
        switch(actionY)
        {
          case 1:
            currentLocationX += movementLenght*Time.deltaTime;
            //Debug.Log("X+");
            break;
          case 2:
            currentLocationX -= movementLenght*Time.deltaTime;
            //Debug.Log("X-");
            break;
        }

        cube.localPosition = new Vector3(currentLocationX,0,0);

        //float distanceToTarget = Vector3.Distance(cube.localPosition,
sphere.localPosition);

        float differenceX = Mathf.Abs(targetLocationX - currentLocationX);

        if(currentLocationX > 1.2f || currentLocationX < -1.2f)
```

```
        {
          AddReward(-1.0f);
          EndEpisode();
        }

        if (differenceX <= 0.002f)
        {
            solve ++;
            AddReward(1.0f);
            EndEpisode();
        }

        if(lastDifferenceX > differenceX)
        {
          AddReward(0.02f);
        }
        if(lastDifferenceX < differenceX)
        {
          AddReward(-0.02f);
        }

        lastDifferenceX = differenceX;

    }

    public override void CollectObservations(VectorSensor sensor)
    {

        sensor.AddObservation(targetLocationX);
        sensor.AddObservation(currentLocationX);
    }

    public override void Heuristic(float[] actionsOut)
    {


        actionsOut[0] = 0;
        if(Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0] = 1;

        }

        if(Input.GetKey(KeyCode.LeftArrow))
        {
          actionsOut[0] = 2;

