Joose Nurminen

Säätöskriptien kirjoittaminen laboratorioautomaatiomoduuleille

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka, monimuoto

Insinöörityö

30.4.2024

Tiivistelmä

Tekijä:	Joose Nurminen
Otsikko:	Säätöskriptien kirjoittaminen laboratorioautomaatiomoduuleille
Sivumäärä:	51 sivua + 4 liitettä
Aika:	30.4.2024
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaaja:	Vanhempi ohjelmistoarkkitehti Mikko Pulli Osaamisaluejohtaja Janne Salonen

---

Insinöörityö on suoritettu osana normaalia kokopäiväistä palkkatyötä Thermo Fisher Scientific Finlandin Vantaan toimipisteessä. Työn tavoitteena oli luoda säätöskriptit, joiden avulla laboratorioautomaatiomoduuleja voisi säätää melkein kuka tahansa skriptien ohjeistusta seuraamalla sekä tarvittaessa muokata ja luoda omia skriptejään niiden pohjalta. Alun perin ideana oli saada skriptit valmiiksi mahdollisimman nopeasti, noin kuudessa viikossa, minkä jälkeen ne oli ollut tarkoitus luovuttaa moduuleja valmistavalle alihankkijalle, jotta he voisivat jatkossa säätää valmistamansa moduulit itse, jolloin R&D-osaston ohjelmistokehittäjät ja V&V-insinöörit voisivat keskittyä muihin asioihin moduulien säätämisen sijaan.

Työssä tutkitaan skriptikirjoitusprosessin eri vaiheita, refaktorointiprosesseja, hyväksi havaittuja käytäntöjä sekä skriptausprosessin haasteita mahdollisine ratkaisuineen. Teoriaosuudessa keskitytään niihin periaatteisiin, joilla skriptejä pyrittiin tekemään mahdollisimman ymmärrettäviksi ja helppolukuisiksi loppukäyttäjiä ajatellen.

Projektin alkuperäinen kuuden viikon aikataulu oli liian kunnianhimoinen, ja skriptien valmistuminen ja luovuttaminen alihankkijalle lykkääntyi erinäisistä syistä useita kuukausia. Tänä aikana skriptit ehdittiin kuitenkin testata, hioa ja korjata virheiltä paljon parempaan ja toimivampaan muotoon, kuin mitä ne olivat ensimmäisten skriptiversioiden aikana.

Avainsanat: C#, Docker, Roslyn, skriptaus, laboratorioautomaatio, refaktorointi, Clean Code

---

Abstract

Author:	Joose Nurminen
Title:	Writing Adjustment Scripts for Laboratory Automation Modules
Number of Pages:	51 pages + 4 appendices
Date:	30 April 2024
Degree:	Bachelor of Engineering
Degree Programme:	Information Technology
Professional Major:	Software Engineering
Supervisors:	Mikko Pulli, Senior Software Architect Janne Salonen, Director of School

---

The engineering thesis was conducted as part of regular full-time employment at Thermo Fisher Scientific Finland's office in Vantaa. The aim of the work was to create adjustment scripts that would allow almost anyone to adjust laboratory automation modules by following the instructions in the scripts, as well as modify and create their own scripts based on them. Initially, the idea was to complete the scripts as quickly as possible, within about six weeks, and then hand them over to the module manufacturer so that they could adjust the modules themselves in the future. This would allow the R&D software developers and V&V engineers to focus on other tasks instead of adjusting modules.

The work explores the different stages of the script writing process, refactoring processes, best practices, and challenges of the scripting process along with their possible solutions. The theoretical part focuses on the principles used to make scripts as understandable and readable as possible for end-users.

The original six-week schedule for the project proved to be too ambitious, and the completion and handover of the scripts to the module manufacturer were delayed for several months due to various reasons. However, during this time, the scripts were thoroughly tested, refined, and corrected to a much better and functional form than they were in the initial script versions.

Keywords: C#, Docker, Roslyn, scripting, laboratory automation, refactoring, Clean Code

Sisällys

Lyhenteet

AIM: *Analyzer Interface Module*. Laboratoriomoduuli, joka toimii rajapintana eri analysaattorien (esim. ACLTOP, Vitros) välillä. Vastaa vanhan TCA-järjestelmän Bypass-moduulia.

CAN: *Controller Area Network*. CAN-väylä on sarjaliikenneprotokolla, joka mahdollistaa mikrokontrollerien keskinäisen kommunikaation.

DLL: *Dynamic-link library*. Windows-järjestelmissä käytetty tiedostomuoto, jonka avulla voidaan jakaa ohjelmakoodia ja dataa, jotta eri ohjelmat voisivat käyttää niitä samanaikaisesti.

GUI: *Graphical User Interface*. Graafinen käyttöliittymä, jossa koneiden ja ohjelmistojen kanssa kommunikointi perustuu kuvakkeiden ja muiden graafisten elementtien käyttöön.

IDE: *Integrated Development Environment.* Integroitu ohjelmointiympäristö on ohjelmisto, joka tarjoaa ohjelmistokehittäjälle työkaluja, kuten esimerkiksi koodieditorin ja virheenjäljitystyökalun, joiden avulla hän voi suoriutua työstään tehokkaammin.

IMT: *Installation and Maintenance Tool*. Laboratoriomoduulien säätö- ja testiskriptien ajamiseen tarkoitettu ohjelmisto, jota ajetaan omassa Docker-kontissaan (container) ja johon ollaan yhteydessä selaimen kautta.

IOS: *Input-Output Sorter*. Laboratoriomoduuli, jonka kautta koeputket syötetään järjestelmään ja jota kautta ne myös poistuvat järjestelmästä. Vastaa vanhan TCA-järjestelmän ES Flex -moduulia.

JSON: *JavaScript Object Notation*. Tekstipohjainen ihmisen luettavissa oleva tiedostomuoto tiedonvälitykseen ja tallennukseen.

LAM: *Laboratory Automation Module*. Laboratorioautomaatiomoduuli, kuten IOS tai AIM.

R&D: *Research and development*. Tutkimus- ja kehitystyö.

RF: *Robot Framework*. Avoimen lähdekoodin testiautomaatiokehys, jolla voidaan automatisoida toiminnallisuuden testausta.

SAFe: *Scaled Agile Framework*. Viitekehys skaalautuvan ja ketterän (agile) ohjelmistokehityksen toteuttamiseen suurissa organisaatioissa.

SSH: *Secure Shell*. Protokolla, jonka avulla käyttäjä voi ottaa yhteyttä tietokoneeseen suojaamattoman verkon kautta.

STAT: *STATim* (lat. ”välittömästi”). Käytetään priorisoitavista näytteistä, jotka täytyy käsitellä mahdollisimman nopeasti.

STO: *Safe Torque Off*. Moottorien turvallisuusominaisuus, jossa virta katkaistaan moottoriin, muttei koko järjestelmään. Hätäseis-painikkeen painaminen on esimerkki STO-signaalin aktivoimisesta.

TCA: *Thermo Clinical Automation*. Thermo Fisherin nykyinen laboratorioautomaatiojärjestelmä.

V&V: *Verification and Validation*. Termillä viitataan verifiointi- ja validointiprosessiin kuin myös kyseisen prosessin parissa työskenteleviin insinööreihin (V&V-insinööri).

1 Johdanto

Tämän Thermo Fisher Scientific Finlandilla tehdyn työn yhteydessä toteutetun insinöörityön tavoitteena oli luoda tarvittavat säätöskriptit IOS- ja AIM-laboratorioautomaatiomoduuleille. Tarkoituksena oli, että säätöskriptien avulla moduulit voisi jatkossa säätää helposti kuka tahansa skriptien kehoteikkunoiden ohjeistusta seuraamalla. Tällaisia tahoja ovat muun muassa yrityksen omat ohjelmistokehittäjät ja V&V-insinöörit kehitysvaiheessa, yrityksen omat tuotannon laitekoestajat, laitteita valmistavan alihankkijan työntekijät sekä laitteita käyttävien ja loppuasiakkaina olevien sairaaloiden ja klinikoiden luona työskentelevät asiakasyritysten kenttähuoltoinsinöörit.

