Jaakko Leinonen

Postprosessorin ohjelmointi Visual Components-siin

Insinööri (AMK)

Konetekniikka

Syksy 2024

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Leinonen Jaakko

Työn nimi: Postprosessorin ohjelmointi Visual Componentsiin

Tutkintonimike: Insinööri (AMK), konetekniikka

Asiasanat: Visual Components, robotiikka, ohjelmointi, voimaohjaus

Tässä opinnäytetyössä selvitettiin, miten Visual Componentsista saadaan käännettyä voimaohjattuja ohjelmia. Työhön kuului olemassa olevan robottisolun mallintaminen, postprosessorin ohjelmointi, kalibrointi, testiohjelmien tekeminen ja niiden ajaminen.

Työn toimeksiantaja oli Kajaanin ammattikorkeakoulu. Kajaanin ammattikorkeakoululla oli valmis robottisolu, jossa oli ABB:n IRB 2400 -robotti ja IRBP A250 -kappaleenkäsittelylaite. Robotin työkalulaipassa oli voima-anturi, ja voima-anturin jatkeena oli sähkökara. Robottisolu mallinnettiin Visual Componentsiin.

Visual Components ei tue tarvittavia voimaohjauskäskyjä, joten Visual Componentsin kääntäjää eli postprosessoria piti muokata. Postprosessori on Python-ohjelmointikielellä kirjoitettu ohjelma, joka tulostaa robotin ohjelman sen omalla ohjelmointikielellä. Uusi postprosessori pystyy tulostamaan tarvittavat voimaohjauskäskyt.

Postprosessorin ohjelmoinnin ja solun mallintamisen jälkeen solumalli kalibroitiin vastamaan fyysistä robottisolua. Työn lopussa tehtiin testiohjelmia, joilla voitiin todeta postprosessorin toimivuus. Uudella postprosessorilla käännetyt testiohjelmat ajettiin oikeassa ympäristössä.

Kajaanin ammattikorkeakoulu pystyy käyttämään solumallia ja postprosessoria robotiikan opetuksessa. Muut pystyvät käyttämään postprosessoria roboteilla, jotka tarvitsevat samoja voimaohjauskäskyjä kuin tässä työssä käytetty robotti.

KAMK • University of Applied Sciences

Abstract

Author: Leinonen Jaakko

Title of the Publication: Programming Post Processor for Visual Components

Degree Title: Bachelor of Engineering, Mechanical Engineering

Keywords: Visual Components, robotics, programming, force control

The aim of this thesis was to modify the post processor of Visual Components to enable the generation of programs containing force control instructions in ABB's RAPID programming language. The work involved modeling an existing robot cell, programming the post processor, creating test programs, and executing these programs to validate the functionality.

This thesis was commissioned by Kajaani University of Applied Sciences, which has a robot cell with ABB IRB 2400 robot and an IRBP A250 workpiece positioner. The robot's tool flange was equipped with a force sensor and an electric spindle. The robot cell was modelled using Visual Components.

Since Visual Components does not natively support the force control commands required by the robot controller, the post-processor, which is a Python script, was modified to translate the robot's program representation into ABB's RAPID language with the necessary force control instructions.

At the end of the work test programs were made to ensure that the post processor works as intended. Test programs were translated by the new post processor and then run in the real environment.

Kajaani University of Applied Sciences can now utilize the cell model and post-processor in robotics education, and the post-processor can be applied in other robot cells requiring similar force control commands.

Sisällys

1 Johdanto

Visual Components on robottien simulointi- ja etäohjelmointiohjelmisto. Visual Components tukee useita robottimerkkejä. Jokaisella robottimerkillä on oma ohjelmointikielensä, ja se tuo haasteita, kun Visual Componentsissa tehty ohjelma halutaan viedä oikealle robotille. Visual Components on ratkaissut tämän ongelman kääntäjillä. Englannin kielessä tästä käytetään nimitystä post processor. Tässä opinnäytetyössä kääntäjästä käytetään nimitystä postprosessori.

Visual Componentsin omat postprosessorit tukevat tavallisia liikekäskyjä ja joitain sovelluskohtaisia käskyjä, kuten hitsauskäskyjä. Tähän opinnäytetyöhön liittyy robottisolu, jossa tarvitaan voimaohjauskäskyjä. Visual Components ei näitä tue, joten postprosessoria joudutaan muokkaamaan. Postprosessori on Python-ohjelmointikielellä kirjoitettu ohjelma, joka kääntää Visual Componentsin esityksen robotin ohjelmasta robotin omalle ohjelmointikielelle.

Työn tilaaja on Kajaanin ammattikorkeakoulu. Kajaanin ammattikorkeakoululla oli valmis robottisolu, jossa oli ABB:n IRB 2400 -robotti ja IRBP A250 -kappaleenkäsittelylaite. Robotin työkalulaippaan oli kiinnitetty voima-anturi, ja voima-anturiin oli kiinnitetty sähkökara. Voima-anturi oli ABB-merkkinen, ja se tuki RAPID-ohjelmointikielen voimaohjauskäskyjä. Tässä opinnäytetyössä postprosessoria muokattiin niin, että postprosessorin tulostamassa koodissa on mukana tarvittavat voimaohjauskäskyt.

Robottisolusta ei ollut valmista mallia Visual Componentsissa, joten se mallinnettiin osana tätä opinnäytetyötä. Mallintamisen jälkeen solumalli piti kalibroida, jotta se vastaisi riittävän hyvin fyysistä robottisolua.

Työn lopussa postprosessorin toimivuus varmistettiin kolmella erilaisella testiohjelmalla. Työn tilaajalle annettiin ohjeet postprosessorin käyttöön.

2 Teollisuusrobotit

Teollisuusroboteiksi lasketaan koneet, joille on yhteistä uudelleen ohjelmoitavuus, yleiskäyttöisyys, vähintään kolmen vapausasteen määrä ja että niissä on työkalu. Yleisiä robottityyppejä ovat SCARA-, delta-, suorakulmainen- ja nivelvarsirobotti. Tässä opinnäytetyössä käsitellään kuusiakselista nivelvarsirobottia. Robotteja käytetään esimerkiksi hitsauksessa, kokoonpanossa ja hionnassa. Robotit parantavat tuottavuutta, koska ne voivat työskennellä taukoamatta. Robotit pitää kuitenkin huoltaa tietyin väliajoin riippuen käyttötunneista. Robotin valinnassa mietitään sen toistotarkkuutta, hyötykuormaa ja ulottumaa. [1.]

Tämän opinnäytetyön robottisolussa on ABB:n IRB 2400 -robotti (kuva 1). Sen toistotarkkuus on 0,03 mm, hyötykuorma 16 kg ja ulottuma 1,55 m [2]. Robotin mukana tulee IRC5-kontrolleri (kuva 3), joka suorittaa robotin ohjelmaa ja ohjaa robottia. Kontrolleriin on tallennettu robotin tietoja, kuten kalibrointitiedot, robotin ohjelmat, koordinaatistot, paikoituspisteet ja robottiin liitetyt ulkoiset akselit. Robotin ulkoisia akseleita ovat robottiin yhdistetyt ulkoiset servo-ohjaimet. Robotti on voitu asettaa radalle, jonka mukana robotti liikkuu [1, s. 136]. Toinen esimerkki ulkoisista akseleista on kappaleenkäsittelylaite. Tämän opinnäytetyön robottisolussa on kuvassa 2 näkyvä kappaleenkäsittelylaite IRBP A250, jossa on kaksi akselia: kääntöakseli (englanniksi plate) ja kallistusakseli (englanniksi arm).

Kuva 1 IRB 2400

Kuva 2 IRBP A250

Kontrollerin mukana tulee käsiohjain, joka näkyy kontrollerin päällä kuvassa 3. Käsiohjainta käytetään robotin ohjelmoimiseen, kalibroimiseen ja asetusten muuttamiseen. Käsiohjaimella voidaan luoda työkohdekoordinaatistoja ja paikoituspisteitä. Robotin työkalun määritykset kuten TCP:n ja massadatan määritys, tehdään käsiohjaimella. Näytön sivussa on sauvaohjain, jonka avulla robottia voidaan liikuttaa. Robottia tarvitsee liikuttaa, kun muokataan tai luodaan uusia paikoituspisteitä ja koordinaatistoja. Robottia liikutellaan myös silloin, kun määritetään työkalupistettä eli TCP:tä. Käsiohjaimella robottia voidaan liikuttaa kahdella tavalla: koordinaatiston suhteen tai akseleiden suhteen. Kun robottia liikutellaan koordinaatiston suhteen, valitaan ensin, minkä koordinaatiston suhteen. Eri koordinaatistoja ovat esimerkiksi maailmakoordinaatisto, työkohdekoordinaatisto ja työkalukoordinaatisto eli TCP:n koordinaatisto.

Käsiohjaimella voidaan liikuttaa myös ulkoisia akseleita. Käsiohjaimen sauvaohjaimella voidaan ohjata kolmea asiaa kerralla. Sauvaohjainta voidaan kääntää vasemmalle, oikealle, ylös ja alas. Lisäksi sauvaohjainta voidaan kääntää myötä- ja vastapäivään. Jos robottia liikutellaan koordinaatiston suhteen, jossa z-akseli osoittaa ylöspäin, niin robotti liikkuu ylöspäin vastapäivään käännettäessä ja alaspäin myötäpäivään käännettäessä. Käsiohjaimen alapuolella on sallintapainike, jonka pitää olla painettuna robottia liikuteltaessa ja myös silloin, kun robotin ohjelmaa testataan. Käsiajossa robotin liikenopeudet on rajoitettu nopeuteen 250 mm/s [1, s. 122].

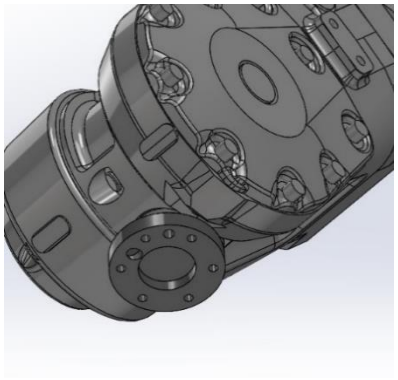
Kuva 3 IRC5-kontrolleri

Ennen kuin robotti voi tehdä mitään hyödyllistä, siihen on kiinnitettävä työkalu. Työkalu kiinnitetään robotin työkalulaippaan (kuva 4). Työkalu voi olla esimerkiksi tarttuja, hitsauslaite, ruuvinväännin tai kara (kuva 6). Robotin tiedoissa annettu hyötykuorma kertoo, kuinka suuri massa työkalulaipassa saa olla. Esimerkiksi, jos robotin hyötykuorma on 10 kg ja työkaluna olisi tarttuja, joka painaa 4 kg, niin tarttuja voi nostaa 6 kg:n painoisia kappaleita. Työkalulaippaan voidaan kiinnittää myös työkalunvaihtaja, jonka avulla robotti pystyy vaihtamaan itse työkalunsa ilman ihmisen

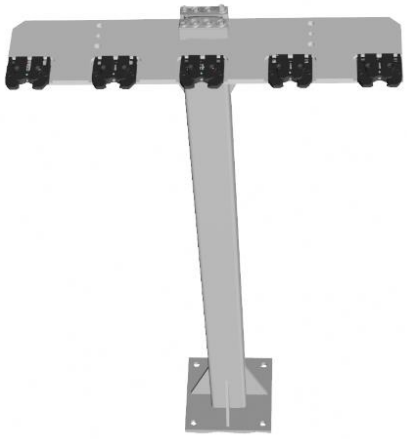
apua. Työkalunvaihtajat ovat sähkö- tai paineilmatoimisia. Työkalulaipan ja työkalun välissä voi olla myös erilaisia sovitelevyjä tai esimerkiksi voima-anturi. [1, s. 201–226.]

Tämän opinnäytetyön robotin työkalulaippaan on kiinnitetty voima-anturi, sovitelevyjä karaa ja johtoja varten sekä kara. Karaan pystyy kiinnittämään erilaisia työkaluja, kuten hiontapäitä tai jyrsin- ja poranteriä. Karassa on paineilmajäähdytys. Maksimikierrosnopeus on 24 000 kierrosta minuutissa. Karan työkalunlukitusmekanismi käyttää hyväkseen paineilmaa.

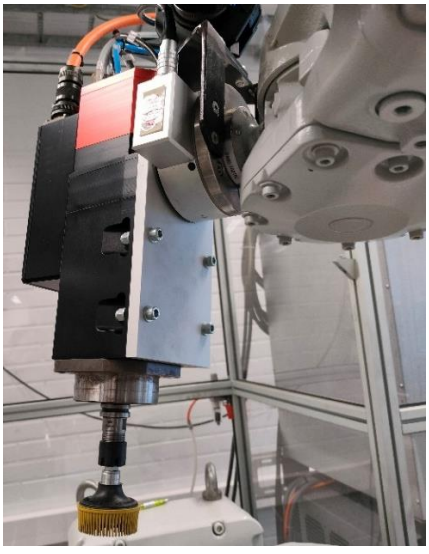
Robottisolussa on työkaluteline, jossa on viisi eri paikkaa. Kun työkalu vaihdetaan, robotti vapauttaa työkalun tyhjälle paikalle ja ottaa tilalle uuden työkalun toisesta paikasta. Tämän onnistumiseksi robotilla on oltava tieto työkalujen paikoista.



Kuva 4 Robotin työkalulaippa



Kuva 5 3D-malli työkalutelineestä



Kuva 6 Robottiin kiinnitetty kara

3 Voimaohjaus

Robotin työkalulaipan jatkeeksi voidaan kiinnittää voima-anturi (kuva 7). Voima-anturiin voidaan edelleen kiinnittää esimerkiksi kara, kuten aikaisemmin kuvassa 6. Voima-anturin avulla työstettävään kappaleeseen voidaan kohdistaa haluttu voima. Tämä on yksi tapa toteuttaa voimaohjaus. Toinen tapa olisi käyttää ulkoista laitteistoa. Voimaohjausta käytetään erityisesti hiontasovelluksissa. [1, s. 168–169, 224.]

Kuva 7 Työkalulaipan jatkeeksi kiinnitetty voima-anturi

Tämän opinnäytetyön robotin voimaohjaus on tehty ABB:n valmiilla ratkaisulla. Tämä tarkoittaa sitä, että mukana ei tule pelkästään voima-anturia, vaan myös voimaohjauskäskyt, jotka helpottavat ohjelmointia. Voimaohjauskäskyt mahdollistavat esimerkiksi lineaariliikekäskyn, jossa työkalu painaa työstettävää kappaletta halutulla voimalla ja liikkuu samaan aikaan halutulla nopeudella haluttuun paikkaan. [3.]

4 Robottien ohjelmointi

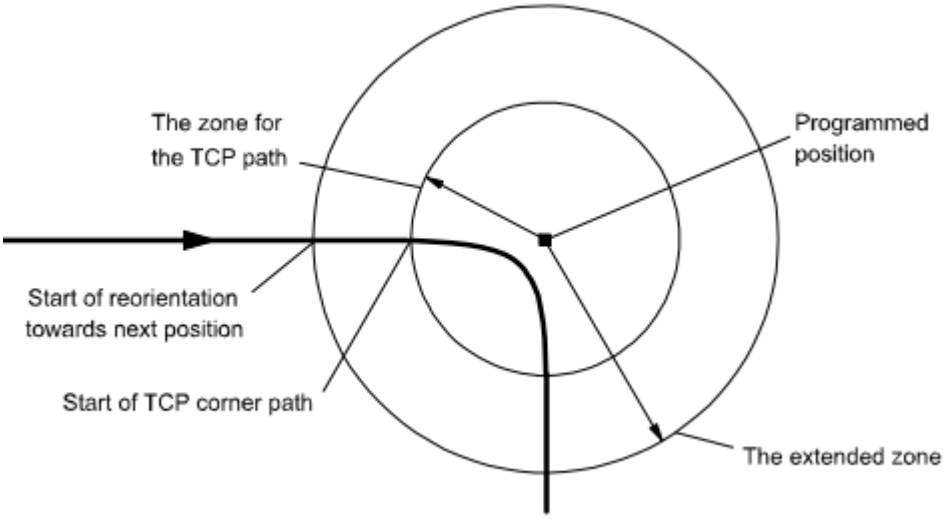
Ohjelmoinnin tavoite on saada robotti suorittamaan tehtävänsä. Yleensä robotteja ohjelmoidaan tekstipohjaisella ohjelmointikielellä, joka riippuu robotin valmistajasta. Esimerkiksi ABB:n robotteja ohjelmoidaan ABB:n omalla RAPID-ohjelmointikielellä. Ohjelmointikielissä eri robottivalmistajien välillä on hyvin samanlaiset käytännöt. Tässä opinnäytetyössä käsitellään ABB:n robottia, joten robottien ohjelmointia käsitellään ABB:n ja RAPID-ohjelmointikielen näkökulmasta.

Robotin ohjelmassa on aina pääohjelma, josta ohjelman suoritus alkaa. Yksinkertaisimmillaan robotin ohjelmassa on pääohjelma, jonka sisällä on liikekäskyjä liikkeitä varten ja IO-käskyjä tarttujan – tai jonkun muun robotin työkalun – ohjausta varten. Pääohjelman lisäksi voi olla myös aliohjelmia, joita kutsutaan pääohjelmasta. Työkalun eri toiminnot on voitu kirjoittaa omiin aliohjelmiinsa. Esimerkiksi tarttujan tapauksessa voisi olla kiinni- ja auki-asennnolle omat aliohjelmansa. Tarttujassa voi olla antureita, jotka tunnistavat tarttujan asennon. Aukaisu-aliohjelmaan voidaan kirjoittaa ensin IO-käsky, jolla tarttuja aukaistaan, minkä jälkeen voidaan odottaa, kunnes anturi antaa tiedon, että tarttuja on auki-asennossa. Tässä opinnäytetyössä karan käynnistäminen ja pysäyttäminen ovat omissa aliohjelmissaan. Jos aliohjelmia ei olisi, niiden sisältö pitäisi kirjoittaa sinne, missä niitä kutsutaan. Tämän takia aliohjelmat vähentävät koodirivien määrää ja tekevät siten koodista helpommin hallittavaa. [1, s. 242.]

