Opinnäytetyö AMK

Konetekniikka

2023

Miikka Taskinen

# Parametrinen OLP-malli lattiaratahitsausrobotille

– Pemamek Oy

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

Opinnäytetyö (AMK) | Tiivistelmä

Turun ammattikorkeakoulu

Konetekniikka

2023 | 77 sivua

Miikka Taskinen

# Parametrinen OLP-malli lattiaratahitsausrobotille

Tämän opinnäytetyön aiheena oli kehittää 3D-malli Pemamek Oy:n vakioidulle lattiaratahitsausrobotille, jolla voidaan tehdä robottihitsausohjelmia, ja jonka piirteitä ja varustelua voidaan muokata yksinkertaisista valikoista. Tarkoituksena oli saada aikaiseksi malli, jolla voitaisiin luoda useille yrityksille samalta pohjalta hitsausratkaisuja ja yksinkertaistaa samankaltaisten projektien kehitystä jatkossa. Samalla tehtäisiin uuden vakioradan kanssa yhteensopivia kääntöpöytiä, joiden on tarkoitus helpottaa turva-aitojen ja työalueen suunnittelua. Lopullisilla malleilla tulisi pystyä säästämään aikaa suunnittelussa, simulointivalmistelussa ja testauksessa. Opinnäytetyön työvaiheiden selityksen olisi tarkoitus toimia potentiaalisena tukimateriaalina samankaltaisten projektien tekemiseen ja antaa esimerkkiä käytettävien ohjelmistojen mahdollisuuksista.

Opinnäytetyöhön sisältyy aihetta alustavaa teoriaa ja kerrotaan mallin tekemisen työvaiheista. Pääkohtina opinnäytetyössä toimivat työvaiheselitykset etäohjelmoitavan 3D-mallin luomiselle Visual Components-ohjelmistolla, tarkemmat selitykset piirteiden muutoksille ohjelmiston työkaluilla sekä lisäohjelmoinnin läpikäynti. Esimerkkeinä käytetään piirteitä ja ohjelmakoodia opinnäytetyössä kehitetystä etäohjelmointimallista, joka on mallin valmistumisen jälkeen otettu käyttöön Pemamek Oy:n suunnittelussa ja robottiohjelmoinnissa. Näihin lukeutuu ohjelmistovikojen välttäminen, poikkeustilanteiden läpikäynti ja käyttöliittymien tiedonkäsittelyn hyödyntäminen simulaatiomaailmassa.

Asiasanat:

3D-mallinnus, simulointi, hitsaus, robotiikka, ohjelmointi, parametrinen

Bachelor’s Thesis | Abstract

Turku University of Applied Sciences

Mechanical Engineering

2023 | 77 pages

Miikka Taskinen

# Parametric OLP model for a robot welding station

The objective of this thesis was to create a 3D model for the standardized robot welding station made by Pemamek Oy. The model can be used for robot welding programming and the features and equipment can be changed from a simple menu. The purpose of this model is to work as a base for welding solutions for clients and to simplify developing similar projects in the future. Turntables compatible with the new standard welding station will be made in tandem so that work environments and safety fences can be designed more easily. The end product should reduce time that is spent on designing the models, preparing them for simulations and testing them. The explanations for the steps in this thesis should double as potential instructional material for similar projects and give examples of the possibilities with the programs used.

This thesis provides some base level theory for the subjects handled in it and explains the work process behind the model. The main focus was on the steps for creating an offline-programmable 3D-model using Visual Components and explaining the modification of features with the program’s tools and additional programming. The examples include features and code from the offline-programmable model made for this thesis, which has been deployed by Pemamek Oy for designing and programming as of its completion. These examples include avoiding program errors, handling plausible abnormalities and utilizing the data processing of interfaces in a simulation.

Keywords:

3D modeling, simulation, welding, robotics, programming, parametric

# Sisältö

# Kuvat

# Käytetyt lyhenteet tai sanasto

| Lyhenne | Lyhenteen selitys (Lähdeviite) |
|---|---|
| B-rep | Boundary representation (Rajan edustus) |
| CAD | Computer-aided design (tietokoneavusteinen suunnittelu) |
| CAM | Computer-aided Manufacturing (tietokoneavusteinen valmistus) |
| CSG | Constructive Solid Geometry (yksinkertaisten geometrioiden yhdistelmä) |
| OLP | Offline programming (etäohjelmointi ulkoisella tietokoneella) |
| PnP | Plug and Play (komponenttien siirron ja liittämisen työkalu) |
| RWS-FG | Robot Welding Station – Floor Gantry |
| Sx | Scale X (koon muutos X-suunnassa) |
| Sy | Scale Y (koon muutos Y-suunnassa) |
| Sz | Scale Z (koon muutos Z-suunnassa) |
| TCP | Tool Center Point (Robotin työkalun keskipiste) |
| Tx | Translational X (siirto X-akselilla) |
| Ty | Translational Y (siirto Y-akselilla) |
| Tz | Translational Z (siirto Z-akselilla) |

# 1 Johdanto

Opinnäytetyössä käydään läpi parametrisen etäohjelmoitavan mallin kehittämistä Pemamek Oy:n lattiaratahitsausrobotista, RWS-FG:stä. Perehdytään etäohjelmoitavan 3D-mallin luomisen teoriaan ja esitellään ohjelmistoja, joita käytetään mallin tekemisessä. Työ tehdään Pemamek Oy:lle osana laajempaa vakiointiprojektia, jossa pyritään kehittämään olemassa olevia konsepteja hitsausradoista kääntöpöytiin. Vakiomallien on tarkoitus sisällyttää mahdollisimman laaja katalogi vaihtoehtoja, joita voidaan käyttää myynnistä tehdasasetteluihin ja lopulliseen hitsausohjelmien luontiin minimaalisilla muokkauksilla ja yksinkertaisilla käyttäjän syötöillä. Valmiilla komponenteilla tarvitsee vain valita näiden sisäisistä valikoista haluttavat dimensiot, joka mahdollistaa mallin räätälöinnin halutun kaltaiseksi käytettävien ohjelmistojen, kuten WeldControl 300:n, perustason lisenssien kanssa. Tällä voidaan säästää tiedostojen suunnitteluun, mallintamiseen, siirtoon ja muokkaukseen käytettävässä ajassa.

Teorian on tarkoitus tarjota yleispohjaa relevantille informaatiolle, jota työssä tarvitaan, sekä esittelemään toimeksiantajan toimialaa. Työn tekemisen osuudessa oletetaan lukijalla olevan valmiina CAD-malli, jonka pystyy tallentamaan sopivaan muotoon ja tuomaan Visual Components -ohjelmaan. Kerrotaan OLP-mallin työvaiheista Visual Componentsissa ja alustavista toimista WeldControl-ohjelmistolla ja sen jälkeen selitetään tarkemmin parametristen ominaisuuksien lisäyksistä ja hyödyistä. Näiden on tarkoitus toimia tämän työn selvityksen lisäksi opettavaisena materiaalina muille samankaltaisille projekteille. Samaa tietoa voidaan hyödyntää muissakin malleissa, mutta työvaiheita saattaa joutua muuttamaan ja esimerkiksi eri parametrisoinnin metodien laajuus voi vaihdella merkittävästi. Visual Components -ohjelmistoon pääasiassa keskityttäessä tullaan myös tämän eri toimintoja käymään läpi tarkemmin ja pyritään kertomaan yleisimpien ominaisuuksien toimintaperiaatteista, jotta toivottavasti työn lopuksi uusi käyttäjä kykenisi soveltamaan näitä jollain tasolla omassa työssään. Lopuksi

tulee vielä mainita, että jotta etäohjelmoitavaa mallia voisi luotettavasti käyttää se vaatisi vielä työssä läpikäytävien työvaiheiden lisäksi mallin kalibroimisen ja pulssien määrittämisen. Nämä eivät enää sisälly omaan työnkuvaani ja vaativat arvoja, joita voidaan saada vain fyysisistä kokoonpanoista, jolloin näiden määrittämisen tulee tapahtua erikseen jokaisen kokoonpanon kohdalla. Sen sijaan lattiaradan OLP-malli tulee olemaan yleinen pohja, jolla on mahdollista tehdä hitsausohjelmia robotille, mutta niitä ei kyseisessä muodossa vielä todellisuudessa käytettäisi epätarkkuuden vuoksi.

Puhuttaessa Visual Componentsin työvaiheista tullaan käyttämään enemmän lainasanoja, koska pyritään viittaamaan suoraan toimintoihin ohjelmistossa, jonka kieliasetukset on asetettu englanniksi ja johon on saatavilla eniten tukimateriaalia kyseisellä kielellä. Parametrisuuden parantamista käsitellessä tullaan enemmän kääntämään termistöä, jotta voidaan paremmin selittää miksi ja miten ohjelmiston ominaisuuksia käytetään. On myös hyvä saada selitettyä tarkemmin toiminnot, joiden kääntämättä jättäminen saattaisi haitata tekstin ymmärtämistä. Opinnäytetyö kokonaisuudessaan luottaa siihen, että lukijalla on kattavat pohjatiedot erityisesti Visual Componentsista ja Python-ohjelmoinnista.

# 2 3D-mallinnus ja simulointi

3D-mallinnuksella viitataan tietokoneavusteiseen geometriseen mallinnukseen. Geometrisia muotoja voidaan luokitella laskutoimituksiksi ja 3D-mallinnuksella näitä laskutoimituksia voidaan esittää digitaalisessa, visuaalisessa muodossa. Laajin käytetty kolmiulotteisen geometrisen mallinnuksen muoto on tilavuusmallinnus, jossa objekti määritetään sen pintojen, kulmien ja pisteiden, eli solmujen, perusteella. Lopputuloksena on kiinteä malli, jonka geometrisia muotoja CAD-ohjelmisto kykenee laskemaan ja jolle ohjelmisto kykenee antamaan ominaisuuksia. Malli voidaan esittää ”*CSG*”- tai ”*B-rep*”-muodossa. *CSG*- eli ”*Constructive Solid Geometry*”-malli muodostaa objektin yhdistelemällä yleisiä kiinteitä muotoja, kuten laatikoita, kartioita, sylintereitä ja palloja. ”*B-rep*”- eli ”*Boundary representation*”-malli pursottaa objektin pisteiden, kulmien ja pintojen välille yhdistävät piirteet, jotka muodostavat saumattoman, kiinteän mallin. (Prescient Technologies.)

Kehittyneillä ohjelmistoilla virtuaalisille objekteille, jotka rinnastavat todellisia kappaleita, voidaan muodostaa simuloitavia prototyyppejä ja tehdä testejä ilman tarvetta fyysiselle mallille. Mallit ovat kuitenkin raskaita prosessoida ja mallien ja simulaatioiden vaativuuteen vaikuttaa esimerkiksi tarkasteltavien piirteiden määrä, näiden liitokset toisiinsa ja liikkeiden rajoitukset. Ohjelmistoja ja niiden ominaisuuksia kehitetään insinööri- ja suunnittelualoille, jotta voidaan optimoida ohjelmistovaatimukset nykyteknologian kanssa ja saada tehtyä mahdollisimman johdonmukainen tuoteprosessi suunnittelusta valmistukseen. (Jankowski & Doyle 2008, Chapter 1.)

Tilavuusmalleja käytetään eniten kolmiulotteisessa geometrisessa mallinnuksessa, koska nämä mahdollistavat ominaisuuksien laskennan, dynaamisen analyysin, robottisimulaatiot, ohjelmointituen sekä geometrisen ja topologisen informaation säilyttämisen.  Suurin osa simulaatioista päätyy käyttämään yksinkertaisuuden nimissä *B-rep* -malleja. (Prescient Technologies.)

CAD-ohjelmiston ominaisuuksien salliessa samasta 3D-mallista voidaan simulaatioiden lisäksi luoda staattisia kuvia 3D-renderöinnillä, jonka on tarkoitus esittää lopullista tuotetta. Vastakohtana tälle toimii 3D-skannausteknologia, joka sallii tuotteiden 3D-mallien luomisen valmiiden tuotteiden tai prototyyppien skannaamisen avulla. Näitä yhdistelemällä voidaan luoda lähes täydellinen digitaalinen kaksonen objekteista, joilla voidaan suunnitella valmistusprosessia sekä suorittaa testejä ja simulaatioita. (Siemens Digital Industries Software.)

## 2.1 SolidWorks – tietokoneavusteinen suunnittelu

SolidWorks on vuonna 1995 julkaistu 3D-mallinnuksessa, sähköisissä piirustuksissa ja simuloinnissa käytettävä tietokoneohjelmisto. Sen kehitys on alkanut tavoitteesta tehdä 3D-suunnittelusta insinöörialoilla ja muotoilussa helpommin lähestyttävää. Ohjelmaa kehitetään aktiivisesti ja laajennukseen on vaikuttanut sen helppokäyttöisyys ja tehokkuus, jotka ovat tehneet siitä houkuttelevan valinnan yrityksille ja yksilöille, ja jatkuva kehitys tekee siitä erityisen kilpailukykyisen. SolidWorks-versioita löytyy eri työtehtäviin, mutta toimintaperiaatteiden puolesta eroavaisuudet on pyritty minimoimaan, jotta siirtymät versioiden välillä eivät olisi liian monimutkaisia. Tämä selkeyttää monissa tapauksissa tuoteprosessia, kun esimerkiksi suunnittelijoiden on helpompi hahmottaa työjärjestystä ja kuinka kappaleita olisi loogista valmistaa ja koota, kun heidän ohjelmistoversionsa edellyttää samanlaista ajattelua kuin muilla aloilla. SolidWorksilla pystytään mallintamaan todellisuutta vastaavia virtuaalisia prototyyppejä kappaleista, joita voidaan simuloida niin realistisesti kuin virtuaalinen maailma ja teknologia sallivat. Ohjelmistolla voidaan yksinkertaistaa mallit ja tehdä simulaatioita ja testauksia ilman tarvetta luoda jokaista iteraatiota varten fyysistä prototyyppiä. (Jankowski & Doyle 2008, Chapter 1 - Chapter 2.)

SolidWorks sallii tiedostojen tallentamisen useissa eri muodoissa, jotka tukevat osakokoonpanojen tallentamista, komponenttien päivittämistä yhdessä kokoonpanon kanssa sekä näiden tallentamista piirustusmuodossa (Jankowski & Doyle 2008, Chapter 2). Omien tiedostomuotojen lisäksi SolidWorks tukee

neutraaleja tiedostomuotoja, joista yksi suosituimmista on ISO-standardisoimisjärjestön kehittämä STEP. STEP, eli ”*Standard for The Exchange of Product model data*”, tukee kaikkia teknisen tuotetiedon muotoja kansainvälisesti, joka on johtanut sen eri tiedostotyyppien käyttöön lähes jokaisessa CAD- ja CAM-järjestelmässä (STEP Tools Inc).

## 2.2 Visual Components – 3D-simulointi

Visual Components on tuotantoautomaatioprosessin simulointiin ja robottiohjelmointiin kehitetty ohjelmisto, joka sisältää laajan katalogin automaatioteollisuudessa käytettäviä robotteja, koneita ja tuotannossa esiintyviä linjastoja. Ohjelmassa suurelle osalle koneista ja rakenteista löytyy valmiita omia toiminnallisuuksia ja näitä voidaan soveltaa automaatiosimulaatioiden opettamisessa opiskelijatasolla sekä tehdasasettelussa asiakasdemonstraatioita tai tuotannon kehitystä ja seurantaa varten. Visual Components sallii myös uusien laitteiden tekemisen CAD-mallien pohjalta ja toiminnallisuuksien luomisen uusille laitteille opetusvideoiden ja tukifoorumien avulla. Samalla Visual Components tarjoaa vapauden osaavammalle käyttäjälle muokata, luoda ja seurata kokonaan uusia laitteiden ominaisuuksia yrityskohtaisten vaatimusten mukaisesti. Esimerkiksi ohjelman omat etäohjelmointityökalut sallivat nopeamman robottien käyttöönoton ja näiden seurannan graafisella käyttöliittymällä ja VR-työkalut nopeamman perehdyttämisen yksikön toiminnasta. (Visual Components.)

Ulkoisten CAD-mallien tuominen Visual Componentsiin edellyttää yhteensopivia tiedostomuotoja ja tuettujen tiedostomuotojen määrä riippuu ohjelmistoversiosta. Viimeisimpien ohjelmistopäivitysten puuttuessa esimerkiksi kansainvälisten standardien mukainen STEP-tiedostomuoto on yleisesti käytössä. Visual Components sallii ”STEP AP 203”-, ” STEP AP 214”- ja ”STEP AP 242” -tiedostojen tuomisen, koska nämä säilyttävät suurien geometriakokoonpanojen piirteet ja rakenteet. Johtuen piirteistä, joita Visual Componentsilla malleihin lisätään, STEP-tiedostoista vain ”AP 242”-tiedoston

vieminen on mahdollista. Tämä periaatteessa sisältää molemmat aikaisemmat tiedostotyypit, eli säilyttää kaiken simulaatiomallin tiedon varmemmin.

## 2.3 Pema WeldControl 300 – Offline-ohjelmointi

Pema WeldControl 300 on Visual Componentsin pohjalta tehty, graafisella käyttöliittymällä varustettu offline-ohjelmointi- ja myyntityökalu. Ohjelma on erityisesti luotu Pemamekin hitsausautomaatiota varten ja kehitetty olemaan optimaalinen yrityksen laitteille, tehokas ja mahdollisimman yksinkertainen kilpailukyvyn ja käyttökokemuksen vuoksi. Ohjelma sisältää Visual Componentsin sisältämät komponentit ja sallii Pemamekin omat mallit toimintoineen, sekä ohjelmoinnin ja lisäosat. Ohjelmalla on mahdollista luoda hitsausohjelmien lisäksi yksinkertaisia demonstraatioita suunnitellun tehdasasettelun koosta, soveltuvuudesta asiakkaan työympäristöön ja robotin laajimmasta mahdollisesta toiminta-alueesta robottisolussa, joka auttaa turva-alueiden suunnittelussa. Ohjelmasta on olemassa versioita, joista offline-ohjelmointiin soveltuu myös WeldControl 200, joka erikoistuu telakkalaitteiden toimintaa ja seurantaa varten. (Visual Components 2020.)

Yksipaneelinen ohjausjärjestelmä helpottaa prosessin seurantaa ja automaattinen diagnostiikka nopeuttaa tuotantoa samalla jokaisessa robotisoidussa hitsaus- ja tuotantoratkaisussa. Ohjelmiston versiot takaavat yksinkertaisen käyttöliittymän jokaiseen käyttökohteeseen, jolloin esimerkiksi laivanrakennuksessa ja konepajateollisuudessa ei ole käytössä tismalleen samaa versiota, joka vähentää tarpeettomien toimintojen määrää operaattoreiden käyttöliittymässä. (Pemamek WeldControl.)

## 2.4 Python-komentokehotekieli

Python on Guido van Rossumin luoma komentokehotekieli, jota voidaan käyttää luomaan ja muokkaamaan ohjelmia tai apuohjelmia ja suorittamaan yksinkertaisia ja erityisesti toistuvia komentoja. Koska Python kykenee

tehokkaasti suorittamaan laskutoimituksia sekä käsittelemään ja vertailemaan erilaisia datakirjastoja ja merkkijonoja se on yhdessä yksinkertaisen lauseoppinsa vuoksi yksi suosituimmista ohjelmointikielistä. Tavallisia sovelluksia Pythonille ovat interaktiiviset, ohjelmoitavat käyttöliittymät, koska sen kyky käsitellä datakirjastoja sallii enemmän taipuvuutta ja yksinkertaistaa ohjelmistojen sisäisten tiedostojen hyödyntämistä yksilökohtaisiin käyttötarkoituksiin. Se sisältää myös laajan kirjaston omia toimintoja, jotka helpottavat sen opettelua aloitteleville ohjelmoijille ja dynaaminen tyyppitarkastus helpottaa vikatarkastelua, tarjoten koodin ajon aikana esimerkkejä tietotyyppien muunnoksien merkityksestä. (Python Software Foundation 2023.)

## 2.5 Parametrinen malli

Parametrilla ei ole tarkkaa määritelmää, mutta sillä voidaan viitata matemaattisiin muuttujiin, joilla voidaan vaikuttaa suureisiin tai yhtälöihin, joista muut suureet ovat jollain tasolla riippuvaisia. Termi parametrinen puolestaan viittaa joukkoon suureita, jotka parametrit saavat aikaiseksi. (Davis 2013.)

Kun puhutaan parametrisesta mallista, viitataan malliin, jossa on mahdollista suorittaa geometrioiden dimensioiden ja esitysmuodon muutoksia annettujen rajoitusten sisällä. Esimerkiksi yksittäisen parametrin muutos saattaa vaikuttaa mallin piirteen kokoon ja kytkettyjen piirteiden sijaintiin tietyllä akselilla tai kokoon suhteessa piirteeseen. Muuttamalla yhtä parametria muutetaan jokaista tästä jollain tasolla riippuvaista osaa kokoonpanossa. Toisin sanoen geometriset muodot voidaan ajatella laskutoimituksina sekä parametreinä ja jokainen parametrien summa, eli mallin eri visuaalinen representaatio, on yksi parametrisen mallin suure. (Davis 2013.)

Parametrisilla vapauksilla ja kytköksillä on tärkeä merkitys, kun ajatellaan mallin toiminnallisuutta ja mahdollisuuksia tämän muutokseen jatkossa. Tekemällä yksittäisestä osasta mallissa riippuvaisen liian monesta tekijästä vaikeutetaan jatkomuutoksia, joka voi johtaa tarvittaessa revisioihin mallista. Toisaalta liian vähäinen määrä parametrejä johtaa pienempään lukumäärään parametrisia suureita, eli rajoittaa mallin mahdollisuuksia. Mallin parametrisoinnin työjärjestystä ja laajuutta tulee suunnitella etukäteen, jotta voidaan välttyä liian monelta ristiriitaisuudelta suunnittelussa ja määrittää osakokonaisuuksien riippuvuuksien laajuudet suhteessa toisiinsa. (Davis 2013.)

# 3 Robotin ohjelmointi

Teollisuusala kehittyy jatkuvasti ja automaation osuus teollisuudessa kasvaa merkittävämmäksi. Tuotantoautomaatio takaa tasaisemman ja nopeamman työtuloksen, pidemmät yhtäjaksoiset tuotantoajat ja laskee työntekijöiden terveysriskejä. Yhdistelemällä konenäköä, automatisoitua prosessinaikaista järjestelmävalvontaa, korkeatasoista robotiikkaa ja viimeisimpiä turvamenetelmiä voidaan saada aikaiseksi työsolu, jossa säännöllisellä kunnonvalvonnalla voidaan varmistaa minimaalisella työvoimalla pyörivä, nopea valmistusprosessi suurille ja pienille tuotteille. Automaation suurin etu tulee mahdollisuudesta suorittaa nopeasti toistuvia työtehtäviä, joissa ei tapahdu muutoksia tai poikkeuksille on voitu ohjelmoida omat toimenpiteet. Adaptiiviseen työhön, jopa silloin kun fyysinen työ hoidetaan automaatiolla, tarvitaan aina joku ohjelmoimaan toimenpiteet prosessin aikana. Ihanteellisessa tilanteessa automaatio ja robotit eivät korvaa ihmisiä työpaikoilla, vaan muuttavat näiden työnkuvaa esimerkiksi linjastokokoamisesta järjestelmähuoltoon, prosessin valvontaan tai robotin ohjelmointiin.

## 3.1 Robotti

Robotiksi määritellään automaattisesti toimiva monitoimilaite, joka kykenee tekemään toistuvaa työtä ihmisen tavoin sille annetun informaation avulla, ja jonka toiminta muuttuu vastaanotetuilla viesteillä, käskyillä ja uudelleenohjelmoinnilla (Herath & St-Onge 2022, 5). Robotin fyysisen rakenteen määritelmä on karkeasti jäykkien rakenteiden kinemaattinen ketju, jossa osat yhdistyvät, ja joiden liikkeet rajoittuvat, nivelien avulla (Herath & St-Onge 2022, 267). Robotin toimintaperiaate teollisuus- tai harrastuskäytössä riippuu sen käyttöönottajan opetuksesta, joka tulee tavallisesti tapahtumaan jonkin tasoisen ohjelmoinnin kautta (Herath & St-Onge 2022, 83–84).