        }
    }

}
```

Lisa 2. Kahe telje treeningu agendi kood

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;

public class TwoVectorLearn : Agent
{
    public GameObject cube;
    public Transform sphere;
    public float currentLocationX;
    public float targetLocationX;
    public float currentLocationY;
    public float targetLocationY;
    public int solve = 0;
    public float movementLenghtX = 0.5f;
    public float movementLenghtY = 0.5f;
    public float lastDifferenceX = 0f;
    public float lastDifferenceY = 0f;
    public int lillteReward = 0;

    public float lastDiffernceDistance;


    public override void OnEpisodeBegin()
    {
        targetLocationX = UnityEngine.Random.RandomRange(-1f, 1f);
        targetLocationY = UnityEngine.Random.RandomRange(-1f, 1f);
        sphere.localPosition = new
Vector3(targetLocationX,targetLocationY,0);


        currentLocationX = 0f;
        currentLocationY = 0f;
        cube.transform.localPosition = new
Vector3(currentLocationX,currentLocationY,0);

        lastDifferenceX =0f;
        lastDifferenceY =0f;

        movementLenghtX = 0.5f;
        movementLenghtY = 0.5f;
        lillteReward = 0;

    }

    public override void OnActionReceived(float[] vectorAction)
    {
```

```
        AddReward(-0.005f);
        var actionY = (int)vectorAction[0];
        switch(actionY)
        {
          case 1:
            currentLocationX += movementLenghtX*Time.deltaTime;
            //Debug.Log("X+");
            break;
          case 2:
            currentLocationX -= movementLenghtX*Time.deltaTime;
            //Debug.Log("X-");
            break;
          case 3:
            currentLocationY += movementLenghtY*Time.deltaTime;
            //Debug.Log("X+");
            break;
          case 4:
            currentLocationY -= movementLenghtY*Time.deltaTime;
            //Debug.Log("X-");
            break;
          case 5:
            currentLocationX += movementLenghtX*Time.deltaTime;
            currentLocationY += movementLenghtY*Time.deltaTime;
            //Debug.Log("X+ ja Y+")2;
            break;
          case 6:
            currentLocationX += movementLenghtX*Time.deltaTime;
            currentLocationY -= movementLenghtY*Time.deltaTime;
            //Debug.Log("X+ ja Y-");
            break;
          case 7:
            currentLocationX -= movementLenghtX*Time.deltaTime;
            currentLocationY += movementLenghtY*Time.deltaTime;
            //Debug.Log("X- ja Y+");
            break;
          case 8:
            currentLocationX -= movementLenghtX*Time.deltaTime;
            currentLocationY -= movementLenghtY*Time.deltaTime;
            //Debug.Log("X- ja Y-");
            break;


        }

        cube.transform.localPosition = new
Vector3(currentLocationX,currentLocationY,0);
        //cube.AddForce(new Vector3(currentLocationX,0,0),
ForceMode.VelocityChange);
        float distanceToTarget =
Vector3.Distance(cube.transform.localPosition, sphere.localPosition);

        float differenceX = Mathf.Abs(targetLocationX - currentLocationX);
        float differenceY = Mathf.Abs(targetLocationY - currentLocationY);

        if(differenceX > 0.05 && differenceX < 0.2)
        {
          movementLenghtX = 0.2f;
        }
        if(differenceY > 0.05 && differenceY < 0.2)
```

```
{
  movementLenghtY = 0.2f;
}

if(differenceX < 0.05)
{
  movementLenghtX = 0.02f;
}
if(differenceY < 0.05)
{
  movementLenghtY = 0.02f;
}


if(currentLocationX > 1.2f || currentLocationX < -1.2f)
{
  AddReward(-1.0f);
  EndEpisode();
}

if (distanceToTarget <= 0.002f)
{
    solve ++;
    AddReward(1.0f);
    EndEpisode();
}


if(lastDifferenceX > differenceX)
{
  AddReward(0.002f);
}

if(lastDifferenceY > differenceY)
{
  AddReward(0.002f);
}

if(distanceToTarget < lastDifferнceDistance )
{
  lillteReward++;
  AddReward(0.002f);
  //EndEpisode();
}
if(lillteReward == 1000)
{
  //AddReward(-.5f);
  EndEpisode();
}
if(distanceToTarget > lastDifferнceDistance )
{
  lillteReward--;
  AddReward(-0.002f);
  //EndEpisode();
}
if(lillteReward == -50)
{
  //AddReward(-1.0f);
  EndEpisode();
```

```
        }

        lastDifferenceX = differenceX;
        lastDifferenceY = differenceY;
        lastDiffernceDistance = distanceToTarget;

    }

    public override void CollectObservations(VectorSensor sensor)
    {

        sensor.AddObservation(targetLocationX);
        sensor.AddObservation(currentLocationX);
        sensor.AddObservation(targetLocationY);
        sensor.AddObservation(currentLocationY);
    }

    public override void Heuristic(float[] actionsOut)
    {


        actionsOut[0] = 0;
        // X axis
        if(Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0] = 1;

        }

        if(Input.GetKey(KeyCode.LeftArrow))
        {
          actionsOut[0] = 2;

        }
        // Y axis
        if(Input.GetKey(KeyCode.UpArrow))
        {
          actionsOut[0] = 3;
        }

        if(Input.GetKey(KeyCode.DownArrow))
        {
          actionsOut[0] = 4;
        }

        // X and Y
        if(Input.GetKey(KeyCode.UpArrow) && Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0]=5;
        }

        if(Input.GetKey(KeyCode.DownArrow) &&
Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0]=6;
        }

        if(Input.GetKey(KeyCode.UpArrow) && Input.GetKey(KeyCode.LeftArrow))
        {
```

```
            actionsOut[0]=7;
          }

          if(Input.GetKey(KeyCode.DownArrow) &&
Input.GetKey(KeyCode.LeftArrow))
          {
            actionsOut[0]=8;
          }
      }

}
```