Robottia ohjataan eri paikkoihin liikekäskyillä. Yleisimmät liikekäskyt ovat nivelliike ja lineaariliike. Kumpaakin käskyä käytettäessä määritellään, mihin robotin työkalu halutaan viedä ja missä orientaatiossa työkalun tulee olla. Lineaariliikkeessä robotti liikuttaa työkalua suoraviivaisesti siitä pisteestä, missä se oli ennen liikekäskyä siihen pisteeseen, mikä annettiin liikekäskyssä. Nivelliikkeessä työkalun liike ei ole suoraviivainen, vaan robotti laskee sille mahdollisimman yksinkertaisen tavan liikkua oikeaan paikkaan liikeradasta välittämättä. Nivelliike on robotille helpompi kuin lineaariliike. Nivelliikettä käytetään yleensä silloin, kun lähestytään jotain tarkkuuttaa vaativaa tehtävää. RAPID-kielessä on myös kaariliike, jossa robotin työkalu saadaan liikkumaan ympyrän kaarta pitkin. Kaikkia liikekäskyjä yhdistää se, että niille annetaan paikoituspiste, liikenopeus, aluetarkkuus, työkalupiste eli TCP ja yleensä myös työkohdekoordinaatisto. [1, s. 234–238.]

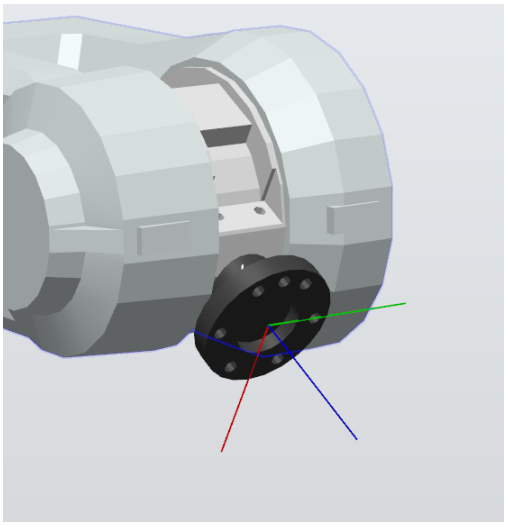
Liikekäskyn aluetarkkuus kertoo, kuinka paljon robotti voi oikoa, kun se muuttaa suuntaansa seuraavaa paikoituspistettä kohti. Jos robotin pitää käydä paikoituspisteen kohdalla, käytetään aluetarkkuutta fine. Teoriassa robotti pysähtyy paikoituspisteessä, ennen kuin se jatkaa seuraa-

vaan pisteeseen. Jos ei käytetä fine-aluetarkkuutta, niin aluetarkkuus voi olla esimerkiksi z10. Silloin robotti alkaa muuttamaan suuntaansa seuraavan liikekäskyn paikoituspistettä kohti 10 mm ennen nykyisen liikekäskyn paikoituspistettä. Kuva 8 näyttää, minkä muotoinen robotin liikerata on, kun aluetarkkuutena ei ole fine. [1, s. 236.; 4, s. 1733–1739.]

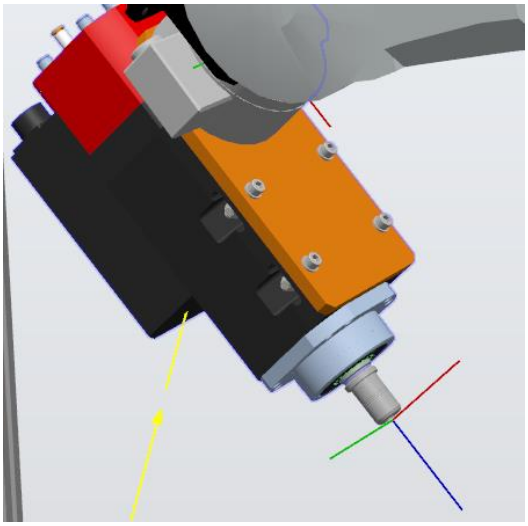


Kuva 8 Aluetarkkuuden vaikutus liikekäskyyn [4, s. 1733]

Liikekäskyssä robotin työkalu halutaan viedä tiettyyn pisteeseen ja tiettyyn orientaatioon koordinaatistossa. Robotin työkalu ei kuitenkaan ole piste, vaan robotin työkalun voidaan ajatella olevan ääretön määrä pisteitä. Tämän takia on määriteltävä työkalupiste eli TCP. TCP:llä on x, y- ja z-koordinaatit, joiden nollakohta on työkalulaipan keskellä. Työkalulaipan keskipistettä kutsutaan tool0:ksi. Kuva 9 näyttää, miten tool0 on määritelty. Punainen on x-, vihreä y- ja sininen z-akseli. TCP:llä on koordinaattien lisäksi orientaatio. Kuvan 10 yläosasta voi nähdä työkalulaipan. Jos sitä verrataan kuvaan 9, voi nähdä, että kuvassa 9 z-akseli on kohtisuorassa laippaan nähden, mutta kuvassa 10 z-akseli on käännetty työkaluun nähden sopivaksi, niin että se osoittaa karan kanssa samaan suuntaan. Näitä koordinaattiakseleiden asentoja kutsutaan orientaatioksi. RAPID-ohjelmointikielessä TCP:n yhteydessä on annettu myös työkalun massaan liittyviä tietoja. Niiden avulla robotti pystyy tunnistamaan törmäykset ja säilyttämään liiketarkkuuden arvioimalla akseleiden tarvitseman sähkövirran määrää.



Kuva 9 TCP tool0

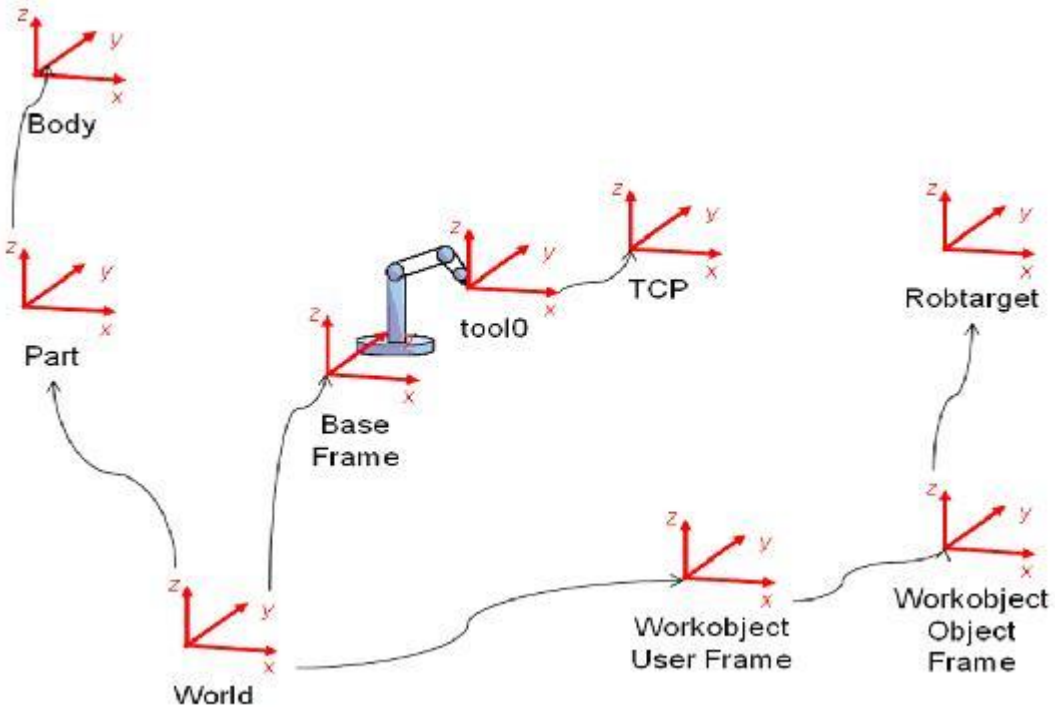


Kuva 10 Työkalun kärkeen määritelty TCP

Paikoituspisteet ovat robotin ohjelmassa määriteltyjä pisteitä, joihin robotin on tarkoitus liikkua ohjelman ajon aikana. RAPID-kielessä paikoituspiste sisältää x-, y- ja z-koordinaatit, orientaation, akselikonfiguraation ja ulkoisten akseleiden asennot. Paikoituspisteitä ei ole sidottu mihinkään koordinaatistoon eikä TCP:hen, vaan ne annetaan vasta liikekäskyn yhteydessä. Sen takia paikoituspisteitä tehtäessä on muistettava valita oikea TCP ja työkohdekoordinaatisto. [1, s. 233.]

Minkään paikoituspisteen koordinaatit ja orientaatio ei kerro mitään, jos ei tiedetä, minkä koordinaatiston suhteen ne on annettu. Robottien ohjelmoinnissa on käytössä monenlaisia koordinaatistoja. Kaikilla niillä on orientaatio ja nollapiste eli origo. Erityinen koordinaatisto on robotin maailmankoordinaatisto. Kaikki muut koordinaatistot ovat suoraan tai välillisesti yhteydessä maailmankoordinaatistoon (kuva 11). Base Frame on robotin peruskoordinaatisto, joka on sidottu robotin jalustaan. Jos robotti asennetaan kallelleen, peruskoordinaatisto kääntyy mukana.

Liikekäskyissä annetaan yleensä työkohdekoordinaatisto. Työkohdekoordinaatisto asetetaan tarkempaa työtä vaativan kohteen lähelle. Työkohdekoordinaatistoja voi olla useampia. Esimerkiksi tämän opinnäytetyön robottisolussa työkalutelineelle ja työstettävälle kappaleelle on omat työkohdekoordinaatistonsa. Työkohdekoordinaatistoista käytetään RAPID-ohjelmoinnissa nimitystä Workobject ja paikoituspisteistä nimitystä Robtarget. Myös paikoituspisteet voidaan ajatella koordinaatistoina, koska niillä on origo ja orientaatio. RAPIDissa on RelTool-käsky, jolla voidaan liikkua paikoituspisteen koordinaatistossa [4, s. 1371].



Kuva 11 Koordinaatistoja [5]

Liikekäskyssä annettu liikenopeus kertoo robotille, kuinka nopeasti se voi työkalua liikuttaa. Liikenopeus voi olla esimerkiksi v100, joka tarkoittaa, että robotin tulee liikuttaa työkalua nopeudella 100 mm/s. RAPIDissa on valmiina määritelty joitakin liikenopeuksia, kuten v100, v200, jne. Jos sopivaa nopeutta ei löydy valmiina, voidaan tehdä uusi speeddata -tyyppinen muuttuja, johon liikenopeus asetetaan yksikössä mm/s. Kuva 12 näyttää, miten tämä voitaisiin tehdä. Kuvassa toinen lukuarvo on työkalun orientaation muutos yksikössä °/s. Kaksi viimeistä ovat ulkoisten akseleiden liikenopeuksia. Toiseksi viimeinen on lineaaristen akseleiden nopeus yksikössä mm/s, ja viimeinen on pyörivien akseleiden liikenopeudet yksikössä °/s. [4, s. 1674.]

```
VAR speeddata vNopeus:=[25, 500, 5000, 1000];
```

Kuva 12 Liikenopeusmuuttujan määritteleminen

RAPID-ohjelmointikielessä muuttujalla on nimi, tyyppi ja arvo. Yleisimpiä ovat num-, bool- ja string-tyyppiset muuttujat. Num on lukuarvojen varastointia varten. Bool-tyyppinen muuttuja voi saada vain kaksi arvoa: tosi ja epätosi. String-tyyppi on tekstin varastointia varten. RAPID-kielessä on myös monimutkaisempia tyyppejä, kuten kuvan 12 speeddata. Kun muuttujan arvoa halutaan käyttää käskyssä, muuttujan nimi kirjoitetaan sille paikalle. Jokainen muuttuja on määriteltävä ennen kuin niihin viitataan ohjelmassa. Muuttujilla vaikutetaan ohjelman suoritukseen, ja muuttujien avulla ohjelman voi kirjoittaa niin, että suuria koodimuutoksia ei tarvita ohjelman kulun

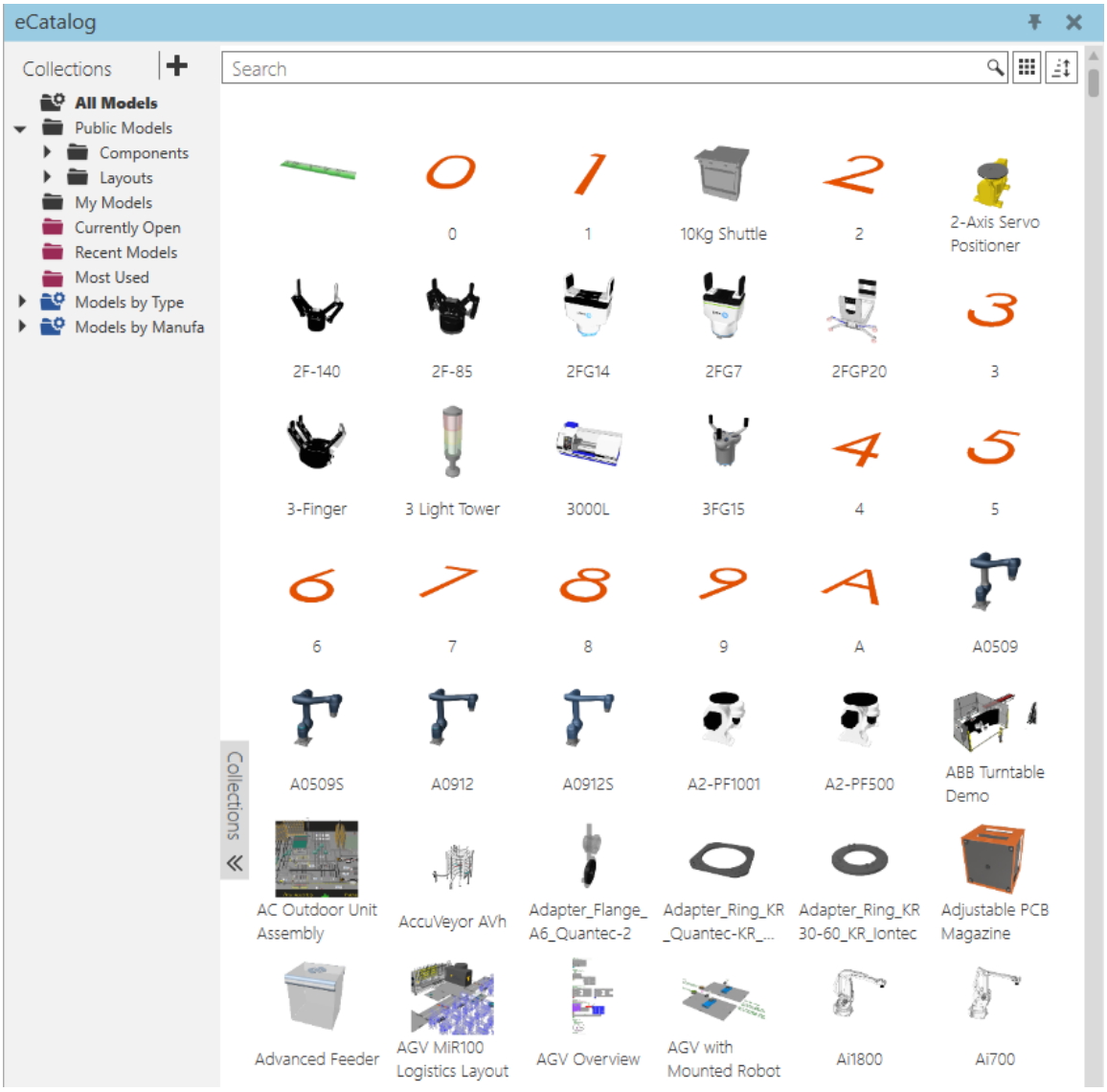
muuttamiseksi. Tässä voidaan käyttää esimerkkinä kuvan 12 vNopeus-muuttujaa. Jos liikekäskyjä olisi useita, voitaisiin jokaiseen liikekäskyyn nopeudeksi asettaa vNopeus. Silloin kaikkien liikekäskyjen liikenopeuksia voidaan ohjata pelkästään muuttamalla vNopeus-muuttujan arvoa. Usein, kun puhutaan muuttujista, tarkoitetaan vakiota. Vakio määritellään ja alustetaan melkein samalla tavalla kuin muuttuja. Ero tulee muuttujan määrittelyssä, jossa käytetään VAR-, PERS- tai CONST-sanaa. Näistä kahta ensimmäistä pystyy muuttamaan määrittelyn jälkeen, ja PERS säilyttää arvonsa, vaikka robotin ohjaimesta kävisi virrat pois. CONST-sanalla merkityn muuttujan arvoa ei pysty muuttamaan ajon aikana.

Koodia voi kirjoittaa monella tavalla ja sitä on haluttu yhdenmukaistaa nimeämiskäytännöillä [1, s. 230]. Yleensä jokaisessa yrityksessä on omat nimeämiskäytäntönsä. Kun koodi on samanlaista kaikkialla, sitä on helpompi lukea ja ymmärtää. Tämän opinnäytetyön robottisolussa on käytetty esimerkiksi sellaisia nimeämiskäytäntöjä, että työkohdekoordinaatiston nimi alkaa aina pienellä w-kirjaimella ja sen jälkeen tulee työkohdekoordinaatiston nimi, jonka ensimmäinen kirjain on iso kirjain. Toinen esimerkki voisi olla TCP:n nimeäminen. TCP:n nimi alkaa aina pienellä t-kirjaimella, ja sen jälkeen tulee TCP:n nimi samalla tyylillä kuin työkohdekoordinaatiston nimessä.

5 Robottijärjestelmien simulointi

On olemassa tietokoneohjelmia, joilla voidaan simuloida robottijärjestelmiä. Robottisolu mallinnetaan simulointiohjelmaan, minkä jälkeen robotille tehdään ohjelmia, jotka ajetaan simulointiohjelmassa. Simulointivaiheessa voidaan nähdä, pystyykö robotti suorittamaan sille ohjelmoituja liikeratoja. Liikeradan suorittaminen voi estyä esimerkiksi törmäykseen tai kulmarajoihin. Nämä voidaan ratkaista muokkaamalla liikeradan paikoituspisteitä tai muuttamalla robotin akselikonfiguraatiota. Simulointiohjelmat mahdollistavat liikeratojen luomisen käyttämällä apuna työstettävien kappaleiden geometriaa. Kappaleesta voidaan valita esimerkiksi särmä, josta simulaatio-ohjelma tekee liikeradan. Mukana voi tulla lähestymis- ja poistumispisteet. Liikerataa luotaessa voi ottaa myös ulkoiset akselit käyttöön. Näin voidaan kääntää esimerkiksi kappaleenkäsittelylaitetta paikoituspisteiden välillä. [1, s. 252–260.]