Robotin ohjaus tapahtuu joko online- tai offline-ohjelmoinnilla. Online-ohjelmointi edellyttää aktiivisen yhteyden robottiin ja tämä tapahtuu yleensä

käsiohjaimien kanssa, kun taas offline-ohjelmointi voidaan suorittaa 3D-malleilla simulointiohjelmilla ja robotin ajot testata ennen ajo-ohjelman siirtoa robotille.

## 3.2 Online-ohjelmointi

Online-ohjelmointi tarkoittaa robotin ohjelmointia aktiivisen yhteyden aikana, eli robotti tulee ottaa pois tuotantokäytöstä ja vaihtaa opetustilaan uuden toiminnon opetuksen ajaksi. Tavallisesti robotin liikkeiden opetus tapahtuu käsiohjaimen kanssa tai valitsemalla robotille opetuspisteet raahaamalla tämä käsin sijaintiin opetustilassa ja tallentamalla pisteet ohjelmaan. Online-ohjelmointi edellyttää vain käsiohjaimen, jotka ovat suhteellisen yksinkertaisia opetella ja joiden avulla voidaan nopeasti saada muutoksia aikaiseksi robotin ajo-ohjelmaan, mutta hankaluuksia tässä voi aiheuttaa esimerkiksi tarve opetella robotin valmistajan oma ohjelmointikieli ja järjestelmä. (Robots Done Right.)

## 3.3 Offline-ohjelmointi

Robotin offline-ohjelmoinnilla tarkoitetaan etäohjelmointia, jolloin robotin ajon ohjelmointi voidaan suorittaa keskeyttämästä robotin toimintaa. Mikäli robotin työtehtäviin tulee muutoksia, voidaan 3D-malleilla simuloida todellisuutta vastaavat ajo-ohjelmat ja testata työprosessia ennen sen käyttöönottoa. Uusien työstettävien kappaleiden, työkalujen ja kiinnikkeiden ulottuvuuksia ja näiden vaikutuksia on helpompi tarkastella realistisilla 3D-malleilla ja mahdolliset törmäykset voidaan eliminoida. (FinnRobotics.)

Offline-ohjelmoinnilla voidaan saada aikaiseksi monimutkaisempia robotin ajo-ohjelmia ja toimintoja, mutta verrattain online-ohjelmointiin tarvitaan myös enemmän ulkoisia laitteistoja ja ohjelmistoja, sekä osaavia ohjelmoijia (Robots Done Right.). Robottia ohjelmoidessa halutaan tehdä sen käyttöjärjestelmästä mahdollisimman helppolukuinen ja tehokas, jolloin yksi suosituimmista vaihtoehdoista on Python-ohjelmointikieli, koska tämä vaatii suhteellisen vähän varsinaista koodia suorittaakseen vaikeitakin tehtäviä. Dynaaminen

tietotyyppien tarkastelu sallii esimerkiksi silmukoiden aikana muuttujien tietotyyppien käsittelyn ja vasta tuntemattoman muuttujan ilmetessä keskeyttää ajon. Yhdessä sisennetyn koodirakenteen ja minimaalisen välimuistin roskan kanssa Python tekee robotin ajamisesta nopeampaa ja vikojen löytämisestä ja selvittämisestä helpompaa. (Herath & St-Onge 2022, 86–87).

Offline-ohjelmoinnissa ei välttämättä tarvitse itse luoda koodia, koska simulointiohjelmiston omilla ominaisuuksilla voidaan saada luotua täydet robotin ajot aikaiseksi, mutta mahdollisuus muuttaa ja jatkaa koodia tuo lisämahdollisuuksia tulevaisuuden kannalta ja opettaa enemmän robotin tavasta käsitellä informaatiota. Lisäksi oikeaoppisella tiedonkäsittelyllä voidaan saada robotin toiminnot vaikuttamaan muihin komponentteihin simulaatiomaailmassa. Esimerkiksi Visual Components sallii komponenttien hallinnan käyttöliittymillä, joita voi joko valita ohjelman omista vaihtoehdoista tai luoda itse.

# 4 Pemamek Oy

Pemamek Oy on vuonna 1970 perustettu suomalainen raskasmetalliteollisuuden yritys, joka erikoistuu maailmanluokan hitsaus- ja robottiautomaatioon. Pemalla on toimistoja Suomessa, Yhdysvalloissa ja Brasiliassa ja ympäri eurooppaa, mutta tuotteiden valmistus tapahtuu Pemamekin Loimaan tehtaalla. Pemamek erikoistuu hitsaus- ja tuotantoratkaisujen suunnitteluun ja räätälöintiin asiakkaille sopivaksi ja yli 90% kaikista tuotteista menee vientiin. (Pemamek.)

Pemamek palvelee asiakkaita ympäri maailman, tarjoten laiteratkaisuja ja kokonaistoimituksia laiva-, prosessi- ja energiateollisuuteen, tuulivoimaan ja konepajateollisuuteen sekä kone- ja laitevalmistukseen. Jokaisessa vaiheessa suunnittelusta käyttöönottoon asiakasta palvelee osaava henkilökunta, joka loppuvaiheissa opettaa tarvittavien ohjelmistojen toiminnan operaattoreille sekä tarjoaa etätukea jatkossa. (Pemamek.)

Kehittyneellä 3D-mallinnuksella ja simuloinnilla, yhdessä asiakkaan toiveiden mukaan, kyetään luomaan suunnitelmat, tehdasasettelut ja kapasiteettilaskennat sekä tarjoamaan visuaalinen representaatio lopputuotteesta ja tekemään muutoksia tarvittaessa. Pemamekin tuotekatalogiin sisältyy muun muassa hitsaustornit ja robottilattiradat, kääntöpöydät, rullastot ja ja kompakti Pema Skytrack. Yhdessä WeldControl-hitsausjärjestelmän ja kokeneen henkilöstön kanssa Pema kykenee tuottamaan asiakkaalle tuotantolaitokseen kooltaan ja tehokkuudeltaan optimaalisen automaatioratkaisun ja kouluttamaan uusille operaattoreille nopeasti ohjelmiston käyttöä ja lukua.

# 5 Työn lähtökohdat

Johtuen Pemamek Oy:n toiminnan laajuudesta, kysynnän kasvusta ja työtehtävien monipuolisuudesta tarvitaan mahdollisimman hyvä pohja yrityksen robottihitsauslattiaradalle, jotta voidaan saada käyttöön standardisoidut mallit. Näitä myydään ja tehdään suhteessa enemmän ja on sekä myynnille, suunnittelulle, valmistukselle että käyttöönotolle suotuisaa, että mallien välillä ei olisi suuria eroavaisuuksia. Mallista ollessa esiteltävä versio ja ohjelmoitava kevyt versio voidaan soveltaa saman mallin pohjaa mahdollisimman moneen projektiin. Parametrisella OLP-mallilla voidaan saada aikaiseksi usean osaston hyödyntämä standardipohja, jonka optimaalisen liukuvalikon asetuksia muuttamalla saadaan asiakkaalle räätälöityä ja demonstroitua oikean kokoinen ja oikeat varustukset omaava rata. Samalla pohjalla, jota on jatkokevennetty, voidaan myös etäohjelmoida varsinaista hitsausohjelmaa.

## 5.1 Sovellutus Pemamekin lattiaratamalliin

Jotta robottilattiaradan malli voisi olla monipuolisin mahdollinen parametrinen malli, jota soveltaa useampaan käyttötarkoitukseen, tulee tämän sisältää kaikki tarvittavat, tarjottavissa olevat dimensioihin liittyvät pituuden muutokset, työkalujen, apulaitteiden ja virtalähteiden muutokset sekä vaihtoehtoiset savukaasunpoistot. Samalla tulee varmistaa, että malli pitää sisällään kaikki offline-ohjelmointia varten tarpeelliset piirteet, linkit ja toiminnallisuudet. Pemamekin myyntipuolen versio parametrisesta mallista tulee sisältää kaikki halutut esitettävät piirteet, jotta demonstrointi asiakkaalle voi olla mahdollisimman kattavaa, kun taas robotiikkapuolen versio, eli lopullinen käyttöönotettava ja opetettava versio tulee olemaan välttämättömiltä dimensioiltaan sama, mutta tarpeettomat pintamuotojen yksityiskohdat ja liian monta alakomponenttia sisältävät osat tulee poistaa. Tämä mahdollistaa mallin nopeamman simuloinnin ja muutokset WeldControl-ohjelmiston puolella.

Parametrisen mallin ollessa valmis on käytössä nopein malli, jolla on pienin tiedostomuoto ja helpoin käytettävyys kohderyhmilleen. Ennen parametrista lattiaratamallia jokainen lattiaradan mallin tekeminen, huolimatta projektin samankaltaisuudesta, on lähtenyt liikkeelle suunnittelusta ja 3D-mallintamisesta. Standardimalliin päästyä voidaan suunnittelun, mallinnuksen ja simuloinnin puolella keskittyä helpommin uusiin projekteihin ja tämä tulee nopeuttamaan uusien projektien kehittymistä ja vähentää informaatiovirran hidastumista osastojen välillä, kun esimerkiksi huolimattomuusvirheistä aiheutuvia ongelmia malleissa pystytään karsimaan mahdollisimman paljon. Samalla tietojen pysyessä vakiona pystytään vähentämään virheellisen informaation välittämistä ja vääriä oletuksia tapahtumasta, kun toimitaan valmiiden, olemassa olevien, läpikäytyjen rajoitusten sisällä.

Parametrisella mallilla voidaan siis vähentää saman työ toistoa, vapauttaa simuloijia ja suunnittelijoita muita projekteja varten, vähentää tarpeettomia työtunteja, joita voisi kulua lähes identtisten mallien erilliseen testaamiseen ja palautukseen testauksessa ilmenneen virheen vuoksi sekä antaa realistisempi, muuttumaton yleiskuva kyseisen radan mahdollisuuksista.

Esimerkkinä yksittäiseen malliin kuluvasta ajasta ja vaivannäöstä toimii opinnäytetyön aihe, josta käy ilmi vaiheet ja tarkistukset, joiden tulee tapahtua ennen kuin malli voidaan luokitella edes etäohjelmoitavaksi. Erottamalla parametrisoinnin vaiheet työvaiheista saadaan yleiskuva radan CAD-mallin kehityksestä OLP-malliksi Visual Componentsin avulla. Raportissa läpikäytävien vaiheiden jälkeen mallin viimeistelyyn sisältyy vielä työvaiheita ennen kuin tätä käytetään virallisiin hitsausajoihin, kuten kalibroinnit ja pulssien määritykset. Nämä kuuluvat muiden osastojen työnkuvaan, koska näissä otetaan huomioon todellisessa radassa olevat rakenteelliset epätäydellisyydet. Ilman kalibrointia ja pulssimäärityksiä ohjaukset eivät tule olemaan tarkkoja, koska servot eivät tule vastaanottamaan simulaation tietoa oikeassa suhteessa.

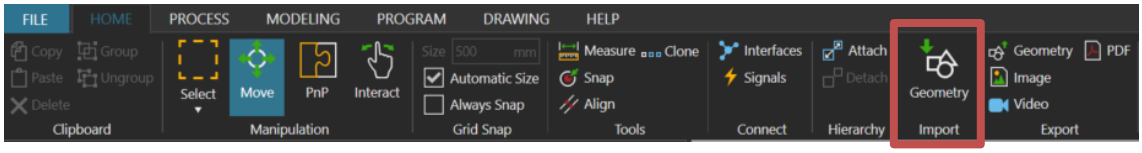
# 6 Työn tekeminen

Parametristen OLP-mallien kehittäminen alkaa standardisoiduista lattiaradan ja kääntöpöytien CAD-malleista. Nämä mallit sisältävät yksityiskohdat, jota tarvitaan yksikön teknisiin tietoihin, kuten käytettävät servomoottorit, liittimet, kiinnitysmenetelmät, osien kytkennät ja osassa malleista sylinterien iskupituudet ja komponenttien liikeradat. 3D-mallia tulee keventää mahdollisimman paljon, eli tästä tulee poistaa SolidWorksissa simulaation kannalta tarpeettomat osat ja piirteet. Työssä käytetty Visual Components 4.2 vaatii mallin olevan STEP-tiedostomuodossa, koska se ei tue vuoden 2020 jälkeisiä SolidWorks-tiedostomuotoja. Pääasiallinen sääntö tällaiselle kevennykselle on poistaa jokainen osa, joka ei hitsauksen ohjelman ajon aikana voi aiheuttaa törmäysvaaraa, mutta säilytetään visuaalinen selkeys kokonaistoiminnasta.

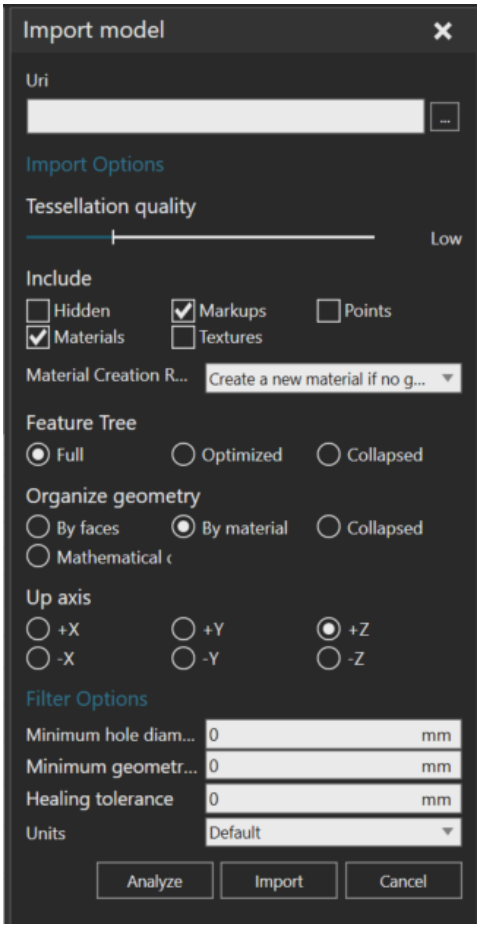
Pienet piirteet, sisäiset piirteet, pyöristykset ja viisteet ovat laskennallisesti raskaita, joten näistä tulee hankkiutua eroon ennen mallin viemistä STEP-tiedostoon, jotta voidaan saavuttaa mahdollisimman pieni tiedostomuoto, helpompi osien jakaminen osakomponentteihin sekä sulavammat simulaatioajot. Osa poistettavista piirteistä tulee sopia erikseen suunnittelijoiden kanssa, koska nämä saattavat olla asiakkaalle yksilöllisiä asioita, jotka on helppo lisätä jälkeenpäin tarvittaessa, mutta yleisessä mallissa saattavat aiheuttaa tarpeetonta ohjelmiston hidastusta tai olla visuaalisesti häiritseviä.

## 6.1 Mallin tuominen ja origon määritys

3D-mallin ollessa riittävän kevyt tuodaan STEP-tiedosto Visual Componentsin maailmaan *Import*-toiminnolla. Valitaan kuvassa 1 näkyvältä *Home*-välilehdeltä Import-osiosta *Geometry* ja valitaan haluttu STEP-tiedosto kansiosta, johon se on tallennettu. Valitaan kuvan 2 mukaiset asetukset.




Kuva 1. Visual Components työkalupalkki.

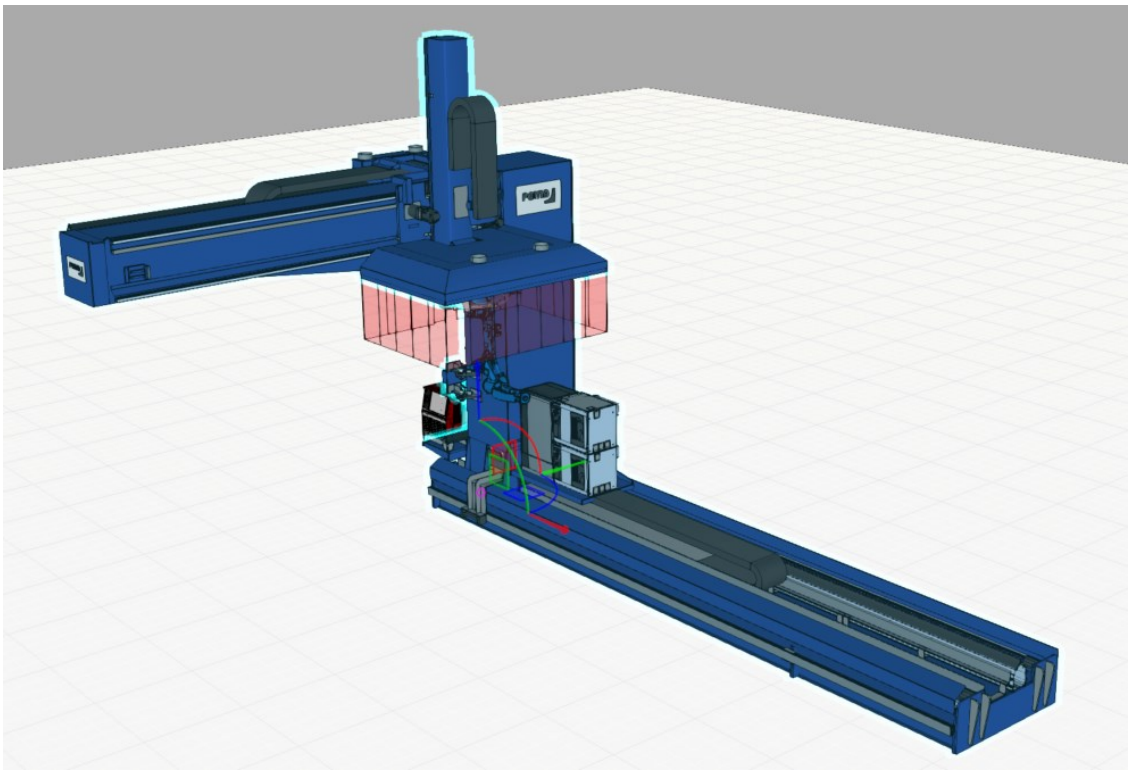



Kuva 2. *Import*-toiminnon suositellut asetukset.

*Import*-asetukset tuovat CAD-mallin riittävän yksinkertaisena, säilyttäen tämän topologian ja asettaen maailmanorigon ”Ylös”-suunnan, johon suhteessa malli tullaan asettamaan.

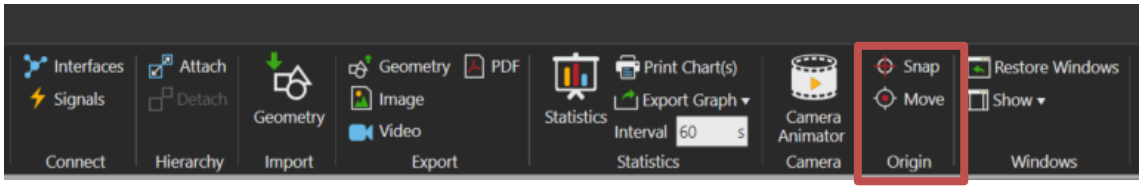
Ensimmäisenä mallin origon orientaatio tulee määrittää oikein. Malli tulee suunnata maailman origoon nähden kuten kuvan 3 tapauksessa, jossa robotti osoittaa positiiviseen X-suuntaan ja riippuen siitä, miten päin robotti asennetaan

rataan, positiivinen Z-akseli osoittaa suuntaan, joka on robottiin suhteessa ylöspäin.



Kuva 3. Mallin origon sijainti ja suunta.

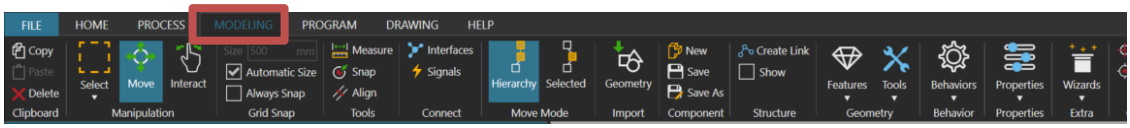
Origon muuttaminen tapahtuu kuvassa 4 näkyvän *Origin*-osion alta löytyvien *Snap*- ja *Move*-toimintojen avulla. Nämä sallivat origopisteen sijainnin muutoksen liikuttamatta osia kokoonpanossa. Kun origon orientaatio on asetettu oikein, tulee painaa *Apply* ja tämän jälkeen nollata koordinaatit ja kulmat *Component Properties* -ikkunasta painamalla X-, Y- ja Z-painikkeita ja näiden rotaatiovariantteja, jotta osien sijoitukset ja rotaatiot päivittyvät asetusten mukaisesti.




Kuva 4. *Origin*-toiminnon sijainti.

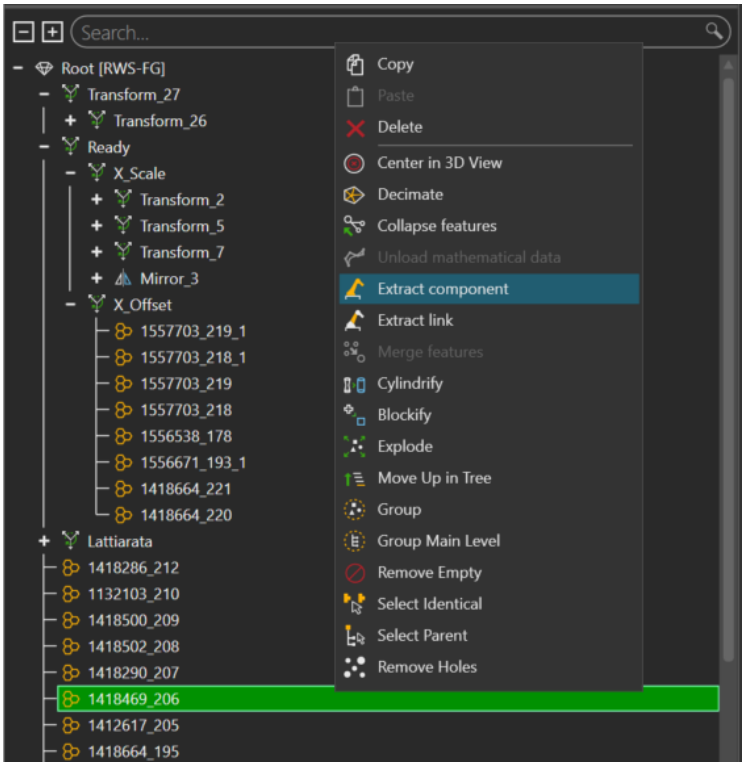
## 6.2 Jako linkkeihin ja osakomponentteihin

Kun mallin orientaatio on tehty, valitaan kokoonpano ja siirrytään *Home*-välilehdeltä *Modeling*-välilehdelle. Kuvassa 5 näkyvällä *Modeling*-välilehdellä kaikki osat, jotka olivat erillisiä piirteitä SolidWorksin puolella, ovat komponentin juuren alla valittavissa. Valitaan osia ja jaetaan näitä omiksi komponenteikseen, jotta voidaan helpommin hahmottaa osakokoonpanoja. Asettamalla erotetuille komponenteille omat origot on helpompi siirtää niitä tarvittaessa.