Alun perin tavoitteena oli saada säätöskriptit valmiiksi mahdollisimman nopeasti, 3 sprintin (6 viikkoa) aikana, jotta alihankkija voisi mahdollisuuksien mukaan säätää jo ensimmäiset valmistamansa pilottimoduulit. Tämä tavoite on kuitenkin lykkääntynyt jatkuvasti muun muassa sen takia, että alihankkijan päässä tapahtuvan käyttöönottoprosessin suhteen ilmeni aukkoja ja ongelmia, kuten mitä kaikkea alihankkija vielä tarvitsisi meiltä skriptien lisäksi. Lisäksi käyttöönottoprosessissa tarvittavien resurssien luominen oli jo ulkoistettu toiselle alihankkijataholle, eikä oman rinnakkaisen tilapäisratkaisun tekemistä asian suhteen pidetty järkevänä resurssien käyttönä, mikä lykkäsi skriptien luovuttamistakarajaa entisestään.

Opinnäytetyön teoriaosuudessa käydään lävitse niitä skriptauksen lähtökohtia, periaatteita ja muita seikkoja, joita skriptien kirjoittamisessa ja refaktoroinneissa olisi hyvä ottaa huomioon. Lisäksi avataan hieman skriptien ajamiseen käytetyn IMT-työkalun toimintaa ja taustaa. Työn pääpaino on kuitenkin itse skriptausprosessin käytännön tekemisen kuvaamisessa sekä sen aikana ilmenneiden ongelmien puimisessa.

2 Skriptauksen lähtökohtia

2.1 Clean Code -periaatteet

Skriptauksessa sovellettiin samoja yrityksessä ja yleisesti käytössä olevia ohjelmistokehitysperiaatteita kuin varsinaisen lähdekoodiohjelmistonkin kehittämisessä: skriptien oli oltava mahdollisimman ymmärrettäviä, helppolukuisia ja helposti muokattavissa olevia. Tämä on erityisen tärkeää lähdekoodin suhteen, koska ohjelmistoa kehittävät eteenpäin ja sitä tulevat ylläpitämään vuosien saatossa lukuisat eri ohjelmistokehittäjä, etenkin kun kyseessä on pitkän elinkaaren tuote 10–20 vuoden elinkaarella. Skriptien kohdalla korostui vielä se, että niitä tulevat käyttämään ja tarvittaessa myös muokkaamaan sellaiset henkilöt, joilla ei ole välttämättä juurikaan kokemusta ohjelmistokehityksestä. Näistä yleisistä ohjelmointiperiaatteista ja käytänteistä puhuttaessa käytetään useimmiten Robert C. Martinin lanseeraamaa Clean Code -käsitettä (”puhdas koodi”).

Skriptien osalta erityisesti seuraavat Clean Code -periaatteet nousivat esille:

- Selkeä nimeämiskäytäntö: funktioiden ja muuttujien nimet heijastavat niiden tarkoitusta (Martin 2009: 18).

- Funktioiden pitäminen yksinkertaisina ja sopivan pieninä (Martin 2009: 34).

- Turhan tiedon abstraktointi pois käyttäjän näkyviltä: toistuvat rakenteet ja muuttujat voidaan sijoittaa omiin kirjastoihinsa (Martin 2009: 94).

Suurin poikkeus Clean Code -periaatteista on runsas kommenttien käyttö, jotta ohjelmointia osaamattomat käyttäjät ymmärtäisivät, mitä missäkin skriptin vaiheessa pitäisi tapahtua. Runsas kommentointi on useimmiten ollut merkki nimeämisen epäonnistumisesta (Martin 2009: 54), kun koodi ei kommentoi itseään, tai liian pitkistä ja kompleksisista rakenteista (Fowler & Beck 2018: 73), joita täytyy selittää auki erillisin kommentein. Toive skriptien eri vaiheiden kommentoimiseen tuli kuitenkin erikseen yrityksen huolto-osaston työntekijöiltä,

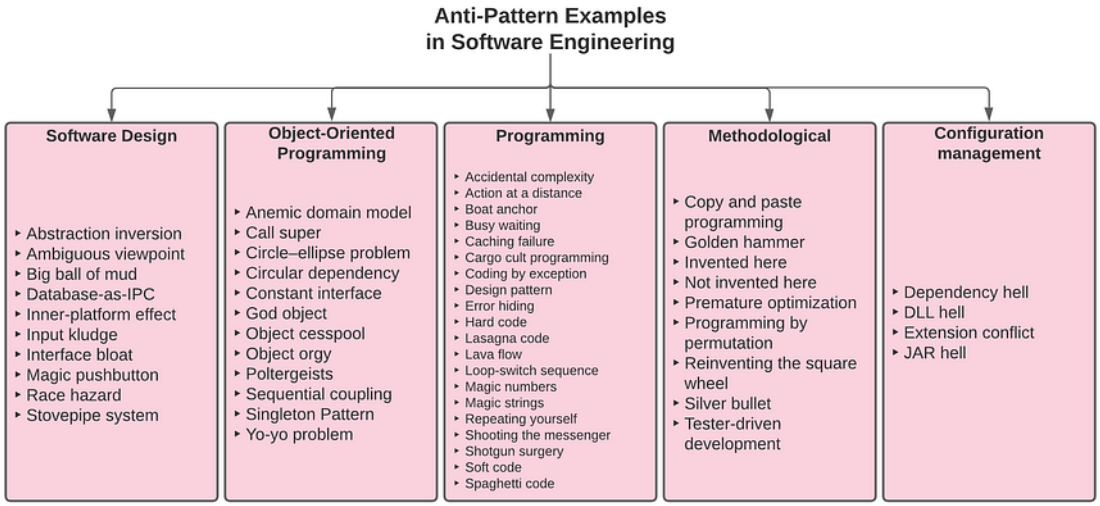
jotka olivat hyödyntäneet kommentointia ennenkin legacy-järjestelmän skriptien kanssa työskennellessään.

Näitä kaikkia Clean Code -periaatteita ei välttämättä ehditty alun kiireen takia toteuttamaan kaikkein uskollisimmalla tavalla, mutta refaktorointivaiheessa skriptien luettavuuteen ja ymmärrettävyyteen ehti kiinnittämään jo enemmän huomiota, kun niitä ehti testaamaan rauhassa moduulien kanssa.

2.2 Refaktorointi ja vastamallien välttäminen

Parhaista periaatteista ja käytänteistä huolimattakin refaktoroinneille tulee käytännössä aina tarvetta koodin rakenteen muuttuessa ja monen eri ohjelmistokehittäjän työskennellessä saman koodin ääressä. Refaktorointia on yrityksessä ollut tapana tehdä sitä mukaan, kun sellaiseen on tarvetta. Kun ”koodihajuja” (code smell) ilmenee, niin ne pitäisi refaktoroida pois saman tien, jos kyseessä ei ole sen isompi urakka. Kiireessä tällainen voi johtaa kuitenkin siihen, että refaktorointi jää kokonaan tekemättä, jos sellaiseen ei ole aikaa ja koodi toimii muuten, kuten pitääkin.

Refaktoroinnilla pyritään välttämään laajemmin haitallisten ”vastamallien” (anti-pattern) juurtuminen lähdekoodiin. Vastamallilla tarkoitetaan useimmiten sellaista ratkaisua, joka aiheuttaa pitemmän päälle negatiivisia vaikutuksia, vaikka se olisi sinänsä käypä ja toimiva ratkaisu (Brown ym. 1998: 6). Tällaisia vastamalleja ovat esimerkiksi ”nuolenpää” (arrowhead), jossa pitkät sisäkkäiset if-else ehtolausekkeet tekevät koodin vaikealukuiseksi (Acharyya 2024). ”Veneankkuri” (boat anchor) on puolestaan vastamalli, jossa tarpeettoman koodin annetaan ikään kuin ankkuroitua lähdekoodiin, koska ohjelmoija saattaa olettaa tarvitsevansa sitä tulevaisuudessa. Tämä on useimmiten virheellinen olettama, mikä puolestaan saattaa hämätä muita ohjelmistokehittäjiä vastaisuudessa (Cimen 2021), etenkin jos alkuperäinen ohjelmoija ei enää työskentele samassa yrityksessä.



Kuva 1. Ohjelmistokehityksen vastamalleja (Cimen 2021).

Olisi hyvä olla tietoinen yleisimmistä vastamalleista ja paremmista ratkaisutavoista eri vastamallien kohdalla, jotta niiden syntyminen voitaisiin havaita ja estää jo kirjoitusvaiheessa. Etenkin skriptaaminen saattaa olla proseduraalisen ja rivi riviltä etenevän luonteensa takia altis virheille tekijän voidessa yrittää nopeasti korjata virheitä leikkaa-liima-menetelmällä tai tilapäisiä muuttujia ja funktioita lisäämällä, jotta skripti toimisi halutulla tavalla ainakin kyseisen ajon aikana.