Robottisolu voidaan mallintaa simulointiohjelmaan ennen fyysistä robottisolua tai vasta fyysisen robottisolun rakentamisen jälkeen. Osa tarvittavista komponenteista löytyy simulointiohjelmiston komponenttikirjastosta (kuva 13). Osat, joita ei löydy komponenttikirjastosta, etsitään komponenttien valmistajien sivuilta tai mallinnetaan itse. Ne komponentit, jotka löytyvät valmiina komponenttikirjastosta, sisältävät kinematiikan valmiina. Niille komponenteille, jotka joudutaan tuomaan simulointiohjelmiston ulkopuolelta, joudutaan kinematiikka rakentamaan yleensä itse. Tässä opinnäytetyössä kappaleenkäsittelylaitteeseen piti rakentaa kinematiikka itse, koska kappaleenkäsittelylaitetta ei löytynyt suoraan komponenttikirjastosta. [1, s. 252–260.]



Kuva 13 Kuvakaappaus Visual Componentsin komponenttikirjastosta

Ennen kuin robotin ohjelma viedään simulaatiosta oikealle robotille, fyysisestä robottisolusta otetaan tärkeät mitat simulaatiomalliin. Tärkeitä mittoja ovat ulkoisten akseleiden sijainnit, TCP:t ja työkohdekoordinaatistot. Simulaatiomallissa kappaleita liikutellaan, niin että niiden paikat ja asennot vastaavat mahdollisimman tarkasti fyysistä robottisolua. Tätä sanotaan kalibroinniksi. Jos kalibrointi jätetään tekemättä, robotti ei välttämättä pysty suorittamaan ohjelmaa lopullisessa ympäristössä, ja ohjelmaan joudutaan tekemään muutoksia. Jos muutoksia joudutaan tekemään, ne olisi parempi tehdä simulaatiomalliin, jotta se vastaisi fyysistä järjestelmää. [1, s. 258–261.]

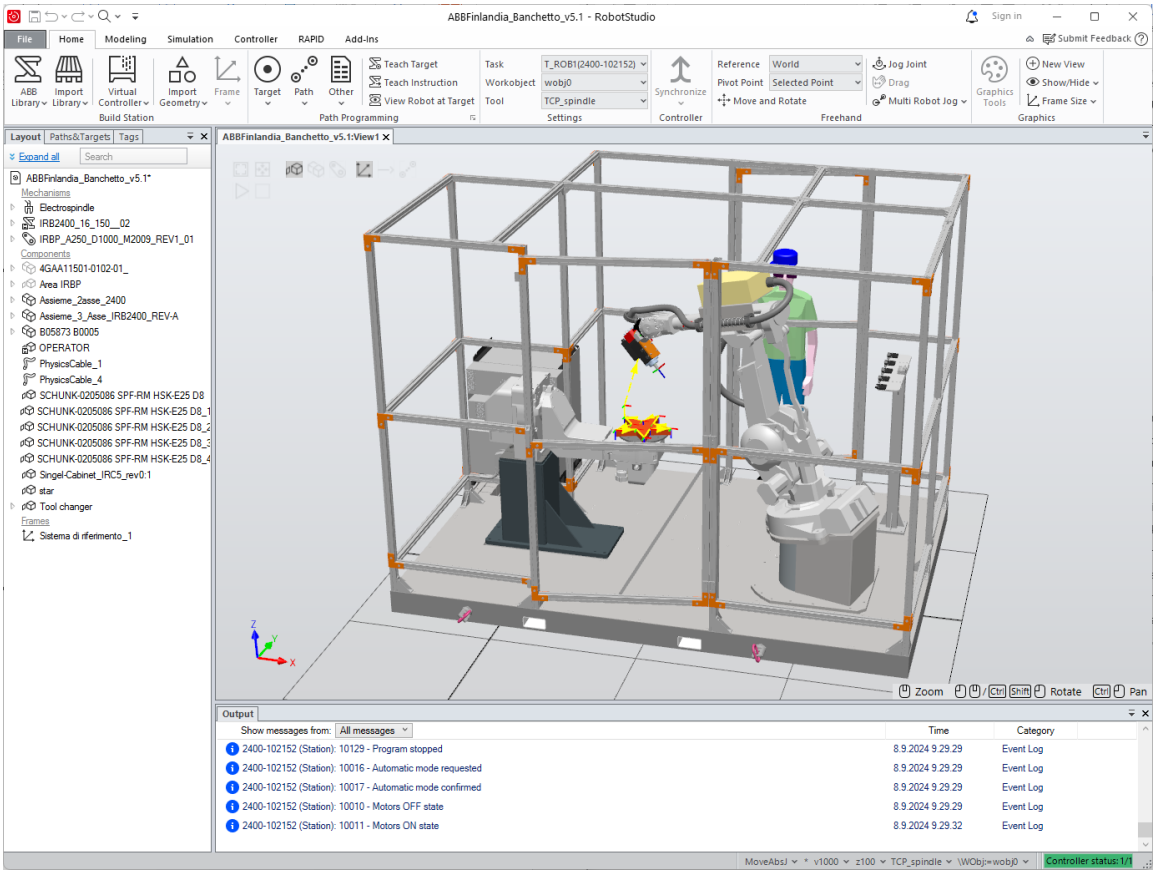
Robottivalmistajilla on omia merkkikohtaisia simulointiohjelmistoja, jotka on tarkoitettu heidän omien robottiensa simuloimiseen ja ohjelmoimiseen. Robottivalmistajien omissa simulointiohjelmistoissa komponenttikirjastot ovat melko suppeita, ja ne sisältävät vain robottivalmistajan omat

robotit ja mahdollisesti simulointiin tarvittavia komponentteja, kuten turva-aitoja ja kuljettimia. Robottivalmistajien omilla simulointiohjelmistoilla ohjelmoidaan robottivalmistajan omalla ohjelmointikielellä. Tämä nopeuttaa ohjelman viemistä tuotantoon, koska koodi on valmiiksi oikeassa muodossa. Robottivalmistajien omien ohjelmistojen lisäksi on yleiskielisiä simulointiohjelmistoja, joissa pystytään simuloimaan useamman eri robottivalmistajan robotteja. Yleiskielisissä simulointiohjelmistoissa on laajemmat komponenttikirjastot, ja niissä on panostettu prosessikohtaisiin ohjelmointimenetelmiin, kuten hitsaukseen. Haasteena on robotin ohjelman kääntäminen robotin omalle ohjelmointikielelle. Aina ei ole mahdollista luoda koodia, joka toimisi oikealla tavalla fyysisessä robotissa. [1, s. 261–263.]

5.1 RobotStudio

RobotStudio on ABB:n oma ohjelmisto, jossa voidaan ohjelmoida ja simuloida vain ABB:n omia robotteja [1, s. 262]. RobotStudiossa robotteja ohjelmoidaan RAPID-ohjelmointikielellä. RobotStudiossa voi käyttää virtuaaliohjaimia, jotka pystyvät ajamaan RAPID-koodia ilman fyysistä ohjainta. RobotStudiossa pystyy luomaan työstöratoja, ja RobotStudio generoi RAPID-koodin näille automaattisesti. Kuva 14 näyttää RobotStudion käyttöliittymän Home-välilehdellä.

Tässä opinnäytetyössä RobotStudiota tarvittiin silloin, kun postprosessorin tulostama ohjelma vietiin oikealle robotille. RobotStudion pystyy yhdistämään robotin ohjaimeen verkkokaapelin avulla.



Kuva 14 RobotStudion käyttöliittymä

5.2 Visual Components

Visual Components on yleiskielinen simulointiohjelmisto, jonka komponenttikirjastossa on yli 1600 robottia 70:ltä eri robottivalmistajalta [6]. Visual Componentsin komponenttikirjaston nimi on eCatalog. Tässä opinnäytetyössä on käytössä Visual Components Premium OLP 4.9. OLP-versiossa on työkaluja, joilla voi luoda liikeratoja esimerkiksi hitsaukseen tai pintakäsittelyyn. Visual Componentsissa robotin ohjelma pitää kääntää robotin omalle ohjelmointikielelle. Englannin kielessä tästä käytetään nimitystä *post processing*. Tässä opinnäytetyössä kääntäjää kutsutaan postprosessoriksi. Koska jokaisella robottivalmistajalla on oma ohjelmointikielensä, jokainen postprosessori on tehty tiettyä robottimerkkiä varten. Visual Componentsissa on 17 robottivalmistajan postprosessorit [6]. Postprosessori on käytännössä Python-ohjelma, joka tulostaa tiedostoon robotin ohjelman robotin omalla ohjelmointikielellä. Jos lopullisessa koodissa tarvitaan käskyjä, joita Visual Componentsin oma postprosessori ei tue, joudutaan tekemään uusi postprosessori.

5.3 Python

Python on yleiskäyttöinen ohjelmointikieli. Python on nopea oppia, ja koodin hallittavuus on helppoa verrattuna moniin muihin ohjelmointikieliin [7]. Pythonissa muistinhallinta on tehty roskienkeruu-menetelmällä, mikä tarkoittaa, että ohjelmoijan ei itse tarvitse huolehtia muistin varaamisesta ja vapauttamisesta.

Python on alustariippumaton ja tulkattu kieli. Koska Python on tulkattu kieli, se vaatii tulkin, joka lukee ohjelmoijan kirjoittaman koodin ja suorittaa sen. Python on käännettäviin ohjelmointikieliin verrattuna hitaampi aikaa vievän tulkkausvaiheen takia. [8.]

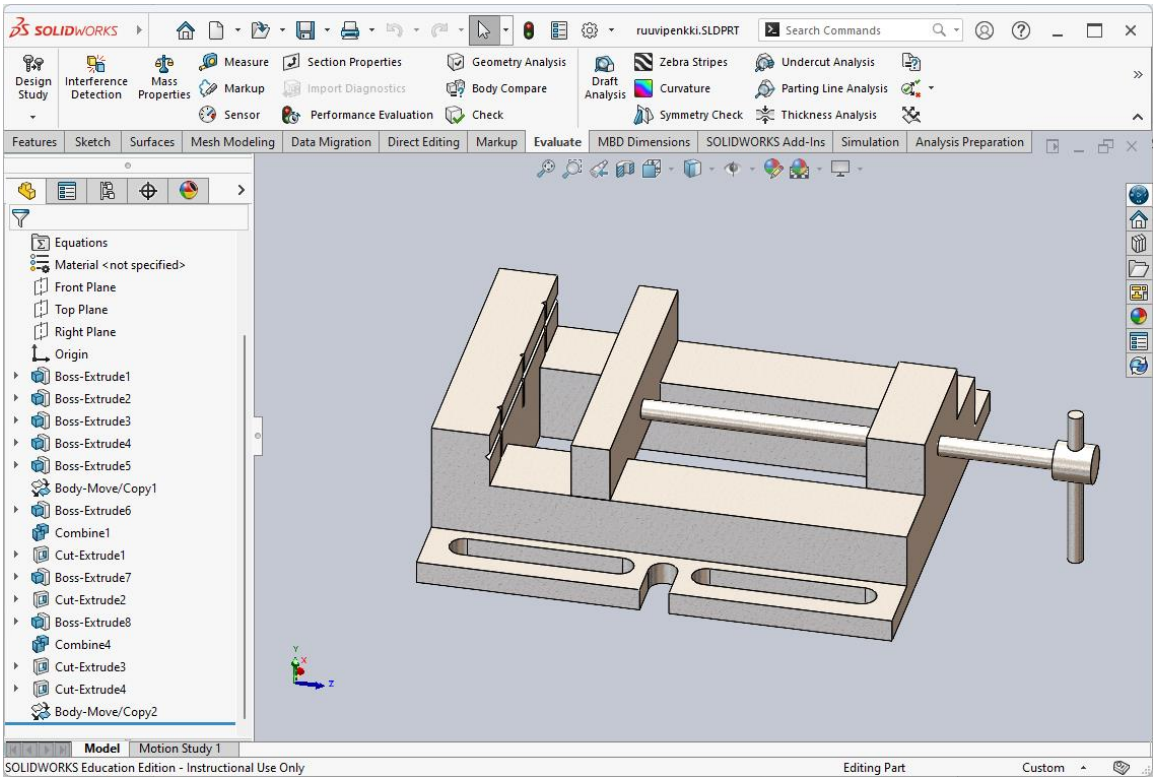
Tässä työssä Pythonia tarvittiin Visual Componentsin postprosessorin ohjelmoimisessa. Visual Componentsissa Pythonia käytetään myös skriptikielenä, mikä mahdollistaa mukautettujen toimintojen tekemisen, joita ei graafisessa käyttöliittymässä pysty tekemään.

6 3D-mallinnus

Robottien simulointiohjelmistoissa käytetään paljon 3D-malleja. Puhutaan myös mallipohjaisesta etäohjelmoinnista [1, s. 251]. Mallitiedon pohjalta työstöratojen luominen onnistuu lähes automaattisesti. 3D-malleja ei käytetä pelkästään työstöratojen suunnittelussa, vaan niitä tarvitaan kaikkialla robottisolun simulaatiomallissa. Yleensä 3D-mallit löytyvät valmiina internetistä valmistajan sivuilta, mutta osan kappaleista voi joutua mallintamaan itse.

Tässä opinnäytetyössä kappaleenkäsittelylaitteeseen oli kiinnitetty ruuvipenkki, josta ei löytynyt valmista 3D-mallia. Ruuvipenkki mallinnettiin SolidWorks-ohjelmistolla. SolidWorksissa pystyy mallintamaan osia (Part), kokoonpanoja (Assembly) ja piirustuksia (Drawing). Tässä opinnäytetyössä tarvittiin ainoastaan Part-tyyppisiä tiedostoja.

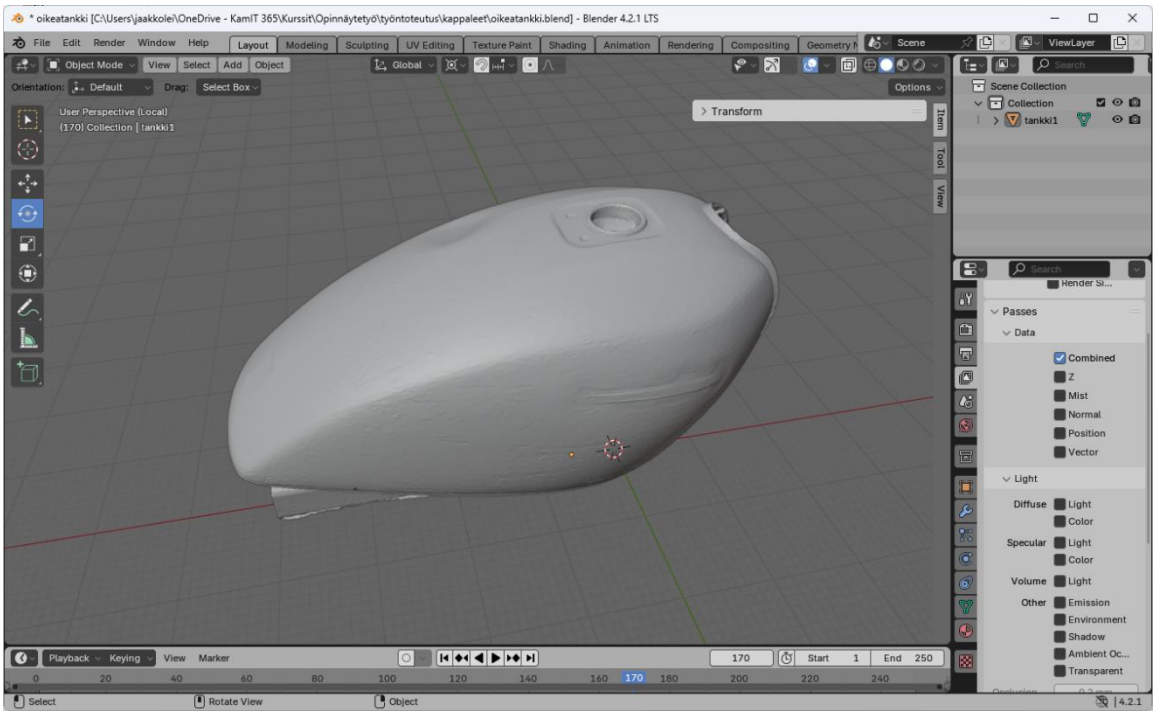
SolidWorksissa osan mallinnus perustuu piirteisiin. Piirrehistoriassa olevia piirteitä pystyy yleensä muokkaamaan ilman, että uudemmat piirteet menisivät rikki. Piirrehistoria näkyy kuvassa 15 vasemmalla.



Kuva 15 Osatiedosto SolidWorksissa

SolidWorksiin pystyy tuomaan 3D-malleja muista ohjelmista. Tässä opinnäytetyössä skannattiin 3D-skannerilla moottoripyörän tankki, ja se oli .obj-muodossa. Toisin kuin SolidWorksin osatiedosto, joka koostuu piirteistä, .obj sisältää 3D-mallin polygonimuodossa. Jokainen polygoni muodostuu vertekseistä eli pisteistä. 3D-skannattu tiedosto voi sisältää niin paljon verteksejä, että sen käyttäminen simulaatio-ohjelmistossa olisi liian raskasta. SolidWorksissa pystyy yksinkertaistamaan tällaisia 3D-malleja. Tämä SolidWorksin ominaisuus oli kuitenkin liian hidas käyttää, joten yksinkertaistaminen tehtiin Blender-ohjelmistolla.

Blender on 3D-mallinnusohjelmisto, joka on suosittu etenkin pelinkehittäjien keskuudessa. Videopelit on optimoitu piirtämään 3D-mallit polygonimalleina. Blender pystyy käsittelemään tehokkaasti raskaitakin polygonipohjaisia tiedostoja. Vaikka kuvassa 16 oleva tankki oli tiedostokooltaan yli 200 megatavua, Blenderin suorituskyky pysyi riittävällä tasolla.



Kuva 16 3D-skannattu moottoripyörän tankki Blenderissä

7 Lähtötilanne

Ennen työn aloittamista Kajaanin ammattikorkeakoululla oli valmis robottisolu, jossa oli ABB:n IRB 2400/16 -robotti ja IRBP A -kappaleenkäsittelylaite. Robotin työkaluna oli ELTE AF80 7/2 CU -kara. Karan ja työkalulaipan välissä oli voima-anturi. Karaan voidaan kiinnittää monenlaisia työkaluja, kuten poranteriä, jyrsinteriä ja hiontapäitä.

Robottisolusta oli valmis malli RobotStudiossa. Robotin työkalulaipassa oli kiinni kappale, joka tuki karalle tulevia kaapeleita. Ensimmäisessä versiossa tuki rajoitti robotin akselin liikealuetta, ja sitä oli mallin tekemisen jälkeen muokattu. Tätä opinnäytetyötä tehdessä muutos otettiin mukaan Visual Componentsiin.