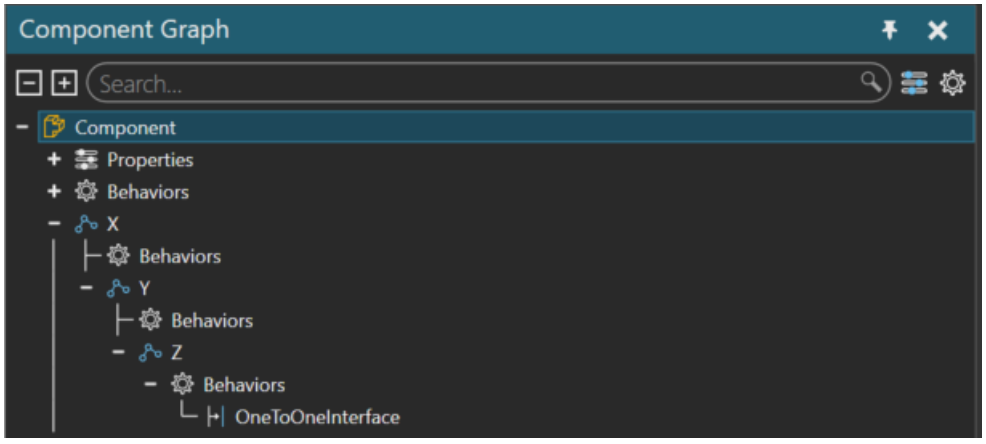

Kuva 5. *Modeling*-työkalupalkin näkymä.

Yhdessä liikkuvat osat tulee asettaa samojen *Transformien* ja linkkien (*Create link*) alle raahamalla ne hiirellä Shift-näppäin pohjassa, jotta osat eivät vahingossa liiku simulaatiomaailmassa. Lisäksi osat voidaan erottaa kokoonpanosta kuvan 6 *Extract Component* -toiminnolla, jolloin kappaleesta tulee itsenäinen *Cell Graph* -ikkunassa.




Kuva 6. Komponentin *root*-näkymä.

Asetetaan linkit liikkumissuuntineen haluttuun järjestykseen. Komponentin alla voi olla kuvan 7 mukaisesti X-, Y- ja Z-linkit, joille asetetaan vapausasteet. Asettamalla osat ensin liikeradoille saadaan varmistettua, että jokaisen liikkeen *Joint*-arvot toimivat halutusti, eli niiden arvo kasvaa niiden liikkuessa oikeaan suuntaan ja että kaikki osat ovat oikeiden linkkien alla. Kuvassa 8 näkyvässä *Link properties* -ikkunassa nähdään linkin asetukset.




Kuva 7. Komponentin linkkejä.

Link Properties:

- Name: Valitaan linkille johdonmukainen nimi.
- Offset: Arvo tai *property*, jonka perusteella linkin origo siirtyy suhteessa sen alkuperäiseen sijaintiin.
- JointType: Linkin liiketyyppi, joka määrittää liikkuuko linkki lineaarisesti, rotaationaalisesti, seuraajana toiselle linkille vai mukautetulla tavalla.

Joint properties:

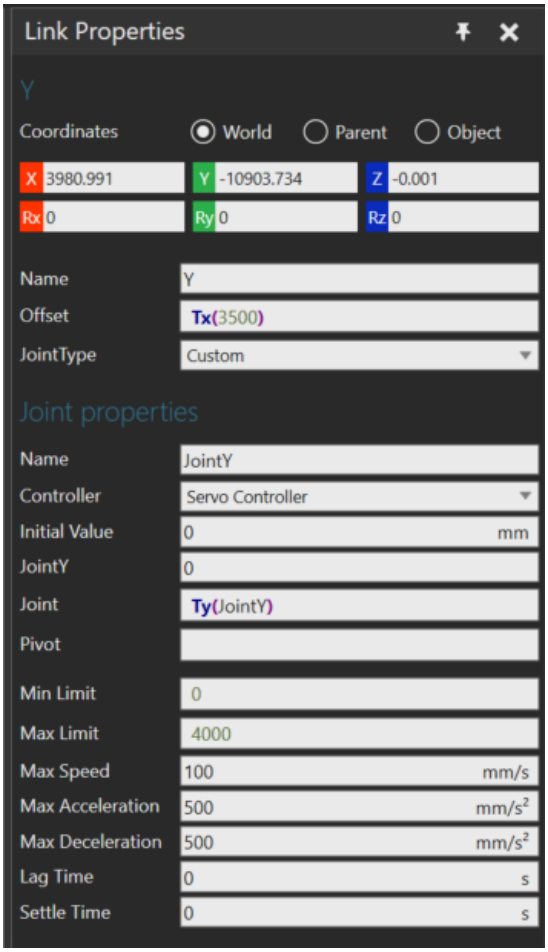
- Name: Valitaan johdonmukainen nimi, jota voidaan hyödyntää loogisesti muissa hierarkian linkeissä.
- Controller: Valitaan minkä servo-ohjaimen alle linkki asetetaan. Saman servon alla olevat linkit voivat hyödyntää toinen toistensa nivelarvoja.
- Initial Value: Määrittää linkin lähtöpisteen arvon.
- JointY: Tämä nimi muuttuu *Joint Name:n* mukaan ja kertoo nivelen ajankohtaisen arvon.
- Joint: Määrää liikesuunnan ja -tyypin ja liikkuuko nivel itsenäisesti vai seuraajana toisen linkin nivelarvon mukaan. Lineaariliikkeet määrätään *Tx*-, *Ty*- ja *Tz*-komennoilla ja pyöritysliikkeet *Rx*-, *Ry*- ja *Rz*-komennoilla. *Custom*-liiketyypillä voidaan asettaa linkki liikkumaan useammalla akselilla kerralla, verrattain *Translational*- ja *Rotational*-liiketyyppeihin, joissa valintana on vai yksi akseli (*Axis*), jonka mukaan linkki liikkuu.
- Pivot: Liikkeen koordinaatiston siirto (*Custom*-vaihtoehto).
- Min Limit: Määrittää nivelen liikeradan minimiarvon.
- Max Limit: Määrittää nivelen liikeradan maksimiarvon.

Linkin liikkeen minimi- ja maksimiarvot ovat välttämättömiä, jotta WeldControllissa ratojen liikkeiden laskenta saadaan toimimaan. Suositeltavaa olisi kehittää tapa arvojen muuttamiselle, jota on helppo myöhemmin päivittää.

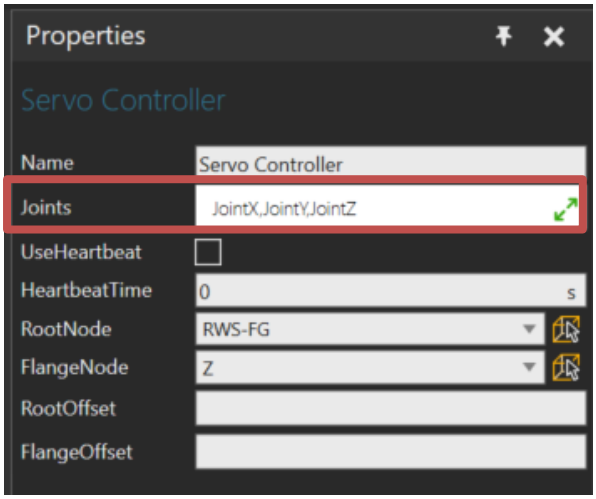
Linkkien järjestyksen pystyy varmistamaan servo-ohjaimen ominaisuuksista (Kuva 9) *Behaviors*-ketjun alta (Kuva 7). Tämä vaikuttaa esimerkiksi siihen, kuinka robotti pyrkii hakemaan omia liikkeitään ollessaan kiinnitettynä XYZ-linkin päähän. Servo-ohjaimesta näkee hierarkian viimeisen linkin

(*FlangeNode*). Jos tähän kiinnittää komponentin, tulee kiinnitetty komponentti noudattamaan viimeisen linkin arvoja.

Toisinaan virheellisesti aseteltu järjestys servossa voi olla hankalampi huomata heti, kun tehdään yksinkertaisia hitsausajoja. Ongelmat voivat ilmetä vasta kun pyritään hitsaamaan kääntöpöydällä pyöritettävää kappaletta ja huomataan, että robotti ei kykene seuraamaan pyörivää kappaletta korrektisti. Tästä syystä testiajoja on joka tapauksessa tehtävä riittävän kattavasti.

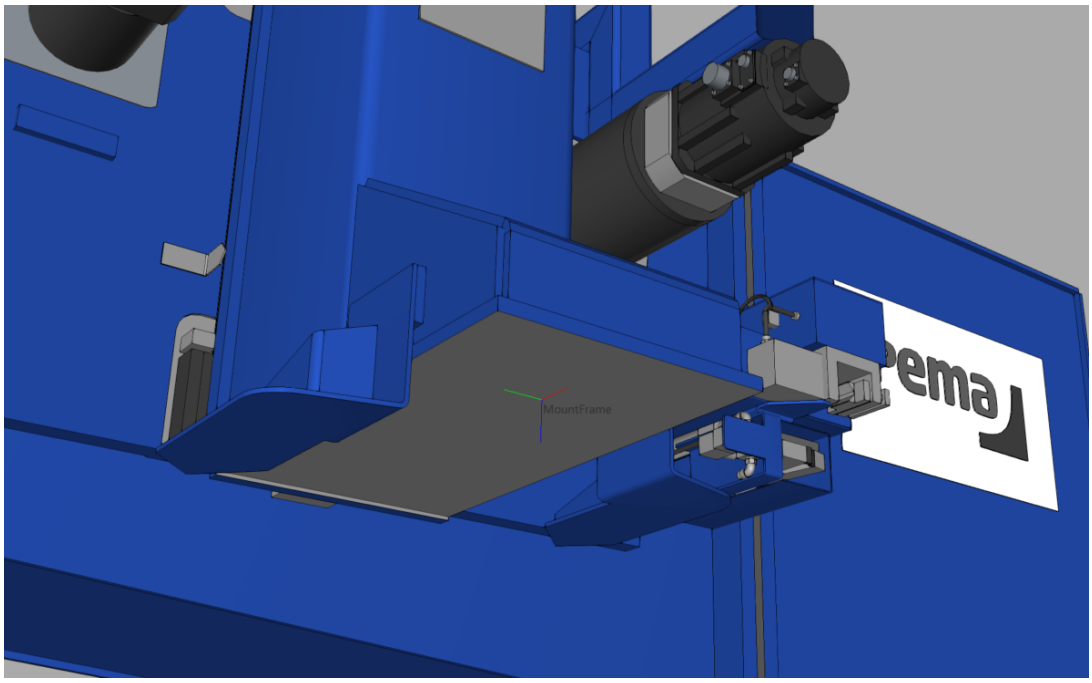



Kuva 8. *Link Properties* -ikkuna.



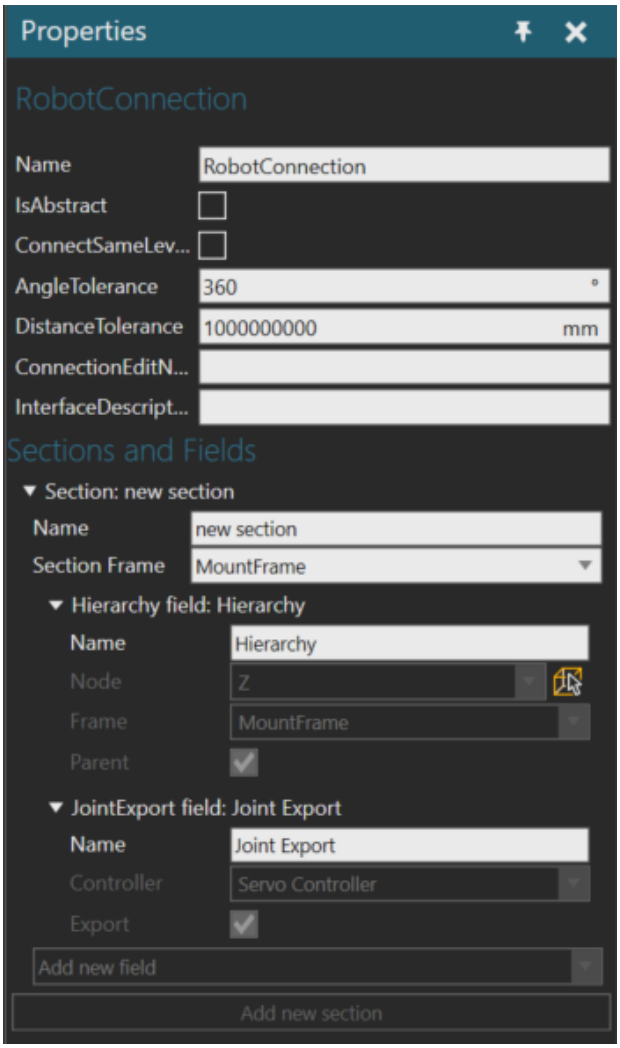

Kuva 9. Servo-ohjaimen asetukset.

Kun linkit on tehty voidaan viimeiseen *child*-linkkiin asettaa ensin *MountFrame* kohtaan, johon robotti on tarkoitus liittää (kuva 10) ja tämän jälkeen lisätä viimeiseen linkkiin *One-to-One* -ominaisuustoiminnolla *RobotConnection*. *MountFramen* orientiaatio tulee asettaa oikein, jotta signaalit välittyvät oikein robotille ja takaisin. Tämä tarkoittaa, että X-akselin tulee osoittaa robotin takaa katsottuna eteenpäin ja Z-akselin robotin pohjasta ylöspäin.




Kuva 10. *MountFrame* sijainti.

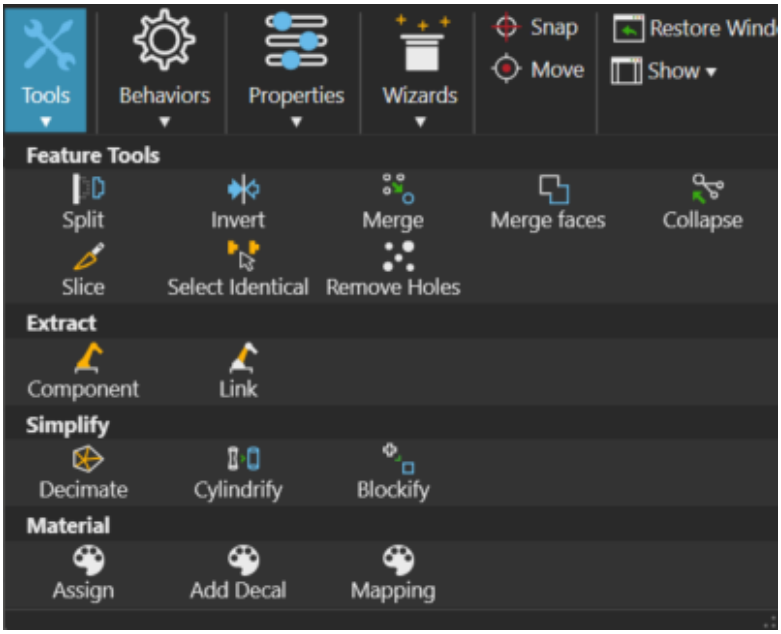
Robotti voidaan asettaa oikeaan sijaintiin ja signaalit välittää, kun *MountFrame* on kohdillaan ja *One-to-One -interfaceen* on asetettu kuvan 11 mukaiset kentät. Asetusten jälkeen siirrytään *Home*-välilehdellä ja *PnP*-valinnalla raahata Visualin omasta kirjastosta tuotu robotti paikalleen. Jos *Frame* ja *Interface* ovat oikein, tulisi robotin sijoittua oikeinpäin. Mikäli robotti kohdistuu automaattisesti väärään suuntaan, tulee tarkistaa *MountFramen* orientaatio. Jos *Interfacessa* on jotain väärin, kuten esimerkiksi viimeinen *Node* on valittu väärin, sen saattaa havaita vasta hitsausajoa tehdessä, mutta se on uudelleen muutettavissa ottamalla robotin pois paikasta.




Kuva 11. *One-to-One -interfacen* vaaditut kenttätiedot.

## 6.3 Geometrioiden muokkaus

Tässä vaiheessa on hyvä keventää enemmän osia, jotka sallivat vain rajoitetun kevennyksen CAD-mallissa. Rajoitettu kevennys osassa voi johtua esimerkiksi järjestyksestä, jossa pursotukset, poraukset ja kulmanpyöristykset on suoritettu. Osa mallien pohjapiirustuksista voi olla *Import face* -muodossa, joka ei salli edes piirustuksien muuttamista, jolloin reiät ja pyöristykset, jotka löytyvät pohjapiirustuksista, tulevat väistämättä mukaan pursotukseen. Visual Componentsin omilla *Remove Holes* -, *Blockify*- ja *Cylindrify*-työkaluilla (kuva 12), jotka löytyvät *Geometryn* alta, voidaan yrittää yksinkertaistaa pintamuotoja lisää. Toiminnot eivät toimi joka kerta, mutta voivat tarjota kuitenkin lisävaihtoehdon osien kevennykselle.



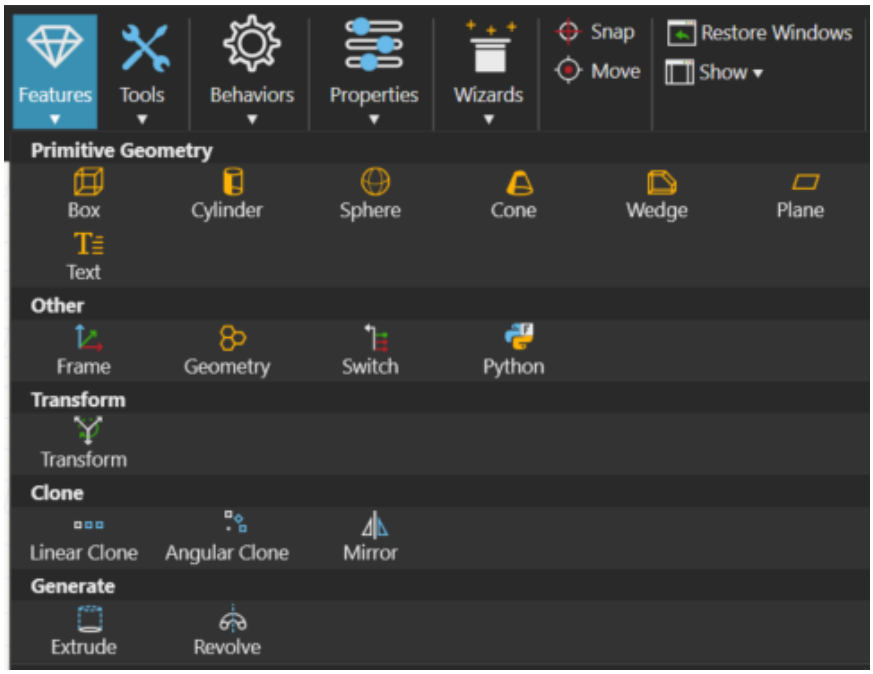

Kuva 12. Työkaluvalinnat.

Kappaleita voi tehdä itse mittaamalla osat, joihin tavalliset kevennysmetodit eivät ole toimineet ja kokoamalla Visualin omista geometriapiirteistä (kuva 13) jotakuinkin osaa vastaavan ”palikkaversion”. Tätä tehdessä tulee osata arvioida ulkoisten piirteiden merkitys simulaation kannalta. Esimerkiksi langansyöttimen tai -katkaisijan kevennys tulee suorittaa tavalla, joka ei anna virheellistä kuvaa

sen uloimmista geometrioista, kun sille suunnitellaan sijaintia, joka voi olla lähellä robotin energiaketjun mukaan ottajaa.

Joskus osat eivät ole tismalleen oikean kokoisia CAD-mallissa, jokin vaihe STEP-tiedoston tuomisessa Visual Componentsiin saattaa hajottaa tai siirtää sitä tai usea peräkkäinen osa voidaan vaihtaa yhdeksi, pidemmäksi osaksi ja tällöin voidaan hyödyntää parhaiten *Transform*-piirrettä (kuva 13). Asettamalla osa *Transformin* alle voidaan tällä muokata osan sijaintia millimetreissä tai skaalaamalla sen kokoa alkuperäiseen verrattuna.

Lisäksi *Transformia*, samoin kuin *Clone*- ja *Mirror*-toimintoa voidaan käyttää, jotta voidaan välttää liian suurta lukumäärää palikoita ja pitää kappaleet varmasti samankokoisina ja oikealla etäisyydellä toinen toisistaan. Tärkeintä parametrisen mallin kannalta on se, että *Transformiin* ja *Cloneen* voidaan asettaa *Property*-muuttujia, joka sallii näiden kokoluokan muokkaamisen *Properties*-valikosta käsin, samalla kun rajoitetaan käyttäjää asettamasta arvoja, jotka saattaisivat rikkoa piirteen toiminnan asettamalla *Property*-muuttujalle arvoväli kohtaan ”*Constraints*”.

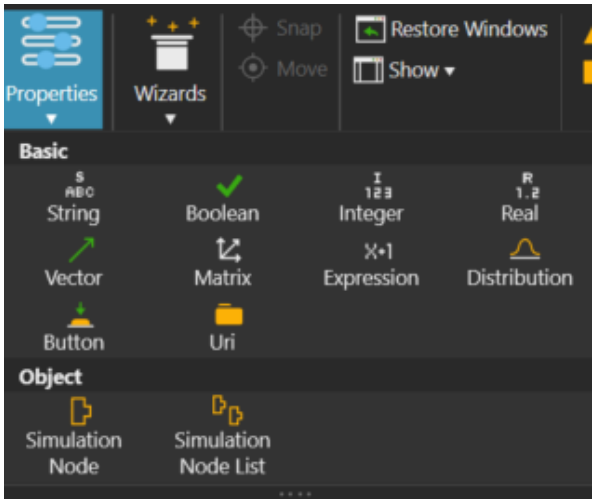



Kuva 13. Piirteiden ja niiden muokkausten valinnat.

Yhdistämällä piirteisiin *Properties*-muuttujia (kuva 14) saadaan laajin mahdollinen mallin käsittely *Modeling*-välilehden ulkopuolella.

Yleisimmät käytetyt muuttujat ja käyttö:

- String: Merkkijono. Voidaan asettaa *Transformiin* tai linkkiin dimensiot, jotka toteutuvat vain tietyn merkkijonon arvon ollessa valittuna.
- Boolean: Totuusarvo. Voidaan asettaa *Switchiin*, jonka alla olevan kappaleen/kappaleiden näkyvyys on riippuvainen totuusarvosta.
- Integer: Kokonaisluku. Käytetään geometrian dimensioiden muuttamisessa. Soveltuu *Transformin* sisäisiin laskutoimituksiin, koska liian suuret jakojäännökset saattavat aiheuttaa virheen ja mitätöidä koko yhtälön vaikutuksen dimensioon.
- Real: Reaaliluku. Käytetään geometrian dimensioiden muuttamisessa, kun kaivataan tarkempia arvoja. Soveltuu erityisesti geometrian siirtämiseen.
- Expression: Lauseke/Ilmaisu. Tähän voidaan yhdistää kahden tai useamman muuttujan laskutoimitus, jotta voidaan lisätä yhden parametrin avulla useamman muuttujan vaikutus geometriaan.
- Button: Kytkin. Voidaan käyttää aktivoimaan *PythonScript*, joka sisältää komponentti- ja ominaisuushaun.




Kuva 14. Perusmuuttujat.

## 6.4 Robotin valmistelu

Luodaan robottiin, sijainnissa *Behaviors→DX200→Bases* itse koordinaatistot ”ROBOT”, ”BASE” ja ”ST1”. ST1 voi olla toistaiseksi *nodessa* ”WORLD” ja ROBOT ja BASE tulee sijaita robotin *PnP*-liitoskohdan keskellä, robotin ensimmäisen nivelen korkeudella ja muuttuu korrektisti aseman parametreja muutettaessa.

ROBOT-koordinaatisto on kiinni robotissa, joka seuraa Z-palkin päässä sijaitsevaa *MountFramea*, eli tämä tulee siirtymään oikeaoppisesti tavallisesti lattiaradan parametrejä muuttamalla. ”ROBOT” toimii koordinaatistona vain robottikädelle. Koska BASE-koordinaatiston *node* on WORLD tämä ei tule liikkumaan portaalin geometrioita tai linkkejä liikuttamalla. Sen sijaan BASE:n ominaisuuksia voidaan muuttaa tietämällä ja etsimällä *PythonScriptin* avulla robotin komponentti simulaatiomaailmassa, tämän käyttäytymiset ja tarkat polut BASE-koordinaatiston luokse. Tällöin voidaan muuttaa BASE-koordinaatiston arvoja aktivoimalla *PythonScript,* joka etsii tuon robotista ja muuttaa sen positiomatriisin arvoja määrätyllä tavalla.

