Jesse Bové

Robottikäden suora- ja käänteiskinematiikan ratkaiseminen ja simulointi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Konetekniikka

Insinöörityö

20.5.2025

Tiivistelmä

Tekijä:	Jesse Bové
Otsikko:	Robottikäden suora- ja käänteiskinematiikan ratkaiseminen ja simulointi
Sivumäärä:	22 sivua + 2 liitettä
Aika:	20.5.2025
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Konetekniikka
Ammatillinen pääaine:	Koneautomaatio
Ohjaajat:	Lehtori Maria Sjöholm

Insinöörityö tehtiin osana Metropolia Ammattikorkeakoulun Robo Garagen Metrover-projektia. Työn tavoitteena oli luoda ja ohjata simulaatiomalli robottikädestä, jota voitaisiin hyödyntää projektissa myös tulevaisuudessa.

Työn alussa vertailtiin neljää eri simulaatio-ohjelmaa, josta yksi valittiin työhön. Simulaatioon tarvittavat 3D-mallit luotiin SOLIDWORKS CAD -mallinnusohjelman avulla. Simulaatio luotiin käyttäen NVIDIA:n Isaac Sim -ohjelmistoa. Työssä perehdyttiin robotiikan suora- ja käänteiskinematiikkaan, ja siihen tarvittaviin matriisilaskumenetelmiin. Näiden laskumenetelmien avulla selvitettiin tässä työssä tehdyn robottikäden suora- ja käänteiskinematiikka.

Työn tulokseksi saatiin toimiva simulaatio sekä suora- ja käänteiskinematiikan laskuri, jolla voidaan laskea liikkeet robottikädelle. Lisäksi reflektoitiin projektissa olleet haasteet sekä suunniteltiin mahdollista jatkokehitystä.

Avainsanat: Suorakinematiikka, käänteiskinematiikka, simulointi

---

Abstract

Author:	Jesse Bové
Title:	Forward and Inverse Kinematics and Simulation of a Robot Arm
Number of Pages:	22 pages + 2 appendices
Date:	20 May 2025
Degree:	Bachelor of Engineering
Degree Programme:	Mechanical Engineering
Professional Major:	Machine Automation
Supervisors:	Maria Sjöholm, Senior Lecturer

This project was completed as part of the Metrover-project, an ongoing project within the Robo Garage of Metropolia University of Applied Sciences. The goal of the project was to create and control a simulation of a robot arm that could be used within the Metrover-project in the future.

At the start of this project, four different simulation software were compared, one of which was chosen to be used. The 3D models used within the project were modeled with SOLIDWORKS CAD. The simulation was created in NVIDIA's Isaac Sim. During the project, forward and inverse kinematics of robotics were studied, as well as the necessary matrix calculations that it requires. Using these calculations, the forward and inverse kinematics for the robot arm created in this project were defined.

The final product of the project was a working simulation and a calculator that can compute the forward and inverse kinematics of the robot arm. The challenges of the project as well as possible future avenues are also reflected upon in the thesis.

Keywords: Forward kinematics, inverse kinematics, simulation

Sisällys

1 Johdanto

Tässä insinöörityössä luodaan simulaatiomalli robottikädestä sekä tehdään simulaatiolle ohjaus. Luodaan myös koodi, jolla voidaan laskea ja määrittää robotin liikkeet. Työn aluksi vertaillaan neljää eri simulaatio-ohjelmistoa ennen kuin valitaan tähän työhön sopivaa. Luodaan yksinkertaiset 3D-mallit ja tehdään takaisinkytkentäohjaus simulaatiosta. Työssä tutustutaan myös robotiikkaan tutkimalla suoraa- ja käänteiskinematiikkaa ja luodaan laskumenetelmät tämän työn simulaatiomalliin.

Tämä insinöörityö tehtiin osana Metropolian Robo Gargaessa olevaa jatkuvaa opiskelijaprojektia, jossa tutkitaan, suunnitellaan ja rakennetaan mobiilirobottia. Projektin nimeksi on annettu Metrover. Projektissa suunniteltu robotti on tarkoitus olla kuudella pyörällä kulkeva ja sen päälle ollaan asentamassa robottikäsi, jolla voidaan vaikuttaa ympäristössä oleviin asioihin.

Robo Garage on laboratoriotila Metropolian Myyrmäen kampuksella, joka perustettiin vuonna 2018. Tilassa opiskelijat voivat tehdä kurssi- ja harrastusprojekteja liittyen robotiikkaan. Metrover on syntynyt halusta lähteä maailmalle kisaamaan muiden yliopistojen kanssa jokavuotisessa tapahtumassa, jossa koetellaan samantyyppisten robottien toimivuutta eri tilanteissa. Tämä jokavuotinen kisa on nimeltään European Rover Challenge (ERC) ja se järjestetään Krakówassa, Puolassa.

2 Teoria

Tässä työssä tutkitaan robotiikan perusteita. Tähän kuuluu suora- ja käänteiskinematiikkaa sekä takaisinkytkentää. Teoriaosiossa perehdytään matriiseihin ja muutamiin niihin liittyviin laskentamenetelmiin, joita tullaan myöhemmin käyttämään suora- ja käänteiskinematiikan ohjelmoinnissa. Takaisinkytkentää hyödynnetään simulaation ohjauksen luomisessa.

2.1 Matriisilaskenta

Matriisi on käsite, jonka avulla voidaan suorittaa vaativampia laskutehtäviä yksittäisinä toimintoina (Launonen ym. 2006: 11). Reaali- tai kompleksiluvuista muodostettu, vaaka- ja pystyriveiksi järjestettyä lukujoukkoa kutsutaan matriisiksi (Launonen ym. 2006: 13).

Kun matriiseja käsitellään, tarkastellaan niiden pysty- ja vaakarivejä. Näitä kutsutaan myös nimityksellä rivit (vaakarivit) ja sarakkeet (pystyrivit). Rivien lukumääränä käytetään *p* ja sarakkeiden lukumääränä käytetään *n*. Tällöin voidaan määrittää matriisin kokoa *p* x *n*.

Tässä työssä käsitellään kahta erikokoista matriisia. Suorassa kinematiikassa hyödynnetään 4 x 4 matriisia. Käänteisessä kinematiikassa hyödynnetään 3 x 5 matriisia.

2.1.1 Matriisin peruskäsitteitä

Neliömatriisi on matriisi, jossa on sama määrä rivejä ja sarakkeita. Matriisin ylävasemmasta paikasta alaoikeaan paikkaan olevat alkiot, muodostavat neliömatriisin päälävistäjä.

Matriisi, jonka kaikki alkiot ovat nollia, sanotaan nollamatriisiksi (Launonen ym. 2006: 15).

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Neliömatriisi, jonka päälävistäjällä olevat alkiot ovat kaikki ykkösiä ja muut alkiot ovat nollia, sanotaan yksikkömatriisiksi (Launonen ym. 2006: 15).

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

2.1.2 Matriisitulo

Jotta matriisitulo olisi mahdollinen, minimissään yhden matriisin sarakkeiden määrä on oltava sama kuin toisen matriisin rivien määrä. Esimerkkimatriisit voisivat olla *p* x *n* ja *n* x *q*. Tällöin matriisitulo on mahdollinen ja tulon tulos on *p* x *q* -kokoinen matriisi.

Kaavat (1) ja (2) havainnollistavat, miten matriisitulo lasketaan.

$$\begin{bmatrix} \vdots & & & \\ \vdots & & & \\ a_{i1} & a_{i2} & \dots & a_{in} \\ \vdots & & & \end{bmatrix} \begin{bmatrix} \dots & b_{1j} & \dots \\ & b_{2j} & \\ & \vdots & \\ & b_{nj} & \end{bmatrix} = \begin{bmatrix} & \vdots & \\ & \vdots & \\ \dots & c_{ij} & \dots \\ & \vdots & \end{bmatrix} \tag{1}$$

$$a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj} = c_{ij} \tag{2}$$

2.1.3 Käänteismatriisi

Tutkittava matriisi *A* on neliömatriisi. Jos on olemassa matriisi *B* siten että

$$AB = BA = I \tag{3}$$

, jossa *I* on yksikkömatriisi, niin *B* on *A*:n käänteismatriisi.