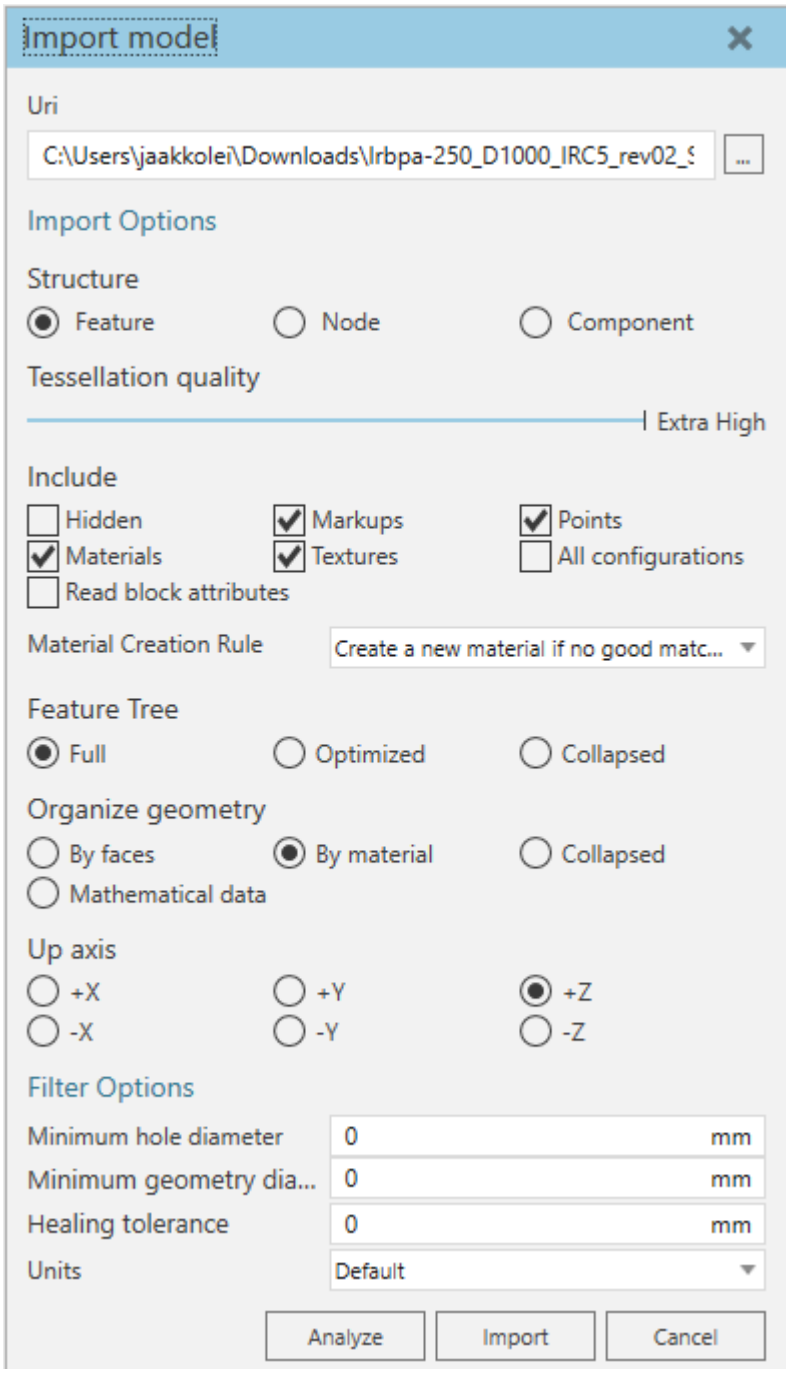
8 Työn toteutus

Työn ensimmäinen vaihe oli rakentaa kappaleenkäsittelylaite, koska sitä ei löytynyt Visual Componentsin komponenttikirjastosta. Kun kappaleenkäsittelylaite oli mallinnettu, ja siinä oli kinematiikka ja muu toiminnallisuus mukana, voitiin komponenttikirjastosta tuoda robotti ja alkaa tekemään robotille ohjelmia. Nämä riittivät postprosessorin ohjelmoimiseen ja yksinkertaiseen testaamiseen. Kun postprosessori oli valmis, robottisolu mallinnettiin loppuun. Valmiiseen malliin suunniteltiin ja ohjelmoitiin testiohjelmia, joiden toimivuus varmistettiin oikeassa ympäristössä.

8.1 Kappaleenkäsittelylaitteen mallintaminen

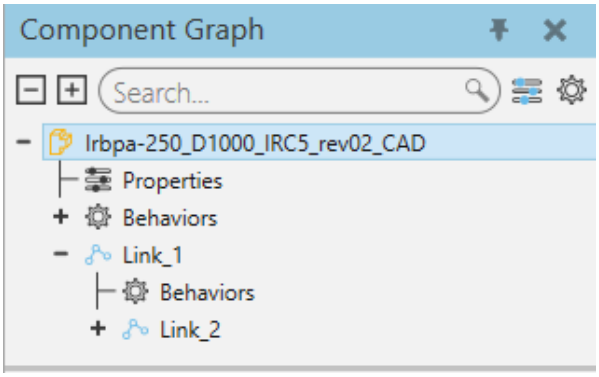
Robottisolussa on ABB:n IRBP A kappaleenkäsittelylaite (aikaisemmin kuva 2), jolla on kaksi akselia. Englannin kielessä näille akseleille käytetään nimityksiä arm ja plate. Visual Componentsin komponenttikirjastosta ei löydy tätä kappaleenkäsittelylaitetta, joten se on mallinnettava itse. Kappaleenkäsittelylaitteen geometria löytyy valmiina ABB:n sivuilta. Tätä ei vielä pysty käyttämään toimivana komponenttina, vaan mallille pitää määritellä akselit.

Mallintamisen ensimmäinen vaihe oli tuoda ABB:n sivulta ladattu 3D-malli Visual Componentsiin. Mallin sai tuotua modeling-välilehden Geometry-napista, jota kautta valitaan tuotava tiedosto. Valinnan jälkeen vasemmalle ilmestyy ikkuna, jossa on erilaisia asetuksia (kuva 17). Structure-kohdassa valittiin Feature, koska se mahdollisti akselien luontivaiheessa yksittäisten osien helpomman käsittelyn.



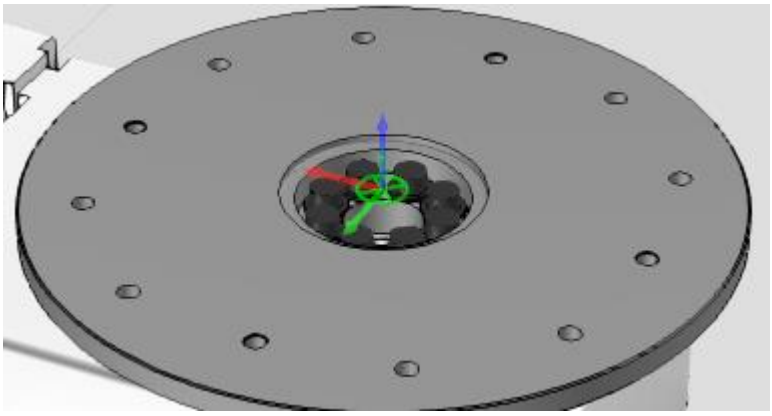
Kuva 17 Geometrian tuominen Visual Componentsiin

Kun malli oli tuotu, määriteltiin kappaleenkäsittelylaitteelle akselit. Akselit määriteltiin Create link -toiminnolla. Visual Componentsissa komponentilla on puurakenne. Komponentissa oleva linkki voi sisältää muita linkkejä. Kuvassa 18 on Link_1, jonka sisällä on Link_2. Link_1 vastaa ARM-akselia ja Link_2 PLATE-akselia.



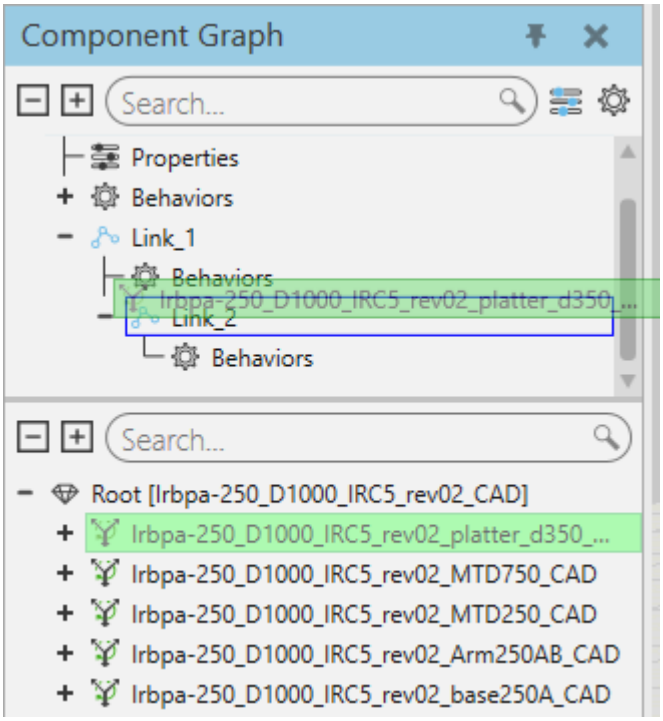
Kuva 18 Kappaleenkäsittelylaitteen puurakenne

Linkkien luomisen jälkeen linkit piti määritellä oikean tyyppisiksi. Link Properties -valikosta Joint-Type -kohdasta kummallekin akselille valittiin Rotational. Tämä tarkoittaa, että linkkiin liitetyt osat pyörivät akselinsa ympäri. Pyörähdysakseli määriteltiin Snap-työkalulla (kuva 19). PLATE-akselille se määriteltiin laipan keskelle ja ARM-akselille varren juureen.



Kuva 19 Linkin origon määrittäminen Snap-työkalulla

Seuraavaksi 3D-mallin osat piti yhdistää osaksi linkkejä. Ilman yhdistämistä osat eivät liikkuisi linkkejä ohjatessa. Kuva 20 näyttää, miten osat yhdistettiin linkkeihin.



Kuva 20 3D-mallin osien raahaaminen linkkeihin

Kun 3D-mallin osat oli raahattu linkkeihin, molempien linkkien asetusvalikosta valittiin ohjaimeksi servo-ohjain. Valinnan jälkeen linkille piti valita kulmarajat ja maksiminopeudet. ABB:n IRBP A:ssa akseleiden kulmarajat on määritelty robotin kontrollerin asetuksissa [9]. ARM-akselilla on myös fyysiset kulmarajat 181° suuntaansa [10, s. 49]. Kulmarajoiksi määriteltiin PLATE-akselille 720° ja ARM-akselille 180° kumpaankin suuntaan nollakohdasta. Maksimikulmanopeus ARM-akselille on 150°/s ja PLATE-akselille 180°/s [10, s. 49]. Linkille olisi voinut määritellä myös maksimikiihtyvyydet, mutta niitä ei löytynyt kappaleenkäsittelylaitteen dokumentaatiosta.

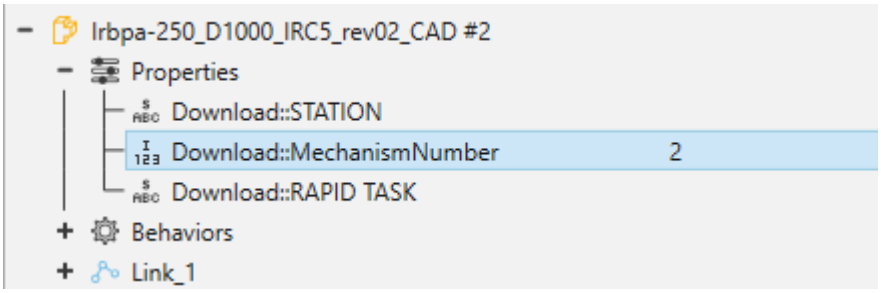
8.2 Robotin tuominen solumalliin

Kun kappaleenkäsittelylaite oli mallinnettu ja tallennettu, otettiin Visual Componentsissa uusi tyhjä maailma, johon tuotiin ABB:n IRB 2400 -robotti ja äskettäin mallinnettu kappaleenkäsittelylaite. Robotti löytyi valmiina Visual Componentsin komponenttikirjastosta, ja se asetettiin maailmankoordinaatiston origoon. Myöhemmässä vaiheessa, kun solun kehikko otettiin mukaan, robotti siirrettiin pois maailmankoordinaatiston origosta.

Kappaleenkäsittelylaite yhdistettiin robottiin Interfaces-toiminnon avulla. Nyt robotille voitiin luoda paikotuspisteitä niin, että paikoituspisteissä on ulkoiset akselit mukana. Kun robotille tehtiin yksinkertainen työkierto ja ohjelma käännettiin RAPID-kielelle, huomattiin, että ulkoisia akseleita ei ollut määritelty oikein. RAPID-kielessä ulkoiset akselit määritellään paikoituspisteessä kuudella lukuarvolla. Visual Componentsin postprosessori kirjoitti ulkoiset akselit, kuten kuvassa 21. 9E9 tarkoittaa, että kyseistä akselia ei ole liitetty robottiin tai sitä ei ohjata [4, s. 1658]. Robotin asetuksissa oli määritelty, että ARM-akselin kulma annetaan toisessa arvossa ja PLATE-akselin kolmannessa. Ensimmäinen arvon kohdalle pitäisi tulla 9E9. Ongelma ratkaistiin niin, että Modeling-välilehdellä klikattiin kappaleenkäsittelylaite aktiiviseksi ja Download::MechanismNumber-kohtaan asetettiin arvoksi 2 (kuva 22).

```
[ 11, 12.3, 9E9, 9E9, 9E9, 9E9]
```

Kuva 21 Ulkoisten akseleiden määritykset RAPID-koodissa



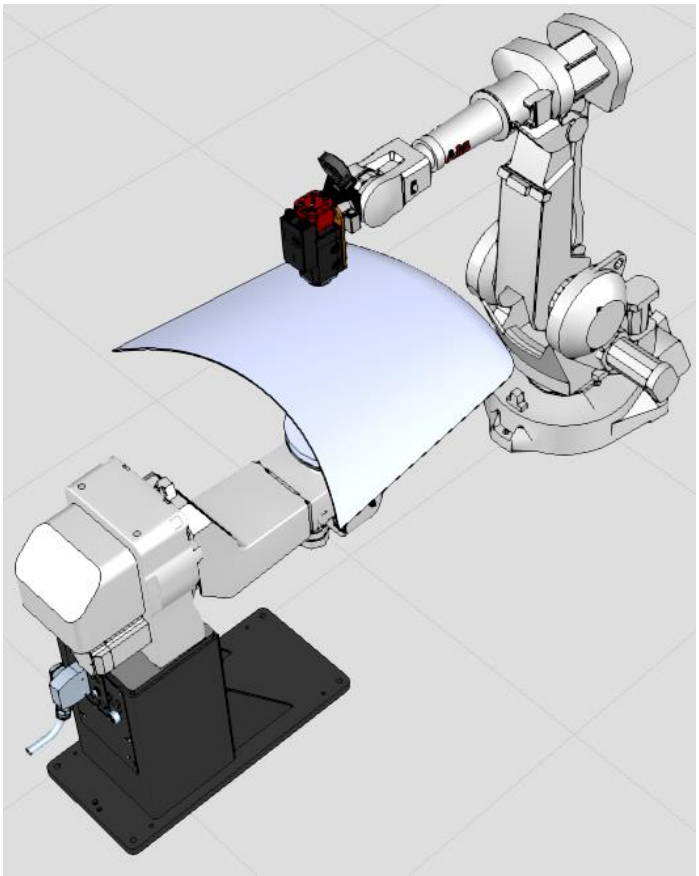
Kuva 22 Ulkoisten akseleiden määritys Visual Componentsissa

8.3 Voima-anturin ja karan tuominen maailmaan

Voima-anturin ja työkalun sai tuotua RobotStudion mallista. Voima-anturi ja kara olivat samassa komponentissa, joten niitä ei tarvinnut yhdistellä Visual Componentsin puolella. Kun 3D-malli oli tuotu Visual Componentsiin, piti määrittää MountFrame eli piste ja orientaatio työkalulaipan kiinnitykselle.

8.4 Postprosessorin ohjelmointi

Tässä kohtaa Visual Componentsin maailmaan oli lisätty robotti, kappaleenkäsittelylaite ja robotille työkalu (kuva 23). Kappaleenkäsittelylaitteeseen lisättiin vielä työstettävä kappale. Työn onnistuminen ei ollut vielä varmaa, ja ajatuksena oli, että jos työtä ei pysty toteuttamaan, niin se on parempi huomata työn alkuvaiheessa. Jos Visual Componentsin solumallissa tehtyä ohjelmaa ei pystyisi kääntämään robotin omalle ohjelmointikielelle eli tässä tapauksessa RAPID-kielelle, niin solumallin tekeminen olisi mennyt hukkaan. Siksi postprosessorin ohjelmointi ja testaus tehtiin jo tässä vaiheessa.



Kuva 23 Robotti, työkalu, kappaleenkäsittelylaite ja työstettävä kappale

Kun ABB:n robotissa on ABB:n oma voima-anturi, voimaohjattu ohjelma menee yksinkertaisimmallaan näin: [3, s. 83]

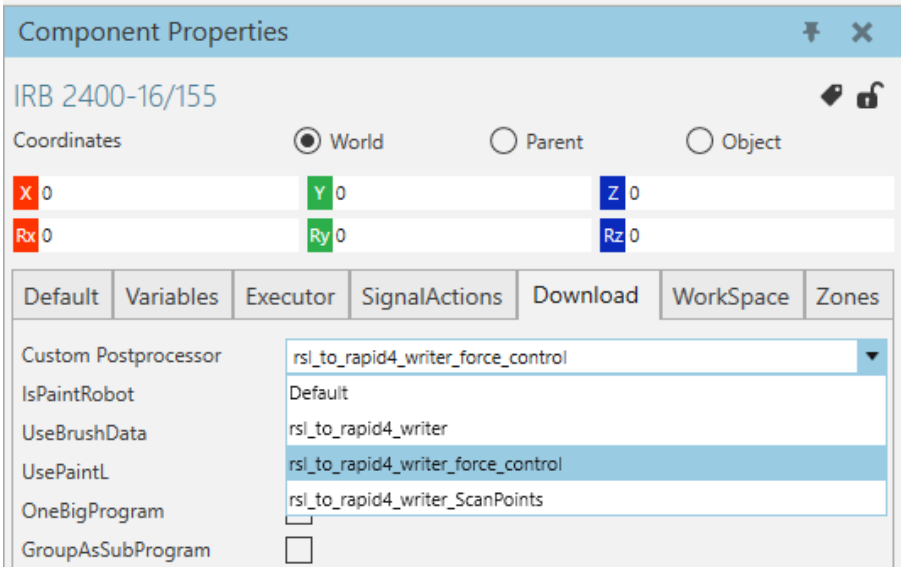
1. Voimaohjauksen aloitus (komennot FCCalib ja FCPress1LStart)
2. Yksi tai useampi liikekäsky (komennot FCPressL ja FCPressC)

3. Voimaohjauksen lopetus (komento FCPressEnd)

Visual Componentsin postprosessori ei tue näitä komentoja suoraan, mutta postprosessoria muokkaamalla ne saadaan mukaan RAPID-koodiin. Postprosessori on käytännössä Python-tiedosto, joka ajetaan siinä vaiheessa, kun käyttäjä haluaa kääntää Visual Componentsin esityksen ohjelmasta robotin omalle ohjelmointikielelle.