Koordinaatistojen ROBOT ja BASE tulee olla samassa pisteessä, robottiportaalin ollessa aloituspisteessään, jotta simulaatio kykenee käyttämään näiden kahden koordinaatiston positiomatriisien erotusta laskeakseen robotin sijainnin simulaatiomaailmassa.

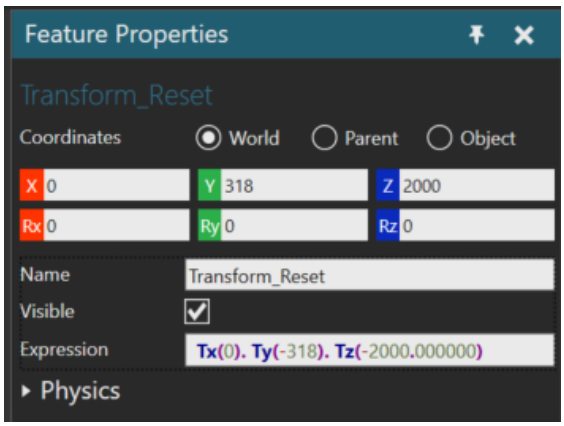
## 6.5 Kääntöpöydän valmistelu

Tehdään erillinen komponentti kääntöpöydästä. Kääntöpöydän tekeminen alkaa robottiportaalin tavoin CAD-mallin kevennyksestä ja tämän STEP-tiedoston tuomisesta Visual Componentsiin, uuteen tiedostoon. Pöydät ovat tavallisesti joko yksi- tai kaksinivelisiä ja niissä tulee suorittaa linkkien luominen ja järjestäminen samoin kuin robottiportaalissa, mutta *Custom*-liiketyypin sijaan linkille asetetaan liiketyypiksi *Rotational*. Tämä edellyttää linkin origon

asettamista oikeaan sijaintiin, oikeassa orientaatiossa. Linkkiä voi siirtää kahdella tavalla:

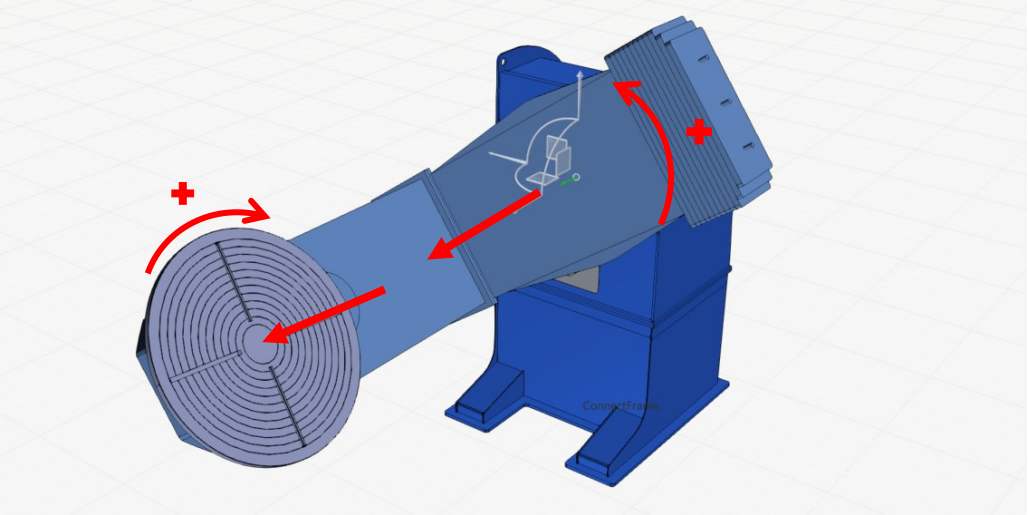
- Snap: Napsauta linkin origo kääntöakselin keskipisteeseen ja jos koordinaatiston orientaatiolla on väliä, tulee pyöräytysasteet nollata.
- Move: Mittaa origon tarkoitetun sijainnin etäisyys maailmanorigosta ja aseta nämä mitat *Offset*-arvoihin. Tämä säilyttää origon alkuperäisen orientaation, mutta sallii myös sen muutoksen tarvittaessa.

Jos linkin alla on jo tuotu geometrioita voi pelkkää origoa siirtää valitsemalla *Move Mode* -osiosta *Selected*. *Selected* sallii siirtää vain tämänhetkistä valintaa. Kun linkin origo on halutussa paikassa ja rotaatiot nollattu vaihdetaan takaisin *Hierarchyyn*, jolloin valintaa siirtämällä siirretään myös kaikkea tämän alle sijoitettua linkeistä geometrioihin. Hieman hankalampi, mutta toimiva tapa siirtää osia on kuvan 15 mukaisesti asettaa ensin origo ja osien ollessa *Transformin* alla näitä voi siirtää asettamalla *Transformin Expressioniin* linkin origon vastaluvut.



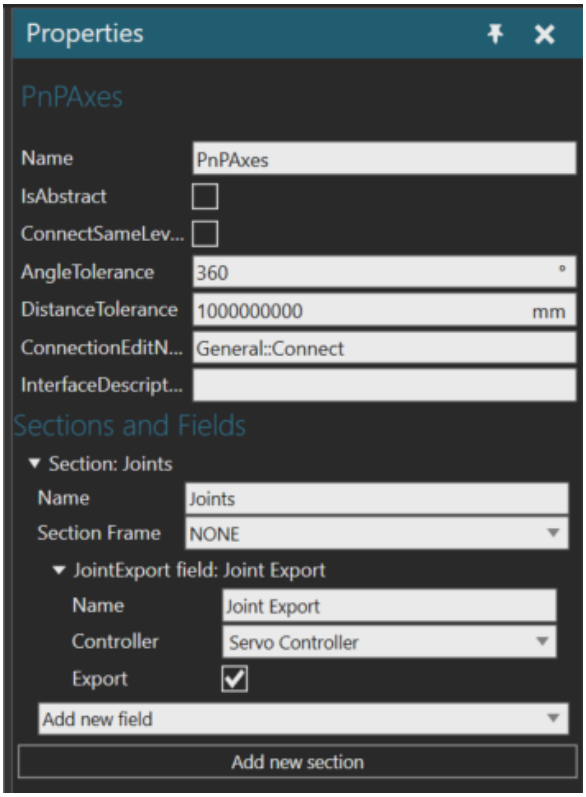

Kuva 15. Siirto *Transform*-lausekkeella.

Kun origo on asetettu oikeaan pisteeseen, tulee valita pyöritysakseli ja tälle pyörityssuunta oikean käden säännön perusteella (Kuva 16). Peukalo osoittaa ulospäin ensimmäisen akselin liitospinnasta ja muut sormet osoittavat positiivisen pyörityssuunnan. Toisen akselin, eli tässä tapauksessa pöytätason, kanssa peukalo osoittaa pintaa kohti.



Kuva 16. Linkin pyöritysakselin sijoitus ja pyörityssuunta.

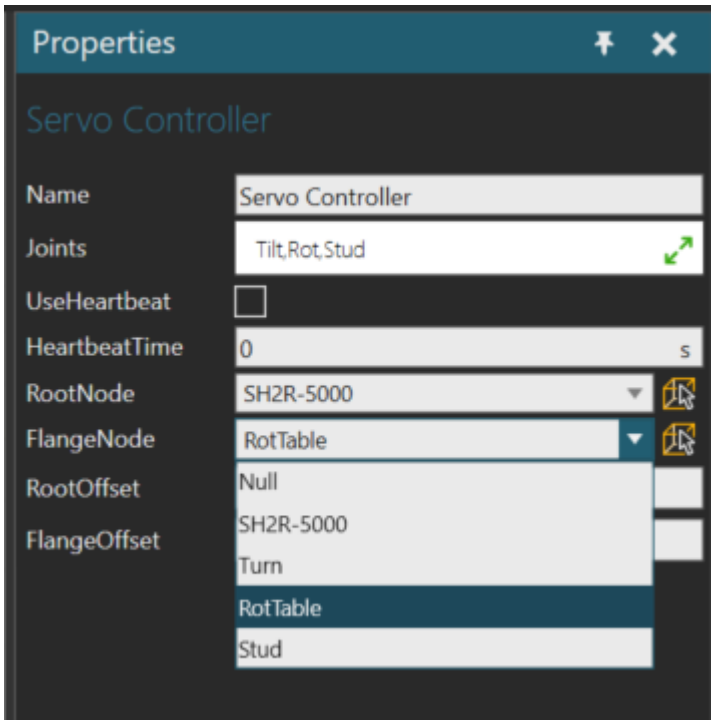
Kääntöpöytä tarvitsee signaalien siirtämistä varten *Interfacen*, joka voidaan yhdistää robottiin. Valitaan komponentin *RootNode*, lisätään *Behaviors*-valikosta *One-To-One -Interface* ja asetetaan tälle kuvan 17 mukaiset asetukset.




Kuva 17. Kääntöpöydän *Interface*-asetukset.

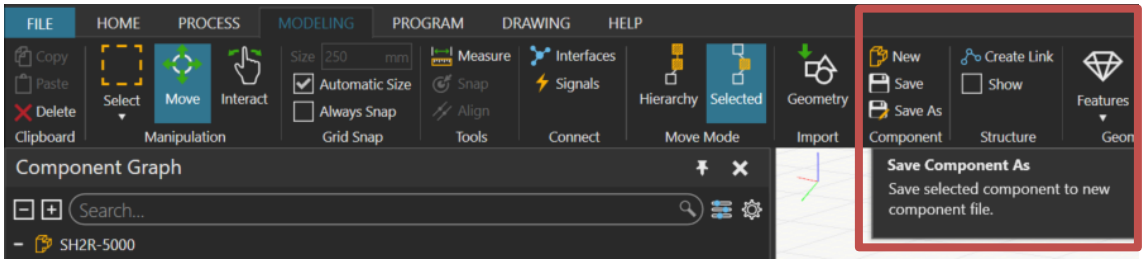
*Interface* ei ole tällä kertaa viimeisen linkin *nodessa*, johtuen järjestyksestä, jolla haluamme simulaation ohjaavan kääntöpöydän akseleita hitsatessaan kappaletta. Annetaan *Interfacelle* nimeksi “PnPAxes” ja *ConnectionEditName*-kohtaan nimeksi “General::Connect”. Kohta “General::” nimessä tuo *Interfacen* esille kun *Home*-välilehdellä valitaan komponentteja ja pyritään liittämään niitä toisiinsa ja ”Connect” on yksinkertainen, kuvaava nimeämistapa *Interfacen* toimintaperiaatteelle. Lisäys ”ConnectionEditName”-kohtaan takaa, että ainoastaan liitäntäpaikat, joiden halutaan näkyvän komponentteja yhteen liitettäessä, esiintyvät *Interface*-valikossa.

Pöydän signaalit siirretään *JointExportin* avulla, jonka alle on asetettu servo-ohjaimen ominaisuuksien hallinta. Tämä sallii ulkoisen komponentin pääsyn servo-ohjaimen alle asetettuihin niveliin. Jälleen tulee varmistaa, että servossa viimeiseksi *nodeksi* on valittu kääntöpöydän viimeinen akseli, ei komponentin viimeiseksi lisätty linkki. Kuvan 18 esimerkkitapauksessa nähdään kuinka *FlangeNode*-listan viimeinen vaihtoehto ei ole kääntöpöydän viimeinen akseli, mutta koska se on lisätty näiden jälkeen, se näkyy listassa alimmaisena.




Kuva 18. Kääntöpöydän servon asetukset.

Komponentin ollessa valmis tallennetaan tämä kuvan 19 mukaisesti ”*Save Component As*”-valinnalla haluttuun tiedostosijaintiin. Tällä tavalla tallennetaan komponentti ja kaikki siinä *Attachilla* kiinni olevat osat.



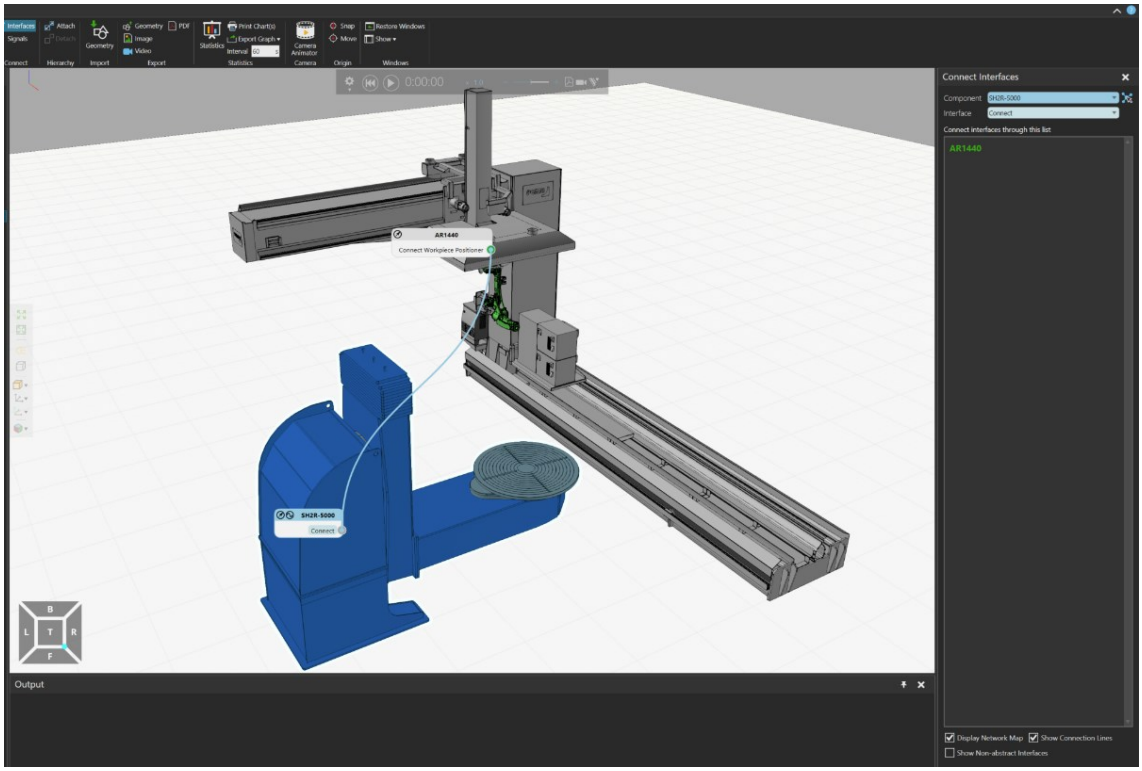

Kuva 19. Komponentin tallennus.

## 6.6 Kokoonpanon asettelu

Komponenttien ollessa valmiita kootaan ne kaikki samaan malliin. Tämä tehdään avaamalla uusi, tyhjä malli ja raahataan tallennetut komponentit joko *eCatalogista*, jos ne on lisätty tiedostosijaintiin, johon *eCatalogilla* on pääsy, tai raahataan ne suoraan tallennuskansiosta resurssienhallinnasta. Siirretään rata ja kääntöpöytä suurin piirtein niiden tarkoitettuun asetteluun. Raahataan *PnP*-valinnalla robotti kiinni portaaliin ja lisätään tähän työkalu. Asetetaan kääntöpöytä oikeinpäin, eli siten miten se tulisi olemaan todellisen maailman tarkastelussa. Mitataan etäisyydet ja asetetaan tarkat koordinaatit origon koordinaatteihin. Valmiina oleviin arvoihin voi summata tai niistä erottaa mittauksen tulokset.

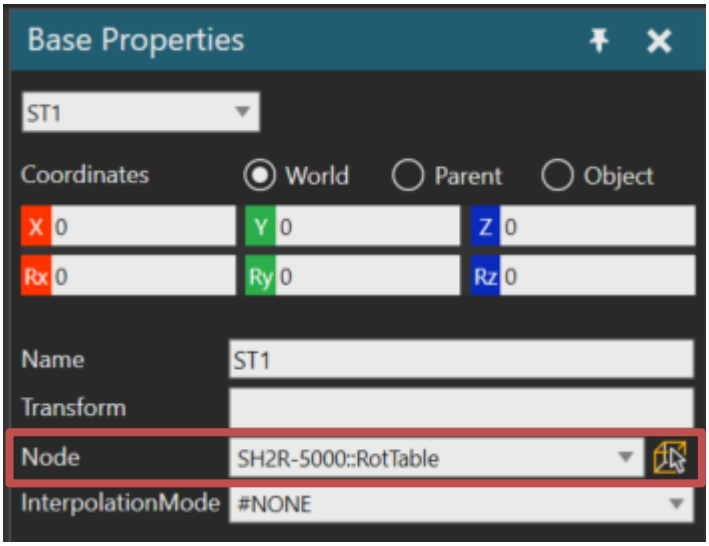
Osien välillä pääsee vaihtamaan *Cell Graph* -ikkunassa, *Home*-välilehdellä, ja näiden tarkempia ominaisuuksia pääsee edelleen muuttamaan *Modeling*-välilehdellä. Muutosten tekeminen ei tule päivittämään mitään alkuperäiseen tallennettuun tiedostoon, ellet ole tallentanut komponenttia ”*Save Component As*”-metodilla ja päivitä sitä uudessa asettelussa ”*Save*”-komennolla samasta *Component*-valikosta, ja tällöin kannattaa tarkistaa, että tallennuskohde on edelleen sama tai Visual Components saattaa tallentaa komponentin omaan vakiokansioonsa.

Yhdistetään kääntöpöytä kuvassa 20 näkyvällä *Interface*-näkymällä. Liitetään ”Connect” kohteeseen ”Connect Workpiece Positioner” raahaamalla liitäntä pisteestä pisteeseen tai napsauttamalla molempia kohteita. Robotissa voi olla näkyvillä 2 liitäntämahdollisuutta ja väärän valinnan voi peruuttaa uudestaan napsauttamalla pistettä.




Kuva 20. Komponenttien *Interface*-liitäntä.

Radan ja pöydän ollessa paikallaan ja *Interfacen* ollessa liitetty valitaan komponentiksi robotti ja siirrytään *Modeling*-välilehdelle, jossa valitaan *Behaviors→DX200→Bases→*ST1. ST1:n *Base Properties* -ikkunassa (kuva 21) valitaan kohtaan *Node* kääntöpöydän pöytätason linkki joko valitsemalla se liukuvalikosta tai ottamalla sen oikealta puolelta aktiiviseksi *Property Pick* ja napsauttamalla kursorilla kohdetta simulaatiomaailmassa.




Kuva 21. ST1-koordinaatiston asetukset.

## 6.7 Tarkastuslista

Visual Componentsin puolella voi suorittaa seuraavia välivaiheita, jotta voidaan varmistua asetelman toiminnasta. Näiden jälkeen solun toiminta riippuu vain WeldControllin puolella tehtävistä työvaiheista ja muutoksista. WeldControl on tehty Visual Componentsin pohjalta, eli sillä on mahdollista tehdä perustason muutoksia malliin *Home*-välilehdellä, mutta koska *Modeling*-ominaisuuksien käyttö edellyttää Visual Componentsin Premium-lisenssiä ja ohjelmiston tietotaitoa, ei tavallisella käyttäjällä ole keinoa muuttaa *nodeja* tai *Interface*-asetuksia, joten nämä tulee hoitaa kuntoon ennen siirtymistä pois Visual Componentsista.

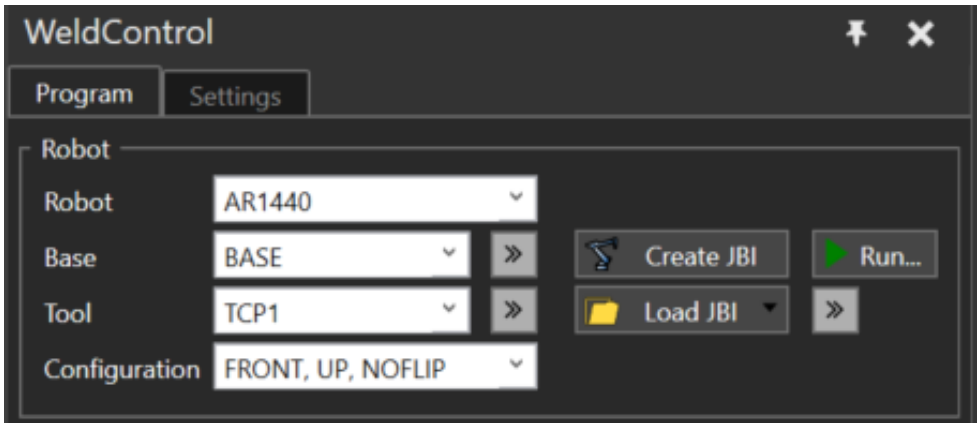
OLP-mallin toiminnan varmistaminen:

1. Katso että robottiportaalin ja kääntöpöydän linkit ovat kaikki halutussa järjestyksessä ja että servo-ohjaimessa niiden järjestys on sama, ja *RootNode* ja *FlangeNode* ovat oikein.
2. Molemmissa komponenteissa on tehty *One-To-One -interface* oikein ja ne sijaitsevat oikean *Noden* alla, eli portaalissa sijainti *FlangeNode* ja kääntöpöydässä *RootNode*.

3. Kääntöpöytä on yhdistetty robottiin *Interface*-toiminnolla, ”Connect Workpiece Positioner” -kohtaan.
4. Robotissa, *Modeling*-puolella ST1-koordinaatiston *node* on asetettu kääntöpöydän pöytälevyn linkkiin (Polku *Behaviors→DX200→bases*).
5. Aseta työkappale kääntöpöydän pöytätasoon ja yhdistä se *Attach*-toiminnolla siihen.
6. Tarkista *Interact*-toiminnolla, että kappale kääntyy pöydän mukana ja positiivinen pyörimissuunta on oikein.
7. Tallenna mallin tiedosto kaikkien liitettyjen komponenttien kanssa. Jos käyttäjällä ei ole WeldControllissa samoja käyttöoikeuksia verkkolevyille kuin mallintajalla Visualissa, ei malli kykene lataamaan kaikkia piirteitä ja voi jäädä osittain tyhjäksi.

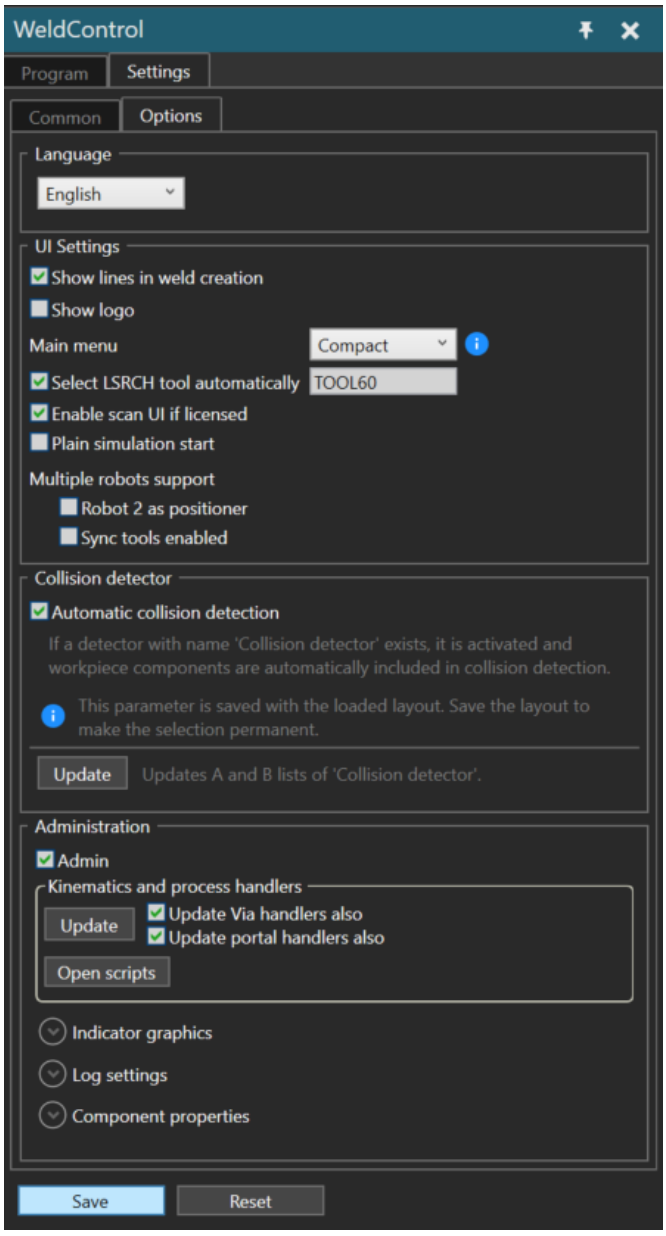
## 6.8 Mallin valmistelu WeldControllissa

Mallin ollessa valmis ja tallennettu avataan se WeldControl-ohjelmistolla. Varmistetaan että kuvassa 22 näkyvässä *Base*-laatikossa on peruskoordinaatistovalintana BASE ja *Tool*-laatikossa oikea työkalupiste, joka esimerkkitapauksessa on TCP1. Nämä määrittävät robotin aloituspisteen ja sallivat robotin tarkan poltinpään seurannan.