Lisa 3. Kolme telje treeningu agendi kood

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;

public class DemoAgent : Agent
{
    public float currentLocation;
    public float targetLocation;
    public float currentLocationX;
    public float currentLocationY;
    public float currentLocationZ;
    public float targetLocationX;
    public float targetLocationY;
    public float targetLocationZ;
    public int solve = 0;
    public float movementLenghtX = 0.5f;
    public float movementLenghtY = 0.5f;
    public float movementLenghtZ = 0.5f;
    public float lastDifferenceX = 0f;
    public float lastDifferenceY = 0f;
    public float lastDifferenceZ = 0f;
    public int lillteReward = 0;
    public Transform cube;
    public Transform sphere;


    public override void OnEpisodeBegin()
    {
      //targetLocation = UnityEngine.Random.RandomRange(-1f, 1f);
      targetLocationX = UnityEngine.Random.RandomRange(-1f, 1f);
      targetLocationY = UnityEngine.Random.RandomRange(-1f, 1f);
      targetLocationZ = UnityEngine.Random.RandomRange(-1f, 1f);
      sphere.localPosition=new
Vector3(targetLocationX,targetLocationY,targetLocationZ);
      //currentLocation = 0f;
      currentLocationX = 0f;
      currentLocationY = 0f;
      currentLocationZ = 0f;
      cube.localPosition=new
Vector3(currentLocationX,currentLocationY,currentLocationZ);
      //Debug.Log(sphere.localPosition);
      //Debug.Log(cube.localPosition);
      lastDifferenceX =0f;
      lastDifferenceY =0f;
      lastDifferenceZ =0f;

      movementLenghtX = 0.5f;
```

```
  movementLenghtY = 0.5f;
  movementLenghtZ = 0.5f;
  lillteReward = 0;

}
public float lastDiffernceDistance;

public override void OnActionReceived(float[] vectorAction)
{
  AddReward(-0.005f);
  var actionY = (int)vectorAction[0];
  switch(actionY)
  {
    case 1:

      currentLocationY += movementLenghtY*Time.deltaTime;
      //Debug.Log("Y+");
      //currentLocation -= 0.01f;
      break;
    case 2:
      currentLocationY -= movementLenghtY*Time.deltaTime;
      //Debug.Log("Y-");
      //currentLocation += 0.01f;
      break;
      case 3:
        currentLocationX += movementLenghtX*Time.deltaTime;
        //Debug.Log("X+");
        break;
      case 4:
        currentLocationX -= movementLenghtX*Time.deltaTime;
        //Debug.Log("X-");
        break;
      case 5:
        currentLocationZ += movementLenghtZ*Time.deltaTime;
        //Debug.Log("Z+");
        break;
      case 6:
        currentLocationZ -= movementLenghtZ*Time.deltaTime;
        //Debug.Log("Z-");
        break;
        //X and Y
      case 7:
        currentLocationX += movementLenghtX*Time.deltaTime;
        currentLocationY += movementLenghtY*Time.deltaTime;
        //Debug.Log("X+ ja Y+")2;
        break;
      case 8:
        currentLocationX += movementLenghtX*Time.deltaTime;
        currentLocationY -= movementLenghtY*Time.deltaTime;
        //Debug.Log("X+ ja Y-");
        break;
      case 9:
        currentLocationX -= movementLenghtX*Time.deltaTime;
        currentLocationY += movementLenghtY*Time.deltaTime;
        //Debug.Log("X- ja Y+");
        break;
      case 10:
        currentLocationX -= movementLenghtX*Time.deltaTime;
        currentLocationY -= movementLenghtY*Time.deltaTime;
```