On myös mahdollista, että skripteistä on vaarana tulla ”purkkaa”, etenkin kiireessä, jolloin päätavoitteeksi tulee sen rakenteen ja luettavuuden optimoinnin sijaan valmiin tuotteen lähettäminen asiakkaalle tiukan aikataulun sanelemana. Skripteille annetun 6 viikon aikataulun puitteissa skriptien kunnollinen testaaminen ei ainakaan ollut mahdollista, eikä siten ollut myöskään mahdollista tunnistaa mahdollisia virhetilanteita ja tehdä virheenhallintaa tällaisten tilanteiden osalta.

2.3 UX ja loppukäyttäjäkokemus

Säätöskriptejä tulevat käyttämään sellaiset käyttäjät, joilla ei ole todennäköisesti lainkaan ohjelmointikokemusta, ymmärrystä skriptien toiminnasta tai mahdollisuutta edes korjata niitä, joten on tärkeää, että ne toimisivat mahdollisimman

moitteettomasti tilanteessa kuin tilanteessa. Useimmat säätöskriptien loppukäyttäjät tuskin tulevat muokkaamaan skriptitiedostoja tai lukemaan niiden koodia tarkemmin, minkä takia olisikin tärkeää panostaa siihen, että käyttäjää ohjeistetaan riittävästi skriptien ajon aikana, jotta hän käyttäytyisi skriptien kannalta odotetulla tavalla. Pelkkä moduulin oven auki unohtuminen voi aiheuttaa virheen, minkä myötä säätöprosessi täytyy aloittaa alusta asti uudestaan.

Käyttäjää pitää neuvoa selkeästi, mitä hänen tulee skriptin ajon aikana tehdä ja milloin eikä mitään saisi jättää olettaman varaan. Skriptejä kirjoittaessa ja ajaessa ohjelmistokehittäjälle piirtyy selkeä kuva siitä, mistä oikein on kyse, mutta loppukäyttäjä ei välttämättä tiedä juuri mitään siitä, mitä hän on tekemässä, mitä moduulin osia hän on (ensimmäistä kertaa) säätämässä ja millä logiikalla skriptit oikein toimivat. Virhetilanteissa skriptiä ajava tavallinen loppukäyttäjä ei ilman ohjeistusta todennäköisesti tiedä, mitä hänen tulisi tehdä virhetilanteesta selvitäkseen.

Käyttäjä tekemät virheet voidaan jakaa kahteen luokkaan: hän voi joko tehdä epähuomiossa lapsuksen (slip) tai erheellisen uskomuksen perusteella virheen (mistake) (Main 2020). Näitä molempia virhetyyppejä voidaan yrittää minimoida skriptin eri vaiheissa käyttäjälle esitetyn ohjeistuksen avulla.

Suurimmat lapsukset lienevät sellaisia, ettei käyttäjä jaksa lukea ohjeita, ainakaan joka kerta, ja hän tulee klikanneeksi liian nopeasti skriptiä eteenpäin, jolloin joku vaihe jää tekemättä ja skripti joutuu virhetilaan. Ratkaisuna tähän voisi pohtia, olisiko yhden suuren tekstiseinän sijaan paikallaan vaikkapa kolmen erillisen tekstikehotteen näyttäminen vaiheittain (esimerkkikoodi 1). Edellisessä on riskinä, ettei käyttäjä lue ohjeita loppuun asti. Jälkimmäisessä hän saattaa rutiinin myötä alkaa klikkailemaan kehotteita liian nopeasti lävitse.

```
await InvokeConfirmPromptAsync($"Avaa moduulin ovi, aseta kalibrointi-
työkalut paikoilleen ja sulje moduulin ovi.”);
//tai
await InvokeConfirmPromptAsync($"Avaa moduulin ovi.”);
await InvokeConfirmPromptAsync($"Aseta kalibrointityökalut paikoil-
leen.”);
await InvokeConfirmPromptAsync($"Sulje moduulin ovi.”);
```

Esimerkkikoodi 1. Ylemmässä esimerkissä käyttäjälle annetaan kolme ohjetta yhdellä kehotteella. Alemmassa esimerkissä samat ohjeet annetaan erikseen kolmena eri kehotteena. Jokainen kehote pitää hyväksyä erikseen Confirm-nappia painamalla.

Varsinainen virhe käyttäjältä olisi vaikkapa se, jos hän laittaisi kalibrointityökalun väärään kohtaan ja yrittäisi paikata tätä liian myöhään avaamalla moduulin ovet erheensä korjatakseen, jolloin STO-signaali aktivoituisi, moduuli pysähtyisi ja skriptin ajo pysähtyisi virhetilaan, jota ei ole otettu skriptissä huomioon. Tällaisessa tapauksessa käyttäjä saa jo nyt toki virheilmoituksen siitä, kuinka esimerkiksi moottorikomento ei mennyt perille, mutta se ei välttämättä kerro käyttäjälle, mitä hänen pitäisi seuraavaksi tehdä, vaan hänelle pitäisi todennäköisesti näyttää kustomoitu virheilmoitus jatko-ohjeineen.

Koska käyttäjää ohjeistetaan skriptin aikana yksinomaan tekstikehottein, ilmaisun tulisi olla selvää ja yksiselitteistä. On myös varauduttava siihen, etteivät Suomessa valmistettavien ja koestettavien moduulien säätäjät osaa englantia, minkä takia tekstit on myös käännetty suomeksi. Kuvalla selostaminen olisi ollut parhain tapa selostaa joitakin asioita esimerkiksi sitä, mihin kohtaan käyttäjän tulee laittaa kalibrointityökalu tai mihin kohtaan kiekkojen tulisi asettua säädön jälkeen, jos ne on säädetty oikein. Tuen lisäämistä kuvaohjeille ei kuitenkaan ole ollut aikomusta lisätä ainakaan vielä tässä vaiheessa.

3 IMT-säätötyökalun käyttö ja toimintaperiaate

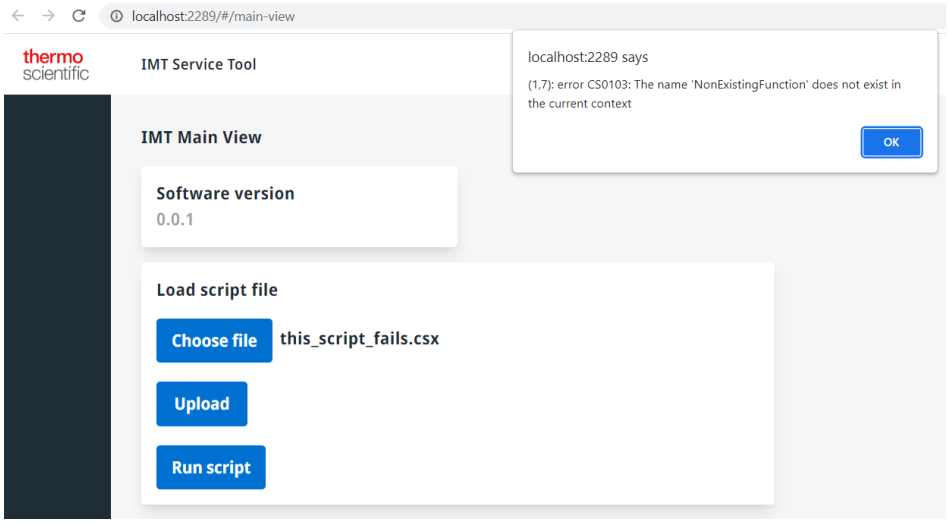
Säätöskriptien ja skriptaamisen ymmärtämiseksi on hyvä käydä lyhyesti lävitse säätötyökalun sekä sen taustalla toimivan skriptijäsentimen toimintaa.

3.1 IMT:stä yleisesti

Säätöskriptejä ajetaan IMT-säätötyökalulla, joka toimii Docker-kontissa, joka puolestaan on yhteydessä LAM-kontissa sijaitsevaan SignalR-hubiin. SignalR-protokollan välityksellä IMT-kontti voi lähettää jäsentämiään skriptikomentoja LAM-kontille, jolla sijaitseva LAM-ohjelmisto voi joko suorittaa ne itse (säätö- ja konfiguraatioarvoihin liittyvät luku- ja päivitysoperaatiot) tai välittää ne edelleen CAN-tasolle CAN-komentoina laitteiston suoritettavaksi. Näin C#-pohjaisten IMT-skriptien avulla LAM-moduulien yksittäisille noodeille (esim. poimijarobotin X-akselin moottorinoodi) pystytään antamaan CAN-komentoja säätö- tai testaustarkoituksessa. IMT ikään kuin ”ohittaa” LAM-ohjelmiston normaalin toiminnan antaen sen kautta käskyjä suoraan laitteiston noodeille.