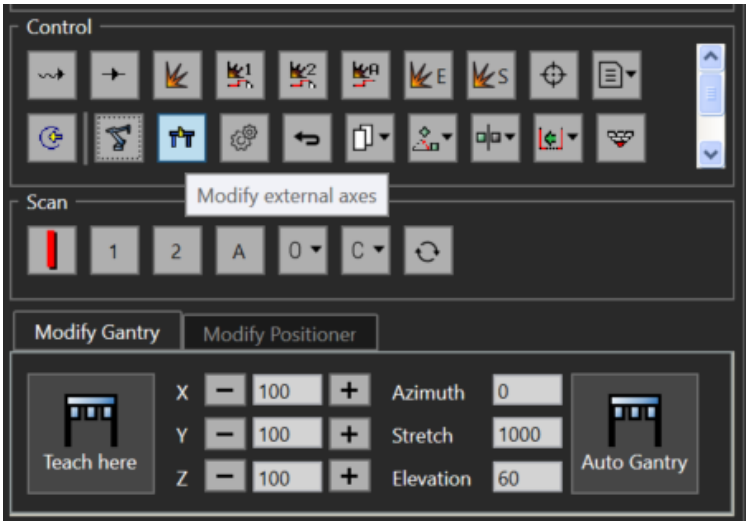

Kuva 22. Robotin asetukset WeldControllissa.

Kun nämä on valittu, siirry kohteeseen Settings→Options→Administration (Kuva 23) ja valitse laatikot ”Admin”, sen esille tuomat laatikot ”Update Via Handlers also” ja ”Update portal handlers also” ja päivitä kinematiikat *Update*-napista. Tämän jälkeen tallenna asetukset *Save*-napilla ja tallenna koko tiedosto *File*-valikosta ja avaa malli ohjelmassa uudelleen. Mallin avaaminen uudelleen ei ole välttämättä tarpeellinen välivaihe, mutta toimii varmistuksena muutosten ja päivitysten käyttöönotolle.



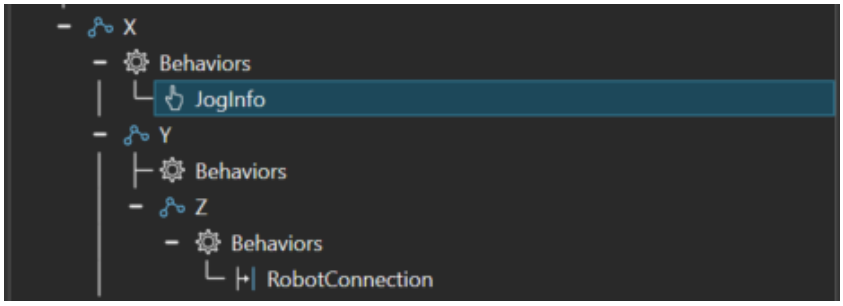

Kuva 23. Kinematiikkojen päivitys.

Ulkoisten liikeakselien oikeat suunnat voidaan varmistaa testaamalla niitä valitsemalla *Program→Controls→Modify external axes*. Kuvassa 24 näkyvistä *Modify Gantry* ja Modify positioner-valikoista voidaan muuttaa *Joint*-arvoja ja mikäli arvoja muuttamalla saadaan tarkoitetut nivelet liikkumaan, on nivelten toiminta määritetty oikein.



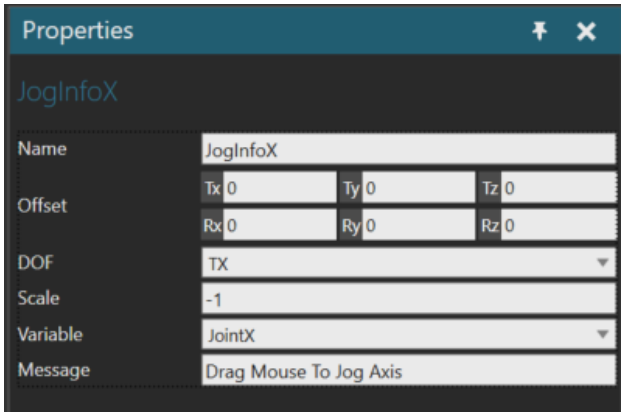

Kuva 24. Ulkoisten akselien hallinta.

WeldControllin puolella akseleiden liikuttaminen suoritetaan *Jog*-toiminnolla, joka on toimintaperiaatteeltaan samankaltainen kuin Interact. Mikäli Jog-toimintoa käyttäessä vaikuttaa siltä, että jokin akseli liikkuu eri suuntaan kuin hiiri, kannattaa loitontaa kuvaa mallista ja kokeilla uudestaan, että kyseessä ei ole vain hetkellinen bugi. Jos ongelma jatkuu, voi vialliselle linkille luoda ”*Jog Info*”-ominaisuuden Modeling-*välilehdellä* valitsemalla ensin linkin ja sitten ominaisuuden *Behaviors*-valikosta. Tämä luo *Jog Infon* linkin alle (kuva 25) ja tämän asetuksilla voidaan kääntää linkin liike suhteessa hiireen.



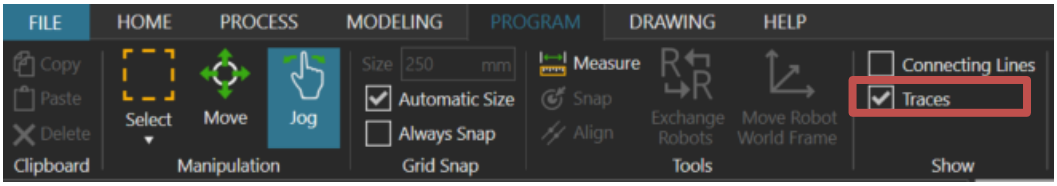

Kuva 25. *JogInfo* linkin alla.

Kuvassa 26 näkyy, kuinka linkille on määritetty *JogInfoon* asetukset. *DOF* (*Degrees Of Freedom*) eli liikevapaudet on rajoitettu arvoon *TX* (*Translational X*) eli lineaariselle X-akselille. Asettamalla *Scale*-arvoksi -1 voidaan muuttaa *JogInfon* saama signaali käänteiseksi ja muuttujaksi vastaanotettavalla signaalille halutaan linkin *Joint Name*, joka tulisi näkyä *Variablen* liukuvalikosta.



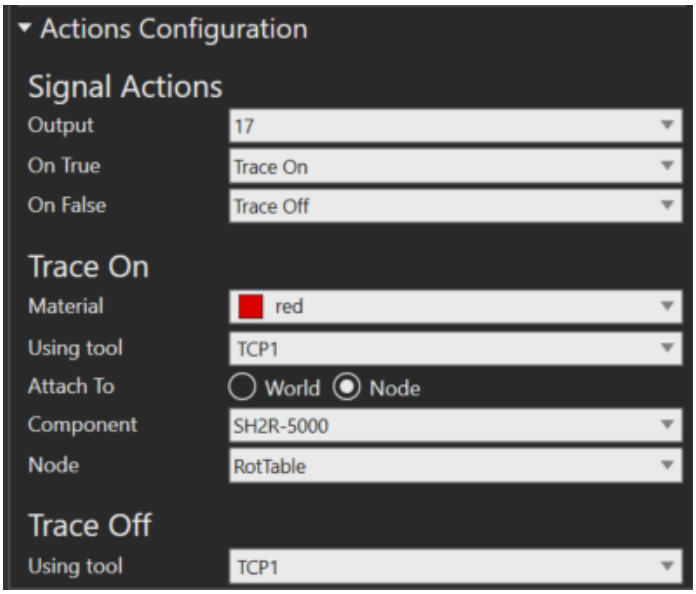

Kuva 26. *JogInfo* asetukset.

Jos hitsausohjelmaa ajaessa haluaa seurata tarkasti poltinpäätä, tulee palata vielä Visual Componentsin puolelle asettamaan ”*Traces*” aktiiviseksi *Program*-välilehdeltä (kuva 27). Tämä vaihe edellyttää, että käytetyissä komponenteissa on jokin työkalupiste, jota seurata.




Kuva 27. *Traces Program*-välilehdellä.

Valinnan ollessa aktiivinen ohjelma piirtää työkalun keskipistettä (*TCP*) jäljittävän viivan. *TCP*:n tulisi löytyä tietyistä komponenteista automaattisesti, kuten robotista ja polttimista, tai sen voi luoda itse luotuun komponenttiin. *TCP* sijaitsee polttimen komponentin viimeisessä linkissä, poltinlangan päässä, joten jos *Traces* piirtyisi väärin, olisi jotain polttimen liitännässä tai geometriassa väärin ja tämä aiheuttaa törmäysriskin. Valitaan siis robotin komponentti ja Component Properties -ikkunan alta löytyvä *Actions Configuration* ja sieltä valitaan seuraavat asetukset.




Kuva 28. Toimintojen *Tracer*-asetukset.

*Output* ”17” määrittää signaalin hitsiä jäljittävälle viivalle ja ”18” määrää signaalin kappaleen haulle. Molemmille tulee asettaa samat asetukset. *On True* ja *On False* määrittelevät onko *Tracer* päällä signaalin ollessa aktiivinen tai epäaktiivinen eli vakioltaan ne tulisi olla oikein. *Material* määrittää *Tracerin* värin

tai materiaalin ja *Using tool* työkalun, jonka mukaan *Tracer* piirtyy. *Using tool* on tässä tapauksessa ”TCP1” ja se asetetaan ”*Tracer On*”- ja ”*Tracer Off*”-valintoihin, jotta ohjelma tietää myös keskeyttää piirtämisen. Valitaan *Attach to* -kohtaan *Node*, jonka jälkeen etsitään listasta käytettävän kääntöpöydän komponentti ja tämän viimeinen linkki, jotta Tracer tulee seuraamaan oikeaa työpöytää ja tämän pöytätasoa. Näiden toimenpiteiden jälkeen malli on simulointiosaston puolesta valmis ja siirretään seuraavalle osastolle pulssien määritykseen ja kalibrointiin.

# 7 Mallin parametrisuuden parantaminen

Opinnäytetyössä tehtiin Visual Componentsissa yleiset työvaiheet, joita sisältyy OLP-mallin tekemiseen, mutta usea välivaihe tuli tehdä ajatuksella, että malliin tulee pystyä lisäämään muutoksia parametrien muutoksilla. Linkit ja geometriat tuli valmistella tavalla, joka sallii näiden siirtämisen käyttäjän syötöillä. Käyttäjäsyötöt eivät kuitenkaan saa muuttaa tai kadottaa vääriä geometrioita, siirtää linkkejä tai häiritä ohjelman toimintaa simuloidessa. Osa parametrien testauksista pystyttiin suorittamaan vasta kun robotti ja kääntöpöytä oli yhdistetty lattiarataan, jolloin perinteinen ominaisuuksien testaus jouduttiin suorittamaan epätavallisessa järjestyksessä, koska esimerkiksi rajapintaliitäntöjen (*Interface*) muokkaus ei ole mahdollista, kun siihen on yhdistetty *Child*-komponentti.

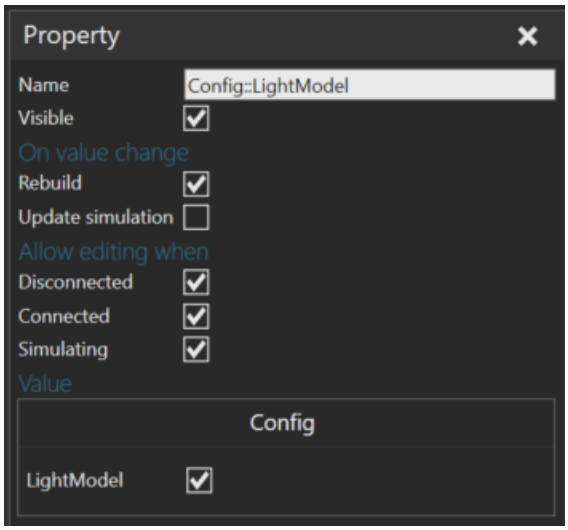
Parametrien asettaminen järkevästi rakenteellisesti ja visuaalisesti auttaa operaattoreita, jotka tulevat käsittelemään ohjelmaa lisenssillä, joka ei salli mallien muokkausta Modeling-välilehdellä. Monissa näistä malleista kuitenkin tulee muistaa ensisijaisesti standarditavat nimetä muuttujia, jotta operaattoreiden ei tarvitse liikaa ajatella näiden toimintaa joko käyttääkseen niitä oikein tai osatakseen välttää niitä, mikäli muuttujat eivät ole heille tarpeellisia. Jos tarvitsee kehittää uusia muuttujia, joita ei aikaisemmissa malleissa ole vielä ollut käytössä niin tulee pyrkiä nimeämään muuttujat mahdollisimman selkeillä tavoilla. On myös suotuisaa pyrkiä asettamaan uusia ominaisuuksia vanhojen muuttujien alle, jos tämä on mahdollista ilman aikaisemman muuttujan toimintaperiaatteen muuttamista.

Esimerkiksi jos malliin halutaan lisätä uusi vaihtoehto virtalähteelle; Tämä tulee pyrkiä lisäämään aikaisemmin olemassa olevaan valikkoon ja mielellään samaan sijaintiin mallissa kuin missä muut virtalähteet ovat. Ensisijaisesti tehdään vain lisäys valikkoon, jonka ei ole siis tarkoitus tehdä mitään muutoksia siihen, kuinka valintaa luetaan tai kuinka fundamentaalisesti geometrian muutos mallissa tapahtuu. Jos mallissa olisi kuitenkin aikaisemmin ollut vain yksi virtalähde ja vaihtoehto virtalähteelle on ollut ”ShowPowerSupply”, eli on valittu

virtalähteen näyttämisen ja piilottamisen välillä boolean-muuttujalla on vain loogista tehdä tämän tilalle merkkijonomuuttuja, jossa valitaan vaihtoehtojen ”Ei virtalähdettä”, ”Virtalähde A” ja ”Virtalähde B” välillä. Tässä tapauksessa nimi tulee muuttaa johonkin paremmin muuttujaa kuvaavaksi, kuten ”PowerSupply”, ja sijoittaa se valikossa vanhan vaihtoehdon tilalle, jolloin huolimatta muuttujan tyyppimuutoksesta sama käyttötarkoitus on loogisesti johdettavissa, johtuen sen nimestä ja sijainnista valikoissa.

## 7.1 Muuttujavalikot

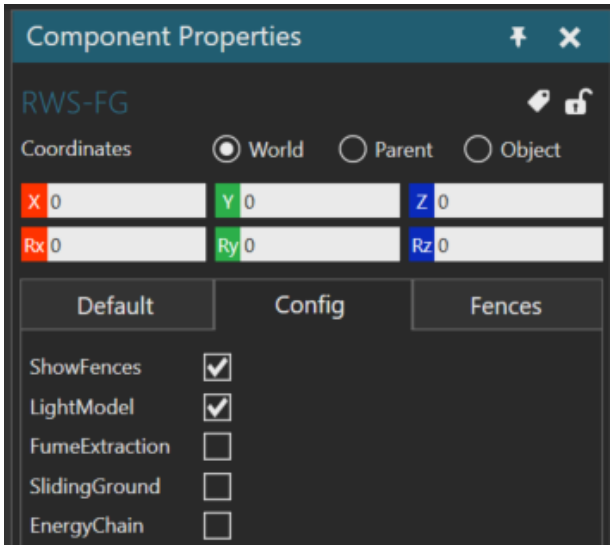
Visual Componentsissa muuttujia lisätessä niitä voi nimetä tavalla, joka luo komponentille uuden *Property*-ikkunan ja tässä työssä lisäsimme erilliset ikkunat Visuaalisten elementtien konfiguraatiolle ja *PythonScriptillä* luotavien aitojen jatkomuutoksille. Aloittamalla muuttujan nimen kuten kuvan 29 esimerkissä nimellä ”Config::” voidaan muuttuja asettaa valikon ”Config” alle. ”::”-komento luo valikon ja nimeää valikon sen vasemmalla puolella olevan sanan mukaiseksi ja oikealle tulee nimi, jolla muuttuja näkyy valikossa.




Kuva 29. totuusarvomuuttujan luominen.

Valikoita voi luoda useamman, kuten kuvan 30 tapauksessa, jos kokee tämän selkeyttävän valikoita visuaalisesti tai loogisesti. Esimerkiksi halutessaan voi

asettaa kaikki totuusarvomuuttujat omaan valikkoonsa ja reaaliluvut omaansa. Muuttujien järjestystä valikossa voi muuttaa raahaamalla niitä *Modeling*-puolella *Properties*-osion alapuolella ja jos haluaa poistaa mahdollisuuden muokata muuttujaa, mutta säilyttää sen myöhempää käyttöä varten voi tämän piilottaa valikosta näkyvistä poistamalla valinnan laatikosta *Visible*.

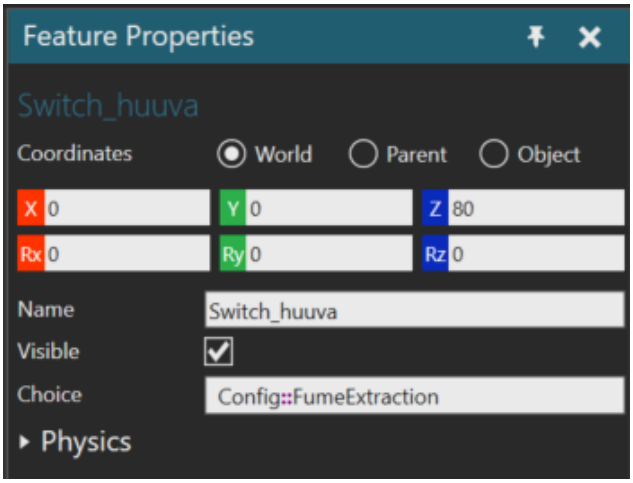



Kuva 30. *Components Properties* -valikoiden jaottelu.

## 7.2 Totuusarvot

*Boolean*- eli totuusarvomuuttuja ei yksin kykene suoraan vaikuttamaan geometrioihin, vaan sen tulee vaikuttaa kappaleeseen *Switch*- eli vaihtopiirteen tai *PythonScript*-komentokielen avulla. Kun geometrian asettaa vaihtopiirteen alle tämä muuttuu näkymättömäksi. Vaihtopiirre vaatii jonkin kokonaisluvun valinnakseen (*Choice*) ja koska valinnan vakiotila on 0 ei se alkutilassa näytä mitään. Ensimmäinen sen alle pistettävä geometria vaatii näkyäkseen *Choice*-arvon 1 ja toinen geometria arvon 2. Sama toistuu järjestyksessä seuraavaksi lisättyjen piirteiden kanssa. Jos tietyn geometrian, joka löytyy vaihtopiirteen alta, haluaa esille, tulee tietää sen järjestysnumero ylhäältä alaspäin katsottuna ja asettaa se valinnaksi. Totuusarvomuuttuja, kuten kuvassa 31, on yksiselitteisiä toimintoja varten parempi, kun esimerkiksi halutaan käyttäjältä kysyä vain, että näkyykö geometria vai ei. Totuusarvomuuttujalla on numeeriset arvot, *False* = 0 ja *True* = 1, eli se toimisi tässä tapauksessa yksinkertaisena kytkimenä geometrian näkyvyydelle. Jos haluaa saada usean eri geometrian piilotettua

samalla komennolla voi nämä siirtää ensin *Transfrom-* eli muunnoslausekkeen alle ja siirtää tämän vaihtopiirteeseen, jolloin kaikkien lausekkeen alla olevien geometrioiden näkyvyys riippuu samasta vaihtopiirteen arvosta.

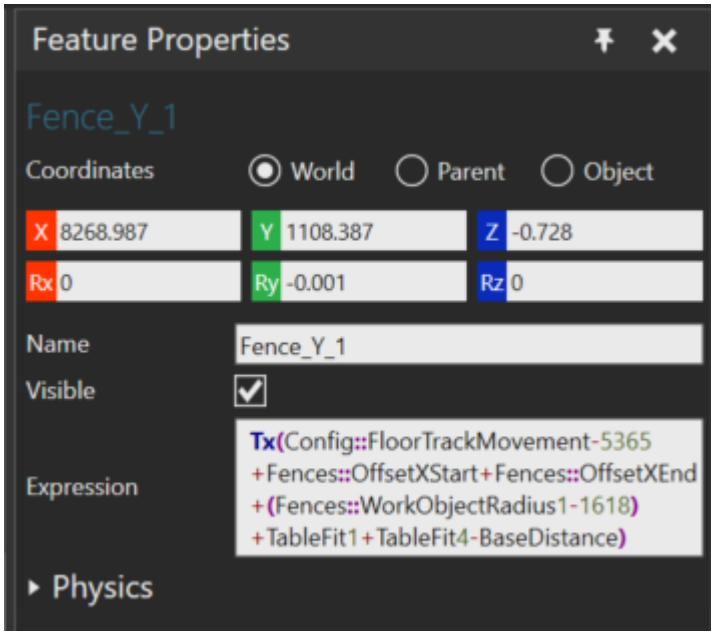



Kuva 31. Muuttuja vaihtopiirteessä.

## 7.3 Lukuarvo-muuttujat

*Integer-* ja *Real*-muuttujat, eli kokonaisluvut ja reaaliluvut, voivat antaa arvon totuusarvolle tai suoraan geometrisen piirteen koon muutokselle, muunnospiirteen siirrolle tai skaalaukselle. Muuttujan nimi tulee asettaa joko muunnoslausekkeeseen tai jos kyseessä on Visual Componentsin oma geometrinen piirre, kuten laatikko, tulee muuttuja asettaa esimerkiksi sen korkeudeksi *Feature Properties* -ikkunasta. Geometrian ollessa muunnospiirteen alla voidaan arvoja ja lausekkeita asettaa ja yhdistää, jotta geometriaa voidaan muokata joko yksittäisellä muunnospiirteellä, jossa on vain yksi pitkä lauseke, tai usealla muunnospiirteellä, joissa jaetaan jokainen akseli omaan muunnoslausekkeeseensa. Yksittäinen muunnospiirre pitkän lausekkeen kanssa, kuten kuvassa 32, voi helpottaa geometrioiden seuraamista geometriapuussa ja laskee ohjelmistobugien riskiä. Riskinä näissä toimii huono luettavuus ja muokattavuus uusia muuttujia lisätessä. Esimerkin kuvassa on lauseke turva-aidan siirrolle X-akselilla, joka on riippuvainen suojattavan radan pituudesta, käyttäjän erikseen syöttämistä reaaliluvuista ja rataan yhdistetyn

kääntöpöydän vaatimasta tilasta. Lausekkeen pituudesta ja huonosta luettavuudesta huolimatta muunnospiirre on turvallisempi siirtää, koska ohjelmiston ei tarvitse ottaa 7 erillisen muunnospiirteen origojen sijaintia huomioon tätä siirrettäessä. Jos geometrioita koittaa siirtää piirteestä toiseen samalla kun toisen tai molempien piirteiden alla on aktiivisesti geometrian kokoon tai sijaintiin vaikuttava lauseke, on olemassa riski, että geometria pysyvästi hajoaa ja se joudutaan erikseen tuomaan malliin tai luomaan uudelleen.