2.1.4 Pseudokäänteismatriisi

Matriisille, joka on muotoa *p* x *n*, on olemassa pseudokäänteismatriisi, jonka muotoa on *n* x *p*, kun se täyttää Moore-Penrose-ehdot. Tässä työssä ei syvennytä näihin ehtoihin, mutta tätä pseudokäänteismatriisia sovelletaan myöhemmin työssä.

2.2 Koordinaatistot

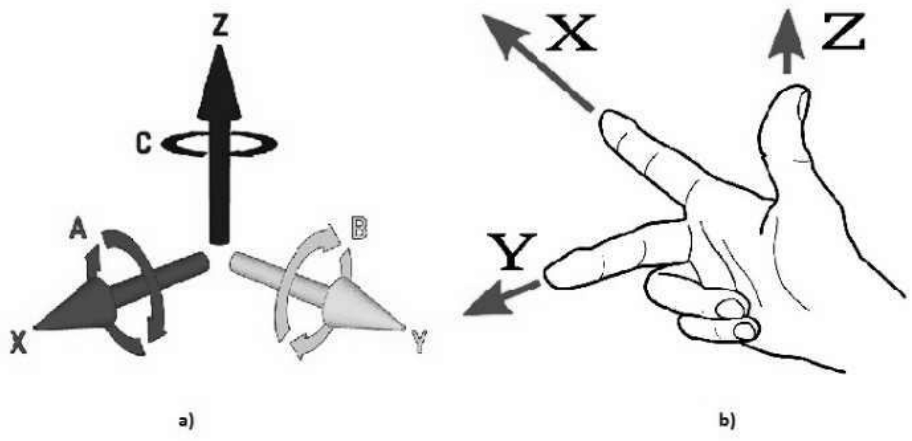
Kinematiikassa voidaan hyödyntää erilaisia koordinaatistoja. Tässä työssä hyödynnetään Cartesian ja Jacobin koordinaatistoja.

2.2.1 Cartesian koordinaatit

Kappaleen paikkaa avaruudessa määritellään Cartesian koordinaateilla (McKerrow, 1991: 134).

Kappaleen avaruutta voidaan määrittää oikean käden säännöllä. Tällä säännöllä määritellään kappaleen koordinaatiston positiiviset suunnat käyttäen oikean käden sormia.

Osoita etusormi eteenpäin, osoita keksisormi 90 asteen kulmaan etusormesta ja osoita peukalo suoraan ylöspäin. Kuvitellaan, että etusormi on x-akseli, keksisormi on y-akseli ja peukalo on z-akseli. Kuvassa 1 b-puolella nähdään miten tätä havainnollistaan.



Kuva 1. Oikean käden sääntö

Näiden koordinaattien kautta voidaan tutkia myös kappaleen orientaatiota eli suuntaa avaruudessa. Kuvassa 1 a-puolella voidaan havainnollistaa tätä.

Kappaleen roll, pitch ja yaw on sen rotaatiota x-, y- tai z-akselin suuntaan. Kuvassa roll, pitch ja yaw ovat merkitty A, B ja C.

2.2.2 Jacobin koordinaatit

Jacobin koordinaatit kuvastavat robotin akselien kulmamuutosten aiheuttamaa loppupään sijainnin ja nopeuden muutosta. Jacobin koordinaatit kuvastetaan usein matriisina. Matriisin sarakkeiden määrä kuvastaa, kuinka monta akselia robotilla on ja rivien määrä kuvastaa, kuinka monta vapausastetta robotilla on.

Tässä työssä tutkitaan viiden akselin robottia, jolla on mahdollisuus liikkua kuuden vapausasteen suuntaisesti. Tällöin siitä muodostettaisiin seuraavanlainen 5 x 6 -matriisi (kaava 4):

$$J = \begin{bmatrix} d_{x1} & d_{x2} & d_{x3} & d_{x4} & d_{x5} \\ d_{y1} & d_{y2} & d_{y3} & d_{y4} & d_{y5} \\ d_{z1} & d_{z2} & d_{z3} & d_{z4} & d_{z5} \\ \delta_{x1} & \delta_{x2} & \delta_{x3} & \delta_{x4} & \delta_{x5} \\ \delta_{y1} & \delta_{y2} & \delta_{y3} & \delta_{y4} & \delta_{y5} \\ \delta_{z1} & \delta_{z2} & \delta_{z3} & \delta_{z4} & \delta_{z5} \end{bmatrix} \quad (4)$$

Matriisin arvot edustavat jokaiselle akselille niiden Cartesian koordinaatit. Ylemmät kolme riviä kertovat jokaiselle akselille niiden sijainnin ja alemmat kolme riviä kertovat suunnan.

Tässä työssä ei tarkastella suuntaa, joten myöhemmin hyödynnetään Jacobin koordinaatteja varten 3 x 5 -matriisia, jossa huomioidaan vain kolme ylintä riviä yllä mainitusta matriisista.

2.3 Kinematiikka

Kinematiikka on toimijan linkin suhde paikkaan, nopeuteen ja kiihtyvyyteen, jossa toimija on käsi, sormi tai jalka (McKerrow 1991: 176). Linkki on se osa robottia, joka yhdistää kaksi akselia toisiinsa. Esimerkki linkistä ja akselista ihmiskehossa on kyynärvarsi ja kyynärpää.

Kinematiikassa on kaksi erillistä ongelmaa ratkaistavana. Suorakinematiikka ja käänteiskinematiikka. Jokaiselle robottikädelle on oma suora- ja käänteiskinematiikka. Eli jos yhdelle robotille on selvitetty kinematiikat, se ei takaa, että se soveltuu toiselle robotille.

2.3.1 Suorakinematiikka

Suorakinematiikassa hyödynnetään yleistä A-matriisia, joka on yhdistelmä paikka- ja rotaatiomatriiseja. Nämä paikka- ja rotaatiomatriisit määrittävät linkin suunnan ja sijainnin avaruudessa. Alla oleva kaava (5) on tämä yleinen A-matriisi ja sen alla oleva taulukko 1 selventää, mitä matriisin eri muuttujat edustavat. Kun tunnetaan kaikki tarvittavat muuttujat, voidaan laskemalla selvittää linkin suunta ja sijainti.

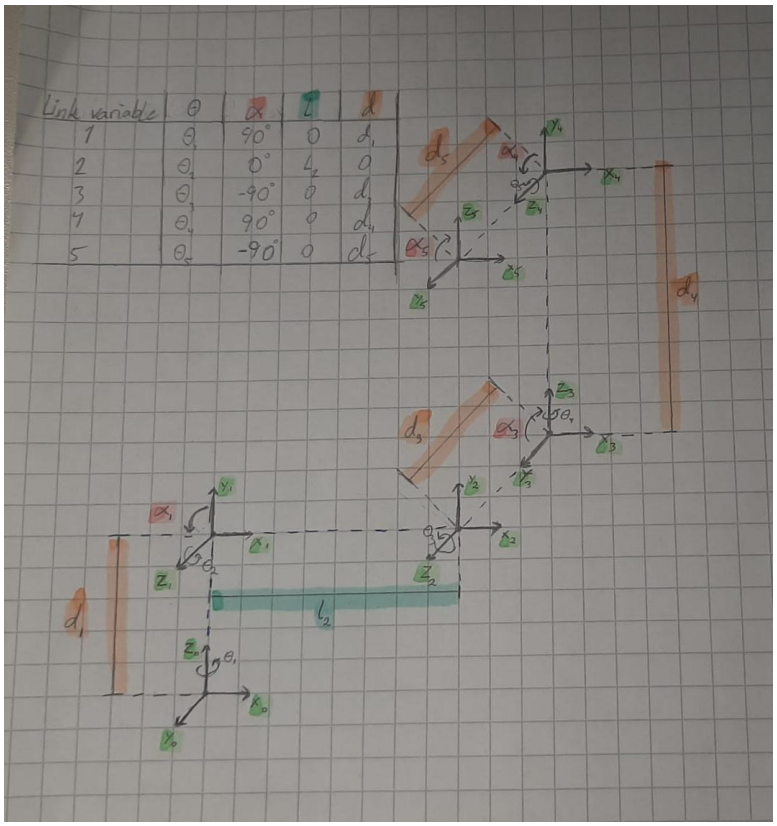
$$\begin{bmatrix} \cos(\theta) & -\sin(\theta)\cos(\alpha) & \sin(\theta)\sin(\alpha) & lcos(\theta) \\ \sin(\theta) & \cos(\theta)\cos(\alpha) & -\cos(\theta)\sin(\alpha) & lsin(\theta) \\ 0 & \sin(\alpha) & \cos(\alpha) & d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

Taulukko 1. Kaavan (5) muuttujien selitykset.