Seuraavaksi piti selvittää, miten postprosessoria tulisi muokata, jotta voimaohjaus saadaan toimimaan. Jokaiselle robottimerkille on oma postprosessorinsa. Kun käytössä on versio OLP 4.9, ABB:n postprosessori löytyy polusta *C:\Program Files\Visual Components\Visual Components Premium OLP 4.9\Python 2\Commands\Olp\OlpTranslators\ABB*. Tähän polkuun piti luoda uusi tiedosto, jonka nimi alkaa *rsl_to_rapid4_writer*, ja päättyy *.py*. Alkuperäinen postprosessori on *rsl_to_rapid4_writer.py*-nimisessä tiedostossa. Alkuperäinen postprosessori kopioitiin, ja tiedostolle annettiin nimeksi *rsl_to_rapid4_writer_force_control.py*. Alkuperäisessä postprosessorissa on lähes 2000 riviä koodia, ja suurin osa siitä säilyy ennallaan. Tämän takia voimaohjauksen postprosessori tehtiin kopioimalla alkuperäistä tiedostoa. [11, s. 29.]

Ennen kuin uuden postprosessorin saa käyttöön, robotin ohjelma pitää kääntää yhden kerran alkuperäisellä postprosessorilla. Kääntämisen aikana Visual Components tutkii, onko postprosessori-kansioon tullut uusia tiedostoja. Robottia klikkaamalla näytön vasempaan reunaan avautuu ikkuna, josta valitaan käytettävä postprosessori (kuva 24).



Kuva 24 Postprosessorin valitseminen

Kuvassa 24 näkyy valintalaatikot IsPaintRobot ja UsePaintL (jotka peittyvät Custom Postprocessor -valikon alle). Nämä ovat oletuksena päällä, ja ne vaikuttavat siihen, mitä postprosessorin funktioita Visual Components kutsuu, kun kääntämistä tehdään. Jos IsPaintRobot on valittuna, kutsutaan postprosessorin WritePaintL-funktiota. WritePaintL-funktion sisällä generoidaan joko PaintL- tai MoveL-käskyjä riippuen siitä, onko UsePaintL valittu aktiiviseksi. Ensimmäinen yritys luoda voimaohjauskäskyjä oli korvata PaintL-käskyjen kirjoittaminen voimaohjauskäskyillä. Vaikutti siltä, että WritePaintL-funktiossa generoidut käskyt eivät menneet suoraan lopulliseen tiedostoon, vaan niitä jotenkin jatkokäsiteltiin. Koska voimaohjauskäskyissä on eri määrä argumentteja kuin PaintL-käskyissä, postprosessori ei pystynyt tekemään käännöstä.

Seuraavaksi selvitettiin, miten RAPID-koodi rakentuu, kun IsPaintRobot ei ole valittuna. Huomattiin, että PaintL-käskyt korvautuvat hitsaukseen liittyvillä ArcL-käskyillä. Funktiot, joissa postprosessori kirjoittaa ArcL-käskyjä, ovat WriteWeldStart, WriteWeld ja WriteWeldEnd. Kuva 25 näyttää, miten postprosessori tarkistaa, minkä kontrollerityypin käyttäjä on valinnut, ja kirjoittaa RAPID-käskyn sen perusteella. Kun Visual Componentsissa klikkaa robottia, näytön vasempaan reunaan ilmestyy kuvan 24 mukainen ikkuna, jossa on ControllerType-niminen alasvetovalikko. Jos ControllerType-kohtaan valitaan CUSTOM, postprosessori ajaa kuvassa 25 näkyvän else-haaran. Else-haaran koodia voi muokata niin, että lopullisessa RAPID-koodissa on voimaohjauskäskyjä. Koodin yksinkertaistamiseksi else-haaran yläpuolella olevat kontrollerityypin vertailut poistettiin, ja postprosessori kirjoitettiin niin, että voimaohjauskäskyt kirjoitetaan välittämättä kontrollerin tyypistä.

rsl_to_rapid4_writer_force_control.py 9+ ×

rsl_to_rapid4_writer_force_control.py > WriteWeldStart

```
1078   def WriteWeldStart(modFile, cWeld, cPoint
1141 >     if ControllerType == "IRC5":···
1156
1157 >     elif ControllerType == "S4":···
1166
1167 >     elif "WolfPack" in ControllerType:···
1178
1179       else:
1180         #create your custom commands here
1181         sData = "MoveL " + PointData + ","
1182         sData = sData + ";"
```

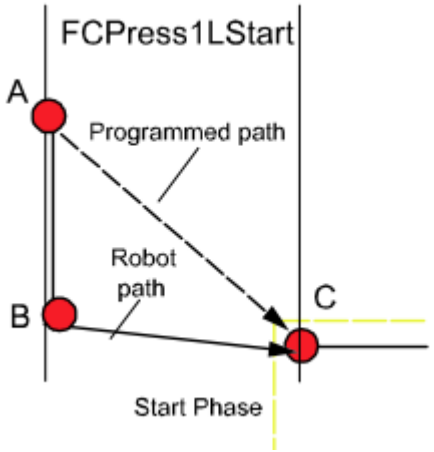
Kuva 25 Kontrollerin tyyppi vaikuttaa, miltä RAPID-koodi tulee näyttämään

Ennen voimaohjauksen aloittamista voima-anturi kalibroidaan FCCalib-käskyllä. FCCalib-käskylle annetaan loaddata-tyyppinen argumentti, jossa on mukana työkalun massa, massakeskipiste ja

hitausmomentti [4, s. 1607]. Näiden tietojen perusteella robotti pystyy huomaamaan, milloin työkaluun kohdistuu ulkoisia voimia.

Voimaohjaus aloitetaan FCPress1LStart-käskyllä. Ennen tätä käskyä robotti ei vielä kosketa työstettävää kappaletta. Käskyn aikana robotti liikuttaa työkalun kiinni työstettävään kappaleeseen ja alkaa painamaan sitä, kunnes haluttu voima saavutetaan.

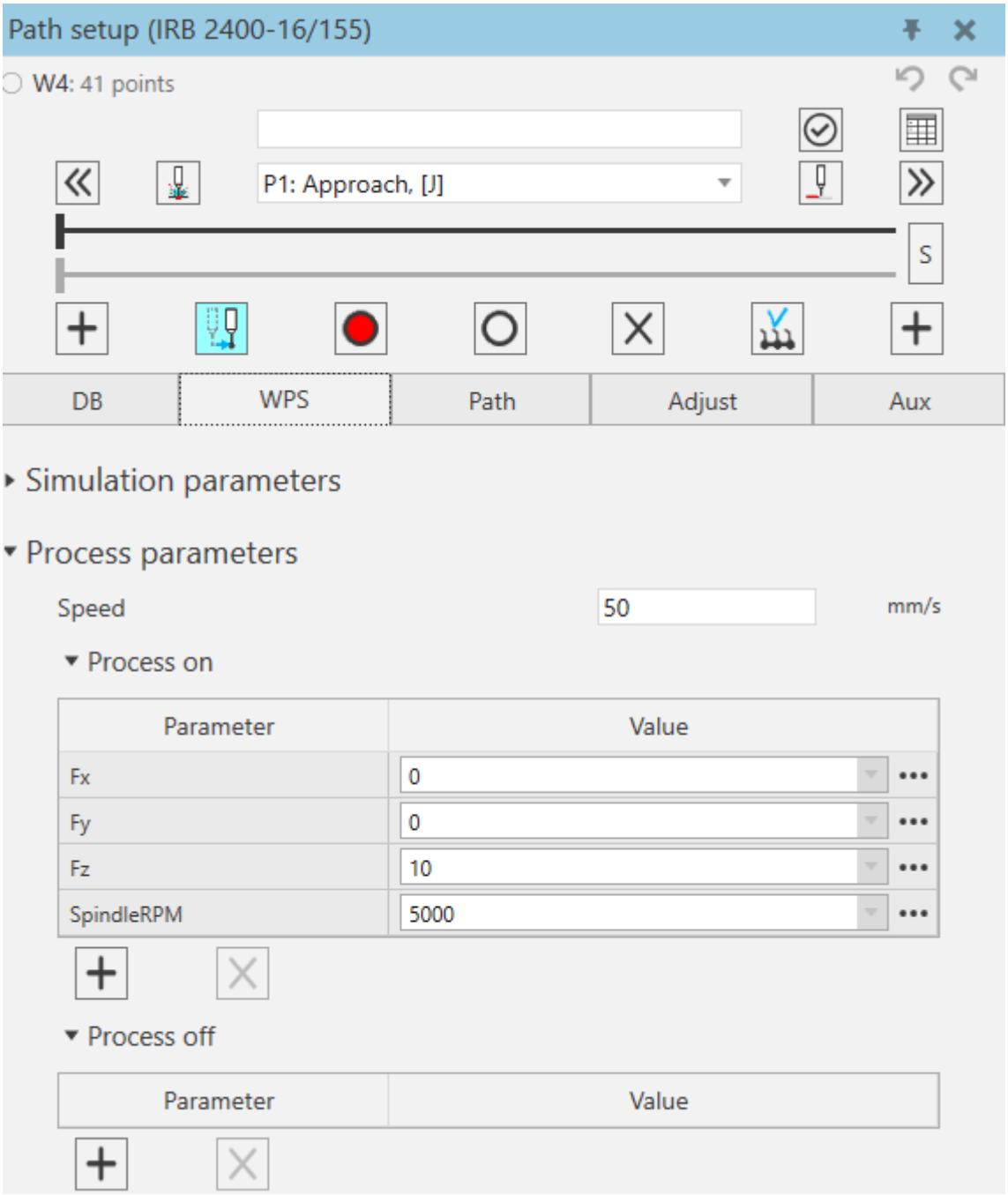
Kuva 26 havainnollistaa FCPress1LStart-käskyä. A on se piste, jossa robotti on ennen FCPress1LStart-käskyä. B on piste, jossa robotti koskettaa työstettävää kappaletta. Pisteestä B pisteeseen C robotin aiheuttama voima työstökappaleeseen kasvaa. FCPress1LStart muistuttaa perinteistä liikekäskyä, mutta se saa lisäksi argumenteikseen voimavektorin ja prosenttiluvun, joka voimavektorin suuruudesta pitää saavuttaa ennen kuin robotti alkaa liikkua pisteestä B kohti pistettä C. Pisteessä C on saavutettu voima, joka annettiin FCPress1LStart-käskylle argumenttina.



Kuva 26 Voimaohjauksen aloitus [3, s. 144]

On joitakin parametrejä, joita käyttäjän pitää pystyä määrittämään Visual Componentsin puolella. FCPress1LStart-käskyllä on vielä enemmän parametreja, mitä edellä on mainittu. Yksi on esimerkiksi se, minkä koordinaatiston suhteen voimavektori annetaan. Tässä sovelluksessa riittää, että käyttäjä määrittelee voimavektorin x-, y- ja z-koordinaatit TCP:n koordinaatistossa.

Käyttäjä pystyy määrittelemään jokaiselle liikeradalle omat parametrinsa (kuva 27). Tässä järjestelmässä oleellisimmat ovat voiman suunta, voiman suuruus ja karan pyörimisnopeus. Kuva 28 näyttää, miten parametrit luetaan postprosessorin puolella. Lainausmerkeissä oleva 5000 tarkoittaa oletusarvoa. Sitä käytetään, jos käyttäjä ei ole antanut parametrille arvoa.



Kuva 27 Prosessiparametrien määrittäminen

```
SpindleRPM = getArcAdvancedProperty(cStatement.ArcOnItems,"SpindleRPM","5000")
```

Kuva 28 Prosessiparametrin lukeminen postprosessorin puolella

Postprosessori saatiin kirjoittamaan FCPress1LStart-käsky muokkaamalla WriteWeldStart-funktiota. Postprosessorin koodi, joka kirjoitti RAPID-koodia kontrollerityypin perusteella, poistettiin. Poistetun koodin tilalle ohjelmoitiin FCCalib- ja FCPress1LStart-käskyn kirjoittaminen. Kuva 29

näyttää miten postprosessori kirjoittaa FCPress1LStart-käskyn. Lopullinen toteutus WriteWeldStart-funktiosta löytyy liitteestä 1.

```
sData = "FCPress1LStart "
sData += PointData
sData += ", v%.0f" % cPoint.Speed
sData += " \Fx:=Fx\Fy:=Fy\Fz:=Fz"
sData += ", 50"
sData += ", " + cPoint.Zone
sData += ", " + cPoint.Tool
sData += "\WObj:=" + cPoint.Base
sData += ";"
modFile.write("  %s%s\n" % (empties,sData))
```

Kuva 29 FCPress1LStart-käskyn kirjoittaminen

Seuraavaksi postprosessori ohjelmoitiin kirjoittamaan voimaohjauksen liikekäskyt FCPressL ja FCPressC. Voimaohjauksessa FCPressL vastaa MoveL-käskyä ja FCPressC vastaa MoveC-käskyä. Jos valinnaisia argumentteja ei oteta huomioon, voimaohjauskäskyissä ylimääräisenä tulee ainoastaan voiman suuruus. Postprosessori kirjoittaa lineaari- ja kaariliikkeen WriteWeld-funktiossa. Kuvassa 25 oli cPoint-parametri. WriteWeld-funktiossa on samanniminen parametri, jolla on Motype-kenttä, joka kertoo, pitääkö kirjoittaa lineaari- vai kaariliikekäsky. Kaariliikekäskyssä tarvitaan kaksi paikoituspistettä. WriteWeld-funktiossa on viaPoint-parametri, josta saadaan ensimmäinen paikoituspiste. cPoint-parametria käytetään lineaariliikekäskyssä ja kaariliikkeen toisena paikoituspisteenä. Itse käskyjen kirjoittaminen tapahtui samalla tavalla kuin kuvassa 29. Lopullinen toteutus on liitteessä 2.

Voimaohjaus päättyy käskyyn FCPressEnd. Robotille annetaan viimeinen paikoituspiste, jota kohti se lähtee liikkumaan. Kun robotti liikkuu kohti viimeistä paikoituspistettä, voiman suuruus pienenee. FCPressEnd-käskyssä on annettu voiman suuruus, jossa voimaohjaus lopetetaan. Postprosessori kirjoittaa FCPressEnd-käskyn WriteWeldEnd-funktiossa. Alkuperäisessä postprosessorissa oli tuki kaariliikkeelle, mutta voimaohjaus ei voi päättyä kaariliikkeeseen. Postprosessoriin kirjoitettiin tarkistus, jossa annetaan virheilmoitus, jos liiketyyppi on jotain muuta kuin lineaarinen. FCPressEnd-käskyn kirjoittaminen meni samalla tavalla kuin aikaisempien voimaohjauskäskyjen. Lopullinen toteutus on liitteessä 3.

Postprosessorin loppupäässä oli kaksi funktiota (kuva 30), joista WritePathStart ajetaan ennen liikeradan liikekäskyjä ja WritePathEnd ajetaan liikekäskyjen jälkeen. Näitä piti muokata niin, että

lopullisessa RAPID-koodissa ennen liikekäskyjä kara käynnistetään ja liikekäskyjen jälkeen pysäytetään. Robotin koodeissa oli valmiina aliohjelmat karan ohjaamiseen (kuvassa 30 SpindleStart2 ja SpindleStop).

Vaatimuksena oli myös kirjoittaa voiman suuruus omaan muuttujaan. Voiman suuruus tulee voimavektorin x-, y- ja z-komponenteista, ja voiman suuruus lasketaan Pythagoraan lauseella. Yhdessä liikeradassa voi olla kymmeniä liikekäskyjä, joten on käytännöllistä laittaa voiman suuruus omaan muuttujaan. Kun voiman komponentit ovat omissa muuttujissaan, käyttäjä voi muuttaa voiman suuruutta ja suuntaa yhdessä paikassa, ja koodi laskee voiman suuruuden automaattisesti. Näin voiman suuruutta ei tarvitse päivittää jokaiseen voimaohjattuun liikekäskyyn erikseen.