3.2 IMT:n GUI:n kuvaus

Csx-muotoiset skriptitiedostot ladataan IMT:n käytettäväksi selaimessa toimivan graafisen käyttöliittymän (GUI, kuva 2) avulla. Skriptin latauksen yhteydessä ScriptEvaluator-funktio varmistaa skriptitiedoston virheiden varalta ja hylkää sen ilmoittamalla mahdollisista virheistä. Tämän jälkeen skriptitiedosto voidaan ajaa, jolloin se välittää sisältämänsä komennot ja käskyt LAM-moduulin toteutettavaksi.



Kuva 2. GUI-näkymä ja virheellisestä skriptistä tullut virheilmoitus.

3.3 IMT:n skriptikielen rakenne

IMT:n skriptikieli perustuu C#-kielelle, mikä on luonteva valinta, koska pääohjelmiston bisneslogiikkakerros on kirjoitettu samalla kielellä. Suurimpana erona C#-pohjaisen lähdekoodin toimintaan on se, että IMT:n C#-skripteissä ei tarvitse määritellä luokkia tai nimiavaruuksia erikseen skriptissä (Vogel 2021). Esimerkiksi pelkkä konsolilla toteutettu ”Hello, World” -koodinpätkä voi muodostaa täysin validin ja ajettavissa olevan csx-skriptitiedoston, vaikka se ei tulostaisikaan mitään käyttäjälle.

```
Console.WriteLine("Hello, World");
```

Esimerkkikoodi 2. Konsolilla tulostettu ”Hello, World” -tuloste ei näy IMT:n käyttäjälle, mutta koska koodi on syntaksisesti oikein, skripti voidaan ajaa onnistuneesti, minkä myötä käyttäjä saa GUI:n kautta ”Script has finished successfully” -ilmoituksen.

Säätöskriptejä varten on luotava kustomoituja funktioita, joita voidaan kutsua skripteistä antamaan komentoja LAM-moduulin moottorinoodeille.

```
await MotorRotateToPosition(71, 100);
```

Esimerkkikoodi 3. Tässä skriptikomennossa X-akselin moottori, jonka noodin tunnistenumero on 71, ajetaan positioon 100. Noodin tunnistenumeron suoran käyttämisen sijaan olisi suotavampaa sijoittaa se kuvaavampaan muuttujaan, kuten ”robotXMotor”, jotta koodi olisi luettavampaa.

IMT:n palvelinpuolella kyseinen funktio lähettää MotorRotateToPositionCommand-komennon LAM-moduulille.

```
public async Task MotorRotateToPosition(int node, float position)
{
    var command = new MotorRotateToPositionCommand(node, position);
    _ = await InvokeCanCommandAsync(command);
}
```

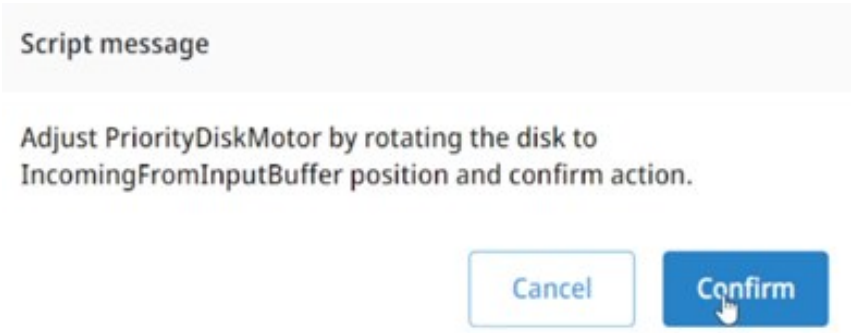
Esimerkkikoodi 4. MotorRotateToPosition koodi IMT:n puolella. Funktio ei palauta arvoa, joten voimme käyttää discard-muuttujaa ”_” pelkän komennon lähettämiseksi.

LAM vastaanottaa komennon lähettääkseen sen eteenpäin tarvittavalle noodille.

```
public async Task<CommandResponse> SendMotorRotateToPositionCom-
mand(MotorRotateToPositionCommand motorRotateToPositionCommand)
{
    return await hardwareConnection.SendMessageAsync(motorRotateToPo-
sitionCommand);
}
```

Esimerkkikoodi 5. Ennen lähettämistä LAM:lle MotorRotateToPositionCommand-funktiokutsuun liitetään vielä ”Send” alkuun. Tämän jälkeen komento lähetetään eteenpäin itse noodille.

CAN-komentojen lähettämisen lisäksi skriptikäskyjen avulla voi lukea ja päivittää tietokannassa sijaitsevia moduulin säätöarvoja (adjustments) sekä konfiguraatioarvoja (configurations). Käyttäjää voi myös ohjeistaa tekstikehottein (prompt) tekemään jotain tietyssä vaiheessa skriptiä (kuva 3).



Kuva 3. Käyttäjälle näkyvä ohjeistava tekstikehote. Käyttäjä voi joko jatkaa painamalla "Confirm" tai peruuttaa skriptin ajon painamalla "Cancel".

Suuri osa tarvittavista komennoista implementoitiin jo vanhan sekä uuden skriptijäsentimen implementoinnin yhteydessä, mutta tarvittaessa niitä voitiin luoda lisää lisäämällä LAM:n puolelle tarvittavat funktiot, jotka olivat yhteisten rajapintojen avulla myös IMT:n käytettävissä. Kun skriptijäsentimen asetuksiin lisättiin viittaukset tarvittaviin dll-tiedostoihin, voitiin IMT:n sisältämiä funktioita käyttää skripteissä sellaisenaan.

4 Skriptausprosessin kulku

4.1 Säätöskriptien tarkoitus

Säätöskriptien avulla on tarkoitus selvittää mitkä ovat moduulin poikkeama-arvot (offset) reaalimaailman toleransseista. Tietokannasta ja teknisistä piirustuksista löytyvät mitat ja arvot eivät ole käytännössä koskaan yksi yhteen valmistettujen laitteiden kanssa, vaan niissä voi olla muutamia millejä heittoa suuntaan tai toiseen, millä on kuitenkin merkitystä, kun kyse on koeputken poimimisesta tietystä kohtaa tai sen sijoittamisesta tiettyyn paikkaan. Jos laitteen pöytä on esimerkiksi hieman kalteva tai koeputkilaatikko ei ole jostain syystä asennettu aivan kohdilleen, tulee sen koordinaatiston XYZ-arvoihin muutoksia. Lisäksi kiekkomoottorien lopulliset positiot pitää aina laskea kaavalla sen perusteella, mihin asentoon kiekko asettuu, kun se ajetaan kotipositioon. Tämän takia tietokantaan tarvitsee asettaa poikkeama-arvoja koordinaatiston XYZ-arvoille ja kiekoille, jotta voitaisiin laskea tarkkaan, mihin kohtaan poimijarobotti päätyy tai mihin kohtaan kiekko lopulta asettuu moottorinoodin alustuksen (initialization) yhteydessä.

Koska säätöskriptit on tarkoitettu myös ohjelmointikokemusta vailla olevien ns. ”maallikoiden” ajettaviksi, on ollut tärkeää, että käyttäjää ohjeistetaan riittävillä kehotteilla, kun hänen tarvitsee tehdä jotakin skriptin ajon aikana, esimerkiksi laittaa kalibrointityökalu paikoilleen. Lisäksi huolto-osaston työntekijöillä oli toivomuksena, että skriptejä kommentoitaisiin riittävästi, minkä takia niihin on lisätty paljon kommentteja selittämään eri funktioiden toimintaa ja tarkoitusta sekä skriptin eri vaiheita. Myöhemmin käyttäjälle osoitetut kehotteet käännettiin myös suomeksi, jotta niitä varmasti ymmärtäisivät niin englannin- kuin suomenkielentaitoiset työntekijät esimerkiksi moduuleja valmistavalla alihankkijalla.

Skriptausprosessi itsessään oli varsin suoraviivainen, mutta matkalla esiin tulleiden ongelmien takia se koki muutamia viivästyksiä ja koodia jouduttiin refaktoroimaan muutamaan otteeseen skriptirajapinnan muutoksien, tarkentuneiden vaatimusten sekä moduuleissa huomattujen eroavaisuuksien myötä.