Tunnus	**Selitys**
θ	Akselin kulmamuutos z-akselin suhteen
α	Akselin orientaatiomuutos aikaisempaan akseliin nähden, x-akselin suhteen
l	Akselin paikkamuutos aikaisempaan akseliin nähden, x-akselin suuntaan
d	Akselin paikkamuutos aikaisempaan akseliin nähden, z-akselin suuntaan

A matriisin ensimmäiset kolme saraketta ja kolme riviä määräävät linkin päätypisteen orientaation aikaisempaan akseliin nähden. A-matriisin viimeisen sarakkeen kolme ylintä arvoa määräävät linkin päätypisteen sijainnin avaruudessa. Jos halutaan selvittää linkkiketjun loppupäädyn suunta ja sijainti, voidaan suorittaa matriisitulo useasta A-matriisista.

Kuvassa 2 nähdään tässä työssä tehdyn robotin kinemaattinen kaavio. Kuvassa 2 olevassa taulukossa on lueteltu jokaiselle akselille omat arvot. Kuvassa 2 oleva koordinaatistoketju edustaa sitä linkkiketjua, josta työn robotti koostuu. Siinä havainnollistetaan koordinaatistojen suunnan ja paikan muutokset.



Kuva 2. Kinemaattinen kaavio

Suorakinematiikassa on tärkeää huomioda se, että kaikki pyöriväliike suoritetaan z-akselin ympäri. Tällöin tehdyt orientaatiomuutokset ovat tärkeitä huomioida.

2.3.2 Käänteiskinematiikka

Käänteiskinematiikka on tapa tutkia robotiikkaa. Usein käänteiskinematiikan tavoite on selvittää, mitkä Jacobin koordinaatin arvot tarvitaan saavuttaakseen toivottua Cartesian koordinaatin suuntaa ja sijaintia. Jacobin koordinaatit voidaan selvittää laskennallisesti tai iteroivilla arvauksilla. Useissa käänteiskinematiikan

laskuissa Jacobin koordinaatit eivät ole yksiselitteisiä. Näiden koordinaattien selvittely laskennallisesti muuttuu haastavammaksi, mitä enemmän akseleja robotilla on tutkittavana.

Laskennallisesti selvitetty käänteiskinematiikka vaatii suorakinematiikan A-matriisin käänteismatriisin selvittelyä jokaiselle akselin matriisille. Tässä työssä, jos yritettäisiin laskennallisesti selvittää käänteiskinematiikan, se vaatisi viittä eri käänteismatriisia.

Voidaan hyödyntää iteroivaa arvausta, jossa muutetaan nykyiset kulma-arvot ja tutkitaan, mihin suuntaan robotin loppupää liikkuu. Tätä iteroivaa arvausta voidaan tehdä käsin, mutta useammin siinä käytetään ohjelmointia apuna. Tässä työssä hyödynnetään iteroivaa arvausta, koska robotilla on tarpeeksi monta akselia, jolloin laskennallisesti selvittäminen olisi liian vaativa.

2.4 Takaisinkytkentä

Ohjauksessa yleisesti käytetty tekniikka on takaisinkytkentä. Siinä määritellään arvo, joka halutaan saavuttaa (ohjearvo), mitataan nykyistä tilannetta (oloarvo) ja verrataan näiden kahden arvon eroa. Jos arvot eivät täsmää, suoritetaan toimintoa, jolla oloarvoa koitetaan saada lähemmäs ohjearvoa.

Yksinkertainen esimerkki takaisinkytkennästä on mekaaninen termostaatti. Ohjearvoksi annetaan toivottu lämpötila huoneelle. Termostaatti mittaa huoneen lämpötilaa, eli oloarvoa. Jos mitattu lämpötila on alle toivotun lämpötilan, laitetaan lämmöt päälle. Jos mitattu lämpötila on sama tai suurempi kuin toivottu lämpötila, laitetaan lämmöt pois päältä.

3 Ohjelmistot

3.1 Simulaatio-ohjelmiston valinta

Työn alussa vertailtiin neljää simulaatio-ohjelmistoa; Siemens NX, Matlab Simulink, NVIDIA Isaac Sim ja Unity. Vertailussa huomioitiin ohjelmiston kattavuus, tuttavuus, saatavuus ja kyky käyttää ulkoista koodia.

Kattavuudella tarkoitetaan, onko ohjelmistolla tarpeeksi työkaluja, jotta simulaation luominen olisi mutkatonta. Tuttavuudella tarkoitetaan, onko ohjelmisto jo entuudestaan tuttu minulle työn tekijänä. Saatavuudella tarkoitetaan, vaatiiko ohjelmisto lisenssiä vai ei. Jos ohjelmisto ei vaadi lisenssiä, se nähtiin positiivisena asiana, sillä sitä olisi mahdollisuus jatkokehittää valmistumisen jälkeen. Kyky käyttää ulkoista koodia halutaan, jotta voidaan suorittaa suora- ja käänteiskinematiikan laskuja toisessa ohjelmistossa ja tuoda nämä laskut suoraan simulaatioon.

NX:n hyvät puolet olivat ne, että ohjelmisto oli jo tuttu. Siinä on kattavat työkalut sekä Myyrmäen kampukselta löytyi paljon ohjelmiston osaavia, joten oli mahdollisuus saada opastusta. Ohjelmisto on myös hyvin kattava, sillä siinä on jo suoran- ja käänteiskinematiikan ohjausta. Sen huonot puolet olivat ne, että ohjelmisto vaatii lisenssin ja ulkoisen koodin liittäminen ei ollut suoraan mahdollista.

Simulinkin hyvät puolet olivat ne, että ohjelmisto oli jo tuttu, Myyrmäen kampukselta löytyi ohjelmiston osaavia ja ohjelmisto suostuu käyttämään ulkoista koodia. Sen huonot puolet olivat ne, että ohjelmisto vaatii lisenssin ja siinä ei ole tarpeeksi kattavat työkalut.

Isaac Simin hyvät puolet olivat ne, että ohjelmisto on täysin ilmainen, on hyvin kattava työkalujen suhteen ja ohjelmisto voi hyödyntää ulkoista koodia. Sen huonot puolet olivat ne, että ohjelmisto ei ollut laisinkaan tuttu.

Unityn hyvät puolet olivat ne, että ohjelmisto on täysin ilmainen, Arabian kampukselta löytyi ohjelmiston osaavia ja ohjelmisto suostuu käyttämään ulkoista

koodia. Sen huonot puolet olivat ne, että ohjelmisto ei ole tuttu eikä se ole tarkoitettu robotiikkaan, vaan suuntaa enemmän pelifysiikan suuntaan.

Alla olevassa taulukko 2:ssa on havainnollistettu nämä hyvät ja huonot puolet jokaisesta ohjelmistosta. Taulukon plusmerkki edustaa hyvää piirrettä ja miinusmerkki edustaa huonoa piirrettä.

Taulukko 2. Vertailutaulukko neljästä ohjelmistosta

	Siemens NX	**Matlab Simulink**	**NVIDIA Isaac Sim**	**Unity**
Kattava	+	-	+	-
Tuttu	+	+	-	-
Saatavuus	-	-	+	+
Ulkoisen koodin hyödyntäminen	-	+	+	+

Vertailussa ensimmäinen ohjelmisto, joka tippui kyydistä, oli Unity. Sen hyödyt suuntautuvat enemmän pelien kehitysten suuntaan, eikä ollut tähän projektiin tarpeellisia. Toisena tippui Simulink. Sitä ei nähty yhtä kattavana kuin muut vaihtoehdot. Lopuksi jäi NX ja Isaac Sim. Päätettiin tutustua Isaac Simin ohjelmistoon yhden viikon ajan ennen lopullisen valinnan tekemistä.