```
  //Debug.Log("X- ja Y-");
  break;
  //X and Z
case 11:
  currentLocationX += movementLenghtX*Time.deltaTime;
  currentLocationZ += movementLenghtZ*Time.deltaTime;
  //Debug.Log("X+ ja Y+");
  break;
case 12:
  currentLocationX += movementLenghtX*Time.deltaTime;
  currentLocationZ -= movementLenghtZ*Time.deltaTime;
  //Debug.Log("X+ ja Z-");
  break;
case 13:
  currentLocationX -= movementLenghtX*Time.deltaTime;
  currentLocationZ += movementLenghtZ*Time.deltaTime;
  //Debug.Log("X- ja Z+");
  break;
case 14:
  currentLocationX -= movementLenghtX*Time.deltaTime;
  currentLocationZ -= movementLenghtZ*Time.deltaTime;
  //Debug.Log("X+ ja Z-");
  break;
  //Y and Z
case 15:
  currentLocationZ += movementLenghtZ*Time.deltaTime;
  currentLocationY += movementLenghtY*Time.deltaTime;
  //Debug.Log("Z+ ja Y+");
  break;
case 16:
  currentLocationZ += movementLenghtZ*Time.deltaTime;
  currentLocationY -= movementLenghtY*Time.deltaTime;
  //Debug.Log("Z+ ja Y-");
  break;
case 17:
  currentLocationZ -= movementLenghtZ*Time.deltaTime;
  currentLocationY += movementLenghtY*Time.deltaTime;
  //Debug.Log("Z- ja Y+");
  break;
case 18:
  currentLocationZ -= movementLenghtZ*Time.deltaTime;
  currentLocationY -= movementLenghtY*Time.deltaTime;
  //Debug.Log("Z- ja Y-");
  break;
case 19:
  currentLocationX += movementLenghtX*Time.deltaTime;
  currentLocationZ += movementLenghtY*Time.deltaTime;
  currentLocationY += movementLenghtZ*Time.deltaTime;
  //Debug.Log("Z- ja Y-");
  break;
case 20:
  currentLocationX -= movementLenghtX*Time.deltaTime;
  currentLocationZ += movementLenghtY*Time.deltaTime;
  currentLocationY += movementLenghtZ*Time.deltaTime;
  //Debug.Log("Z- ja Y-");
  break;
case 21:
  currentLocationX += movementLenghtX*Time.deltaTime;
  currentLocationY -= movementLenghtY*Time.deltaTime;
```

```
            currentLocationZ += movementLenghtZ*Time.deltaTime;
            //Debug.Log("Z- ja Y-");
            break;
          case 22:
            currentLocationX -= movementLenghtX*Time.deltaTime;
            currentLocationY -= movementLenghtY*Time.deltaTime;
            currentLocationZ += movementLenghtZ*Time.deltaTime;
            //Debug.Log("Z- ja Y-");
            break;
          case 23:
            currentLocationX += movementLenghtX*Time.deltaTime;
            currentLocationY += movementLenghtY*Time.deltaTime;
            currentLocationZ -= movementLenghtZ*Time.deltaTime;
            //Debug.Log("Z- ja Y-");
            break;
          case 24:
            currentLocationX -= movementLenghtX*Time.deltaTime;
            currentLocationY += movementLenghtY*Time.deltaTime;
            currentLocationZ -= movementLenghtZ*Time.deltaTime;
            //Debug.Log("Z- ja Y-");
            break;
          case 25:
            currentLocationX += movementLenghtX*Time.deltaTime;
            currentLocationY -= movementLenghtY*Time.deltaTime;
            currentLocationZ -= movementLenghtZ*Time.deltaTime;
            //Debug.Log("Z- ja Y-");
            break;
          case 26:
            currentLocationX -= movementLenghtX*Time.deltaTime;
            currentLocationY -= movementLenghtY*Time.deltaTime;
            currentLocationZ -= movementLenghtZ*Time.deltaTime;
            //Debug.Log("Z- ja Y-");
            break;

      }


      cube.localPosition = new
Vector3(currentLocationX,currentLocationY,currentLocationZ);


      float distanceToTarget = Vector3.Distance(cube.localPosition,
sphere.localPosition);

      float differenceX = Mathf.Abs(targetLocationX - currentLocationX);
      float differenceY = Mathf.Abs(targetLocationY - currentLocationY);
      float differenceZ = Mathf.Abs(targetLocationZ - currentLocationZ);

      if(differenceX > 0.05 && differenceX < 0.2)
      {
        movementLenghtX = 0.2f;
      }
      if(differenceY > 0.05 && differenceY < 0.2)
      {
        movementLenghtY = 0.2f;
      }
      if(differenceZ > 0.05 && differenceZ < 0.2)
      {
        movementLenghtZ = 0.2f;
```