Tarkastelen ja kommentoin skriptien osalta itse tekemiäni skriptejä sekä muiden tekemiä skriptejä sen osalta, miten olen niitä refaktoroinut sitä mukaan kuin on ollut tarpeellista. Seuraavat säätöskriptit tein itse alusta loppuun:

- AIMAdjustmentScript.csx: AIM-moduulin kiekkojen säätöskripti.

- IOSInitialization.csx[1]: IOS-moduulin moottorinoodien (robotti, kiekot) alustusskripti noodien alustamiseksi.

- IOSRobotAccessPositionAdjustmentScript.csx: IOS-moduulin robotin access-paikkojen säätöskripti.

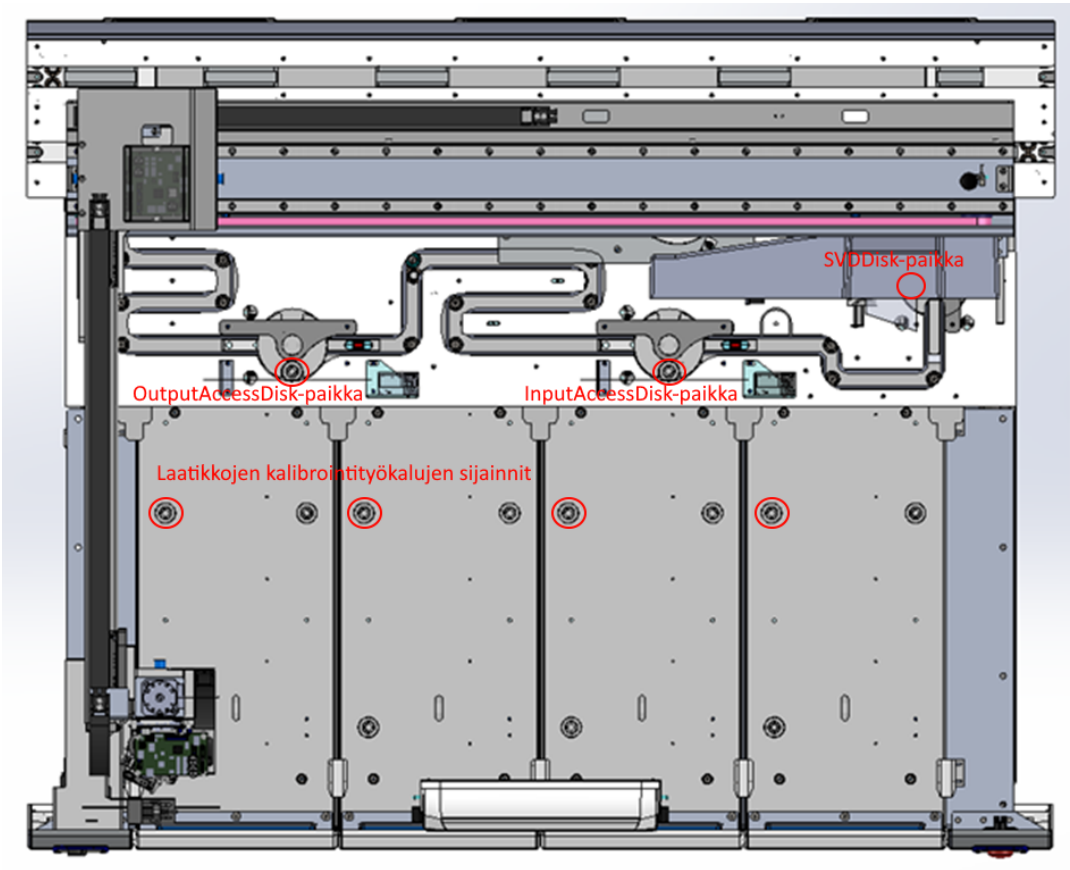
Seuraavat säätöskriptit ovat muiden aloittamia, mutta kirjoitin ne ajan myötä lopulliseen muotoonsa:

- IOSdiskAdjustment.csx: IOS-moduulin kiekkojen säätöskripti.

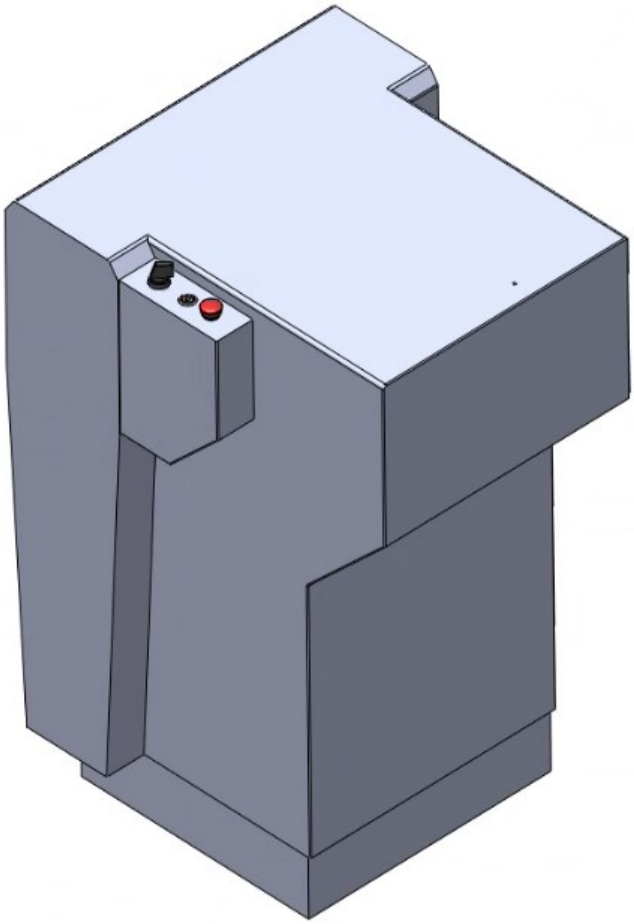
- IOSDrawerCoordinateAdjustment.csx: IOS-moduulin laatikon koordinaattien säätöskripti.

[1] Kirjoitin tämän skriptin aluksi vanhemmalle jäsentimelle, minkä jälkeen sen refaktoroi toinen ohjelmistosuunnittelija uudelle jäsentimelle, minkä minä lopulta refaktoroin sen lopulliseen muotoonsa.

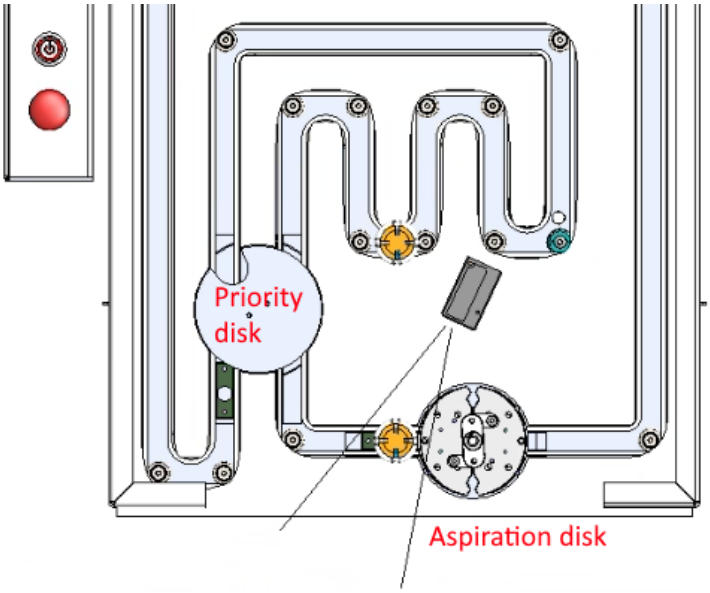
Kuva 4. IOS-moduuli ulkoapäin kuvattuna.



Kuva 5. Kalibrointityökalulla säädettävien paikkojen sijainnit.