Tutustumisviikon aikana huomattiin, että Isaac Sim oli tähän työhön oiva ohjelmisto. Vaikka ohjelmisto ei ollut tuttu, siitä löytyi tarpeeksi opastusvideoita ja tietoa.

3.2 SOLIDWORKS CAD

Työssä käytettiin Dassault Systemes:n SOLIDWORKS CAD -ohjelmistoa. Tällä ohjelmistolla luotiin simulaation 3D-mallit, jotka tuotiin simulaatio-ohjelmistoon.

Tämä ohjelmisto valittiin, koska se oli valmiiksi usealla laitoksen tietokoneilla ja se oli minulle tutuin mallinnusohjelmisto.

3.3 VScode

Työssä käytettiin VScode-ohjelmointisoftaa. Softassa on mahdollista käyttää monia eri koodikieliä virtuaaliympäristössä. Ohjelmisto valittiin siitä syystä, että AIoT Garagen henkilökunta suosi sitä labran projekteissa.

4 Simulointi

4.1 Simulaation mallit

Simulaatiota varten laadittiin yksinkertaisia 3D-malleja. Mallien mitat valittiin mielivaltaisesti, mutta huomioiden Metroverin lopulliset dimensiot. Alla olevaan taulukkoon (Taulukko 3) on kerätty osien ulkomitat.

Taulukko 3. 3D-mallien mitat taulukoituna.

Osan nimi	**Pituus (mm)**	**Leveys (mm)**	**Korkeus (mm)**
Base	600	300	340
Shoulder Joint	140	140	110
Shoulder Link	480	80	80
Elbow Joint	80	80	110
Elbow Link	490	80	80
Hand	80	80	160

Base on palikkamainen kappale. Sen päälle mallinnettiin ympyränmuotoinen koroke, jotta seuraavalla osalla olisi visuaalinen liitoskohta simulaation rakentamisessa.

Shoulder Joint on ympyräpohjainen kappale. Siihen mallinnettiin u-malliset liitoskohdat, jotta olisi visuaalinen liitoskohta seuraavaan osaan.

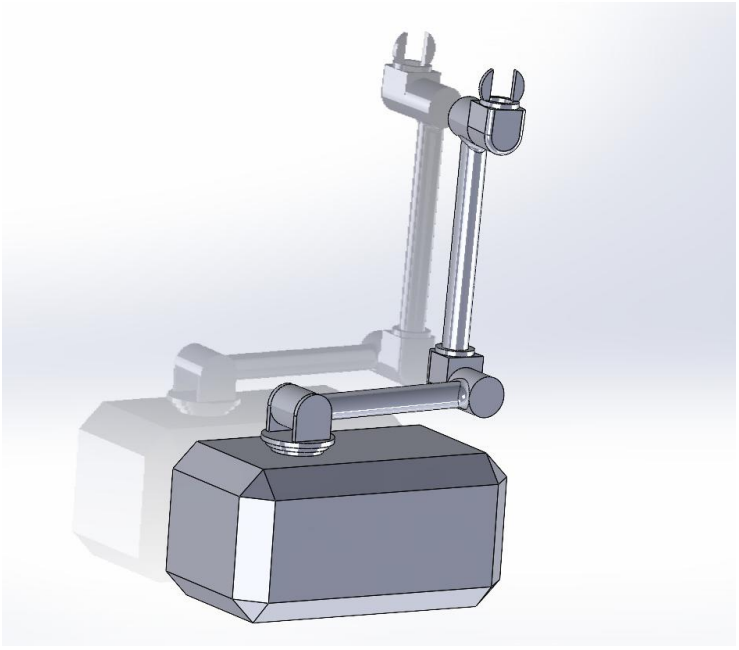
Shoulder Link on putkimainen osa. Sen keskikohta on putki, jonka molemmissa päissä on 90-asteenkulmassa lieriöt.

Elbow Joint on u-mallinen kappale. Siihen mallinnettiin ympyrämäinen liitoskohta, jonka tarkoitus on visualisoida liitosta seuraavaan osaan.

Elbow Link on putkimainen osa. Putken toisessa päädyssä mallinnettiin 90-asteen kulmaan lieriö, jonka tarkoitus on visualisoida liitosta seuraavaan osaan.

Hand on u-mallinen kappale. Osaan mallinnettiin rapusaksen näköiset osat visualisoidakseen käden päätypistettä.

Kuvassa 3 näkyy robotin kokoonpano.



Kuva 3. Robotin kokoonpano

4.2 Simulaation rakentaminen

Ensimmäisenä luotiin simulaation pohja. Tässä simulaatiossa käytettiin valmista vihreää ruutupohjaa.

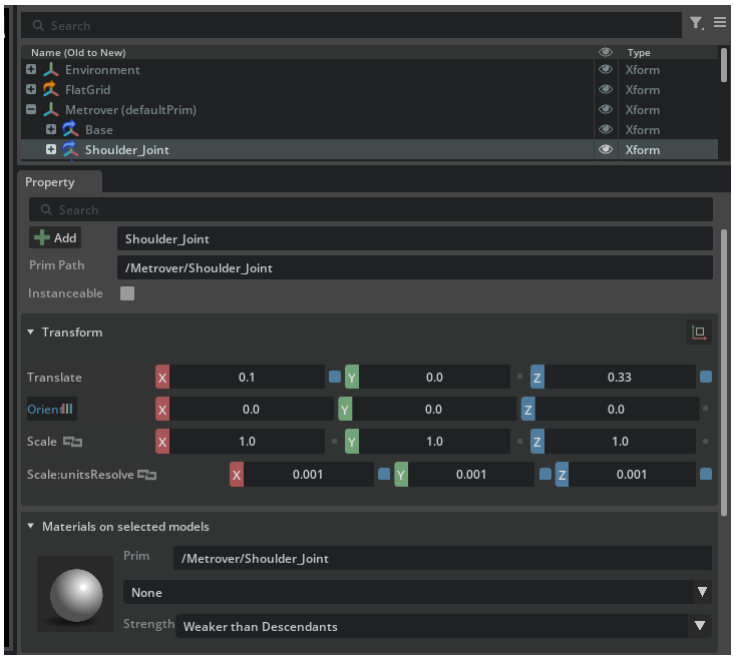
Seuraavaksi luotiin robotille Prim, joka edustaa koko robottia. Tällöin sitä on helpompi käsitellä ohjauksessa, kun laitetaan kaikki robottiin kuuluvat osat tämän Primin alle. Sitten tuotiin osat yksitellen simulaatioon. Toimintajärjestys oli seuraavan luettelon mukainen.

- Tuo osa simulaatioon
- Luo osalle fysiikat
- Määritä osan suunta ja sijainti

Isaac Sim sallii helposti tuoda SOLIDWORKS 3D -mallit simulaatioon. Ohjelmiston alareunassa sijaitsevassa Content-välilehdeltä voi löytää tietokoneen tiedostot. Tältä välilehdeltä käytetään Import-toiminto ja valitaan haluttu tiedosto tuotavaksi simulaatioon.

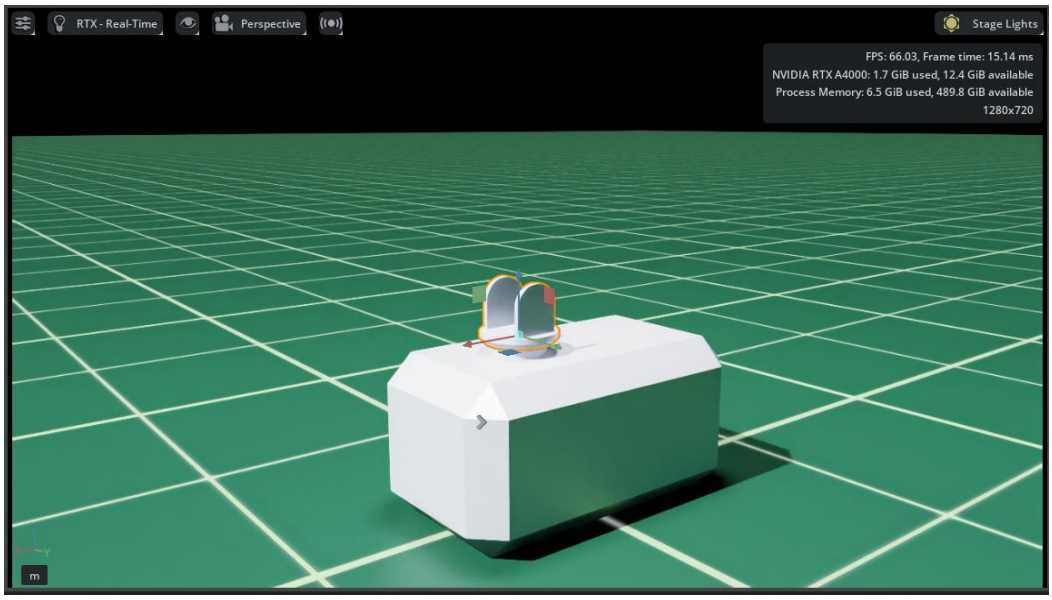
Osien fysiikat luodaan valitsemalla hiiren oikealla painikkeella osaa. Valikosta Add, Physics, Solid Body/Collision Body. Tässä simulaatiossa Collision Body ei ole täysin tarpeellinen jokaisessa osassa. Siitä on jopa haittaa tietyissä kohdissa, joten se laitettiin pois päältä kaikilta osilta, paitsi Base- ja Hand-osille.