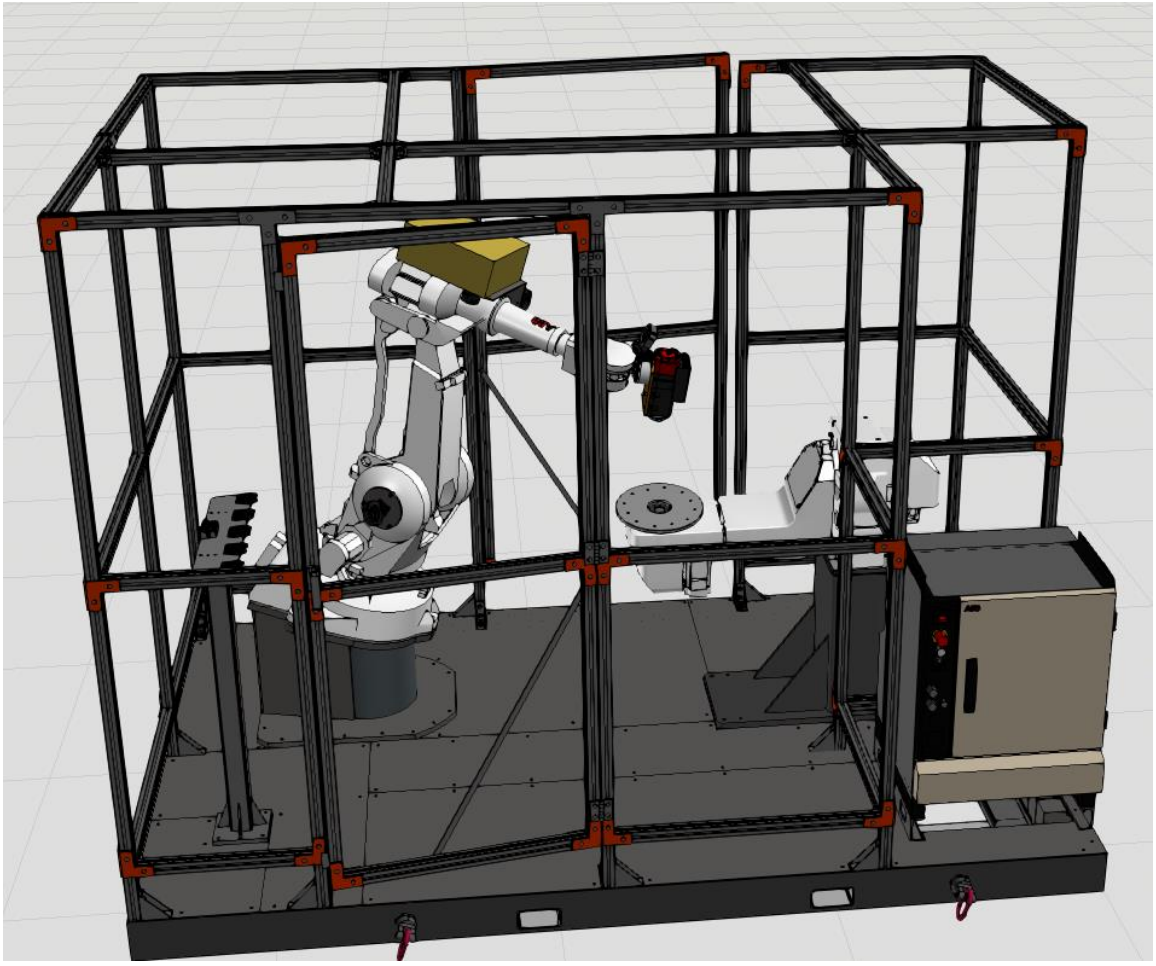
```
# ======================================
#   write: Path Start, access to write custom code at the start of a path
# ======================================
def WritePathStart(modFile, cStatement):
  empties = GenEmpties(cStatement.IndentLevel)

  Fx = getArcAdvancedProperty(cStatement.ArcOnItems,"Fx","0")
  Fy = getArcAdvancedProperty(cStatement.ArcOnItems,"Fy","0")
  Fz = getArcAdvancedProperty(cStatement.ArcOnItems,"Fz","10")

  modFile.write("  %sCONST num Fx := %s;\n" % (empties, Fx))
  modFile.write("  %sCONST num Fy := %s;\n" % (empties, Fy))
  modFile.write("  %sCONST num Fz := %s;\n" % (empties, Fz))
  modFile.write("  %sVAR num Force;\n" % (empties))
  modFile.write("  %sForce := sqrt(Fx * Fx + Fy * Fy + Fz * Fz);\n\n" % (empties))

  SpindleRPM = getArcAdvancedProperty(cStatement.ArcOnItems,"SpindleRPM","5000")
  modFile.write("  %sSpindleStart2 \\rpm:=%s;\n" % (empties, SpindleRPM))

# ======================================
#   write: Path End, access to write custom code at the end of a path
# ======================================
def WritePathEnd(modFile, cStatement):
  empties = GenEmpties(cStatement.IndentLevel)
  modFile.write("  %sSpindleStop;\n" % (empties))
```

Kuva 30 Liikeradan aloitus ja lopetus postprosessorissa

8.5 Solun mallintaminen

Kun oltiin varmoja, että postprosessori toimii, solumalli rakennettiin loppuun. Solusta oli olemassa valmis malli RobotStudiossa. Kappaleenkäsittelylaite ja robotti olivat tässä vaiheessa Visual Componentsin puolella valmiina. Kaikki loput solun geometriasta saatiin tuotua RobotStudiosta. Kuvassa 31 näkyy, miltä solumalli tuli näyttämään Visual Componentsissa. Oikeassa solussa karaan on kytketty kaapeleita, jotka ovat suojusten sisässä. Nämä suojukset olivat mukana RobotStudion mallissa, mutta niitä ei otettu mukaan Visual Componentsiin.

Ennen mallinnusta oikeasta robottisolusta käytiin ottamassa mittauksia. Mallintamisen jälkeen varmistettiin, että solumallin mitat vastaavat oikean solun mittoja.



Kuva 31 Solumalli Visual Componentsissa

8.6 Kalibrointi

Simulaatiomallin ja oikean robottisolun mitat eivät vastaa koskaan täydellisesti toisiaan. Jos simulaatiomalli poikkeaa liikaa oikeasta robottisolusta, robotti ei toimi odotetusti, ja ohjelmaan joudutaan tekemään muutoksia käyttöönoton aikana. Kalibroinnissa fyysisen robottisolun mitat viedään simulaatiomalliin.

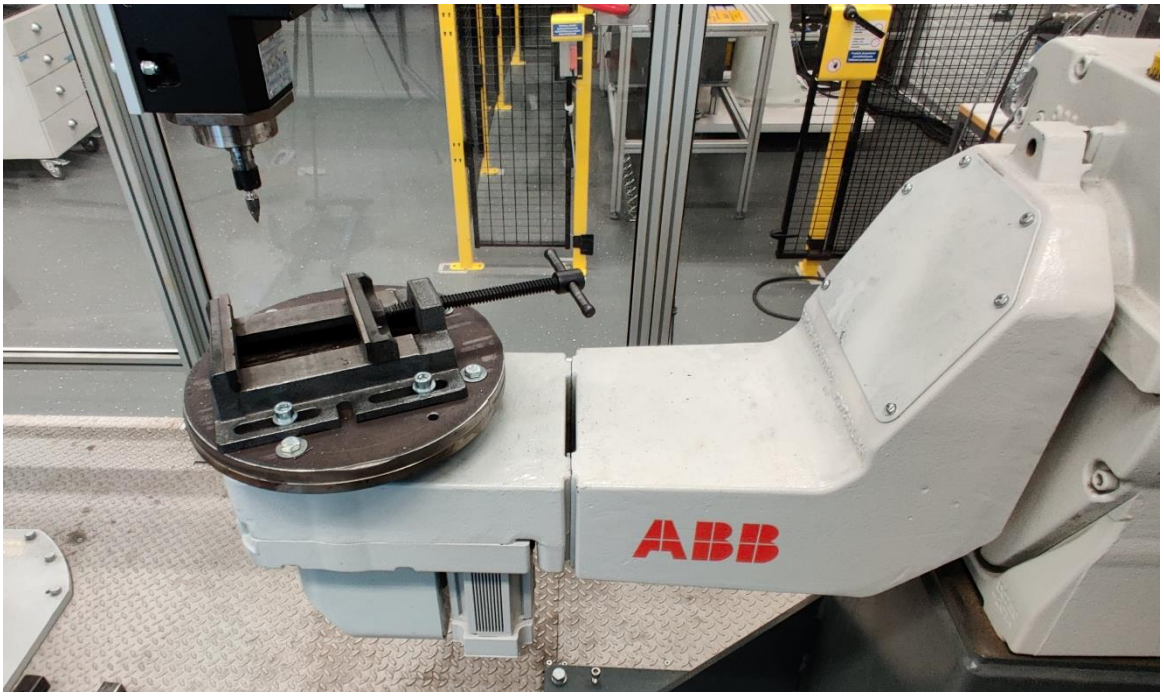
Robotin kalibrointitiedot on tallennettu robotin kontrolleriin. ABB:n roboteissa robotista voidaan ottaa varmuuskopio, josta löytyy nämä tiedot. Ensimmäinen vaihe kalibroinnissa oli varmuuskopion ottaminen.

Varmuuskopion mukana tuli tiedosto MOC.cfg, jossa on määritelty ulkoisten akseleiden sijainnit ja orientaatiot. Sijainnit on määritelty xyz-koordinaatteina ja orientaatiot kvaternioina. Robotti oli asennettu jalustan päälle, jonka yläosa oli noin 15°:n kulmassa lattian suhteen. Maailmankoordinaatiston on tarkoitus olla suorassa. MOC.cfg-tiedostossa tämä on otettu huomioon siten, että robotille on määritelty peruskoordinaatiston orientaatio. Kun robotin tuo Visual Componentsiin, ja sitä kääntää, maailmankoordinaatisto kääntyy mukana. Tämän takia maailmankoordinaatisto piti kääntää suoraan MOC.cfg-tiedostossa annetun peruskoordinaatiston perusteella. Maailmankoordinaatisto saatiin suoraan laskemalla peruskoordinaatiston orientaatiosta käänteiskvaternio. Visual Componentsissa Program-välilehdeltä löytyy Extras-nappi, jonka alta löytyy Convert-työkalu. Convert-työkalun avulla kvaternion saa muunnettua Visual Componentsin ymmärtämään muotoon.

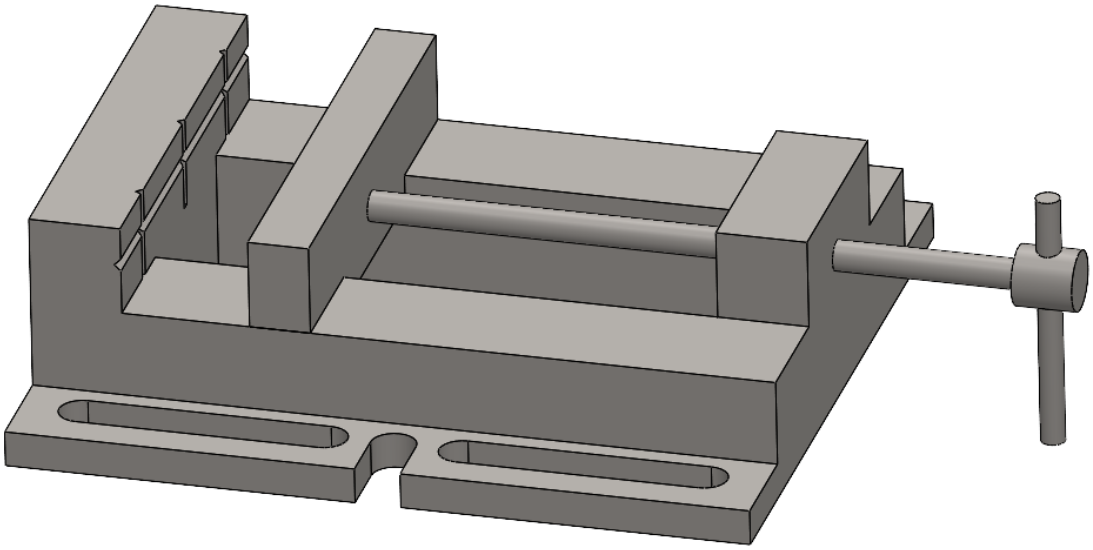
Kun maailmankoordinaatiston asento oli oikein, kalibroitiin kappaleenkäsittelylaite. Kalibroinnin jälkeen huomattiin, että laipan akseli heitti noin 11 mm alkuperäisestä. Kappaleenkäsittelylaitteen ARM- ja PLATE-geometriat pidettiin kiinni toisissaan, ja kalibrointivirheen voi nähdä pienenä rakona ARM-akselin ja kappaleenkäsittelylaitteen rungon välillä.

Kappaleenkäsittelylaitteeseen ei pysty suoraan kiinnittämään työstettävää kappaletta. Tässä robottisolussa asia oli ratkaistu, niin että kappaleenkäsittelylaitteeseen oli pultattu ruuvipenkki (kuva 32). Ruuvipenkistä ei löytynyt valmista CAD-mallia, joten se mallinnettiin SolidWorksissa (kuva 33). Ruuvipenkistä katsottiin, mitkä osat ovat merkittäviä paikoituksen kannalta. Ensimmäinen tärkeä mitta oli ruuvipenkissä olevan tason etäisyys laippatasoon. Tällä tavalla ruuvipenkkiin asetetut kappaleet saadaan vaakatasoon ja pystysuunnassa oikeaan kohtaan. Toinen tärkeä mitta saatiin ruuvipenkin päädystä. Päätyyn kohdistettuna kappaleen asento saadaan lukittua koko-

naan. Tässä kohtaa ruuvipenkkiin kohdistettu kappale pystyy vielä liikkumaan sivusuunnassa. Päädyn keskellä oli pystysuuntainen ura, jonka etäisyys reunasta mitattiin, ja otettiin mukaan CAD-malliin.

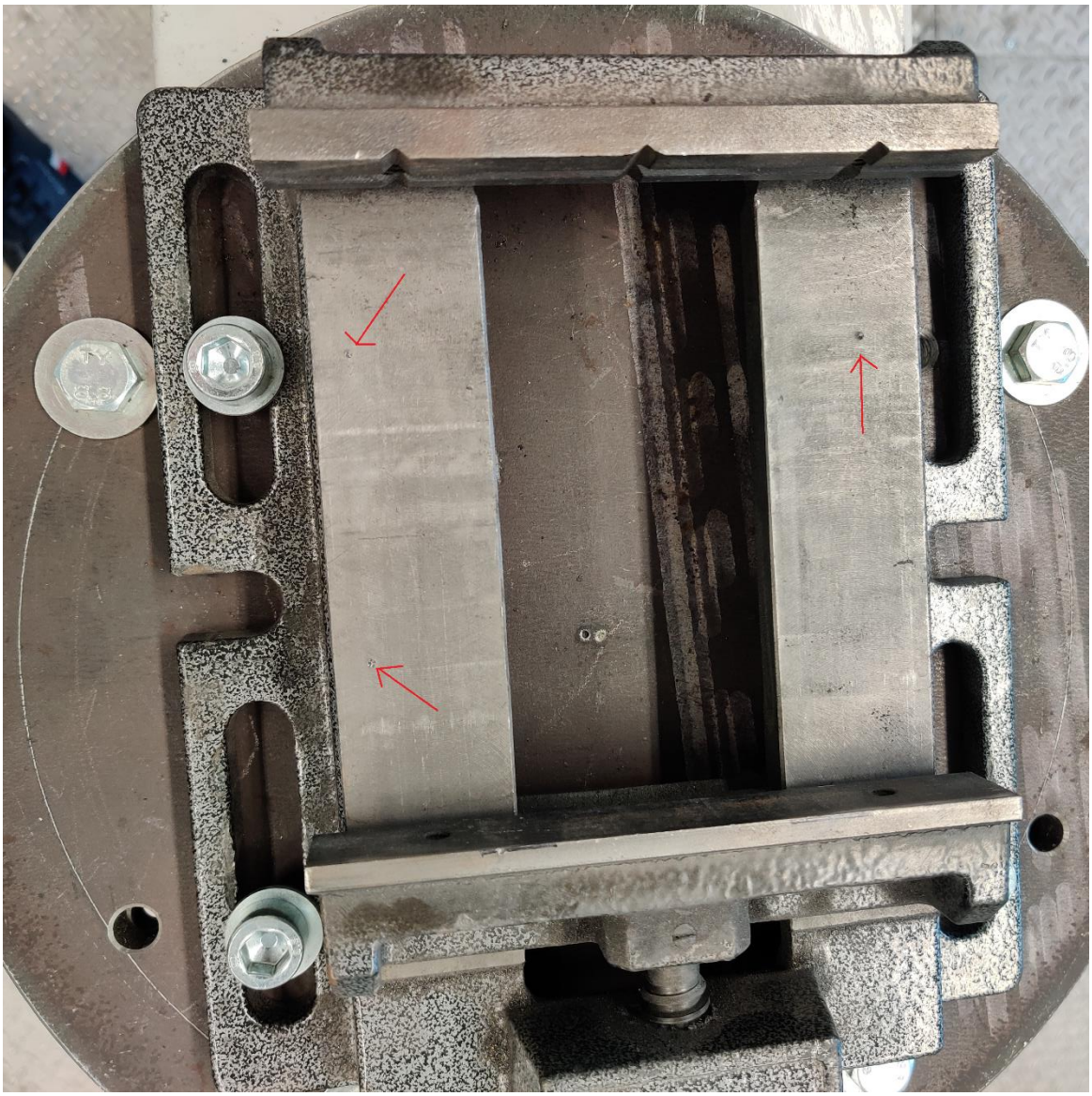


Kuva 32 Ruuvipenkki kappaleenkäsittelylaitteessa



Kuva 33 SolidWorksissa mallinnettu ruuvipenkki

Ruuvipenkkiin piti jotenkin saada työkohdekoordinaatisto. Robotille voidaan tehdä uusi työkohdekoordinaatisto kolmen pisteen avulla. Ruuvipenkkiin lyötiin pistepiikillä kolme pistettä (kuva 34). Kaksi ensimmäistä olivat samalla etäisyydellä päädystä, niin että pisteet olivat eri puolilla ruuvipenkkiä. Kaksi ensimmäistä pistettä määrittivät y-akselin suunnan. Kolmas piste lyötiin toiselle tasolle, niin että se on ensimmäisen pisteen kanssa kohtisuorassa päätyä vasten. Kuvassa 34 vasemmalla olevat pisteet määrittävät x-akselin suunnan. X-akselin arvot kasvavat kuvassa alaspäin ja y-arvot oikealle päin. Z-akseli osoittaa kameraa.



Kuva 34 Työkohdekoordinaatistoon otetut pisteet

Pistepiikillä lyödyistä pisteistä tehty työkohdekoordinaatisto vietiin Visual Componentsiin. Työkohdekoordinaatiston pisteet riittivät orientaation määrittämiseen ja z-koordinaatin määrittämiseen. X- ja y-koordinaatit otettiin talteen siten, että robotti liikutettiin päädyn keskellä olevaan

uraan, ja siitä pisteestä otettiin koordinaatit ylös. Myöhemmin Visual Componentsissa ruuvipenkin yläosassa uran kohdalla oleva piste voitiin liikuttaa robotista otettuihin koordinaatteihin. Kun työkohdekoordinaatisto ja ruuvipenkki oli kalibroitu Visual Componentsiin, voitiin alkaa tekemään testiohjelmia. Kun ruuvipenkin mitat olivat oikein, testiajoissa käytettävät kappaleet oli helppo kohdistaa ruuvipenkkiin Visual Componentsin Align-työkalulla, ja testiajoissa ei tarvinnut määritellä enää uusia työkohdekoordinaatistoja.