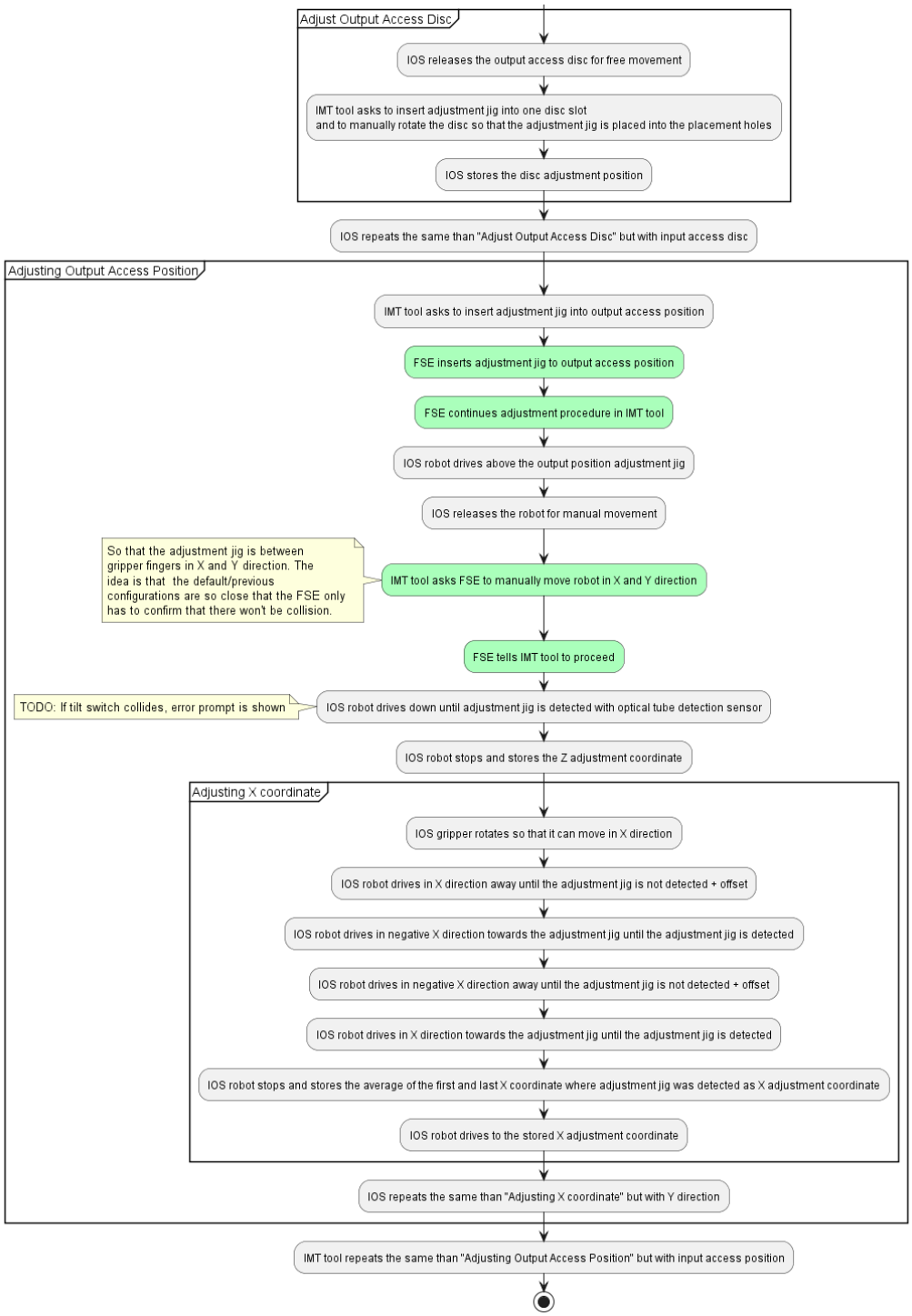
Kuva 6. AIM-moduuli ulkoapäin kuvattuna.



Kuva 7. AIM-moduulin säädettävät paikat.

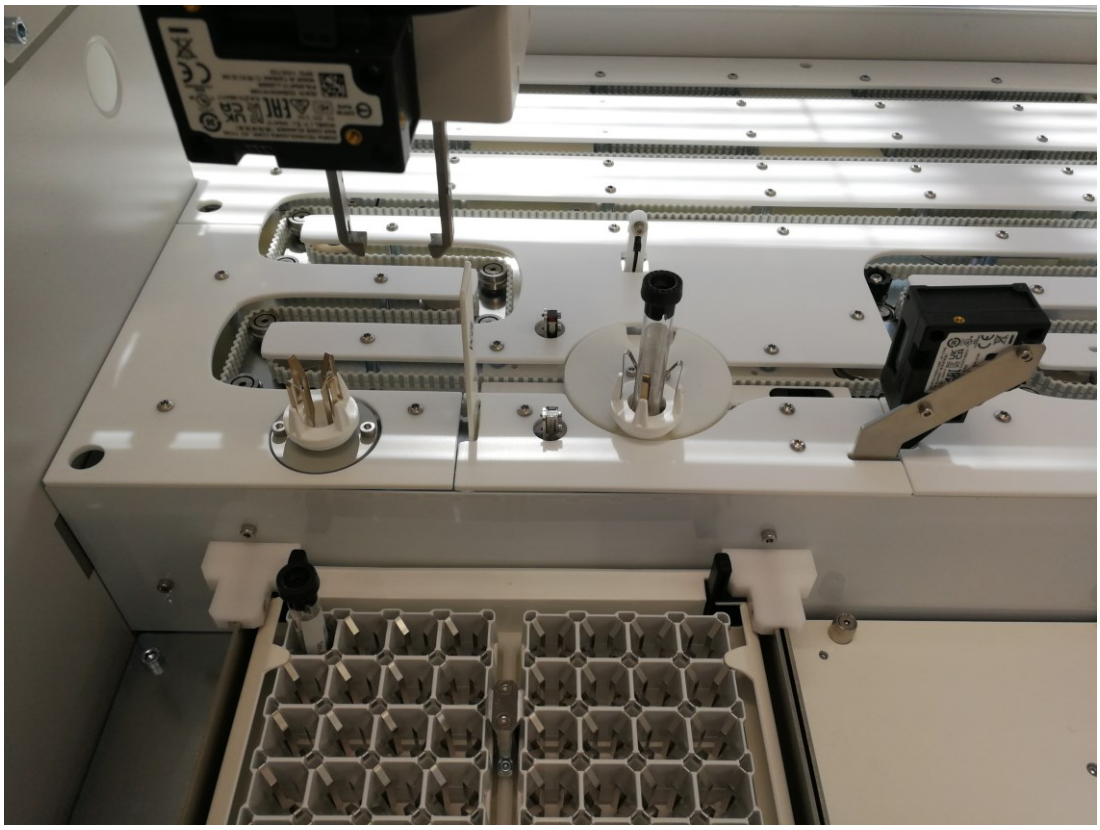
4.2 IOS-robotin access-paikkojen säätöskripti

IOS-robotin access-paikkojen säätöskriptin kirjoittamisessa runkona toimi ohjeistuksena saatu säätöprosessikaavio (kuva 8).



Kuva 8. Input- ja outputpaikkojen säätöprosessikaavio.

IOS-moduulissa on kaksi access-paikkaa tai ”saatavuuspaikkaa”, input ja output, joihin koeputkia mahdollisesti sisältävät kuljettimet (carrier) tulevat. Input, eli syöttöpuolella, kuljettimet ovat tyhjiä, ja poimijarobotti (gripper, specimen robot) poimii koeputket ”koeputkitelineistä” (rack) järjestelmään, kun taas output, tai lähtöpuolella, kuljettimissa mahdollisesti olevat koeputket poimitaan järjestelmästä takaisin telineisiin (kuva 9).



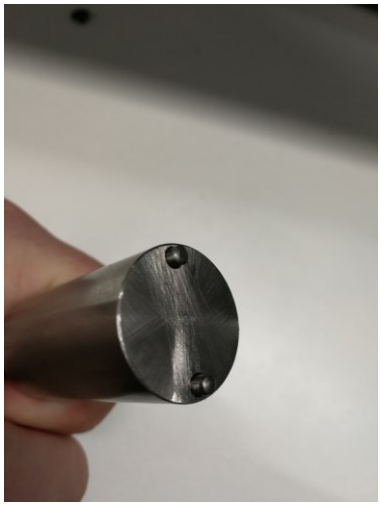
Kuva 9. Poimijarobotti, koeputki laatikon koeputkitelineessä ja toinen koeputki kuljettimessa output-puolen kiekon access-paikassa. Robotin täytyy pystyä poimimaan ja laittamaan koeputki oikeaan kohtaan laatikoissa ja access-paikoissa.