Osan suuntaa ja sijaintia, eli orientation & translate, määriteltiin jokaiselle osalle. Osan suunta voidaan muuttaa pyöräyttämällä x-, y- tai z-akselin ympäri. Osien sijainti on aina simulaation nollapisteen suhteen ja on metri skaalassa. Kuvassa 4 nähdään Property-välilehti Shoulder Joint -osalle, jossa määritetään osalle suunta ja sijainti.



Kuva 4. Shoulder Joint -osan Property-välilehti

Kuvassa 5 nähdään simulaatio, jossa on kaksi osaa, Base ja Shoulder Joint, jotka istuvat vihreällä taustalla. Shoulder Joint -osan suunta ja sijainti ovat kuvassa 4 olevien arvojen mukaan sijoitettu.



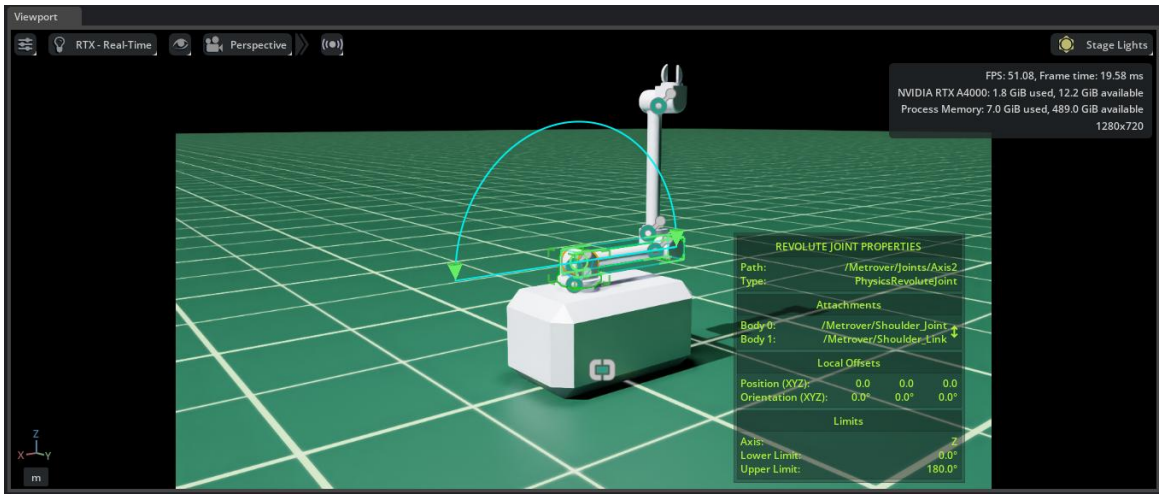
Kuva 5. Simulaatiomallin rakentaminen

Luodaan jokaiselle askelille oma fysiikka nimeltään Revolute Joint. Tämä sallii kappaleen pyöriä määrätyn pisteen ympäri. Samalla tavalla, kun määrätään osien suunta ja sijainti myös tämän nivelen suunta ja sijainti on määrättävä. Nivelen suunta ja sijainti määrätään osien suhteen, ei simulaation keskipisteen suhteen, joten on oltava tarkka, kun määrittää nivelet.

Koska nivel on kahden osan välille luotu, on määritettävä näistä kahdesta osasta se, joka on kiinteä ja se, joka on liikkuva. Kiinteänä osana tarkoitetaan sitä, jonka ympäri pyöritään ja liikkuva on taas se osa, joka pyörii. Simulaatiossa Body 0 on kiinteä osa ja Body 1 on liikkuva osa. Kiinteä osa on tässä työssä aina se osa, joka on lähimpänä Base-osaa linkkiketjussa.

Jotta niveltä voidaan liikuttaa, on lisättävä sille jokin liikuttava voima. Nivelen Property-välilehdellä voidaan lisätä Angular Drive, joka määrittää voiman nivelen ohjaamiseen. On tärkeätä huomata, millä tavalla haluaa ohjata robottia, sillä sen mukaan valitaan nivelelle sen Damping- ja Stiffness-arvot. Näiden tarkoitusta käydään läpi seuraavassa kohdassa tarkemmin.

Kuvassa 6 nähdään Axis 2 -nimisen nivelen visualisointi. Kuvassa 6 esiintyvässä taulukossa nähdään nivelen tyyppi, mitkä osat ovat Body 0 ja Body 1, nivelen suunta ja sijainti simulaation keskipisteen suhteen, sekä mahdollisia liikkeen rajoitteita, joita voisi määritellä. Tälle nivelelle on tehty liikkeen rajoitteet. Sitä voi liikuttaa 0 ja 180 asteen välillä.



Kuva 6. Axis 2 -nivelen visualisointi

4.3 Simulaation testaus

Simulaatiota on mahdollista ohjata käyttäen joko paikka-, nopeus- tai voima-arvoja. Tässä työssä hyödynnettiin paikkaohjausta aluksi testausta varten ja lopuksi nopeusohjausta, kun oli luotu takaisinkytkentä.

Kun simulaatiota halutaan ohjata, on tärkeä tietää, millä ohjauksella halutaan ohjata ja on määriteltävä nivelien Stiffness- ja Damping-arvo sen mukaisesti. Kun halutaan paikkaohjata, nivelelle on laitettava suuri Stiffness-arvo ja pieni Damping. Kun halutaan nopeusohjata, tarvitaan suuri Damping ja pieni Stiffness.

Action Graph -välilehdellä luotiin funktioblokeilla ohjaus. Simulaation ohjausta varten valittiin ohjaamiseen Articulation Control -ohjausblokki. Tässä blokissa määriteetään, mitkä nivelet liikutetaan ja millä ohjearvolla.

Testauksissa ohjattiin paikkaohjearvolla, jolloin simulaation nivelet liikkuivat välittömästi määrättyyn arvoon. Tällöin voitaisiin tarkistaa, toimiko myöhemmin määritetty koodi toivotulla tavalla.

4.4 Simulaation takaisinkytkentä

Testausten jälkeen luotiin takaisinkytkentä käyttäen funktioblokkeja. Funktioblokeilla määrätään simulaatiolle arvoja sen akselien kulmille ja liikutetaan simulaation osia, kunnes saavuttavaan määrätty piste. Simulaatiossa hyödynnetään yhdeksää erilaista funktioblokkia. Alla on luettelo työssä käytetyistä funktioblokeista.