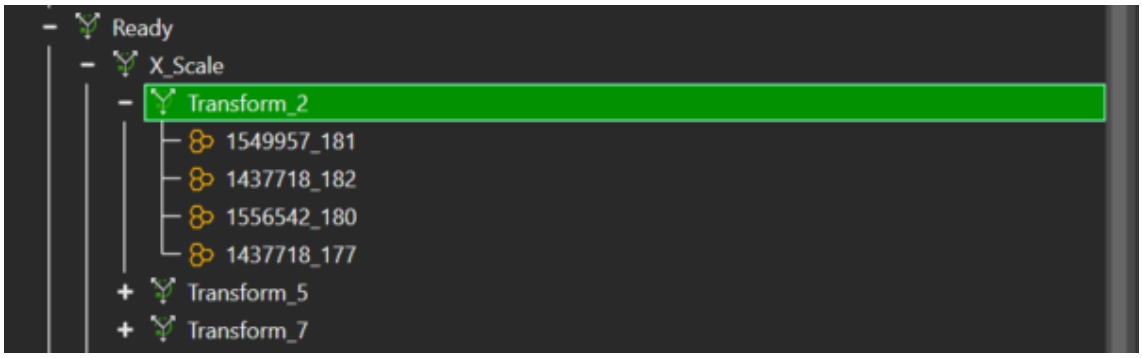



Kuva 32. Parametrien käyttö lausekkeessa.

Useamman muunnospiirteen käyttö, kuten kuvan 33 tapauksessa, voi helpottaa lausekkeiden perässä pysymistä, jos tiedetään, että kappaletta tullaan muuttamaan paljon ja siihen todennäköisesti lisätään jatkossa vielä enemmän muuttujia. Kappaletta voi olla helpompi hallita jakamalla Y-koordinaattiin ja Z-koordinaattiin kohdistuvat muuttujat omien piirteiden alle, mutta jokaista piirrettä kohden on olemassa suurempi riski, että osia tarvitsee siirtää piirteen alta toiseen jossain vaiheessa mallinnusprosessia. Geometrian pysyvästi hajottava ohjelmistobugi ilmenee suurella todennäköisyydellä, kun muunnospiirteen alla oleva lauseke unohdetaan nollata. Lausekkeiden nollaus muuttaa geometrian

takaisin alkuperäiseen muotoonsa ja siirtää sen alkupisteeseensä ja laskee riskiä sen dimensioiden hajottamiseen.

Muunnospiirteen ja muiden piirteiden lisääminen saman geometriapuun alle voi silti hajottaa sen, huolimatta varotoimenpiteistä ja oikeasta työjärjestyksestä. Toisinaan geometriat lakkaavat liikkumasta säännönmukaisesti ilman tarkempaa syytä. Voidaan olettaa, että on ohjelmalle laskennallisesti liian raskasta pysyä jokaisen geometriapuun alla olevan origon perässä ja näiden huomioon ottaminen geometrioita siirtäessä lakkaa kokonaan ohjelman kaatumisen estämiseksi. Näissä tapauksissa edes siirtäminen alkuperäiseen sijaintiin ei korjaa ongelmaa ja joudutaan ottamaan käyttöön viimeisin tallennus tai luomaan geometriapuu kokonaan uudelleen.

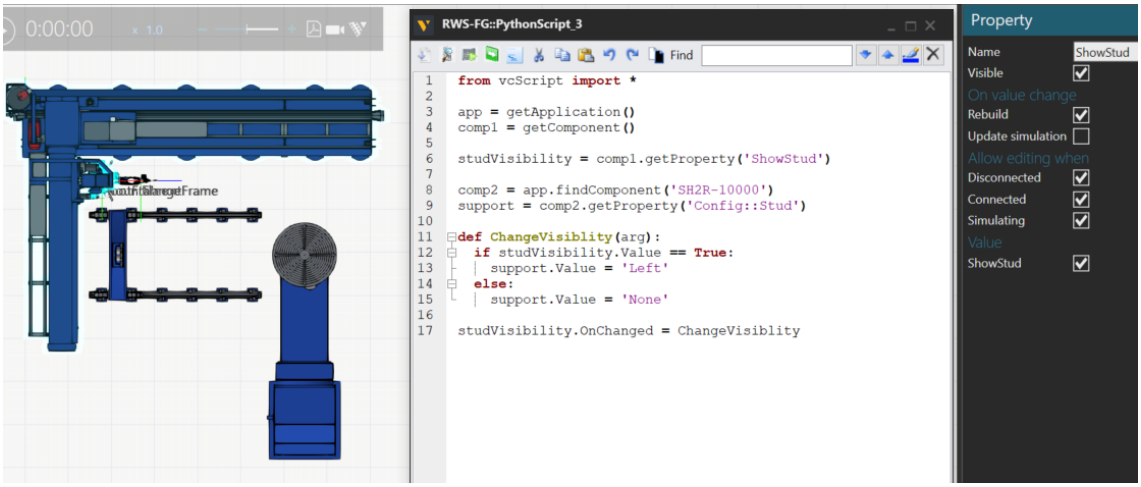



Kuva 33. Muunnospiirteiden asettamista hierarkiaan.

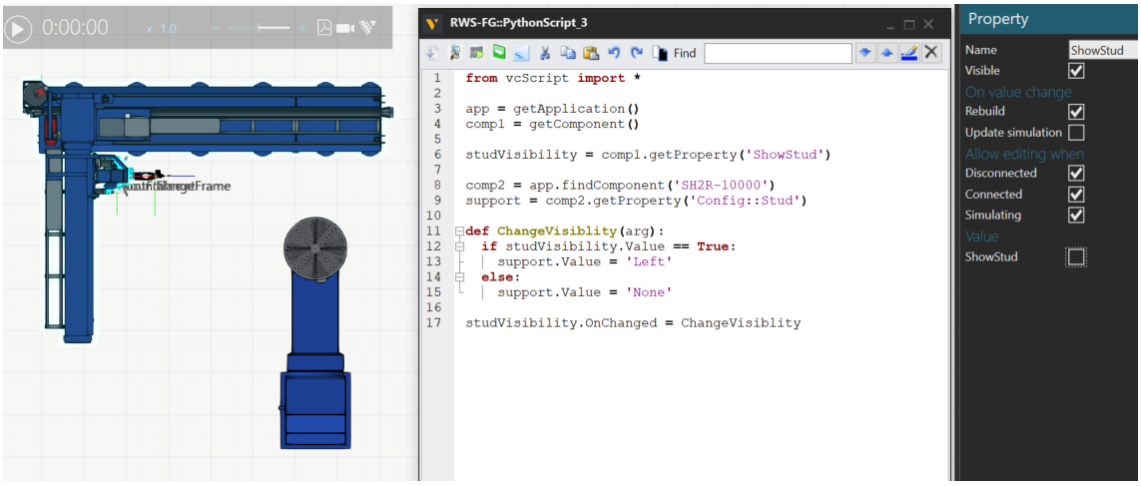
## 7.4 PythonScript-komentokieli

Jos haluaa hallita koko mallin asettelua, ei pelkästään tallennettua komponenttia, voidaan suositella käytettävän *PythonScript*-komentokieltä. *PythonScript* kääntää käyttäjän komennot ohjelmalle luettavaan muotoon, käyttäjän osatessa hyödyntää vcScript-kirjaston import-funktioita. Nämä eivät ole pythonin vaan ohjelman ominaisfunktioita ja antavat käyttäjän hakea ja lukea ohjelmaa laajemmin ja hakemaan asetelman komponentteja. Pythonskriptejä voi olla usealla komponentilla samassa asetelmassa ja ne voivat hallita vain oman komponenttinsa piirteitä, mutta käyttäjän osatessa hakea muita komponentteja samassa asetelmassa niiden nimillä voi useamman

komponentin piirteitä hallita yhden komponentin *Component Properties* -ikkunasta käsin. Kuvissa 34 ja 35 nähdään esimerkki geometrian näkyvyyden totuusarvon muutoksesta pääkomponentin ulkopuolella. Asettamalla lattiaradan valikon muuttujalle *ShowStud* arvoksi joko *True* tai *False* aktivoidaan lattiaradan alle luotu pythonskripti. Muuttujat koodissa kannattaa nimetä tavalla, joka selkeyttää koodin kokonaisuutta. Tämä auttaa muita, jotka saattaisivat haluta lukea koodia tai joskus itse koodin kehittäjää, jos tämä lukee koodia pidemmän tauon jälkeen.




Kuva 34. Tukipöytä esillä.




Kuva 35. Tukipöytä piilotettu.

*PythonScript*-funktiot ja muuttujat, joille näiden paluuarvot määrätään:

- *getApplication():* Etsii pääkokoonpanon/ohjelman ja sallii tämän hallinnan. Määrätään arvo muuttujalle ”app”.
- *getComponent():* Hakee käytettäväksi komponentin, jonka alle *PythonScript* on luotu. Määrätään muuttujalle ”comp1”.
- *comp1.getProperty*: Etsii pääkomponentin attribuutin (Property) tarkalleen sillä nimellä, joka on laitettu sulkuihin lainausmerkkien sisään. Muuttujanimi ”studVisibility”.
- *app.findComponent*: Etsii kokoonpanosta komponentin, joka vastaa annettua nimeä. Muuttujanimi ”comp2”.
- *comp2.getProperty*: Kun ulkoinen komponentti on löydetty, kykenee *PythonScript* hakemaan myös tämän attribuutteja. Muuttujanimi ”support”.
- support.Value: Etsii ja päivittää *”support”*-muuttujan arvon tietotyyppien ollessa täsmäävät.

Määritetään mitä komponentin attribuuttia muuttamalla *PythonScript* aktivoituu, joka tässä tapauksessa on *ShowStud*, pääkomponentin alta. ”studVisiblity.OnChanged” määrää, että attribuutin ShowStud arvon muuttuessa koodi suorittaa määritetyn komennon ”ChangeVisibility”. *ChangeVisibility* tarkistaa attribuutin *ShowStud* arvon ja palauttaa muuttujalle *support* arvon tämän mukaan. Jos komennon suoritettua *ShowStud*-arvo on *True*, asettaa koodi muuttujalle *support* arvoksi *”Left”* ja palauttaa tämän tiedon myös kääntöpöydän komponentille. Jos *ShowStud*-arvo on *False*, palauttaa koodi attribuutille arvon *”None”.*

Huomaa, että tässä muodossa *ShowStud*-totuusarvoattribuutti sallii *support*-muuttujan arvolle arvot ”*Left*” ja ”*None*” totuusarvojen sijaan. Tämä johtuu siitä, että *”Config::Stud”-property* kääntöpöydän komponentin alla on merkkijono. Totuuslauseke määrää siis *support*-muuttujalle arvoksi merkkijonon, jonka ohjelma ensin tarkistaa löytyvän attribuutin alta ja löytäessään valitsee tämän. Jos käyttäjä pyrkii virheellisesti etsimään ja palauttamaan attribuutteja, kuten esimerkiksi kirjoittamalla ”config::Stud” pienellä c-kirjaimella ”Config::Stud”

sijaan, koodi ilmoittaa lukuvirheestä (*Invalid syntax/Syntax error*) tai palauttaa muuttujalle arvon ***None***. Esimerkin totuuslauseessa ”*None*”-arvon määrittäminen lainausmerkeissä tekee siitä merkkijonon. Ilman lainausmerkkejä palauttaisi koodi arvon ***None***, joka tarkoittaa muuttujan arvon olevan tyhjä, eli koodi ei pystyisi enää lukemaan tämän tietotyyppiä eikä asettamaan tätä arvoksi muuttujalle tai attribuutille. Arvoa **None** voidaan käyttää keskeyttämään skripti tai hyppäämään komentojen yli, kun taas ”*None*” on yleinen nimitys, jonka voi asettaa merkkijonoon, kun halutaan vaihtoehto, jossa kaikki piirteet on piilotettu.

Muuttujien määrittäminen tulee tapahtua oikeassa järjestyksessä ja yhdenkin muuttujan määrittäminen väärin keskeyttää ajon.

Mahdollisia syitä keskeytyksille:

- *”getApplication()”* tai *”getComponent()”* on kirjoitettu väärin, jolloin koodi ei kykene palauttamaan funktioita ja määrittämään muuttujia, koska se etsii niitä tyhjän arvon avulla.
- Muuttujia yritetään määrittää väärässä järjestyksessä, esim. *”support”* ennen *”comp2”*-muuttujaa. Vaikka molemmat olisi kirjoitettu täysin oikein, johtuen pythonskriptin tavasta prosessoida koodia rivi kerrallaan tulee muuttujat määritellä ennen niiden käyttöä muiden muuttujien etsimiseen.
- Komponentti, jota koitetaan etsiä, on klooni/kopio kokoonpanossa, jolloin sen nimi on muotoa ”*KomponentinNimi+Järjestysnumero*”. Esimerkin tapauksessa nimen *”SH2R-10000”* sijaan se olisi *”SH2R-10000 #2”.* Löytääkseen mitään mallissa tulee nimen olla tismalleen sama, eli komponentin voi löytää vain tietämällä sen nimen ja sen mahdolliset, loogisesti muodostuvat variantit.
- Sisennykset on tehty väärin, jolloin funktiot eivät tapahdu tai palaudu vain tiettyjen ehtojen täyttyessä.
- Totuusmuuttujaa kysytään vain lausekkeella ”*if studVisiblity == True:”,* eli unohdetaan tarkentaa, että halutaan tietää totuusarvoattribuutin arvo *(”.Value”)*, eikä pelkästään onko attribuutti olemassa. Tämä voi olla hankala huomata skriptissä, koska kyseessä ei ole lauseopillinen virhe

vaan tarkennuksen puute eli looginen virhe, jolloin ohjelman *Output* ei ilmoita ongelmasta.

Perusperiaatteet pythonskriptille pysyvät samoina, mutta tietotyyppejä käsitellään eri laajuudella riippuen tarpeesta: Etsi pääkokoonpano, komponentit, komponenttien attribuutit, linkit, käyttöliittymät, rajapinnat sekä emo- ja lapsiluokkiin lukeutuvat solmukkeet (Parent/Child node). Tietotyyppejä tulee osata lukea ja muuttaa riippuen siitä mihin sitä käytetään. Yksinkertaisimmillaan voit asettaa pythonskriptin asettamaan reaaliluvulle tietyn arvon riippuen siitä, mikä on tietyn totuusmuuttujan arvo. Esimerkki monimutkaisemmasta pythonskriptistä voisi olla reaaliluvun asettaminen ulkoisessa komponentissa etsimällä tämän kokoonpanosta nimellä, jonka se on saanut luettua pääkomponenttiin kytketyistä lapsisolmukkeista.

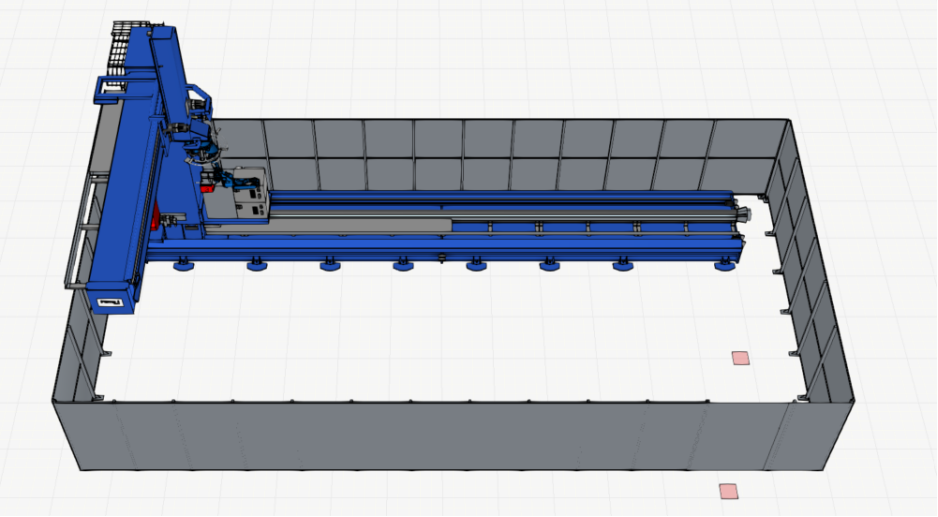
## 7.5 Aidanrakennusohjelma opinnäytetyössä

Opinnäytetyössä yhdistettiin kaikkia aikaisemmin mainittuja vaihtoehtoja parametrisen mallin luomiseksi. Laajin parametrinen ominaisuus mallissa oli lopuksi kehitetty aidanrakennus-pythonskripti. Tämän tuli mahdollisimman tarkasti vastata turvastandardien mukaista aitausta, joka tultaisiin todellisuudessa rakentamaan asiakkaan robottiyksikön ja työalueen ympärille. Aita tulee rakentaa vakioetäisyyksillä kaikista kiinteistä pinnoista yksikössä ja osa liikkuvista elementeistä tulee ottaa huomioon, kuten sähkökaappien teline, joka ääripäässä tulee hieman radan päätyä pidemmälle, vaatien aidalle enemmän etäisyyttä kiinteästä rakenteesta. Vakiomalli sisältää perustan kaikkien aitojen rakennukselle ja aidan rakentuminen tulee riippumaan kääntöpöydästä tai kahden solun mallissa kääntöpöydistä. Kääntöpöytien pöytätason keskipisteen tulee olla hitsausrobottilattiaradan Y-liikkeen keskipisteessä, jolloin erikokoisten kääntöpöytien runkorakenteet tulevat väistämättä vaatimaan omat aitojen dimensiot.

Aidat ovat kokonaan oman linkkinsä alla lattiaradan komponentissa helpomman liikuttamisen vuoksi. Aita on mahdollista piilottaa totuusarvomuuttujan avulla,

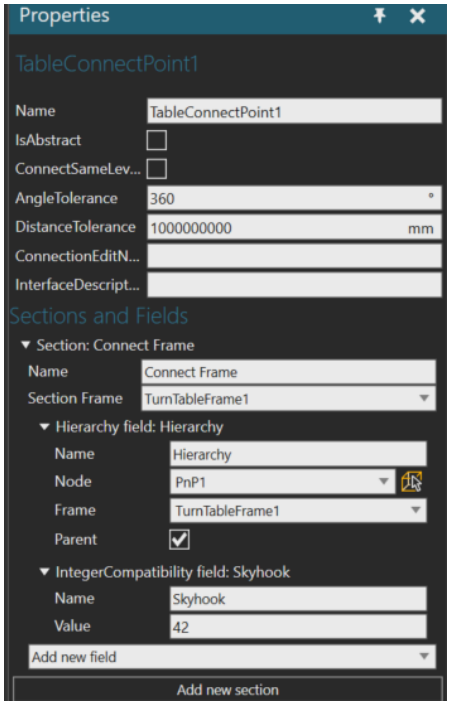
muutettaessa radan pituutta tai leveyttä aita kasvaa automaattisesti mukana, aitauksen voi jakaa kahteen soluun, etuaidan kanssa voi vaihtaa kiinteän ja valoverhon välillä ja jokaista aidan sivua, mukaan lukien vaihtoehtoista keskiaitaa, voi siirtää käyttäjän syötöillä.

Kuvassa 36 näkyy lattiarata ja aitaus neutraalissa lähtöpisteessä. Punaiset neliöt maassa ilmoittavat paikat, joihin kääntöpöydät voidaan raahata *PnP*-toiminnolla. Rataa lähempänä olevaan neliöön voi yhdistää yksiakseliset, eli *Headstock*-kääntöpöydät ja loitompana olevaan yhdistetään kaksiakseliset, eli *Skyhook*-kääntöpöydät. Neliöiden kohdalla sijaitsee omien linkkiensä alla olevat kehyspiirteet (Frame), jotka toimivat liitäntäpisteenä rajapintaliitännälle. Pöydän omatessa *TurnTableFrame*-piirteen kanssa yhteensopivan *ConnectFrame*-piirteen voi sen raahata vakioituun liitäntäpisteeseen ilman tarvetta laskea erikseen pöydän sijaintia suhteessa rataan. Yhteensopivan pöydän tulisi itse hakeutua pisteeseen, kun sen raahaa riittävän lähelle. Jos näin ei käy se tarkoittaa, että pöydällä joko ei ole vaadittua kehystä tai se ei ole yhteensopiva.

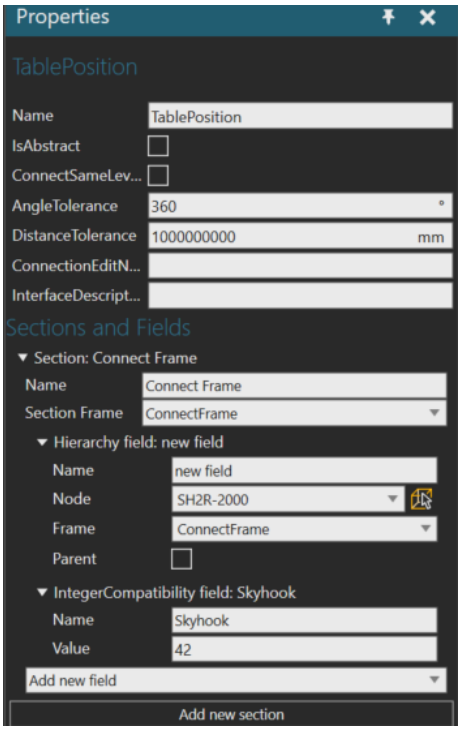


Kuva 36. Lattiaradan neutraali aitaus.

Jotta käyttäjä ei voi vahingossa liittää kääntöpöytää väärään pisteeseen on *Skyhook*- ja *Headstock*-pöytien rajapintojen (Interface) asetuksissa omat arvonsa liittämiselle. Kuvassa 37 ja 38 näkyy asetukset, joilla radan ja pöydän *One-to-One*-rajapinnat saadaan yhteensopiviksi.




Kuva 37. Lattiaradan kiinnityspisteen rajapinta-asetukset.




Kuva 38. Kääntöpöydän kiinnityspisteen rajapinta-asetukset.

Käyttöliittymään tulee luoda uusi osio (*Section*) ja järjestyksessä luoda hierarkia- ja kokonaislukuyhteensopivuuskenttä (*Hierarchy*- ja *IntegerCompatibility-field*). Hierarkiassa määritetään kiinnityspisteelle kehys (Frame) ja mikä linkki on emo tai lapsi. Kokonaislukuyhteensopivuus sallii kahden rajapinnan olevan yhteensopivia vain, jos ne molemmat omaavat saman arvon (*Value*). Lattiaradassa on 4 eri *TableConnectionPoint*-rajapintaa, joilla on jokaisella oltava oma kehys, joka toimii kääntöpöydän kiinnityspisteenä. Pelkän kehyksen liikutus muunnospiirteellä ei liikuttaisi siihen kytkettyä kääntöpöytää, vaan kääntöpöytä tulisi kiinnittää uudelleen ennen kuin sen sijainti päivittyisi. Asettamalla kehys emosolmukkeen alle, kykenee tämä siirtämään lapsisolmuketta, eli pöydän rajapinta-asetuksissa määrättyä juurisolmuketta. Tämä toimii samalla periaatteella kuin robotin kiinnittäminen Z-palkin päähän, mutta tässä emosolmuke on staattinen simulaation aikana ja sen sijaintia voidaan määrätä ja lukita muuttujilla.