```
      }

      if(differenceX < 0.05)
      {
        movementLenghtX = 0.02f;
      }
      if(differenceY < 0.05)
      {
        movementLenghtY = 0.02f;
      }
      if(differenceZ < 0.05)
      {
        movementLenghtZ = 0.02f;
      }

      if(distanceToTarget < lastDiffernceDistance )
      {
        lillteReward++;
        AddReward(0.002f);
        //EndEpisode();
      }
      if(lillteReward == 500)
      {
        //AddReward(-.5f);
        EndEpisode();
      }
      if(distanceToTarget > lastDiffernceDistance )
      {
        lillteReward--;
        AddReward(-0.002f);
        //EndEpisode();
      }

      if(lillteReward == -250)
      {
        //AddReward(-1.0f);
        EndEpisode();
      }

      /*if(lastDifferenceX > differenceX && lastDifferenceY > differenceY &&
lastDifferenceZ > differenceZ && distanceToTarget < lastDiffernceDistance)
      {
        AddReward(0.005f);
      }
          else{
                AddReward(-0.005f);
          }*/

      if(currentLocationX > 1.2f || currentLocationX < -1.2f ||
currentLocationY > 1.2f || currentLocationY < -1.2f || currentLocationZ >
1.2f || currentLocationZ < -1.2f)
      {
        AddReward(-1.0f);
        EndEpisode();
      }

      if(distanceToTarget <= 0.2)
      {
        AddReward(0.001f);
```

```
}
if(distanceToTarget <= 0.1)
{
  AddReward(0.002f);
}
if(distanceToTarget <= 0.05)
{
  AddReward(.003f);
}
if(distanceToTarget <= 0.02)
{
  AddReward(.004f);
}
if(distanceToTarget <= 0.01)
{
  AddReward(0.005f);
}

if (distanceToTarget <= 0.002)
{
    solve ++;
    AddReward(1.0f);
    EndEpisode();
}

if(differenceX <= 0.005)
{
  AddReward(0.05f);
}
if(differenceY <= 0.005)
{
  AddReward(0.05f);
}
if(differenceZ <= 0.005)
{
  AddReward(0.05f);
}



/*if(lastDifferenceX > differenceX)
{
  AddReward(0.001f);
}
if(lastDifferenceY > differenceY)
{
  AddReward(0.001f);
}
if(lastDifferenceZ > differenceZ)
{
  AddReward(0.001f);
}

if(lastDifferenceX < differenceX)
{
  AddReward(-0.001f);
}
if(lastDifferenceY < differenceY)
{
```

```
    AddReward(-0.001f);
  }
  if(lastDifferenceZ < differenceZ)
  {
    AddReward(-0.001f);
  }*/



  lastDifferenceDistance = distanceToTarget;
  lastDifferenceX = differenceX;
  lastDifferenceY = differenceY;
  lastDifferenceZ = differenceZ;
  //Debug.Log("Solve counter: " + solve);
}

public override void CollectObservations(VectorSensor sensor)
{
  sensor.AddObservation(sphere.localPosition);
  sensor.AddObservation(cube.localPosition);
  //sensor.AddObservation(targetLocation);
  /*sensor.AddObservation(targetLocationX);
  sensor.AddObservation(targetLocationY);
  sensor.AddObservation(targetLocationZ);
  //sensor.AddObservation(currentLocation);
  sensor.AddObservation(currentLocationX);
  sensor.AddObservation(currentLocationY);
  sensor.AddObservation(currentLocationZ);*/
}


public override void Heuristic(float[] actionsOut)
{
    actionsOut[0] = 0;
    if(Input.GetKey(KeyCode.UpArrow))
    {
      actionsOut[0] = 1;
    }