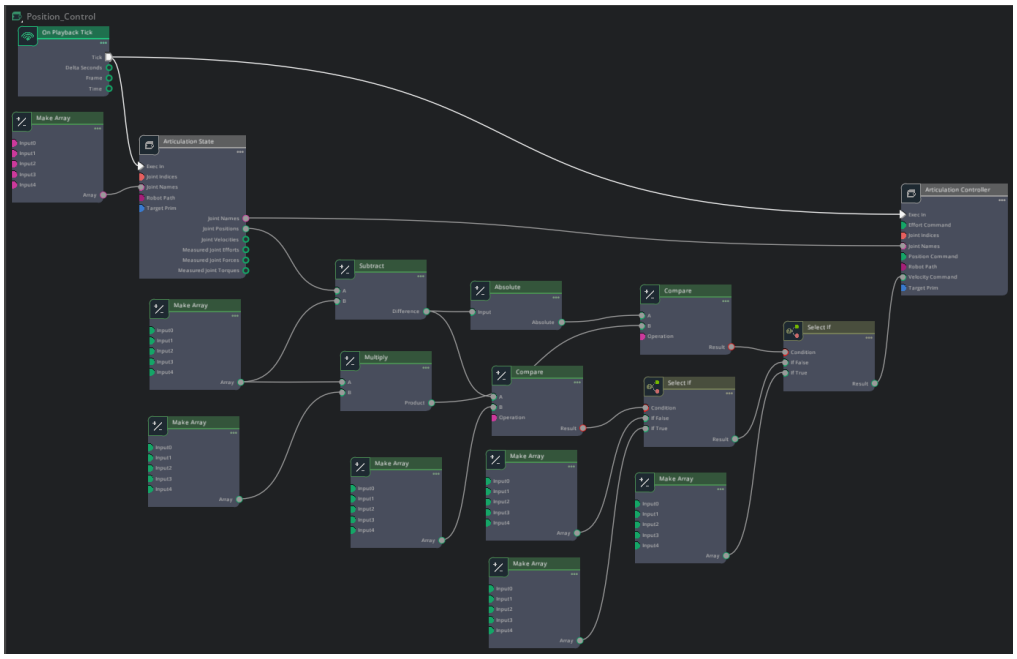
- On Playback Tick
- Make Array
- Subtract
- Absolute
- Multiply
- Compare
- Slecet If
- Articulation State
- Articulation Controller

On Playback Tick määrittää, kuinka usein simulaatio suorittaa tehtävän. Make Array kokoaa useita arvoja yhdeksi tietoketjuksi, jolloin on helpompaa tutkia tietoja seuraavissa funktioblokeissa. Subtract laskee kahden syötetyn arvon erotus. Absolute syöttää annetun arvon absoluuttisen arvon. Multiply kertoo syötetyt arvot yhteen.

Compare vertaa kahta arvoa ja syöttää joko kielteisen (0) tai myönteisen (1) vastauksen. Vertailussa katsotaan, kumpi arvo on suurempi tai pienempi kuin toinen ja syötetään vastaus tämän perusteella. Select If tutkii, onko syötetty vastaus joko 0 tai 1 ja suorittaa toimintoja sen perusteella. Toiminnot määrätään If False ja If True kohdilla.

Articulation State lukee määrätyille akseleille niiden sijainnin, nopeudne, voiman ja väännöt. Tässä työssä hyödynnetään ainoastaan akselien sijaintia. Articulation Controller suorittaa määrätyille akseleille niiden toiminnot, joko paikka- tai nopeusohjauksella.

Kuvassa 7 nähdään, miten näitä funktioblokkeja on hyödynnetty tässä työssä.



Kuva 7. Takaisinkytkennän funktioblokkikaavio

Takaisinkytkentää varten tarvitaan ohjearvo, oloarvo ja toiminto. Ohjearvo on tässä tilanteessa ne akselien kulma-arvot, joita tavoitellaan. Oloarvo on sen hetken kulmien arvot. Toiminto on nopeus, jota määritellään akseleille, kunnes oloarvo saavuttaa ohjearvoa.

Ohjearvot määritellään jokaiselle kulmalle erikseen. Nämä voidaan valita mielivaltaisesti tai voi hyödyntää käänteiskinematiikan koodia, jota myöhemmin on käytetty. Ohjearvot sijoitetaan Make Array -nimiseen funktioblokkiin. Nämä ohjearvot verrataan suoraan oloarvoihin. Oloarvot selvitetään Articulation State -funktioblokista. Toimintoa ohjataan Articulation Controller -funktioblokilla.

Funktioblokkikaaviossa olevat Compare- ja Select If -funktioblokit käytetään määrittämään, liikutetaanko akselit positiiviseen tai negatiiviseen suuntaan tai pysytäänkö paikoillaan. Subtract laskee oloarvon ja ohjearvon erotuksen. Absolute antaa tästä erotuksesta absoluuttisen arvon. Multiply laskee virhearvoa. Tämä arvo määrittää, kuinka lähellä akselin on oltava tavoitepistettä, jotta

liikettä ei enää suoriteta. Tällä on suurempimerkitys, kun ohjausta kehitetään eteenpäin.

5 Ohjelmointi

5.1 Ohjelmointikielen valinta

Työssä tehtävät laskut suora- ja käänteiskinematiikalle vaativat haastavampaa matematiikkaa. Ohjelmointikieleksi valittiin Python, sillä sille on olemassa kattavat koodikirjastot, jotka auttavat suorittamaan näitä matemaattisia laskuja.

Käytetyt kirjastot olivat Math ja NumPy. Math-kirjasto käytetään määrittämään matemaattisia vakioita, kuten pii, ja trigonometrisia funktioita ja kuten sini ja kosini. NumPy-kirjasto käytetään vaativammissa laskuissa. Tässä työssä sitä käytetään matriisilaskujen selvittämiseen.

5.2 Suorakinematiikka

Koodissa tarkastellaan viittä eri matriisia. Jokaisella matriisilla on omat arvot, joita voidaan määrittää koodin alkupäässä. Liitteessä 1 voi nähdä, miten nämä arvot ovat sijoitettu koodiin, sekä miten lainattuihin kirjastoihin on viitattu. Theeta on akselin kulma, alfa on akselin suhde aikaisempaan akseliin, l on etäisyys x-akselin suuntaan aikaisemmasta akselista, d on etäisyys z-akselin suuntaan aikaisemmasta akselista. Alla olevassa esimerkkikoodi 1:ssä nähdään yksittäisen A-matriisin kirjoitusasua. Koodissa oleva m.-osa viittaa käytettyyn Math-kirjastoon.

```
M_1 = [[m.cos(theeta_1), -1*m.sin(theeta_1)*m.cos(alpha_1),
m.sin(theeta_1)*m.sin(alpha_1), l_1*m.cos(theeta_1)],
        [m.sin(theeta_1), m.cos(theeta_1)*m.cos(alpha_1), -
1*m.cos(theeta_1)*m.sin(alpha_1), l_1*m.sin(theeta_1)],
        [0, m.sin(alpha_1), m.cos(alpha_1), d_1],
        [0, 0, 0, 1]]
```

Esimerkkikoodi 1. Python-kielellä kirjoitettu suorakinematiikan A-matriisi.

Jokaiselle akselille tehdään tämän mallinen matriisi, käyttäen jokaiselle akselille omat theeta, alfa, l ja d -arvot. Nämä matriisit kerrotaan yhteen käyttäen NumPy-kirjaston dot-funktiota, joka suorittaa matriisitulolaskun. Alla oleva esimerkkikoodi 2 havainnollistaa, miten tämä lasku on suoritettu tässä työssä. Koodissa oleva np.-osa viittaa käytettyyn NumPy-kirjastoon.

```
M_12 = np.dot(M_1,M_2)
M_123 = np.dot(M_12,M_3)
M_1234 = np.dot(M_123,M_4)
M_12345 = np.dot(M_1234,M_5)
```

Esimerkkikoodi 2. Python-kielellä tehty matriisikertolasku.

Lopuksi koodissa pyydetään tulostamaan näytölle viimeinen matriisitulo. Tätä matriisituloa voidaan tarkastella kuin perinteinen, suorakinematiikan A-matriisi, jossa viimeisen sarakkeen ensimmäiset kolme arvoa on loppupisteen x-, y- ja z-koordinaatit.

5.3 Käänteiskinematiikka

Koodissa hyödynnetään aikaisemmin määriteltyä suorakinematiikan laskua, sekä samat kirjastot ja osa samoista vakioista. Liitteessä 2 nähdään, että koodin alku muistuttaa suorakinematiikan koodia. Koodin alussa määritellään toivottua loppupistettä (target_position) ja lähtötilanteen kulmia (initial_angles). Koodissa on neljä funktiota, joita käytetään selvittämään tarvittavat kulmat jokaiselle akselille, jotta saavutetaan tietyn Cartesian koordinaatti.