Asetusten järjestyksen tulee pysyä samana, jotta *PythonScript* kykenee luotettavasti muuttamaan niitä. Kuvassa 39 on osio pythonskriptistä, joka muuttaa toisen solun rajapinta-asetusten arvot, kun muutetaan aitaus yhden solun muotoon. Tämä asetus paitsi piilottaa visuaalisesti liitospisteet, mutta myös tekee liitospisteistä yhteensopimattomat pöytien kanssa, jotta niihin ei voi vahingossa liittää pöytiä raahaamalla niitä pisteiden yli simulaatiossa.

```
304  #Find and edit sections and fields in interfaces
305 def IntegerChange(arg):
306     sectionList1 = connection1.Sections
307     sectionList2 = connection2.Sections
308     if cellType.Value == 'Dual':
309         sectionList1[0].Fields[1].Value = 42
310         sectionList2[0].Fields[1].Value = 84
311     else:
312         sectionList1[0].Fields[1].Value = 404
313         sectionList2[0].Fields[1].Value = 404
314     comp.rebuild()
315
```

Kuva 39. Rajapinta-asetusten muuttaminen pythonskriptillä.

Ohjelman toimintaperiaate:

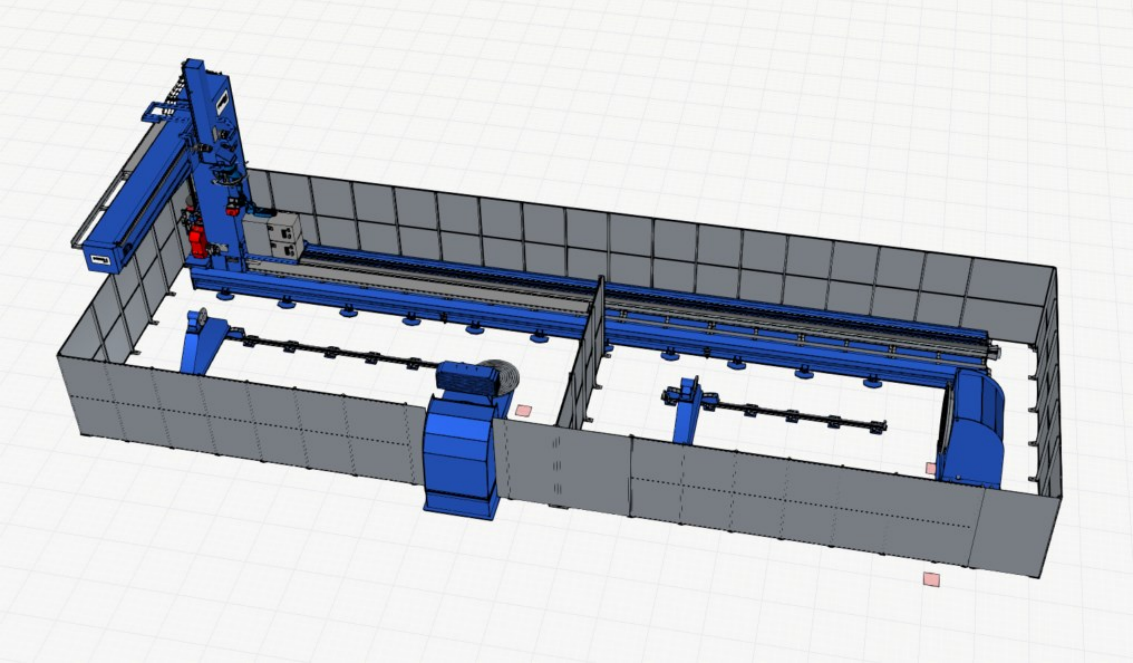
- Rivit 306 ja 307: Hae solun ”2” rajapinnat ja määrää niiden osioille (*Sections*) listamuuttujat. Muuttujanimet *sectionList1* ja *sectionList2*.
- Rivit 308–313: *IF*-lauseke määrittää arvon sen mukaan onko aitauksen malli muotoa *”Dual”* vai *”Single”*, viitaten onko yksi vai kaksi solua.
- Rivit 309, 310, 312 ja 313: Etsitään järjestyksessä rajapintavalintaikkunan alta ensimmäinen osio, sen toinen kenttä ja toisen kentän arvo ja määrätään tähän joko pöydän kanssa yhteensopiva tai yhteensopimaton arvo. Muista, että Pythonissa listan elementtien numerointi aloitetaan nollasta, eli ensimmäisen osiolistan valinnan arvo on 0, toisen on 1 ja niin edelleen!
- Rivi 314: *comp.Rebuild()* varmistaa muutosten käyttöönoton päivittämällä pääkomponentin ja tarkistamalla liitetyt solmukkeet.

Eli jos aitaus on ”*Single*”-muodossa skripti piilottaa toisen solun rajapinnat, väliaidan ja estää käyttäjää liittämästä pöytiä vääriin paikkoihin. Pitämällä osioiden ja kenttien järjestykset ja arvot esimerkin mukaisina voidaan jatkossa skriptiä jatkaa minimaalisilla muutoksilla ja voidaan tehdä enemmän yhteensopivia kääntöpöytiä, jotka voidaan liittää kokoonpanoihin helpommin.

## 7.6 Kääntöpöytien yhteensopivuuden testailu

Aidan rakennukseen luotu pythonskripti lukee radan rajapintojen lapsisolmukkeet, eli jos jokin pöytä on kiinnitetty rajapintaan, tulee skripti päivittämään aitojen koon ja linkkien sijainnin sen mukaan, mikä pöytä on kiinnitetty ja mihin soluun. Kääntöpöytien vakiomalleja tehtiin 7 kappaletta, ja yhdessä näistä oli 2 vaihtoehtoa rungon koolle, joten aidan dimensiot riippuvat 8:n eri pöydän dimensioista. Koska on mahdollista, että kahdessa solussa on samanlainen kääntöpöytä tarkoittaa tämä, että tuli ottaa huomioon 16 kääntöpöydän kaikki mahdolliset kombinaatiot, jotta saatiin aikaiseksi kaikki mahdolliset aitojen dimensiot. Loppujen lopuksi pöytäparit muodostavat 120 uniikkia kombinaatiota, joiden perusteella aidan koko muuttuu.

Kuvassa 40 on esimerkki yhdestä pöytäyhdistelmästä kahden solun radalla. Pöydät on asetettu tavalla, jolla robotti pääsee mahdollisimman helposti pöytätasoon kiinnitetyn työkappaleen luokse, liikkuvien piirteiden osuminen aitaan on hankalaa tai mahdotonta ja liikkuvien osien ollessa ääriasennoissaan ei ihminen voi jäädä näiden väliin puristuksiin.



Kuva 40. Aidan rakennus kääntöpöytien ympärille.

Kuvassa 41 näkyy aidanrakennuspythonskriptin alkuosio. Palautetaan funktioita ja määritetään mahdollisimman monta muuttujaa heti alkuun, jotta voidaan yksinkertaistaa komentojen rakennetta. Aidanrakennus käynnistyy, kun käyttäjä ensin asettaa pöydät rajapintoihin paikoilleen ja tämän jälkeen lattiaradan komponentista painaa ”*BuildFences*”-nappia. Aitojen rakennus olisi ollut mahdollista suorittaa myös välittömästi pöydät yhdistäessä, asettamalla skripti tarkastelemaan rajapintaliitoksissa tapahtuvia muutoksia, mutta tämä on huomattavasti raskaampi tapa. Kahden solun tapauksessa aita rakentuisi kahdesti, joka loppujen lopuksi vaatii enemmän aikaa johtuen joutoajasta, jota skriptin käsittely aiheuttaa. Riveillä 15–27 ja 31 ovat kaikki muuttujat, jotka tulevat jollain tavalla vaikuttamaan aitojen kokoon tai sijaintiin X- ja Y-

suunnassa. Niiden nimiin sisältyy jollain tavalla niiden toiminto, esimerkiksi, *Length*, *Var (Variable)* tai *Offset*. Osa näistä vaikuttaa vain soluun 1, osa vain soluun 2 ja loput molempiin. Jäljellä olevat muuttujat alkuosiosta määrittävät etsityt komponentit, attribuutit, linkit ja rajapinnat, joita toiminnot hyödyntävät.

```
comp = getComponent()
app = getApplication()

button = comp.getProperty('Config::BuildFences')
YMovement = comp.getProperty('Config::YMovement')
fenceType = comp.getProperty('Fences::FrontFenceType')
huuva = comp.getProperty('Config::FumeExtraction')
cellType = comp.getProperty('Config::TurnTableSetup')
trackSafety = comp.getProperty('Config::TrackSafetyCurtain')
innerLight = comp.getProperty('InnerLightCurtain')


fenceLength1 = comp.getProperty('FenceVariable1')
fenceLength2 = comp.getProperty('FenceVariable2')
fenceLength3 = comp.getProperty('FenceVariable3')
tableOffset1 = comp.getProperty('TableFit1')
tableOffset2 = comp.getProperty('TableFit2')
tableOffset3 = comp.getProperty('TableFit3')
tableOffset4 = comp.getProperty('TableFit4')
tableOffset5 = comp.getProperty('TableFit5')
tableOffset6 = comp.getProperty('TableFit6')
tableVar1 = comp.getProperty('TableVariable1')
tableVar2 = comp.getProperty('TableVariable2')
tableVar3 = comp.getProperty('TableVariable3')
tableVar4 = comp.getProperty('TableVariable4')
connection1 = comp.findBehaviour('TableConnectPoint3')
connection2 = comp.findBehaviour('TableConnectPoint4')
lightCurtain = comp.getProperty('LightCurtainDistance')
baseDistance = comp.getProperty('BaseDistance')
pnp1 = comp.findNode('PnP1')
pnp2 = comp.findNode('PnP2')
pnp3 = comp.findNode('PnP3')
pnp4 = comp.findNode('PnP4')

BASECONSTANT = 400

def UpdateFences(arg):
    if len(pnp1.Children)>0:
        baseDistance.Value = 0
        for node1 in pnp1.Children:
            print node1.Name +" Placed"
            stringend = node1.Name.split(' ')
            if node1.Name == 'SH2R-2000':
                fenceLength1.Value = -500
                tableOffset1.Value = 0
                tableOffset4.Value = 157
                tableVar1.Value = 0
                tableVar3.Value = 0
            elif node1.Name == 'SH2R-5000':
                tableOffset1.Value = 139
                tableOffset4.Value = 180
                tableVar1.Value = 24
                tableVar3.Value = -18
                comp1 = app.findComponent('SH2R-5000')
                axis = comp1.getProperty('Config::RotAxisDistance')
                if axis.Value == 2500:
                    fenceLength1.Value = 572.5
                else:
                    fenceLength1.Value = 72.5
            elif node1.Name == 'SH2R-10000':
                fenceLength1.Value = 670
                tableOffset1.Value = 139
                tableOffset4.Value = 206
                tableVar1.Value = 308
                tableVar3.Value = 0
            elif len(stringend)>1:
                if node1.Name == ('SH2R-2000 '+stringend[1]):
                    fenceLength1.Value = -500
                    tableOffset1.Value = 0
                    tableOffset4.Value = 157
                    tableVar1.Value = 0
                    tableVar3.Value = 0
```

Kuva 41. PythonScript aidan rakennukseen.

*PythonSciptin* toiminnasta:

- *UpdateFences* (rivi 39): Komento, joka käynnistyy painamalla RWS-FG-komponentin BuildFences-nappia.
- ***if*** *len(pnp1.Children)>0* (rivi 40): Lukee rajapinnan 1. Se etsii annetun rajapinta-attribuutin komponentista ja lukee, onko tämän lapsisolmukkeiden lista suurempi kuin 0. Mikäli listan pituus on suurempi kuin 0 on tähän yhdistetty jokin yhteensopiva kääntöpöytä.
- *baseDistance.Value* (rivi 41): Tarkoitettu vain yksiakselisille kääntöpöydille solussa 1. PnP1 on kaksiakselisten kääntöpöytien rajapinta, joten tämän ollessa aktiivinen palautetaan yksiakselisten pöytien muuttuja neutraalin tilan arvoon.
- ***for*** *node1* ***in*** *pnp1.Children* (rivi 42): Löytäessään lapsisolmukkeen alkaa vertaamaan tämän nimeä annettuihin vaihtoehtoihin. Skripti sisältää jokaisen vakiomallin vaatiman tilan arvot ja jos yhdistetty komponentti, tai tämän loogisesti numeroitu klooni löytyy listasta päivittää skripti muuttujien arvot if-lausekkeen antamien arvojen mukaisiksi. Jos komponentin nimi ei löydy if-lausekkeista palauttaa skripti aitauksen vakioarvot.
- stringend = node1.Name.split(’ ’) (rivi 44): Jakaa node1-merkkijonon välilyöntien perusteella ja muuttaa jaetun merkkijonon listaksi. Jos merkkijonossa olisi välilyöntejä olisi tämä todennäköisesti kloonikomponentti, koska kloonikomponentin nimen ja järjestysnumeron välillä on välilyönti. Tämä on esimerkki komponentin löytämisestä, vaikka sen nimi ei ole tismalleen sama kuin alkuperäisen etsityn, niin pitkään kuin alkuperäisen nimen variantit ovat loogisesti pääteltävissä. Käyttämällä välilyönnillä jaetun merkkijonon, eli listan, toista elementtiä voidaan löytää vakiomallien klooneja jäljellä olevasta skriptiosuudesta.
- ***elif*** *len(stringend)>1* (rivi 68): Jos listan *stringend* pituus on suurempi kuin 1, on löydetty komponentti todennäköisesti klooni ja vertaa skripti nimeä loppuihin nimiin.
- ***if*** *node1.Name == (’SH2R-2000 ’+stringend[1])* (rivi 69): Yhdistää listan toisen elementin etsittävään nimeen. Ei ole merkitystä, kuinka mones

klooni on kyseessä, niin pitkään kuin ensimmäinen elementti nimestä on oikein, eli sille löytyy *if*-lause.

Opinnäytetyön pythonskripti ei ole ottanut huomioon tilannetta, jossa käyttäjä pyrkisi lisäämään yhteen soluun yksiakselisen sekä kaksiakselisen pöydän samanaikaisesti, koska tätä ei koskaan tehtäisi todellisessa tilanteessa. Jo kahden kääntöpöydän samaan soluun asettaminen mallissa osoittaa kuinka mallien geometriat lävistävät toisensa, joten ei ole tarvetta kuluttaa aikaa poikkeustilanteen laatimiseksi skriptiin. Tarkoituksena on joko yksittäisen solun muodossa tarkastella yhden kääntöpöydän vaatimaa tilaa ja muuttaa aitaa tämän mukaan tai kahden solun muodossa tarkastella kahdessa eri solussa olevan pöydän vaikutusta aitaan ja kiinnityspisteen sijaintiin. Keskiaita ei tule siirtymään ilman erillistä lukuarvon asettamista tämän Offset-attribuuttiin, joten toisen solun pöydän etäisyys tästä varmistetaan siirtämällä tämän kiinnityspistettä.

Mallissa etuaita on suljettu, mutta todellisuudessa tässä tulee olemaan ovi, josta työntekijät voivat mennä soluun sisälle. Mallissa aitojen tulee olla kuitenkin visuaalisen selkeyden vuoksi täysin suljettu ja kaikki välit aidan ja kaksiakselisen kääntöpöydän välillä tulee olla niin ahtaat, että ihmisen raaja ei sinne mahtuisi. Mallin aitaan on asetettu pienet välit, jotka aukeavat tai sulkeutuvat sen mukaan, mikä kääntöpöytä on yhdistetty soluun. Aidan välin koko riippuu kääntöpöydästä ja sen sijainnin muutos solusta. Jos solussa on yksiakselinen kääntöpöytä, on tuo väli kokonaan suljettu. Yksiakselisten kääntöpöytien soluissa rungon takaosan ja aidan väliin on jätetty riittävä väli, jotta laitteen koneistoa pääsee huoltamaan solun sisällä. Solussa ollessa kaksiakselinen kääntöpöytä on aidan väli avattu, koska pöydän rungon takaosa voi olla osassa tapauksista näkyvissä aidan ulkopuolella helpompaa huoltoa varten samalla kun aita on riittävän kaukana liikkuvista osista. Tässä kohtaa tuli esille kuinka tärkeää oli iteroida aidan muuttujien kanssa.

## 7.7 Ongelmien selvitys kääntöpöytien sijoittelussa

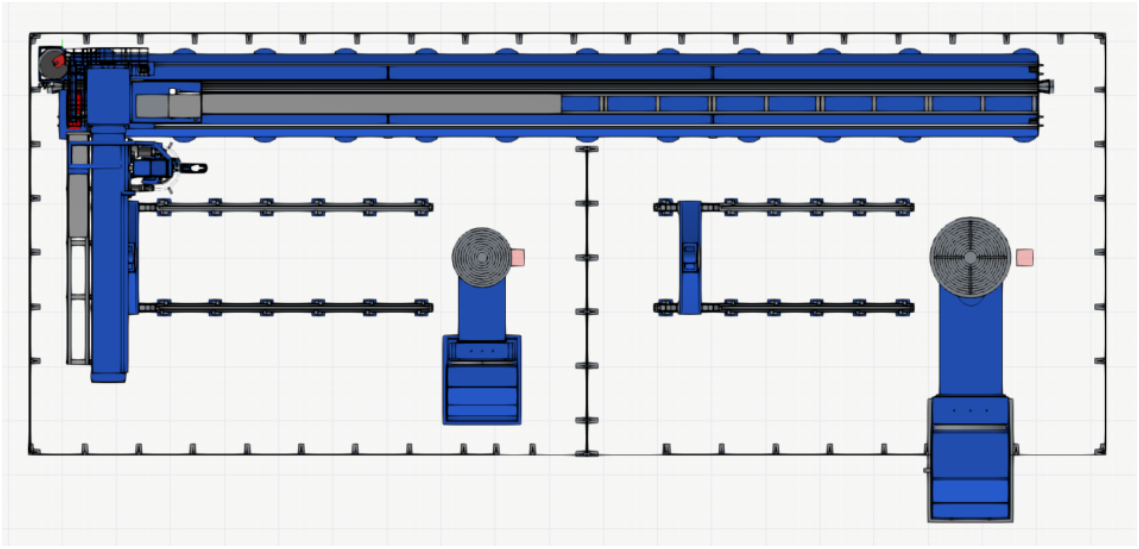
Kääntöpöytien sijoittelu solussa vaati useita eri revisioita, koska välissä tuli varmistaa suunnittelijoilta kuinka tietyissä tilanteissa halutaan käsitellä pöytien sijaintia ja tyhjää tilaa solussa. Samalla lattiaradan vakiomalliin tehtiin pieniä muutoksia ja osa näistä saattoi vaatia aidanrakennuksen muuttujien päivittämistä. Pöytien kiinnitys malliin ei saa häiritä simulointia, estää robotin liittämistä kääntöpöytiin tai häiritä törmäystarkastelua ja mallin tulee kuitenkin tarkasti esittää joko suoraan ratkaisuja tai antaa visuaalinen representaatio ongelmista, joiden avulla suunnittelijat voivat ratkaista ongelmia liittyen työtilaan ja turvallisuuteen. Johtuen aikaisemmin mainitusta mahdollisten parien lukumäärästä oli tärkeää pystyä testaamaan niin monta versiota lattiaradan työtilasta kuin mahdollista.

Esimerkki: Kahden solun mallissa aidan rakennus Y-suunnassa tulee tapahtua sen kääntöpöydän mukaan, jonka runko ylettyy uloimmas. Tämä tulee ottaa huomioon vain, kun toisessa tai molemmissa soluissa on kaksiakselinen kääntöpöytä, koska yksiakseliset pöydät eivät vaadi tilaa, joka ylettäisi Y-palkkia ulommas solussa. Kaksiakselinen kääntöpöytä määrää aidan rakennukselle Y-suunnassa muuttujan arvon. Molemmissa soluissa ollessa kaksiakselinen kääntöpöytä tulee aidan rakennus tapahtua vain yhden muuttujan avulla, ei kahden summalla, joten molemmat solut määräävät muuttujan arvon pöydän mukaan ja suurempi muuttuja vaikuttaa yksin aitaan. Tämä tapahtuu kolmannen muuttujan kautta, jonka arvo määrittyy suuremman muuttujan arvon mukaan. Kuvassa 42 on lisäksi poikkeustilanteet, joissa kolmanteen muuttujaan tehdään jatkomuutoksia riippuen siitä, mikä yhdistelmä kaksiakselisia pöytiä on soluissa. Tämä tulee tehdä siitä syystä, että joidenkin kaksiakselisten pöytien kokoerot ovat niin suuria, että aidan rakentuessa suuremman pöydän mukaan toisen aidan on järkevämpi jäädä suljetun aidan sisälle, sen sijaan että aitaan tehtäisiin syvennys koneistoon pääsyn vuoksi. Tässä ongelmaksi kuitenkin koitui, että suljetun aidan ja pienemmän kaksiakselisen pöydän väliin ei ihan jäänyt tarpeeksi tilaa huoltohenkilöstölle. Ratkaisuna tälle sovittiin suunnittelijoiden kanssa, että koko aitaa siirretään sen verran ulommas, että pienemmän pöydän

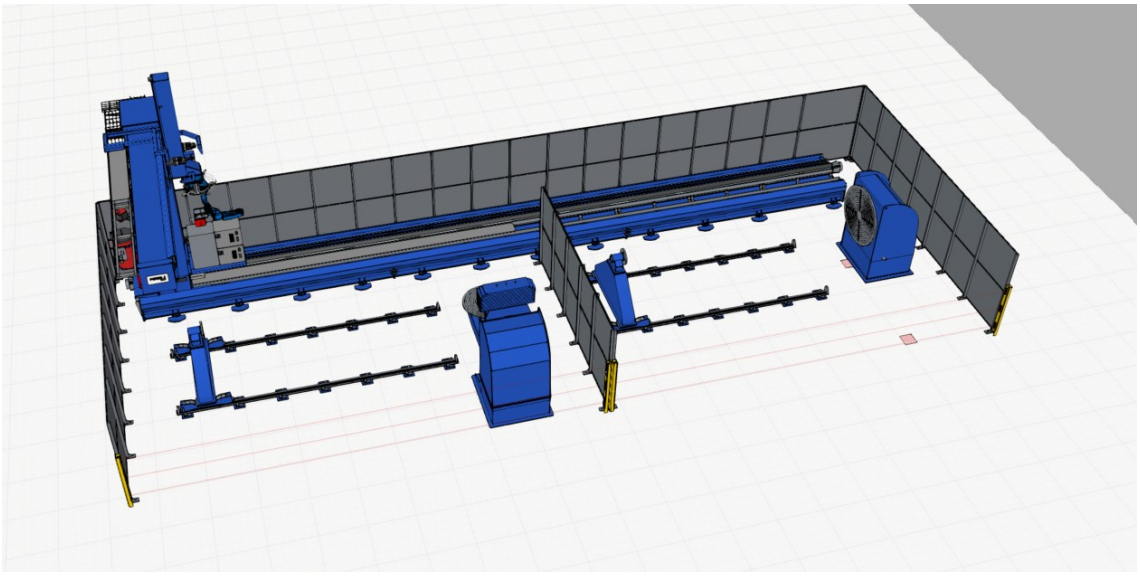
ja aidan väliin jää riittävästi tilaa (kuva 43). Poikkeustilanteisiin lukeutuu myös valoverho etuaitana (kuva 44), jolloin Y-aidan tulee ylettyä tarpeeksi pitkälle, että valoverho tulee standardimäärän verran pidemmälle kuin pöydän rungon takaosa, eli tässä tapauksessa Y-suunnan kolmas muuttuja summataan valoverhon ja pöydän antaman valoverhon arvon kanssa.

```
#Prioritize building fences based on the larger table's dimensions
if fenceLength1.Value>fenceLength2.Value:
    fenceLength3.Value = fenceLength1.Value
elif fenceLength1.Value<=fenceLength2.Value:
    fenceLength3.Value = fenceLength2.Value

if fenceLength1.Value == 572.5 and fenceLength2.Value == -500:
    fenceLength3.Value = fenceLength1.Value + 50
    tableVar4.Value = -1029
elif fenceLength2.Value == 572.5 and fenceLength1.Value == -500:
    fenceLength3.Value = fenceLength2.Value + 50
    tableVar3.Value = -1337

if fenceLength1.Value == 670 and fenceLength2.Value == -500:
    tableVar4.Value = -1029
elif fenceLength2.Value == 670 and fenceLength1.Value == -500:
    tableVar3.Value = -1337

#Increase the front fence's distance to make room behind SH2R-2000
if fenceLength1.Value == 72.5 and fenceLength2.Value == -500:
    tableVar3.Value = -38
    tableVar4.Value = 308
elif fenceLength2.Value == 72.5 and fenceLength1.Value == -500:
    tableVar3.Value = 0
    tableVar4.Value = 408

#Extend fences so that light curtains go 100 mm behind the tables
if fenceType.Value == 'Closed':
    lightCurtain.Value = 0
else:
    if fenceLength3.Value == 670:
        lightCurtain.Value = 1347
    elif fenceLength3.Value == 572.5:
        lightCurtain.Value = 827
    elif fenceLength3.Value == 72.5:
        lightCurtain.Value = 827
    elif fenceLength3.Value == 622.5:
        lightCurtain.Value = 827
    elif fenceLength3.Value == -500:
        lightCurtain.Value = 722
    elif fenceLength3.Value == -610:
        lightCurtain.Value = 1279
    else:
        fenceLength3.Value = 0

```

Kuva 42. Y-suuntaisen muuttujan määritys aidalle.