Tietokannassa on suurpiirteiset oletusarvot (kuva 10) sille, mihin kohtaan access-paikkaa koeputki syötetään input-puolella ja mistä kohtaa se poimitaan output-puolella. Koska koeputkia poimivan poimijarobotin leukojen asento (ns. R-asento, tietokannassa ”Phi”) on aina 45 asteen kulmassa, tietokantaan tarvitsee säätää vain tarvittavat XYZ-arvot.

```
COPY public."ModuleCoordinateAdjustments" ("Id", "X", "Y", "Z", "Phi") FROM stdin;
RobotWaitingCoordinateOffset    0    -30 0    0
Drawer1Coordinate    -8    403 0    0
Drawer2Coordinate    229    403 0    0
Drawer3Coordinate    465    403 0    0
Drawer4Coordinate    703    403 0    0
InputAccessDiskRobotCoordinate    612 450 146 45
OutputAccessDiskRobotCoordinate 173 450 146 45
\.
```

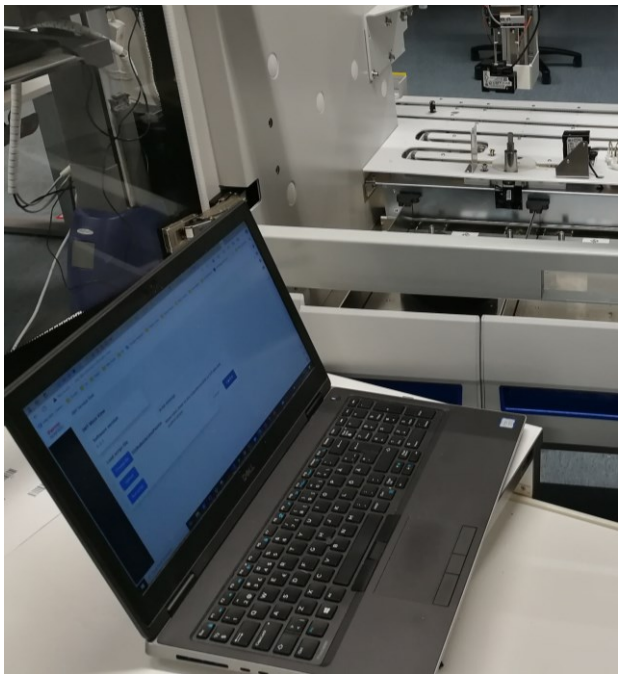
Kuva 10. Input- ja outputpaikkojen oletusarvot tietokannassa.

Tarvittavien XYZ-arvojen selvittämiseksi käytössä oli kalibrointityökalu (kuva 11 ja 12), joka voitiin asettaa access-paikasta löytyviin reikiin oikean paikan varmistamiseksi.



Kuva 11 ja 12. Kalibrointityökalu ylhäältä- ja alhaaltapäin kuvattuna.

Kun kalibrointityökalu olisi paikoillaan, voitaisiin poimijarobottia ajaa sitä kohti XYZ-akseleita pitkin, jotta robotin leukojen välissä oleva sensori voisi tunnistaa tarvittavat reuna-arvot oikeiden XYZ-poikkeama-arvojen selvittämiseksi.



Kuva 13. Tyypillinen säätötilanne työkaluineen. Etualalla olevalla kannettavalla tietokoneella ajetaan skriptitiedostoja IMT:n GUI:n kautta. Taustalla poimijarobotin alapuolella on output-kiekko, johon on laitettu kalibrointityökalu.

4.2.1 Ensimmäinen versio

Ensimmäisessä versiossa opettelin vielä uuden skriptijäsentimen toimintaa: ennen joululomalle lähtöäni käytössä oli ollut vielä TCA-järjestelmälle perustuva skriptijäsennin, joka korvattiin lomieni aikana Microsoftin .NET-kääntäjäalusta Roslynille[2] perustuvalla C#-jäsentimellä. C# skriptausta oli alun perin ajateltu työkaluksi yksittäisten C#- ja .NET-koodinpätkien testausta varten, jottei näitä varten tarvitsisi luoda erillisiä yksikkötestaus- ja projektirakennelmia, jolloin API:tä saattoi testata helpommin (Michaelis 2016), mutta se sopi sellaisenaan myös säätöskriptien kirjoitusta varten. Nykyinen skriptijäsennin osoittautuikin varsin nopeasti helppokäyttöiseksi vanhaan jäsentimeen verrattuna: jäsentimen toimintatavasta ei ollut epäselvyyttä ja siihen oli helppo lisätä uusia

[2] https://github.com/dotnet/roslyn

skriptikomentoja. Ensimmäinen säätöskriptiversio oli valmis kahden viikon sisällä (ks. liite 1).

Suurin osa skriptausajasta meni kirjoittamisen sijasta skriptin käytännön testaamiseen. Käytössä oli toki omalta koneelta lokaalisti ajettavat Docker-kontit, joiden avulla saattoi vielä tarkastaa skriptien syntaksin virheiden varalta, mutta käytännössä suuri osa ajasta piti silti viettää moduulin vieressä skriptin toimintaa seuraten ja sitä mukaa skriptitiedostoja korjaten.

Ensimmäisessä versiossa skripti käytti pitkälti puhtaita CAN-komentoja, kuten MotorRotateToPosition-komentoa robotin liikuttamiseksi. Tässä vaiheessa käytössä oli vain muutamia yhdistelmäkomentoja, joista osa oli sijoitettu skriptitiedostoihin, mikä oli hyvä ratkaisu, ja osa lähdekoodin rajapintaan, mistä taas koituisi tulevaisuudessa ongelmia. Yhdistelmäkomentojen ja muiden apufunktioiden sijoittamisella skriptitiedostoihin lähdekoodin sijasta tavoiteltiin sitä, että skriptit olisivat tarpeen vaatiessa helposti muokattavissa. Muuten riskinä olisi se, että jos jokin skriptikomento ei toimisi halutulla tavalla loppuasiakkaan kohdalla, niin silloin pitäisi tehdä virhepäivitys uusine V&V-testeineen, missä saattaisi mennä useita kuukausia. Lisäksi skriptirajapinta haluttiin pitää ”puhtaana” ylimääräisistä komennoista, jotka eivät varsinaisesti kuuluneet sinne.

Käytetyissä yhdistelmäkomennoissa asetettiin muun muassa moottorinoodien nopeuksia sekä selvitettiin korkeusarvo Z-akselilla ja XY-akselien keskipisteen selvittämiseksi tarvittavat reuna-arvot inkrementoimalla ja dekrementoimalla poimijarobotin liikettä X- tai Y-akselin suuntaisesti, kunnes robotin sensori tunnistaisi kalibrointityökalun, minkä jälkeen kyseisen paikan arvo tallennettaisiin jatkotoimenpiteitä varten. Inkrementointi ja dekrementointi yhdellä yksiköllä (1 mm) toimi kyllä, mutta kyseinen ratkaisutapa sai koko poimijarobotin tärisemään, eikä se näyttänyt kovinkaan sulavalta ja ”ammattimaiselta” ratkaisulta. Lisäksi muutamat tarvittavat nimiavaruudet, kuten System.Threading, oli importoitu vasta skriptitiedostossa sen sijaan, että ne olisi importoitu skriptien käyttöön lähdekoodissa dll-tiedostojen tapaan.

Käskyjen antaminen ja skriptien korjaaminen oli ensimmäisessä versiossa hyvin työlästä ja virhealtista, koska robotin liikuttamiseksi tarvittiin käytännössä neljä skriptikomentoa:

```
//Go to access output position at travel height
await MotorRotateToPosition(ZAxisMotor, ZAxisInitialValue);
await MotorRotateToPosition(XAxisMotor, XAxisInitialValue
await MotorRotateToPosition(YAxisMotor, YAxisInitialValue);
await MotorRotateToPosition(RAxisAndGripper, RAxisInitialValue);
```

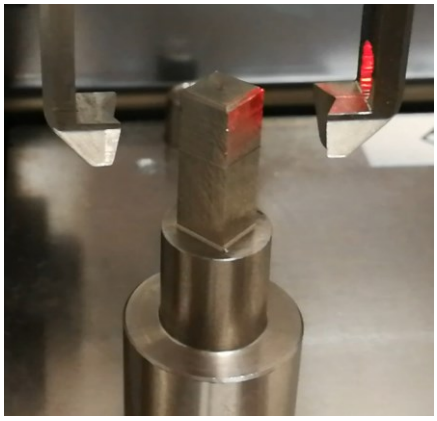
Esimerkkikoodi 6. Ensimmäisen version tapa ajaa poimijarobotti access-paikkaan. Myöhemmin sama hoidettiin yhdellä funktiokutsulla XYZR-kohdearvojen toimiessa parametreina.