Funktio forward_kinematics hyödyntää aiemmin selvitettyä suorakinematiikan koodia ja tuottaa lasketun A-matriisin x-, y- ja z-koordinaatit.

Funktiossa compute_jacobian luodaan ja tutkitaan Jacobin matriisi, jolla suoritetaan käänteiskinematiikka. Funktiossa luodaan 3 x 5 -matriisi, jolla tutkitaan ainoastaan robotin x-, y- ja z-koordinaatit. Lisäksi määritellään myös pieni numeerinen arvo. Tällä arvolla tutkitaan jokaiselle akselille, miten robotti liikkuu, kun liikutetaan akseleja positiiviseen tai negatiiviseen suuntaan.

Funktio wrap_to_range määrittää sen, että koodissa selvitetyt kulmat eivät ole suurempia kuin 360 astetta tai 2pi radiaania.

Funktio inverse_kinematics hyödyntää kaikkia muita funktioita. Haetaan ensin lähtötilanne, joka voidaan selvittää forward_kinematics -funktiosta. Verrataan toivottuun tilanteeseen löytääkseen näiden pisteiden ero. Lasketaan Jacobin avulla, miten on muutettava akseleja, jotta tämä ero pienentyisi. Laskua toistetaan, kunnes ero on tarpeeksi pieni tai on saavutettu laskujen enimmäismäärän.

6 Yhteenveto

6.1 Tulokset

Työssä luotiin simulaatiomalli robottikädestä, joka sijoitettaisiin Metroverin päälle. Isaac Simillä tehty simulaatiomalli hyödyntää takaisinkytkentää liikuttaakseen mallin osia. Työssä on myös selvitetty tälle robottikädelle suora- ja käänteiskinematiikka x-, y- ja z-koordinaattien suhteen.

6.2 Haasteet

Ohjelmiston käyttö koitui työn suurimmaksi haasteeksi. Isaac Sim ei ollut entuudestaan tuttu ja sen opettelu vaati paljon työtä ja testailua. Kerättiin muutamia huomautuksia, joita tehtiin työn aikana. Nämä huomautukset liittyvät enimmäkseen simulaation rakentamiseen.

- Älä tuo simuloitavaa mallia yhtenä kokonaisuutena, vaan osa kerrallaan, kuten työssä on selitetty
- Pidä kirjaa osien mitoista, jotta voidaan helpottaa osien ja akselien sijainnin määrittelyä
- Rakentaessa simulaatiota, pidä mielessä missä suunnassa akseli pyörii

6.3 Jatkokehitystä

Simulaatio suorittaa liikkeet, mutta saapuessaan loppupisteeseen alkaa vapina. Tämä johtuu siitä, että takaisinkytkentä on simppeli päälle-pois ajatuksella luotu. Jotta vapina saataisiin loppumaan, olisi lisättävä jonkinlainen säätö, joka hidastaa liikettä, mitä lähemmäksi mennään toivottua pistettä.

Käänteiskinematiikka on toimiva, mutta sen käyttäminen ei ole implementoitu suoraan simulaatioon. Tällä hetkellä on syötettävä erilliseen koodiin toivotut x-, y- ja z-koordinaatit. Koodi antaa viisi arvoa, joita pitää käsin syöttää simulaatioon. Seuraava askel olisi liittää käänteiskinematiikan kaava simulaatioon.

Käänteiskinematiikan koodia voidaan myös jatkokehittää. Tällä hetkellä koodi pystyy selvittämään ainoastaan käden sijainnin, mutta ei sen suuntaa. Seuraava askel olisi kehitellä jatkoa jo määrättyyn koodiin, jossa voidaan laskea kaikki akselien kulmat toivotun suunnan ja sijainnin perusteella.

Simulaatioon on määrätty joillekin osille Collision Body -fysiikka, jolloin osat törmäävät toisiinsa eikä pääse liikkumaan toistensa läpi. Tämä ei estä simulaatiota yrittämästä liikkumaan itsensä läpi annettaessa ohjeet. Seuraavana askeleena olisi kielletyn alueen luominen käden liikkeille.

Tämä robottisimulaatio käsittelee ainoastaan robottikättä, jonka tarkoitus on olla monipyöräisen mobiilirobotin päällä. Seuraava askel olisi luoda robotille pyörät ja määrittää niiden liikkeet ja ohjaukset.

Lähteet

Cartesian Coordinate System & Right Hand Rule. Verkkoaineisto. ResearchGate. <https://www.researchgate.net/figure/Cartesian-coordinate-system-a-Right-hand-rule-b-6_fig1_326500990>. Luettu 22.4.2025.

Europena Rover Challenge. 2025. Verkkoaineisto. <https://roverchallenge.eu/>. Luettu 2.4.2025.

Launonen, Eero; Sorvali, Esko & Toivonen, Pertti. 2006. Teknisten ammattien matematiikka, Lineaarialgebra. WSOY Oppimateriaalit Oy Helsinki.

Matlab Simulink. 2025. Verkkoaineisto. <https://se.mathworks.com/products/simulink.html>. Luettu 14.2.2025.

McKerrow, Phillip John. 1991. Introduction to Robotics. Addison-Wesley Publishers Ltd.

Moore-Penrose inverse. 2025. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Moore%E2%80%93Penrose_inverse>. Päivitetty 13.4.2025. Luettu 2.5.2025.

NVIDIA Isaac Sim. 2025. Verkkoaineisto. <https://developer.nvidia.com/isaac/sim>. Luettu 14.2.2025.

NX CAD software. 2025. Verkkoaineisto. <https://plm.sw.siemens.com/en-US/nx/cad-online/>. Luettu 14.2.2025.

Unity. 2025. Verkkoaineisto. <https://unity.com/products/unity-personal>. Luettu 14.2.2025.

Suorakinematiikan koodi

```
  # Imported libraries
import numpy as np
import math as m

# Angle of each axis
# Input must be in RADIANS
# Change these values to see the end posittion of the arm
theeta_1 = 0
theeta_2 = 0
theeta_3 = 0
theeta_4 = 0
theeta_5 = 0

# Axis transpotion compared to previous axis
# Input must be in RADIANS
# Values are constant, DO NOT CHANGE
alpha_1 = 0.5*m.pi
alpha_2 = 0
alpha_3 = -0.5*m.pi
alpha_4 = 0.5*m.pi
alpha_5 = -0.5*m.pi

# Length along X-axis compared to previous axis
# Values are constant, DO NOT CHANGE
l_1 = 0
l_2 = 400
l_3 = 0
l_4 = 0
l_5 = 0

# Length along Z-axis compared to previous axis
# Values are constant, DO NOT CHANGE
d_1 = 70
d_2 = 0
d_3 = -80
d_4 = 450 # Check this value, should be from middle of Elbow Joint to
middle of dogbone in Elbow Link
d_5 = 80
```

```
# Position matricies

M_1 = [[m.cos(theeta_1), -1*m.sin(theeta_1)*m.cos(alpha_1),
m.sin(theeta_1)*m.sin(alpha_1), l_1*m.cos(theeta_1)],
        [m.sin(theeta_1), m.cos(theeta_1)*m.cos(alpha_1), -
1*m.cos(theeta_1)*m.sin(alpha_1), l_1*m.sin(theeta_1)],
        [0, m.sin(alpha_1), m.cos(alpha_1), d_1],
        [0, 0, 0, 1]]

M_2 = [[m.cos(theeta_2), -1*m.sin(theeta_2)*m.cos(alpha_2),
m.sin(theeta_2)*m.sin(alpha_2), l_2*m.cos(theeta_2)],
        [m.sin(theeta_2), m.cos(theeta_2)*m.cos(alpha_2), -
1*m.cos(theeta_2)*m.sin(alpha_2), l_2*m.sin(theeta_2)],
        [0, m.sin(alpha_2), m.cos(alpha_2), d_2],
        [0, 0, 0, 1]]

M_3 = [[m.cos(theeta_3), -1*m.sin(theeta_3)*m.cos(alpha_3),
m.sin(theeta_3)*m.sin(alpha_3), l_3*m.cos(theeta_3)],
        [m.sin(theeta_3), m.cos(theeta_3)*m.cos(alpha_3), -
1*m.cos(theeta_3)*m.sin(alpha_3), l_3*m.sin(theeta_3)],
        [0, m.sin(alpha_3), m.cos(alpha_3), d_3],
        [0, 0, 0, 1]]

M_4 = [[m.cos(theeta_4), -1*m.sin(theeta_4)*m.cos(alpha_4),
m.sin(theeta_4)*m.sin(alpha_4), l_4*m.cos(theeta_4)],
        [m.sin(theeta_4), m.cos(theeta_4)*m.cos(alpha_4), -
1*m.cos(theeta_4)*m.sin(alpha_4), l_4*m.sin(theeta_4)],
        [0, m.sin(alpha_4), m.cos(alpha_4), d_4],
        [0, 0, 0, 1]]

M_5 = [[m.cos(theeta_5), -1*m.sin(theeta_5)*m.cos(alpha_5),
m.sin(theeta_5)*m.sin(alpha_5), l_5*m.cos(theeta_5)],
        [m.sin(theeta_5), m.cos(theeta_5)*m.cos(alpha_5), -
1*m.cos(theeta_5)*m.sin(alpha_5), l_5*m.sin(theeta_5)],
        [0, m.sin(alpha_5), m.cos(alpha_5), d_5],
        [0, 0, 0, 1]]

# Forward kinematics matrix product
M_12 = np.dot(M_1,M_2)
M_123 = np.dot(M_12,M_3)
M_1234 = np.dot(M_123,M_4)
M_12345 = np.dot(M_1234,M_5)

# Round the values of the final matrix to 2 decimal points
M_12345 = np.round(M_12345,2)

# Print the values of the final matrix
print (M_12345)
```