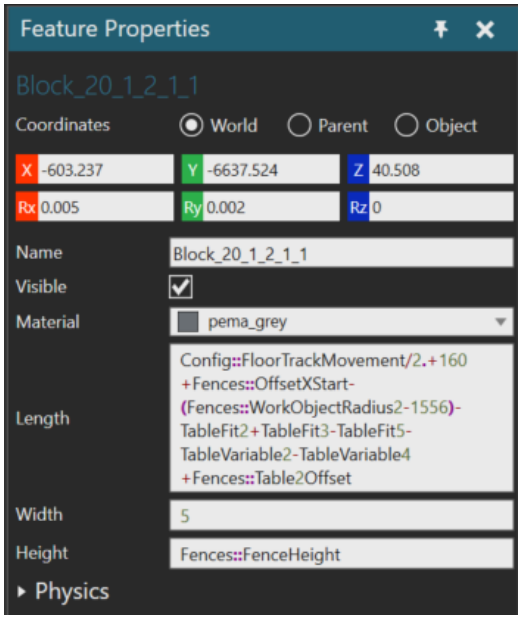
Kuva 43. Poikkeustilanne aidan etäisyyden määrityksessä.



Kuva 44. Valoverho etuaitana mallissa.

Mallissa käsiteltiin aidan rakennus automaattisesti X- ja Y-suunnassa ja mainittuja tapoja käytettiin selvittämään, kuinka aita rakennetaan joko umpeen tai jätetään sopiva väli aidan ja kääntöpöydän väliin ja arvot muuttuvat vielä lisää riippuen siitä onko etuaita valoverho, onko mallissa savukaasunpoisto tai onko kääntöpöytien yhdistelmä poikkeuksellinen. Näiden lisäksi viimeiseksi

annetaan vielä käyttäjälle oma *Fences*-valikko arvojen muuttamiseksi, mikäli tarve tulee siirtää pöytiä tai aitoja tai nostaa aidan korkeutta. Tämän tiedon avulla tarkastellaan kuvan 45 geometrian arvoja, jonka on tarkoitus olla kahden solun tapauksessa toisen solun suurempi seinä. Kyseessä on Visual Componentsin oma *Block*-piirre, jossa vain aidan paksuus säilyy vakiona.




Kuva 45. Konfiguraatioista riippuvainen *Block*-piirre.

Pituuteen vaikuttavat tekijät:

- *Config::FloorTrackMovement/2.+160*: Koko määrittyy automaattisesti radan pituuden mukaan ja kasvaa puolella suhteessa rataan. Kappaleen pituuden tulee lähtöpisteessä olla hieman pidempi kuin radan pituuden arvo jaettuna kahdella ja siksi tähän summataan +160(mm).
- *Fences::OffsetXStart*: Kasvattaa käyttäjän syötöillä aidan kokoa negatiivisessa X-suunnassa. Todellisuudessa tämä siirtää koko aitausta negatiiviseen suuntaan ja kasvattaa tai siirtää elementtejä positiivisessa X-suunnassa, mutta lopputulos on helpompi saada aikaiseksi ja ilmeinen vaikutus pysyy samana.
- *Fences::WorkObjectRadius2-1556*: Käyttäjäsyötöillä siirtää kääntöpöydän liitäntäpistettä, jolloin aidan reunan tulee pysyä

vakioetäisyydellä rungosta. Lähtökohtaisesti kääntöpöytä kykenee käsittelemään työkappaletta, jonka säde on 1556 mm, joten tuo erotetaan arvosta, jotta käyttäjäsyöttöjä tarvitaan vain, kun halutaan määrittää, kuinka paljon suurempi on käsiteltävä työkappale, kuin mitä lähtökohtaisesti voidaan työstää. Eli lyhyesti sanottuna ”Kuinka paljon tarvitsemme lisää tilaa, jotta työkappaletta voidaan käsitellä turvallisesti?”

- *TableFit2*, *-3* ja *-5*, sekä *TableVariable2* ja *-4* ovat kaikki kääntöpöytien antamia muuttujien arvoja ja arvot määräytyvät käyttäjän päivittäessä liitäntöjä. Nämä joko täyttävät aitaan jäävän tyhjän välin tai sovittavat aidan reunan vakioetäisyydelle kääntöpöydästä.
- *Fences::Table2Offset*: Sovittaa aidan reunan jälleen vakioetäisyydelle kääntöpöydän reunasta, kun käyttäjäsyötöillä siirretään kääntöpöytää X-suunnassa.
- *Fences*::FenceHeight: Muuttaa käyttäjäsyötöillä aidan korkeutta.

Tähän kyseiseen kappaleeseen vaikuttaa vielä geometriapuussa useampi muunnos- ja vaihtopiirre (*Transform* ja *Switch*), jotka siirtävät sitä eri suunnissa tai piilottavat sen tarvittaessa. Eri aidan osat toimivat samalla periaatteilla, mutta vaihtelevilla muuttujilla ja esimerkiksi aidan tolpat hyödyntävät arvoja koon muuttamisen sijaan tietääkseen kuinka monta kloonia tolpasta tarvitaan ja kuinka pitkälle välimatkalle. Tolppien määrä tai edes sijainti ei vastaa todellisuutta, mutta kuten aidanrakennusskriptin tarkoitus yleisestikin niiden on tarkoitus antaa visuaalista selkeyttä samalla kun osoitetaan poikkeustilanteista aiheutuvia ongelmia tilan kanssa.

## 7.8 Robotin ohjaus lähtösijainnin muuttuessa

Viimeiseksi esitellään robotin uudelleen sijoittamista parametrisessa mallissa. Aikaisemmin mainittuna robotin paikantaminen simulaatiossa tapahtuu kahden positiomatriisin erotuksen avulla ja nämä tapahtuvat koordinaattien avulla, joista toinen ei liiku automaattisesti linkkejä muutettaessa mallissa. Robottiin tehdyt koordinaatistot ”*ROBOT*” ja ”*BASE*” paikantavat robotin simulaation aikana ja *ROBOT* on kiinni robotissa itsessään, joka on lattiaradan lapsisolmuke eli se

seuraa radan linkkejä ja niissä tapahtuvia muutoksia. *BASE* on kiinni *WORLD*-solmukkeessa, eli sen on tarkoitus pysyä paikallaan, vaikka solmukkeet sen ympärillä liikkuisivat. Tämä tarkoittaa, että *BASE* on luotettava referenssipiste simulaation aikana, mutta samalla tätä ei voi sijoittaa linkkien alle, koska sen siirtyminen niiden mukana estäisi robotin tarkkaa paikannusta. Tämä hankaloittaa sen siirtoa, kun halutaan päivittää sen sijaintia lattiaradan tornin korkeuden mukaan. Jotta *BASE*-koordinaatistoa voidaan siirtää halutulla tavalla, tulee tietää sen sijainti robotin toimintojen alla. Eli etsitään robotin nimellä komponentti simulaatiomaailmasta ja sen *Behaviors*-listasta *DX200* (kuva 46), jonka lapsielementit voidaan löytää vakionimellä *BASE*. *BASE*-koordinaatiston siirtäminen vaatii ensin attribuutit (Property) lattiaradan alle, jotka ovat tässä tapauksessa *RobotBaseHeight*, eli robotin alustan korkeus ja *BMovePrev*, jonka on tarkoitus tallentaa viimeisin robotin alustan korkeus. Tarkoituksena on etsiä *BASE*-koordinaatisto ja erottaa sen nykyisestä sijainnista Z-suunnassa robotin alustan viimeisin korkeus ja summata sen nykyinen valittu korkeus. Eli jos halutaan vaihtaa korkeudesta 4,5 m korkeuteen 3,5 m muuttuu koordinaatiston korkeus metrin verran negatiivisessa Z-suunnassa. Sijainnin muutoksen jälkeen skripti päivittää *BMovePrev*-arvolle sen nykyisen valitun korkeuden, jotta se voi seuraavan kerran käyttää tätä korkeutta muuttaessa.

```
96  #PositionMatrix adjusts the offset in relation to current position in the order(X, Y, Z)
97  def movebase(arg):
98      bases = comp7.findBehaviour('DX200').Bases
99      for base in bases:
100         if base.Name == "BASE":
101             m = base.PositionMatrix
102             m.translateAbs(0,0,-prevmoveB.Value+moveB.Value)
103             base.PositionMatrix = m
104
105     comp7.rebuild()
106     prevmoveB.Value = moveB.Value
```

Kuva 46. Koordinaatiston siirto pythonskriptillä.

## 7.9 Lopputulokset

OLP-mallien kehitys on osa simuloinnin työnkuvaa, joten mallin kehittämisen työprosessi oli suurimmaksi osaksi samankaltainen kuin muiden OLP-mallien, mutta parametrisuuden soveltaminen ihanteellisella tavalla toi omat vaikeutensa ja hyötynsä. Vakiomallien suunnitelmat eivät vielä olleet täysin viimeisteltyjä johtuen useista, odottamattomista muuttujista, jotka voitiin saada selville vain tekemällä kokoonpanoja ja laiteyhdistelmiä simulointipuolella mallien prototyyppien kanssa. Moni puute suunnittelussa voitiin todeta vasta kun malli oli saatu visuaalisesti riittävän pitkälle kehitettyä ja tästä syystä malli piti heti alkuun suunnitella tavalla, joka salli sen laajankin muutoksen myöhemmässä vaiheessa. Esimerkiksi putsausyksiköiden, jäähdyttimien ja lankatynnyrien mukaan ottajien sijainnit vaikuttivat alkujaan siihen, miten aidat tuli sijoittaa suhteessa radan alkupäätyyn, mutta näiden sijaintia mallissa siirrettiin muutamaan otteeseen, jolloin aidat tuli myös sijoittaa uudelleen, jotta turvaetäisyydet pysyisivät samana. Projekti eteni kehittämällä mallia hitaasti ja testaamalla varovaisesti jokaista komponenttiyhdistelmää samalla kun tuli olla valmiina sisällyttämään malliin uusin geometriapäivitys, jotta voitaisiin varmistua, että tämä ei voisi vaikuttaa liikaa muuhun toimintaan. Kaikilta ongelmilta ei vältytty huolimatta varotoimenpiteistä, mutta mallista saatiin aikaiseksi versio, jota ollaan valmiina jatkamaan tulevaisuudessa päivityksillä, mutta tämänhetkinen versio mallista saatiin hyväksyttyä suunnittelijoiden ja testaajien kanssa.

Lopputulos oli toimiva prototyyppi parametrisesta OLP-mallista, jolla voidaan suorittaa testiajoja ja esitellä jokaista varianttia vakioidun hitsausrobottilattiaradan ja kääntöpöytien yhdistelmistä ilman että mikään toiminto ohjelmassa lakkaa toimimasta. Mallista tuli melko raskas, jolloin tästä tuli tehdä vielä erityisen kevyt versio, jota tullaan käyttämään lopullisessa etäohjelmoinnissa, mutta tämä tarkoitti lähinnä robottiohjelmoinnin ja törmäystarkastelun kannalta tarpeettomien piirteiden piilottamista tai poistamista. Esimerkiksi turva-aitojen ei pitäisi enää suunnittelun ja työalueen määrittelyn jälkeen hitsausohjelmoinnissa aiheuttaa riskiä komponenteille tai

työkappaleille, jolloin etäohjelmointia varten nuo ovat tarpeettomia piirteitä. Samalla, koska malli tulisi olla räätälöity asiakkaalle ei olisi tiedostokoon ja simuloinnin kevyen toiminnan kannalta järkevää jättää malliin tarpeettomia piirteitä, jotka saattaisivat hidastaa ohjelmiston toimintaa asiakkaan käytössä. Parametrisen OLP-mallin tuli olla toimiva, visuaalisesti selkeä pohja yksiköiden ja työalueiden suunnitteluun sekä hitsausrobottien etäohjelmointiin. Näissä onnistuttiin ja malli onnistuttiin suunnittelemaan riittävän yksinkertaiseksi, että mikäli jatkossa ilmenee tarvetta lisäyksille ja päivityksille tulisi näiden olla mahdollista suorittaa samojen toimintaperiaatteiden avulla. Pemamek Oy:n tilauksien monimuotoisuuden kasvaessa vakiolaitteet ja malli, jonka pohjalta nämä kaikki voidaan suunnitella, tulee merkittävästi nopeuttamaan suunnittelun ja simuloinnin prosessia ja toivottavasti auttaa välttämään sekä ihmisten että koneiden aiheuttamia virheitä ja viivästyksiä ennen siirtymistä fyysisten laitteiden kanssa toimimiseen. Mallin kokonaishyödyt ovat vielä hankalasti arvioitavissa, mutta esimerkiksi samankaltaisten yksiköiden layouttien pohjasuunnitelmien kehittäminen muuttui tunneista minuutteihin ja simuloinnin puolella voidaan säästää potentiaalisesti kahden päivän työ, jota tavallisesti kuluisi mallin keventämiseen ja valmisteluun.

## 7.10 Esimerkkejä ongelmista mallissa

Inhimillinen virhe: Yhdessä vaiheessa omassa lattiaradan testauksessani unohdin asettaa robotin *Behaviors*-toimintojen alta löytyvän *ST1*-koordinaatiston kääntöpöydän viimeiseen linkkiin, joka johti robotin kyvyttömyyteen löytää testikappaleen saumoja, koska se pyrki etsimään testikappaleen kiinnityskohtaa maailmanorigosta, joka sijaitsi täysin robotin ulottumattomissa. Kun tämä oli asetettu kääntöpöydän tasoon, viimeiset yhtymäkohdat linkattu oikein ja kinematiikat päivitetty robotti onnistui löytämään kappaleen ja jäljittämään saumaa testauksen aikana. Tämä on myös piirre, joka tulee muistaa päivittää, kun vaihdetaan mihin kääntöpöytään robotti on yhdistetty, koska kaksi pöytää ei voi olla yhdistettynä samanaikaisesti.

Ohjelmiston virhe: WeldControllissa lisäosa ei kyennyt lisäämään luomaansa kalibrointiattribuuttia lattiaradan viimeiselle linkille, koska ohjelma ei jostain syystä tunnistanut hierarkian viimeistä linkkiä *Custom*-tyyppiseksi, eikä tästä syystä kyennyt lukemaan *Joint Name*-arvoa ja hyödyntämään tätä nivelen arvon löytämisessä ja muokkaamisessa. Ongelma kyettiin korjaamaan lisäämällä viimeinen attribuutti lausekkeeseen itse, koska lisäosa oli tehnyt jo kaiken muun tarvittavan. Oletettavasti ylimääräisten linkkien lukumäärä aiheutti jonkinnäköistä sekaannusta lisäosan koodissa, joka aiheutti sen, että lisäosa todennäköisesti pyrki lukemaan yhtä energiaketjujen linkeistä listan viimeisenä linkkinä, mutta koska nuo olivat kaikki tyypiltään lukittuja (Fixed), ei ohjelma kyennyt niitä tunnistamaan käyttökelpoisiksi. Ongelma oli onneksi vielä ratkaistavissa omilla toimilla, vaikka syytä tälle ei varsinaisesti saatu selvitettyä.

Ohjelmiston virhe: Vaihdettaessa aidanrakennusskriptissä kahden solun muodosta yhden solun muotoon, mikäli toisessa solussa oli pöytä kiinni, kun rajapinta piilotettiin ei ohjelma jostain syystä irrottanut tätä kokoonpanosta. Tähän kokeiltiin useita variantteja liitoksen purkamisesta skriptissä, kuten käyttämällä komponenttiin *Disconnect*-komentoa ja *Connect/isConnected*-arvon muuttamista epätodeksi, mutta jopa kokoonpanon päivittämisen jälkeen pöytä oli kiinni liitoksessa ja vaati irrottamiseen *PnP*-työkalua. Lopuksi päätettiin, että poikkeus on niin harvinainen ja ratkaisu yksinkertainen, että tälle ei kannattaisi uhrata enempää aikaa skriptin parissa.

Huolimatta varovaisuudesta ohjelmiston käytössä ja työvaiheiden läpikäymisessä listan kanssa voi silti toistuvien työprosessien kanssa unohtaa yhden, pienen välivaiheen, joka voi kuitenkin johtaa mallin toiminnan häiriintymiseen. Kokemus mallien tekemisessä voi toisinaan olla jopa haitaksi, esimerkiksi kun tekee samankaltaista työtä pidemmän aikaa ja luo paljon samanlaisia malleja voi helposti luottaa liikaa, että on muistanut jokaisen välivaiheen eikä katso ohjeita. Tästä syystä työvaiheiden ohessa on selitetty useaan otteeseen, kuinka piirteen lisäämättä jättäminen voi aiheuttaa ongelmia. Lukija voi halutessaan tehdä oman listan essentiaalisista asioista ja kuitata välivaiheita niitä tehdessään.

# 8 Yhteenveto

Työn aloituksessa haluttiin varmistaa, että keskitytään vain olennaisimpiin etäohjelmoitavan mallin kehityksen vaiheisiin ja vältettäisiin näyttämästä yksityistä informaatiota liittyen laitteiden ja työprosessin tietoihin. Tämä tarkoitti pääasiassa keskittymistä Visual Componentsin ominaisuuksista kertomiseen, koska nämä ovat julkista tietoa Visualin omilta sivuilta saatavana, vaikka ne vaativatkin eri tason maksullisia lisenssejä täyden tason käyttökokemusta varten. Simuloinnin työnkuvassa on kuitenkin päädytty tekemään ratkaisuja malleille, joihin ei suoraan löydy apua Visual Componentsin foorumeilta, mutta joihin voi saada lähtötietoja opetusvideoista ja opettelemalla hieman ohjelmointikieliä, sekä läpikäymällä Visual Componentsin omia API-kirjastoja. OLP-mallia tehdessä pyrittiin käsittelemään ohjelmiston toimintaa tavalla, jolla kyettiin kehittämään tämä työ tukimateriaaliksi samankaltaiseen työskentelyyn. Ihanteellisesti lukijan omatessa riittävät lähtötiedot teoriasta, mallinnusohjelmistosta ja ohjelmoinnista voisi näitä askeleita ja esimerkkejä käyttää edes soveltavasti oman mallin tekemisessä ja valmistelussa tai käyttää tietoa ymmärtääkseen piirteitä Visual Componentsissa paremmin välttääkseen tekemästä yleisiä virheitä.

Realistisesti työn ”valmiin” version ei ole tarkoitus pysyä lopullisena visiona, vaan tarkoituksena oli luoda joustava pohja tulevaisuutta ajatellen. Pemamek Oy:n toiminta jatkaa laajentumistaan vuosi vuodelta, jolloin informaation ja mallien vakioiminen on tärkeä osa aikataulussa pysymistä ja jalkojen maassa pitämistä. Myyntiosastolla malli on jo otettu käyttöön johtuen sen kyvystä helposti luoda alustavat minimivaatimukset turvaetäisyyksille soluissa ja tämän pohjalta voidaan luoda lopullinen OLP-malli jokaiselle variaatiolle hitsausradan lopullisesta asettelusta. Mallia enemmän käytettäessä tulevaisuudessa tullaan varmasti löytämään asioita, joita tulee vielä korjata ja piirteitä, joita voidaan lisätä. Jos asiakas haluaa omaan robottihitsausasemaansa lisäpiirteitä, voidaan nämä suunnitella valmiin pohjan päälle, sen sijaan että tulisi tehdä perinteinen työprosessi 3D-mallin kehittämisestä tämän keventämiseen ja simulointimallin

ominaisuuksien lisäämiseen. Kommunikaatio ja realistiset tavoitteet uudistuksissa tulevat olemaan kriittisiä mallin hyötyjen maksimoimiseksi. Vakioasema konseptina pyrki jo parantamaan yleistä tuottavuutta, helpottamaan hitsausohjelmien luontia, minimoimaan arvuuttelua sopivien ratkaisujen välillä sekä vähentämään materiaalihukkaa, kun toimitaan helpommin ennustettavien rajojen sisällä, joten mallin tarjoama pohja ja tämän tekemisen kautta havaittavat mahdollisuudet on välittömästi opinnäytetyön valmistuttua otettu käyttöön.

# Lähteet

Davis, D. 2013. A History of Parametric. Viitattu 22.02.2023. https://www.danieldavis.com/a-history-of-parametric/.

Finnrobotics. Offline ohjelmointi. Viitattu 23.02.2023. http://finnrobotics.fi/suunnittelu/offline-ohjelmointi/.

Herath, D. & St-Onge, D. 2022. Foundations of robotics. E-kirja Finna-tiedonhakupalvelussa. Viitattu 18.04.2023. https://www.finna.fi/Record/samk.991466572905968?sid=2936539977.

Jankowski, G. & Doyle, R. 2008. SolidWorks for dummies, 2nd Edition. E-kirja Finna-tiedonhakupalvelussa. Vaatii kirjautumisen palveluun. Viitattu 14.04.2023. https://www.finna.fi/Record/samk.991414540805968?sid=2931155448.

Pemamek OY. PEMA hitsaus- ja tuotantoautomaatio – Tuotteet & Ratkaisut. Viitattu 21.02.2023. https://pemamek.com/fi/ratkaisut/.

Pemamek OY. Pemamek – Innovaatio, ammattitaito ja luottamus. Kaikki saman katon alla. Viitattu 21.02.2023. https://pemamek.com/fi/yritys/.

Pemamek OY. PEMA Weldcontrol. Viitattu 17.04.2023. https://pemamek.com/fi/ratkaisut/weldcontrol/.

Prescient Technologies. Types of Geometric Modeling. Viitattu 06.04.2023. https://www.pre-scient.com/knowledge-center/geometric-modelling/types-of-geometric-modelling.html.

Python Software Foundation 2023. General Python FAQ. Viitattu 22.02.2023. https://docs.python.org/3/faq/general.html#what-is-python.

Robots Done Right. Online Robot Programming vs Offline Robot Programming. Viitattu 19.04.2023. https://robotsdoneright.com/Articles/online-robot-programming-vs-offline-robot-programming.html.

Siemens Digital Industries Software. 3D Modeling. Viitattu 22.02.2023. https://www.plm.automation.siemens.com/global/en/our-story/glossary/3d-modeling/17977.

STEP Tools Inc. The STEP Standard ISO 10303 – What is STEP?. Viitattu 12.04.2023. https://www.steptools.com/stds/step/step_1.html.

Visual Components 2020. Pemamek Case Study: The Birth & Evolution of PEMA WeldControl. Viitattu 22.02.2023. https://www.visualcomponents.com/resources/case_studies/pemamek-case-study-the-birth-evolution-of-pema-weldcontrol/.

Visual Components. Products. Viitattu 17.04.2023. https://www.visualcomponents.com/products/premium/.