Viimeinen vaihe kalibroinnissa oli TCP:iden kalibrointi. Voimaohjaus ei vaadi niin suurta tarkkuutta kuin monet muut sovellukset, ja testiajoissa riitti, että työkalujen TCP:t määritettiin kokonaan Visual Componentsissa. Tämä säästi aikaa, koska TCP:itä ei tarvinnut määrittää manuaalisesti robotilla. Visual Componentsissa pystyi valitsemaan, että TCP:t siirretään postprosessoinnin yhteydessä tulostettuun koodiin mukaan. Näin TCP:n tietoihin ei tarvinnut koskea ollenkaan postprosessoinnin jälkeen.

8.7 Testaaminen oikeassa ympäristössä

Postprosessorin testaamista varten tehtiin kolme erilaista testiohjelmaa. Testiajot tehtiin, jotta voitiin osoittaa postprosessorin toimivuus. Kahdessa ensimmäisessä testissä työstettävät kappaleet mallinnettiin SolidWorksissa.

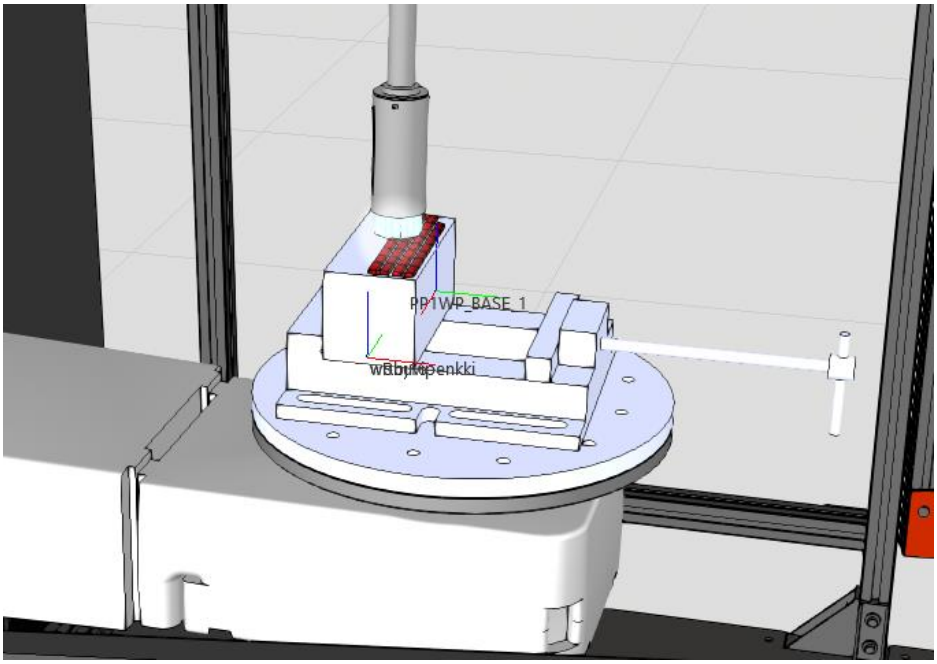
8.7.1 Tasainen pinta

Ensimmäinen testi oli yksinkertaisin. Siinä hiottiin venttiililohkoa (kuva 35) Xebecin harjalla (kuva 36). Tässä testissä kappaleenkäsittelylaitteen akselit eivät olleet käytössä, vaan ainoastaan robotti liikkui.

Kuva 35 Venttiililohko testiajon jälkeen

Kuva 36 Xebecin harja karaan kiinnitettynä

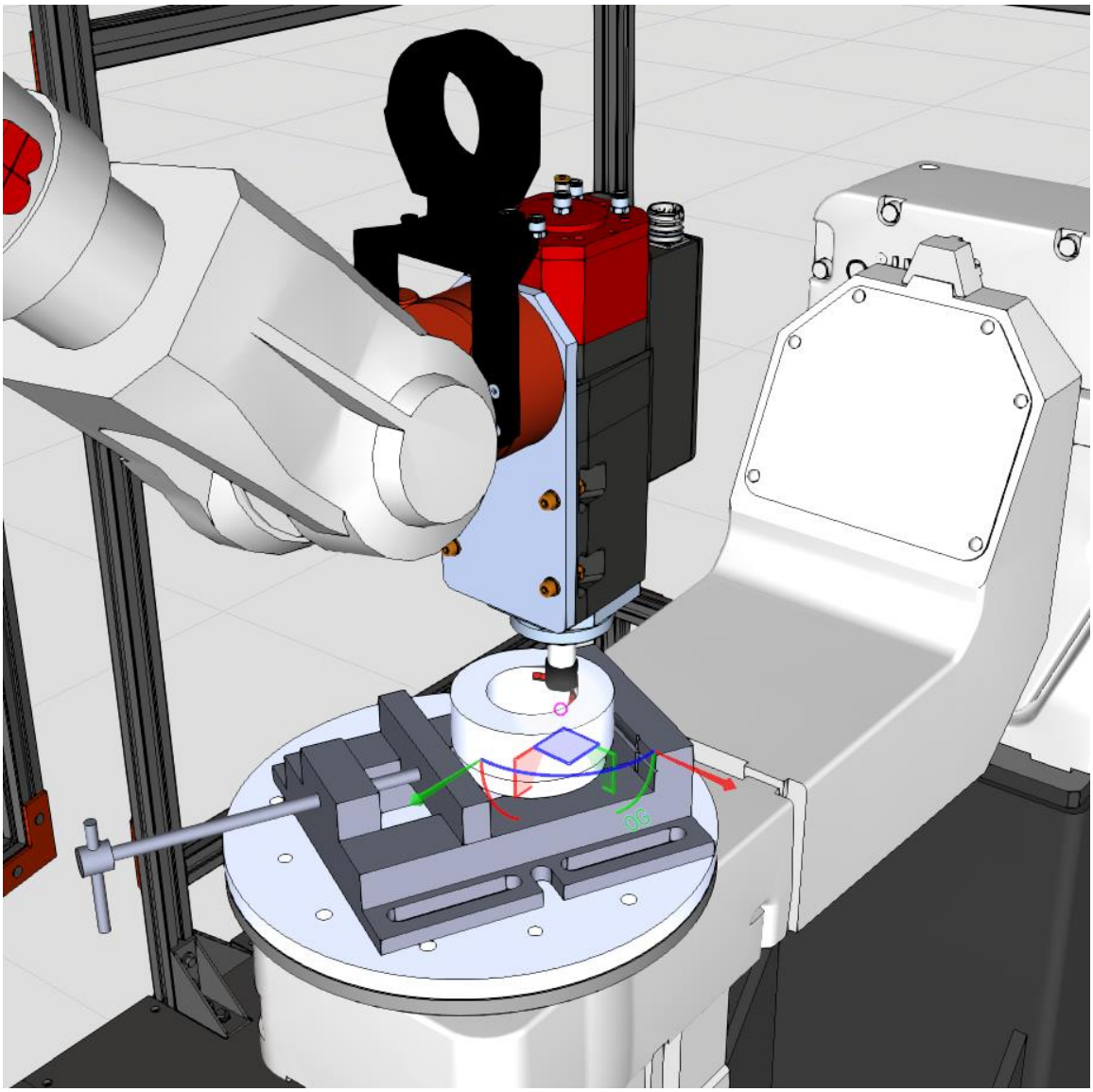
Visual Componentsissa venttiililohko kohdistettiin Align-työkalulla ruuvipenkin reunaan. Kuva 37 näyttää, miten kappaleen pääty ja ruuvipenkin reuna ovat samassa tasossa. Harjan TCP määritettiin Visual Componentsissa, ja postprosessorin asetuksissa valittiin, että TCP:t tulevat mukaan postprosessorin kirjoittamaan koodiin. Koska kalibrointi oli tehty oikein, ohjelman suoritus oikealla robotilla onnistui jo ensimmäisellä yrityksellä.



Kuva 37 Venttiililohko Visual Componentsissa

8.7.2 Kaariliike ulkoisella akselilla

Toisessa testissä kovametalliviilalla hiottiin holkin sisäkehää. Kuvassa 38 näkyy, että TCP on määritetty niin, että x-akseli osoittaa robotilta päin katsottuna oikealle. Z-akseli osoittaa viilan kärjen suuntaisesti. Voimaohjausparametrit asetettiin niin, että voima on x-akselin suuntaan.



Kuva 38 Holkin sisäreunan hionta

Tämän testin ajaminen tuotti eniten haasteita. Testissä käytetyssä kovametalliviilassa oli isot hampaat, ja se tarkoitti sitä, että viilan piti liikkua nopeasti, jotta se ei purisi liian syvälle.

Kun testiohjelma oli saatu Visual Componentsin puolella valmiiksi, se käännettiin postprosessorilla robotille sopivaan muotoon. Aluksi voiman suuruudeksi asetettiin 10 newtonia. Testiä yritettiin ensin ajaa ilman, että kara oli käynnissä.

Heti ensimmäisessä kaariliikekäskyssä suoritus pysähtyi, ja käsiohjaimen näytölle tuli virhe, joka ilmoitti liian nopeasta orientaation muutoksesta. Tämän jälkeen liikekäskyjä hidastettiin 25 prosenttiin käsiohjaimella. Nyt ohjelman ajaminen onnistui, mutta vauhti olisi ollut liian hidas karan pyörittämiselle.

Virheen, joka ilmoitti liian nopeasta orientaation muutoksesta, sai poistettua RAPID-käskyllä FCSupvReoriSpeed. Tämän jälkeen vauhtia kasvatettiin, mutta nyt viila ei enää pysynyt kappaleessa kiinni. Kun ohjelmaa ajettiin askeltamalla, viila ei enää kaarikäskyn lopussa koskettanut kappaletta. Kun käsky oli ajettu, robotti liikutti viilan viiveellä takaisin kappaleeseen. Ongelma oli siinä, että voimaohjaus ei reagoinut riittävän nopeasti voiman muutoksiin.

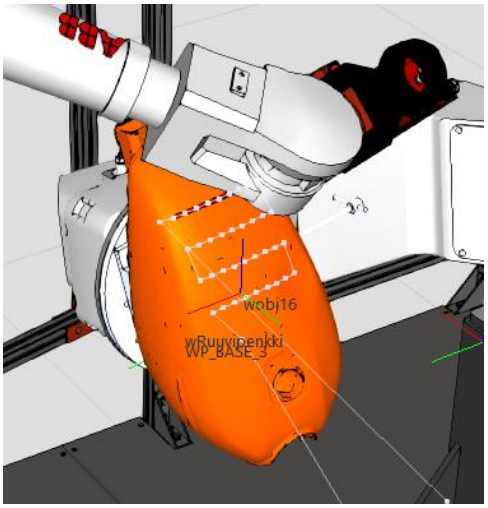
ABB:n voimaohjauksen dokumentaatiosta FCPress1LStart-käskylle löytyi valinnainen asetus \DampingTune, jolla pystyi vaikuttamaan siihen, kuinka herkästi robotti reagoi voiman muutokseen. Arvo 50 oli pienin mahdollinen, ja silloin voimaohjaus oli herkimmillään. Tämä ei ratkaissut ongelmaa kokonaan. Viila pysyi kuitenkin riittävän hyvin kappaleessa, joten tämän jälkeen rata ajettiin niin, että kara oli käynnissä. Kuva 39 näyttää, että radan lopussa (oikealla) viila ei ole enää pysynyt kappaleessa.

Kuva 39 Holkki testiajon jälkeen

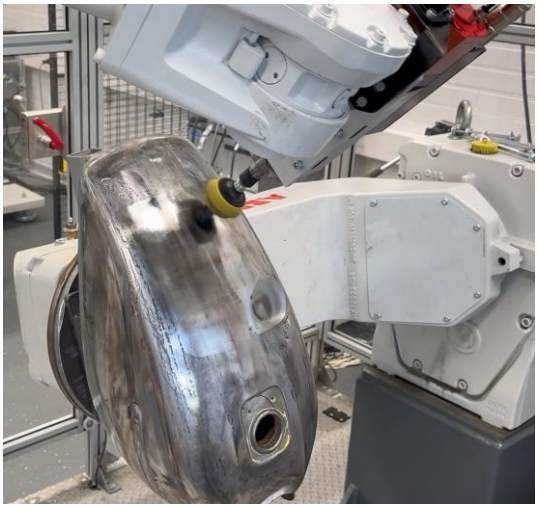
Tämän testin täydelliseen onnistumiseen olisi vaadittu hitaampi vauhti ja hienompihampaisempi työkalu. Testiä voi pitää kuitenkin postprosessorin kannalta onnistuneena, koska RAPID-koodiin tuli mukaan voimaohjauksen kaarikäskyt.

8.7.3 3D-skannattu kappale

Tässä testissä moottoripyörän tankkia kiillotettiin. Ensin tankki skannattiin 3D-skannerilla. Skannattu tiedosto oli yli 200 megatavua, eli sitä ei olisi sellaisenaan pystynyt viemään Visual Componentsiin. Mallia yksinkertaistettiin Blender-ohjelmistolla, ja tiedostokooksi saatiin lopulta noin 4 Mt. Skannaustiedosto vietiin Blenderiin .obj-muodossa, ja Blenderistä yksinkertaistettu malli vietiin Visual Componentsiin .stl-muodossa. Muita tiedostomuotoja käytettäessä Visual Components piirsi mallin polygonimuodossa, ja se vaikeutti liikaa työstöratojen tekemistä. Kuva 40, näyttää miten .stl-muodossa tuotu malli piirtää kappaleen pinnan yhtenäisenä eikä polygoneina. Ja koska pinta on yhtenäinen, Visual Components pystyy rakentamaan työstöradan, jossa z-akseli on aina normaali pinnan kanssa. Muilla tiedostomuodoilla tämä ei onnistunut, koska pinta ei pysynyt yhtenäisenä. Kuva 41 on otettu testiajon aikana.



Kuva 40 3D-skannattu tankki Visual Componentsissa



Kuva 41 3D-skannatun tankin kiillotus

Tämän testin tavoitteena oli käyttää kumpaakin käsittelylaitteen akselia. Testi ajettiin useamman kerran, mutta välillä robotti ei pystynyt sitä suorittamaan. Liikekäskyissä zonedata oli asetettu z10:ksi, ja kun tätä pienennettiin z1:ksi, niin robotti pystyi ajamaan radan ongelmitta. Tankkiin oli hitsattu putkipalkki, jonka avulla tankki saatiin ruuvipenkkiin (kuva 42).

Kuva 42 Tankin kiinnitys ruuvipenkkiin

9 Yhteenveto

Tämän opinnäytetyön tavoitteena oli kirjoittaa Visual Components-ohjelmistolle postprosessori. Postprosessori kirjoitettiin, koska Visual Componentsin oma postprosessori ei pystynyt tulostamaan RAPID-koodia, jossa olisi ollut mukana voimaohjauskäskyjä.

Postprosessorin testaamista varten mallinnettiin olemassa oleva robottisolu Visual Componentsiin. Robottisolussa oli ABB:n IRB 2400/16 -robotti ja IRBP A -kappaleenkäsittelylaite. Robotti löytyi valmiina Visual Componentsin eCatalogista. Kappaleenkäsittelylaite jouduttiin rakentamaan itse valmiista CAD-malleista, jotka löytyivät ABB:n sivuilta.

Postprosessori kirjoitettiin ennen kuin koko solumalli oli valmis. Postprosessorin yksinkertaiseen testaamiseen riitti, että Visual Componentsissa oli robotti ja kappaleenkäsittelylaite. Kun oltiin varmoja postprosessorin toimivuudesta, solumalli rakennettiin loppuun.

Kappaleenkäsittelylaitteessa oli ruuvipenkki, johon työstettävät kappaleet kiinnitettiin. Ruuvipenkki jouduttiin mallintamaan itse, koska valmista mallia ei ollut. Kun ruuvipenkki oli tuotu Visual Componentsiin, tehtiin kalibrointi. Kalibroinnissa oikean maailman mitat siirrettiin Visual Componentsiin. Kun kalibrointi on tehty oikein, postprosessorin tulostama koodi toimii yleensä suoraan, eikä siihen tarvitse tehdä mitään muutoksia.

Kalibroinnissa robotin ohjaimesta otettiin varmuuskopio, jonka mukana tuli tieto robotin orientaatiosta ja kappaleenkäsittelylaitteen akseleiden sijainneista ja orientaatioista. Ruuvipenkin kalibroimista varten ruuvipenkkiin lyötiin pistepiikillä kolme merkkiä, joiden mukaan työkohdekoordinaatisto määriteltiin. Lisäksi robotilla otettiin ylös referenssipiste ruuvipenkistä, joka piti löytää helposti 3D-mallista. Näin työkohdekoordinaatisto saatiin 3D-mallissa samaan paikkaan oikean ruuvipenkin kanssa.

Kalibroinnin jälkeen suoritettiin kolme testiajoa. Ensimmäisessä testiajossa venttiililohkoa hiottiin Xebecin harjalla. Toisessa testiajossa testattiin voimaohjauksen kaariliikekäskyjä. Viimeisessä testissä moottoripyörän tankki skannattiin 3D-skannerilla. Tiedosto oli liian iso, joten sitä ei voitu sellaisenaan tuoda Visual Componentsiin. Mallia yksinkertaistettiin Blender-ohjelmistolla. Tankille tehtiin yksinkertainen työstörata, jossa molemmat ulkoiset akselit olivat käytössä. Testiajojen perusteella postprosessori todettiin toimivaksi. Toimeksiantaja pystyy nyt tekemään robotille voimaohjattuja ohjelmia Visual Componentsissa.