Käänteiskinematiikan koodi

```
# Imported libraries
import numpy as np
import math as m

# Input the target x,y,z values
target_position = np.array([600, 0, 520])

# Input the initial axis angles
initial_angles = np.array([0,0,0,0,0])

# Axis transpotion compared to previous axis
# Input must be in RADIANS
# Values are constant, DO NOT CHANGE
alpha_1 = 0.5*m.pi
alpha_2 = 0
alpha_3 = -0.5*m.pi
alpha_4 = 0.5*m.pi
alpha_5 = -0.5*m.pi

# Length along X-axis compared to previous axis
# Values are constant, DO NOT CHANGE
l_1 = 0
l_2 = 400
l_3 = 0
l_4 = 0
l_5 = 0

# Length along Z-axis compared to previous axis
# Values are constant, DO NOT CHANGE
d_1 = 70
d_2 = 0
d_3 = -80
d_4 = 450
d_5 = 80
```

```
# Position matricies
def forward_kinematics(joint_angles):
    theeta_1, theeta_2, theeta_3, theeta_4, theeta_5 = joint_angles
    M_1 = [[m.cos(theeta_1), -1*m.sin(theeta_1)*m.cos(alpha_1),
m.sin(theeta_1)*m.sin(alpha_1), l_1*m.cos(theeta_1)],
        [m.sin(theeta_1), m.cos(theeta_1)*m.cos(alpha_1), -
1*m.cos(theeta_1)*m.sin(alpha_1), l_1*m.sin(theeta_1)],
        [0, m.sin(alpha_1), m.cos(alpha_1), d_1],
        [0, 0, 0, 1]]

    M_2 = [[m.cos(theeta_2), -1*m.sin(theeta_2)*m.cos(alpha_2),
m.sin(theeta_2)*m.sin(alpha_2), l_2*m.cos(theeta_2)],
        [m.sin(theeta_2), m.cos(theeta_2)*m.cos(alpha_2), -
1*m.cos(theeta_2)*m.sin(alpha_2), l_2*m.sin(theeta_2)],
        [0, m.sin(alpha_2), m.cos(alpha_2), d_2],
        [0, 0, 0, 1]]

    M_3 = [[m.cos(theeta_3), -1*m.sin(theeta_3)*m.cos(alpha_3),
m.sin(theeta_3)*m.sin(alpha_3), l_3*m.cos(theeta_3)],
        [m.sin(theeta_3), m.cos(theeta_3)*m.cos(alpha_3), -
1*m.cos(theeta_3)*m.sin(alpha_3), l_3*m.sin(theeta_3)],
        [0, m.sin(alpha_3), m.cos(alpha_3), d_3],
        [0, 0, 0, 1]]
    M_4 = [[m.cos(theeta_4), -1*m.sin(theeta_4)*m.cos(alpha_4),
m.sin(theeta_4)*m.sin(alpha_4), l_4*m.cos(theeta_4)],
        [m.sin(theeta_4), m.cos(theeta_4)*m.cos(alpha_4), -
1*m.cos(theeta_4)*m.sin(alpha_4), l_4*m.sin(theeta_4)],
        [0, m.sin(alpha_4), m.cos(alpha_4), d_4],
        [0, 0, 0, 1]]

    M_5 = [[m.cos(theeta_5), -1*m.sin(theeta_5)*m.cos(alpha_5),
m.sin(theeta_5)*m.sin(alpha_5), l_5*m.cos(theeta_5)],
        [m.sin(theeta_5), m.cos(theeta_5)*m.cos(alpha_5), -
1*m.cos(theeta_5)*m.sin(alpha_5), l_5*m.sin(theeta_5)],
        [0, m.sin(alpha_5), m.cos(alpha_5), d_5],
        [0, 0, 0, 1]]

    # Forward kinematics matrix product
    M_12 = np.dot(M_1,M_2)
    M_123 = np.dot(M_12,M_3)
    M_1234 = np.dot(M_123,M_4)
    M_12345 = np.dot(M_1234,M_5)

    # Returns (x,y,z) since we are only interested in these values
    return M_12345[:3,3]
```

```
# Jacobian calculation (position only)
def compute_jacobian(joint_angles):

    # Jacobian matrix
    J = np.zeros((3,5))

    # Small value for numeriacal differentation
    epsilon = 1e-6

    for i in range(5):
        joint_angles_plus = joint_angles.copy()
        joint_angles_plus[i] += epsilon
        P_plus = forward_kinematics(joint_angles_plus)

        joint_angles_minus = joint_angles.copy()
        joint_angles_minus[i] -= epsilon
        P_minus = forward_kinematics(joint_angles_minus)

        J[:,i]=(P_plus - P_minus)/(2*epsilon)

    return J

def wrap_to_range(angle, min_value, max_value):
    range_size = max_value - min_value
    wrapped_angle = (angle - min_value) % range_size + min_value
    if wrapped_angle > max_value:
        wrapped_angle -= range_size
    return wrapped_angle
```

```
def inverse_kinematics(target_position, initial_angles, tolerance=1e-6
,max_iterations=1000, learning_rate = 1.0):
    joint_angles = np.array(initial_angles, dtype = np.float64)

    for iteration in range(max_iterations):

        # Get the current position
        current_position = forward_kinematics(joint_angles)

        # Compute error (difference btween current position and tar-
get)
        error = target_position - current_position

        # Check if the error is within the tolerance
        if np.linalg.norm(error) < tolerance:
            print(f"Target position reached in {iteration} itera-
tions.")
            return joint_angles

        # Compute the Jacobian
        J = compute_jacobian(joint_angles)
        # Compute the pseudo-inverse of the Jacobian
        J_inv = np.linalg.pinv(J)   # Pseudo-inverse handles singular-
ities

        delta_theeta = J_inv.dot(error)
        delta_theeta = np.reshape(delta_theeta,-1)

        delta_theeta *= learning_rate
        # Update joint angles
        joint_angles += delta_theeta

        # Enforece the range for axese 1, 3, 4 and 5 using
wrap_to_range
        joint_angles[0] = wrap_to_range(joint_angles[0], -m.pi, m.pi)
        joint_angles[2] = wrap_to_range(joint_angles[2], -m.pi, m.pi)
        joint_angles[3] = wrap_to_range(joint_angles[3], -m.pi, m.pi)
        joint_angles[4] = wrap_to_range(joint_angles[4], -m.pi, m.pi)

        # Fix the angle theeta_2's range as 180 degrees
        joint_angles[1] = np.clip(joint_angles[1], 0, np.pi)

        print(f"Iteration {iteration}: Error = {np.linalg.norm(er-
ror)}")

    print("Maximum iterations reached")
    return joint_angles

final_joint_angles = inverse_kinematics(target_position, initial_an-
gles)
print(np.round(final_joint_angles,2))
```