    if(Input.GetKey(KeyCode.DownArrow))
    {
      actionsOut[0] = 2;
    }

    if(Input.GetKey(KeyCode.RightArrow))
    {
      actionsOut[0] = 3;
    }

    if(Input.GetKey(KeyCode.LeftArrow))
    {
      actionsOut[0] = 4;
    }

    if(Input.GetKey(KeyCode.A))
    {
      actionsOut[0] = 5;
    }
```

```
        if(Input.GetKey(KeyCode.D))
        {
          actionsOut[0] = 6;
        }
        // X and Y
        if(Input.GetKey(KeyCode.UpArrow) && Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0]=7;
        }

        if(Input.GetKey(KeyCode.DownArrow) &&
Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0]=8;
        }

        if(Input.GetKey(KeyCode.UpArrow) && Input.GetKey(KeyCode.LeftArrow))
        {
          actionsOut[0]=9;
        }

        if(Input.GetKey(KeyCode.DownArrow)&& Input.GetKey(KeyCode.LeftArrow))
        {
          actionsOut[0]=10;
        }
        // x and Z
        if(Input.GetKey(KeyCode.A) && Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0]=11;
        }

        if(Input.GetKey(KeyCode.D) && Input.GetKey(KeyCode.RightArrow))
        {
          actionsOut[0]=12;
        }

        if(Input.GetKey(KeyCode.A) && Input.GetKey(KeyCode.LeftArrow))
        {
          actionsOut[0]=13;
        }

        if(Input.GetKey(KeyCode.D) && Input.GetKey(KeyCode.LeftArrow))
        {
          actionsOut[0]=14;
        }

        // Y and Z

        if(Input.GetKey(KeyCode.A) && Input.GetKey(KeyCode.UpArrow))
        {
          actionsOut[0]=15;
        }

        if(Input.GetKey(KeyCode.D) && Input.GetKey(KeyCode.UpArrow))
        {
          actionsOut[0]=16;
        }

        if(Input.GetKey(KeyCode.A) && Input.GetKey(KeyCode.DownArrow))
```

```
        {
          actionsOut[0]=17;
        }

        if(Input.GetKey(KeyCode.D) && Input.GetKey(KeyCode.DownArrow))
        {
          actionsOut[0]=18;
        }

        // XYZ
        if(Input.GetKey(KeyCode.RightArrow) && Input.GetKey(KeyCode.UpArrow)
&& Input.GetKey(KeyCode.A) )
        {
          actionsOut[0]=19;
        }

        // -XYZ
        if(Input.GetKey(KeyCode.LeftArrow) && Input.GetKey(KeyCode.UpArrow)
&& Input.GetKey(KeyCode.A) )
        {
          actionsOut[0]=20;
        }

        // X-YZ
        if(Input.GetKey(KeyCode.RightArrow) &&
Input.GetKey(KeyCode.DownArrow) && Input.GetKey(KeyCode.A) )
        {
          actionsOut[0]=21;
        }

        // -X-YZ
        if(Input.GetKey(KeyCode.LeftArrow) && Input.GetKey(KeyCode.DownArrow)
&& Input.GetKey(KeyCode.A) )
        {
          actionsOut[0]=22;
        }

        // XY-Z
        if(Input.GetKey(KeyCode.RightArrow) && Input.GetKey(KeyCode.UpArrow)
&& Input.GetKey(KeyCode.D) )
        {
          actionsOut[0]=23;
        }

        // -XY-Z
        if(Input.GetKey(KeyCode.LeftArrow) && Input.GetKey(KeyCode.UpArrow)
&& Input.GetKey(KeyCode.D) )
        {
          actionsOut[0]=24;
        }

        // X-Y-Z
        if(Input.GetKey(KeyCode.RightArrow) &&
Input.GetKey(KeyCode.DownArrow) && Input.GetKey(KeyCode.D) )
        {
          actionsOut[0]=25;
        }

        // -X-Y-Z
```

```
        if(Input.GetKey(KeyCode.LeftArrow) && Input.GetKey(KeyCode.DownArrow)
&& Input.GetKey(KeyCode.D) )
        {
          actionsOut[0]=26;
        }
    }
}
```