Lähteet

1. Välimäki K, Niemelä M. Teollisuuden robotiikka. Helsinki: Suomen Robotiikkayhdistys ry; 2023
2. ABB. IRB 2400 Industrial Robot. [Internet]. [viitattu 26.5.2024]. Saatavilla: https://search.abb.com/library/Download.aspx?DocumentID=PR10034EN_R7
3. ABB. Application Manual Force Control. [Internet]. [viitattu 16.5.2024]. Saatavilla: https://library.e.abb.com/public/bde123851c3c419580ee0512a055df98/3HAC050377%20AM%20Force%20Control%20RW%206-en.pdf
4. ABB. Technical reference manual RAPID Instructions, Functions and Data. Saatavilla: https://library.e.abb.com/public/b227fcd260204c4dbeb8a58f8002fe64/Rapid_instructions.pdf.
5. ABB. Coordinate Systems. [Internet]. [viitattu 31.5.2024]. Saatavilla: https://developercenter.robotstudio.com/api/robotstudio/articles/Concepts/Math/Coordinate-Systems.html
6. Visual Components. Robotics OLP. [Internet]. [viitattu 4.6.2024]. Saatavilla: https://www.visualcomponents.com/olp-products/robotics-olp/
7. Peltomäki J. Python 3 -ohjelmoinnin perusteet. Books on Demand; 2023
8. [viitattu 17.9.2024]. Saatavilla: https://www.matikki.fi/guidebook-python/python/guide-basics/
9. ABB. external rotational axis limits. [Internet]. [viitattu 13.5.2024]. Saatavilla: https://forums.robotstudio.com/discussion/11624/external-rotational-axis-limits
10. ABB. Product specification IRBP/D2009. [Internet]. [viitattu 13.5.2024]. Saatavilla: https://search.abb.com/library/Download.aspx?DocumentID=3HAC038208-001
11. Visual Components. ABB Integration and Postprocessing. [Internet]. [viitattu 16.5.2024]. Saatavilla: https://extranet.visualcomponents.com/content/uploads/2021/07/ABB-Robotics-OLP-Integration-and-Postprocessing-1.pdf

```
def WriteWeldStart(modFile, cWeld, cPoint, ControllerType, useRobtargets):

  #if trackdata exists, it is assumed that weavedata also exists
  #if not, error might occur when downloading

  #EXAMPLES HOW TO GET UI DATA FROM WELD WPS
  #if len(cWeld.ArconData) >= 5:
  #  DataOnLine4 = cWeld.ArconData[4]
  #AdditionalParameterStartValue = getArcAdvancedProperty(cWeld.ArcOnItems,"AdditionalParameter",None)
  #AdditionalParameterEndValue = getArcAdvancedProperty(cWeld.ArcOffItems,"AdditionalParameterEnd",None)
  sData = ""
  empties = GenEmpties(cPoint.getIndentLevel())

  if cPoint.Motype == "LINEAR":

    if useRobtargets:
      PointData = cPoint.Name
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)
    else:
      PointData = "[" + cPoint.getPositionData() + "," + cPoint.getConfigData() + "," + cPoint.getExternalA-
xisData() + "]"
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)

    # FCCalib
    sData = "FCCalib "
    sData += cPoint.Tool
    sData += ";"
    modFile.write("  %sFCCalib %s_LD;\n" % (empties,cPoint.Tool))

    # FCPress1LStart
    sData = "FCPress1LStart "
    sData += PointData
    sData += ", v%.0f" % cPoint.Speed
    sData += " \Fx:=Fx\Fy:=Fy\Fz:=Fz"
    sData += ", 50"
    sData += ", " + cPoint.Zone
    sData += ", " + cPoint.Tool
    sData += "\WObj:=" + cPoint.Base
    sData += ";"
    modFile.write("  %s%s\n" % (empties,sData))

  elif cPoint.Motype == "CIRCULAR":
    modFile.write("\n    !Weld Start with CIRCULAR movement not supported\n")

  else:
    modFile.write("\n    !Weld Start with JOINT movement not supported yet --> write code into Writer \n")
```

```
def WriteWeld(modFile, cWeld, cPoint, viaPoint, ControllerType, useRobtargets):
  sData = ""
  empties = GenEmpties(cPoint.getIndentLevel())

  if cPoint.Motype == "LINEAR":

    if useRobtargets:
      PointData = cPoint.Name
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)
    else:
      PointData = "[" + cPoint.getPositionData() + "," + cPoint.getConfigData() + "," + cPoint.getExternalA-
xisData() + "]"
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)

    sData = "FCPressL "
    sData += PointData
    sData += ", v%.0f" % cPoint.Speed
    sData += ", Force"
    sData += ", " + cPoint.Zone
    sData += ", " + cPoint.Tool
    sData += "\WObj:=" + cPoint.Base
    sData += ";"

    #modFile.write("    %s\n" % sData)
    modFile.write("  %s%s\n" % (empties,sData))

  elif cPoint.Motype == "CIRCULAR":

    if useRobtargets:
      PointDataVia = viaPoint.Name
      PointData = cPoint.Name
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)
    else:
      PointDataVia = "[" + viaPoint.getPositionData() + "," + viaPoint.getConfigData() + "," + viaPoint.getExter-
nalAxisData() + "]"
      PointData = "[" + cPoint.getPositionData() + "," + cPoint.getConfigData() + "," + cPoint.getExternalA-
xisData() + "]"
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)

    sData += "FCPressC "
    sData += "%s, %s" % (PointDataVia, PointData)
    sData += ", v%.0f" % cPoint.Speed
    sData += ", Force"
    sData += ", " + cPoint.Zone
    sData += ", " + cPoint.Tool
    sData += "\WObj:=" + cPoint.Base
    sData += ";"
    modFile.write("  %s%s\n" % (empties,sData))

  else:
    modFile.write("\n    !Weld Switch with JOINT movement not supported yet --> write code into Writer \n")
```

```
def WriteWeldEnd(modFile, cWeld, cPoint, viaPoint, ControllerType, useRobtargets):
  sData = ""
  empties = GenEmpties(cPoint.getIndentLevel())

  if cPoint.Motype == "LINEAR":
    if useRobtargets:
      PointData = cPoint.Name
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)
    else:
      PointData = "[" + cPoint.getPositionData() + "," + cPoint.getConfigData() + "," + cPoint.getExternalA-
xisData() + "]"
      if cPoint.syncPointID != -1:
        PointData = PointData + "\ID:=" + str(cPoint.syncPointID)

    sData = "FCPressEnd "
    sData += PointData
    sData += ", v%.0f" % cPoint.Speed
    sData += ", " + cPoint.Tool
    sData += "\WObj:=" + cPoint.Base
    sData += ";"
    modFile.write("  %s%s\n" % (empties,sData))
  else:
    modFile.write("\n    !Weld End with JOINT or CIRCULAR movement not supported \n")
```

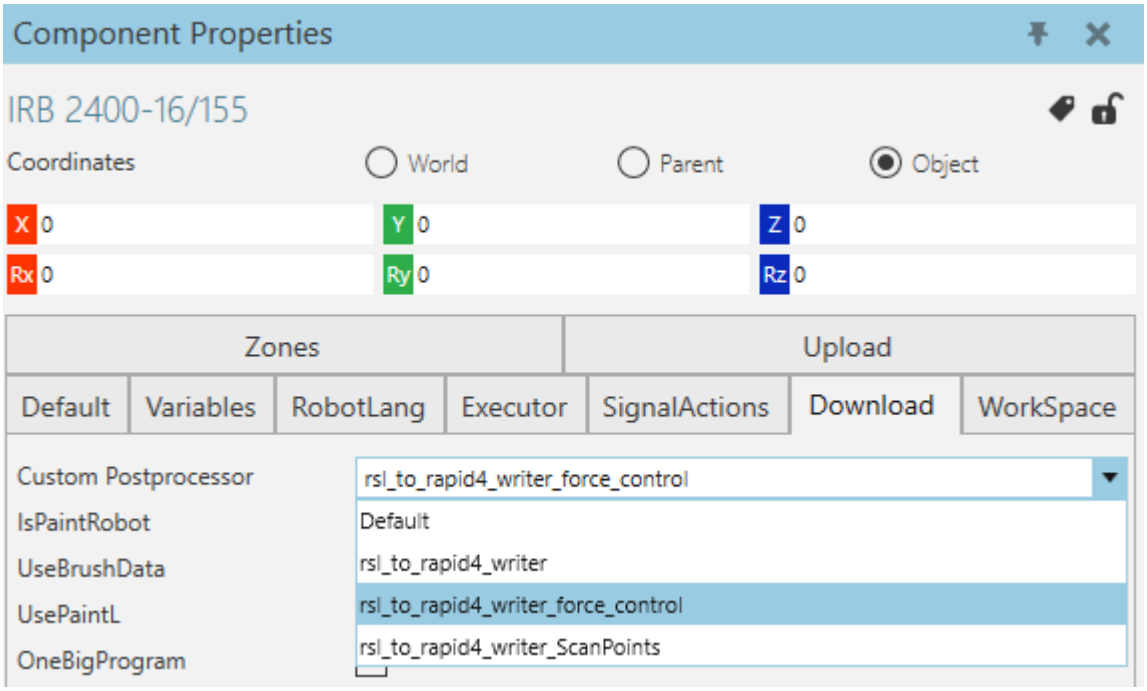
Ohjeet postprosessorin käyttöön

Nämä ohjeet toimivat Visual Components OLP 4.9 -versiossa. Siirry resurssienhallinnassa polkuun *C:\Program Files\Visual Components\Visual Components Premium OLP 4.9\Python 2\Commands\Olp\OlpTranslators\ABB* ja kopioi tiedosto 1 sinne.

rsl_to_rapid4_writer_force_control.py

Tiedosto 1 Postprosessori

Klikkaa Visual Componentsissa Robottia. Siirry Download välilehdelle kuten kuvassa 1. Jos Custom Postprocessor -valikossa ei näy *rsl_to_rapid4_writer_force_control*, valitse Default, ja sen jälkeen klikkaa ikkunan ylhäältä valintanauhasta Postprocess. Valikko avautuu, josta ABB:n alta tulee klikata Download. Visual Components pyytää valitsemaan tiedoston, johon postprosessorin tulostama koodi kirjoitetaan. Kun Visual Components on ajanut postprosessorin, siirry uudelleen kuvan 1 osoittamaan paikkaan. Nyt valikosta pitäisi pystyä valitsemaan *rsl_to_rapid4_writer_force_control*. Tämän jälkeen valintanauhan kautta Postprocess → ABB → Download ja Visual Components käyttää tässä työssä kehitettyä postprosessoria.



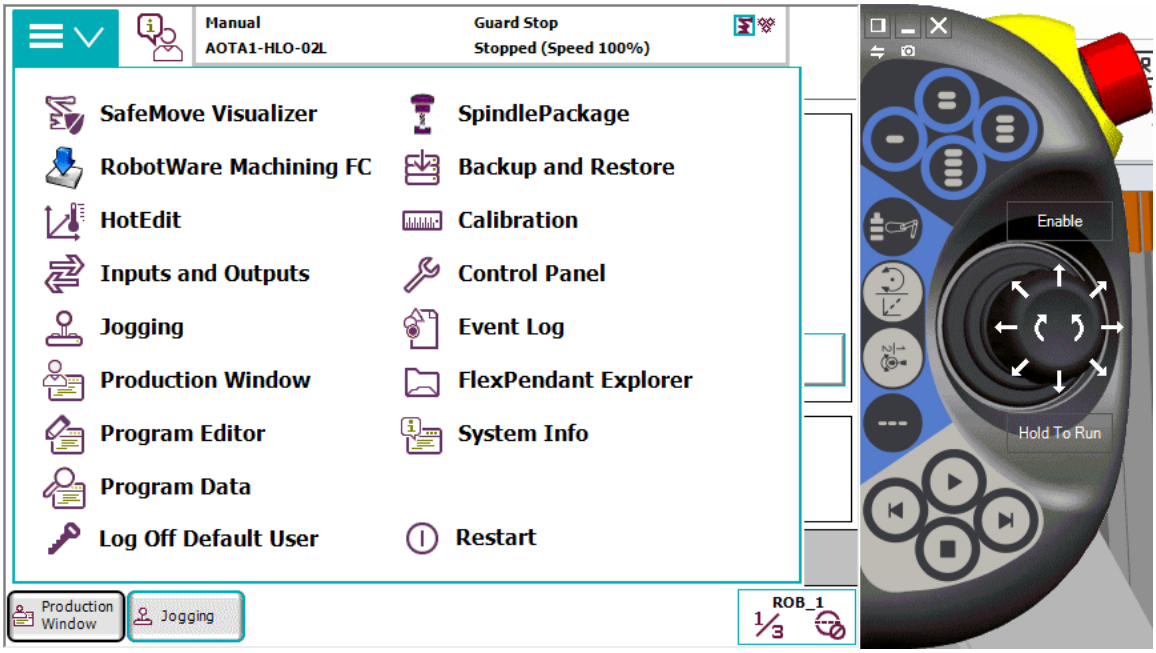
Kuva 1 Postprosessorin valitseminen

Voimaohjaus ei vaadi niin tarkkoja TCP-määrityksiä kuin monet muut sovellukset. Kuvan 1 Download-sivulta löytyy asetus WriteToolData. Tämä on hyödyllinen asetus, koska TCP:tä ei tarvitse määrittää robotilla, mikä on yleensä vaikeaa ja aikaa vievää.

Ruuvipenkin kalibrointi

Jos ruuvipenkki irrotetaan, se joudutaan kalibroimaan uudelleen. Tässä on ohjeet ruuvipenkin kalibroimiselle. Robotissa tulee olla piikkityökalu kalibrointia tehdessä.

Avaa valikko kuten kuvassa 2. Siirry Jogging-välilehdelle.



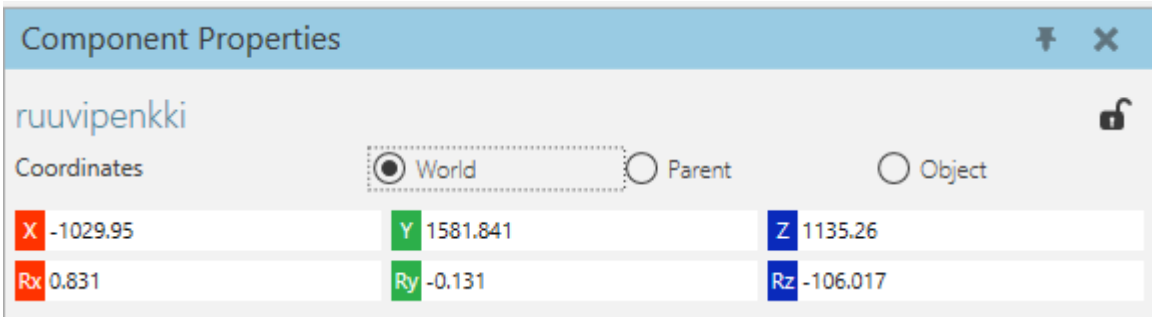
Kuva 2 Käsiohjain

Klikkaa *Work object:* -kohdasta. Valitse wRuuvipenkki. Sitten klikkaa alhaalta Edit ja sitten Define. Etsi ruuvipenkistä kolme pistepiikillä lyötyä piikkiä. Määritä koordinaatisto niin, että origo on nurkkapisteessä (kahden muun pisteen väliin jää 90 astetta). X-akselin tulee osoittaa ruuvin suuntaisesti.

Avaa RobotStudio ja yhdistä robotin ohjain tietokoneeseen. Etsi koodista wRuuvipenkki, jonka pitäisi näyttää kuten alla. ”STN1”-sanan jälkeen on kolmen luvun ryhmä ja neljän luvun ryhmä. Kolmen luvun ryhmä kertoo x-, y- ja z-koordinaatit. Neljä viimeistä on orientaatio.

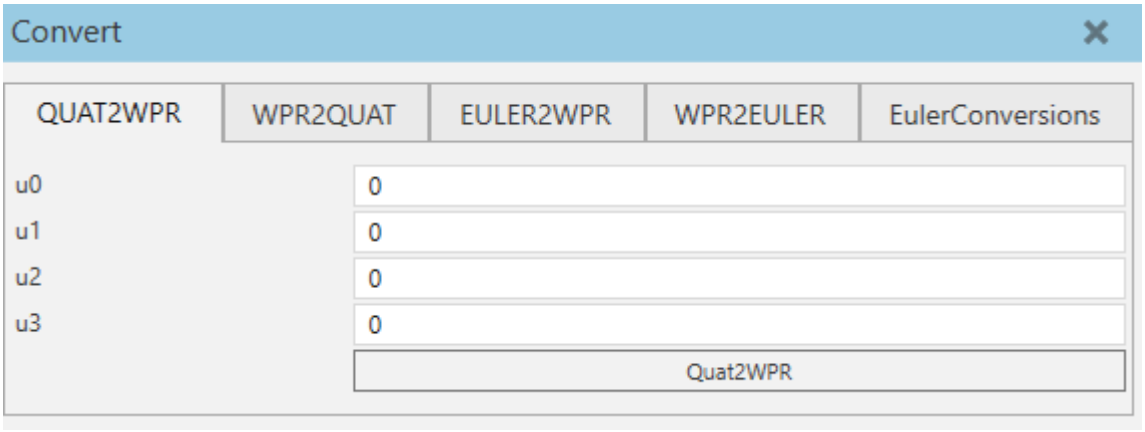
```
PERS wobjdata wRuuvipenkki:=[FALSE,FALSE,"STN1",[[1266.88,-
147.214,515.523],[0.00813022,0.00129205,0.000585722,-
0.999966]],[[0,0,0],[1,0,0,0]]];
```

Klikkaa Visual Componentsissa ruuvipenkkiä. Kuva 3 näyttää miten koordinaatit tulee syöttää. Varmista, että valittuna on World. Rx-, Ry- ja Rz-arvoihin ei tarvitse tässä vaiheessa koskea.



Kuva 3 Ruuvipenkin koordinaattien asettaminen

Visual Componentsissa ylhäältä valintanauhasta löytyy Extras-nappi, jonka alla on Convert. Kuva 4 näyttää millainen ikkuna avautuu. Kirjoita arvojen kohdalle wRuuvipenkki-työkohdekoordinaatiston orientaation arvot.



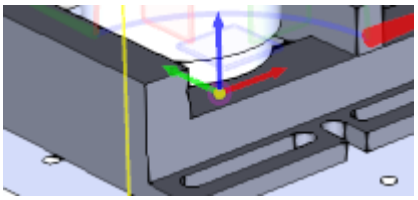
Kuva 4 Kvaternio kulmamuotoon

Nappia painettaessa konsoliin pitäisi tulla kuvan 5 mukainen teksti.

Kuva 5 Kvaternio kulmamuodossa

Sulje Convert-työkalu, ja klikkaa jälleen ruuvipenkkiä. Tällä kertaa Worldin sijasta valitse Object. Kirjoita konsoliin tulleet lukuarvot Rx-, Ry- ja Rz-kohtiin.

Klikkaa robottia, ja valitse valintanauhasta Jog. Valitse oikealle ilmestyneestä ikkunasta Base-kohdasta wRuuvipenkki. Tämän jälkeen klikkaa samalta riviltä hammaspyörän kuvaketta. Klikkaa valintanauhasta Snap-työkalu aktiiviseksi. Valitse Snap Typeksi origin. Varmista, että Settings-kohdassa kaikki valinnat ovat valittuna, ja että Align axis on Z+. Siirrä hiiri ruuvipenkin kohdalle. Näytölle pitäisi tulla ruutu, jossa lukee ”NodeOrigin of ruuvipenkki”. Varmista, että koordinaatisto tulee samaan paikkaan kuin wRuuvipenkki määriteltiin robotille. Kuva 6 näyttää mihin basen pitäisi tulla.



Kuva 6 Basen määrittäminen Align-työkalulla