Tämä ei olisi välttämättä niin väliksi, jos skriptit olisivat pysyväisluonteisia, mutta tarkoituksena on kuitenkin se, että IMT-skriptejä voisivat muokata ja luoda tarvittaessa myös sellaiset käyttäjät, joilla ei ole juurikaan ohjelmointi- tai skriptauskokemusta (V&V-insinöörit, tuotannon laitekoestajat, kenttähuoltoinsinöörit). Tällöin olisi hyvä, jos valmiista skripteistä voitaisiin tehdä mahdollisimman ymmärrettäviä ja helposti muokattavissa olevia, jotta skriptausvirheiden mahdollisuus olisi minimaalinen.

Ensimmäisen version aikana suurimmat ongelmat liittyivät skriptaamisen sijaan pikemminkin laitteen käyttämiseen. IMT:n skripteillä voi antaa komentoja suoraan moottorinoodeille, mikä tarkoittaa myös sitä, että nämä komennot ohittavat kaikki mahdolliset turvallisuusrajoitukset. Tämän takia poimijarobottia voidaan ohjata vaarallisesti siten, että se voi pahimmassa tapauksessa vahingoittaa laitetta tai sillä työskentelevää käyttäjää, minkä takia onkin tarpeellista alentaa skriptitiedostossa poimijarobotin nopeutta sekä seurata sen toimintaa skriptin ajon aikana käsi hätäseis-painikkeella sen varalta, että robotti käyttäytyy skriptausvirheen tai väärinkäsityksen takia arvaamattomasti.

Sain itse rikottua vahingossa poimijarobotin heti ensimmäisen testausistunnon aikana, kun käskytin sitä liikkumaan input-kiekolta suoraan output-kiekolle, jolloin se ei noussut Z-akselilla turvalliseen korkeuteen ennen XY-akseleilla liikkumista, kuten se normaalisti tekisi laitteen ollessa normaalissa käytössä, vaan se liikkui suoraan horisontaalisesti X-akselin suuntaisesti. Tämän seurauksena

poimijarobotti törmäsi matkalla input-paikan takaviivakoodilevyyn ja output-paikan viivakoodinlukijaan, minkä seurauksena robotti vääntyi kieroksi. Robotti suoristettiin kyllä nopeasti, mutta se oli jo siinä määrin taipunut yläpäästä, että se piti lopulta vaihtaa kokonaan. Tämän huomaamiseen meni jonkin verran aikaa, mikä puolestaan vaikutti kehitystyöhön, koska vääntymisen seurauksena robotin leuat eivät osuneet aina aivan kalibrointityökalun keskelle, vaan ne olivat joko liian vasemmalla tai oikealla, kun robottileukoja käänsi R-akselin suuntaisesti 180 astetta. Luulin aluksi vian olleen skriptissäni, joten kirjoitin sitä moneen otteeseen turhaan uusiksi yrittäessäni saada robotin leukoja osumaan tarpeeksi keskelle kalibrointityökalua (kuva 14 ja 15).



Kuva 14 ja 15. Poimijarobotin leuat ovat yläosan vääntyneiden osien takia selkeästi eri kohdassa, kun leukoja kääntää 180 astetta.

4.2.2 Ensimmäinen refaktorointi – asynkroniset operaatiot ja skriptin rakenne

Ensimmäistä skriptiversiota kirjoittaessa SignalR:ää ei ollut vielä konfiguroitu oikealla tavalla asynkronisia operaatioita varten, vaan skriptikäskyjä voitiin suorittaa vain yksi kerrallaan. Kun LAM:n puolella sijainnut SignalR hubi oli konfiguroitu suorittamaan useita paralleeleja invokaatioita, skriptikäskyjä voitiin antaa rinnakkain, jolloin asynkroniset operaatiot tulivat mahdollisiksi.

Tämän muutoksen myötä päätin refaktoroida inkrementointia ja dekrementointia käyttävän reunantunnistusalgoritmin asynkronisia funktioita käyttäen. Nyt oli mahdollista kuunnella sensoria samaan aikaan, kun ajoi robottia

kalibrointityökalua kohti: kun sensorin tila vaihtuisi havaitessaan XYZ-reunan, arvo tallennettaisiin jatkokäsittelyä varten.

Poistin myös lähdekoodin rajapinnasta muutamia sinne päätyneitä yhdistelmäkomentoja ja sijoitin ne suoraan skriptitiedostoon. Tässä vaiheessa skriptitiedostoissa alkoi olla jo sen verran tavaraa, että skriptin rakennetta täytyi pohtia uudestaan, jotta se säilyisi mahdollisimman selkeänä ja loogisena niin ohjelmistokehittäjille kuin muille skriptejä käyttämään tuleville tahoille. Tässä vaiheessa päädyin seuraavanlaiseen jakoon:

- Skriptimuuttujat, -vakiot ja muut arvot: näitä ovat muun muassa kovakoodatut ”pysyvähköt” arvot, kuten noodien id-numerot, sekä sellaiset arvot, joita saattaa tarvita vaihtaa, kuten offset-arvot, joilla robotti ajetaan lähelle kohdetta.

- Apufunktiot: nämä ovat yhdistelmäkomentoja, jotka kokoavat yhteen LAM:lle lähetettäviä komentoja, jotta joitakin toistuvia asioita, kuten robotin liikuttamista ja nopeuksien asettamista, voitaisiin tehdä helpommin ja suoraviivaisemmin.

- Pääfunktio: tämä on varsinainen pääfunktio, jossa säätäminen tapahtuu apufunktioita ja muita suoraan LAM:lle osoitettuja komentoja käyttäen. Pääfunktion avulla voidaan säätä yksi tai useampi säädettävissä oleva paikka.

- Skriptin vaiheet: varsinainen skripti alkaa tästä, kun funktioita aletaan kutsua. Tarvittaessa säätöfunktioita voi kommentoida pois tai päälle, jos haluaakin vain säätää yhden paikan useamman sijasta.

Tässä vaiheessa oli jo selvillä, mitä nimiavaruuksia suurin piirtein tarvittaisiin, joten skriptin alussa tapahtuva importtaus voitiin poistaa skriptistä ja sijoittaa se lähdekoodin puolelle tapahtumaan samassa yhteydessä dll-tiedostojen referoinnin kanssa.

4.2.3 Toinen refaktorointi – STO-signaalin huomioiminen

Toiseen refaktorointiin oli ryhdyttävä kaikkien skriptien osalta, kun kävi ilmi, että lopullisessa moduulissa tulisi ottaa huomioon STO-signaalin aktivoituminen, mikä ei tällä hetkellä ollut toiminnassa prototyyppi- eikä pilottimoduuleissa. STO aktivoituu, kun moduulin ovet avataan, jolloin virta moottoreihin katkeaa ja ne pitäisi käynnistää skriptissä erikseen uudestaan. Skriptit sisälsivät kohtia, joissa käyttäjää neuvottiin avaaman ovet ja asettamaan kalibrointityökalut paikalleen, sekä tarvittaessa avaamaan ovet liikuttaakseen robottia XY-akselilla, jotta robotti menisi oikeassa kohdassa alas Z-akselia pitkin kohti kalibrointityökalua. Nyt näissä tilanteissa moottorit menisivät STO:n myötä kyllä pois päältä, mutta eivät ikinä kytkeytyisi takaisin päälle, jolloin skriptit jumiutuisivat.

STO-signaalin huomioon ottaminen oli verrattain helppoa: moottorit piti vain käynnistää erikseen jokaisen potentiaalisen moduulin ovenavauksen yhteydessä. Sillä ei ollut niin väliä, avasiko käyttäjä todella ovet vai ei, vaan pääasia oli, että moottorit käynnistettiin tällaisen vaiheen jälkeen uudestaan joka tapauksessa varmuuden vuoksi.

Tässä vaiheessa viimeistään alkoi haittaamaan jo aikaisemmin huomattu ohjelmistollinen vika koskien virran katkaisemista moottoreihin ja niiden laittamista takaisin päälle. Ilmeisesti firmware-tasolla oli jonkinlainen ongelma sen suhteen, jos tiettyihin moottoreihin (R-akseli, kiekkojen moottorit) katkaistiin virta, niitä liikuteltiin, ja virta laitettiin takaisin päälle. Tämän jälkeen moottoreille annetut käskyt eivät enää menneet perille, koska moottori oli jostain syystä mennyt vikatilaan. Onnistuin väliaikaisena ratkaisuna selvittämään moottorien vikatilan sulkemalla ja käynnistämällä moottorit uudestaan muutamaan otteeseen (pois päältä, päälle, pois päältä, päälle), mutta perimmäistä ongelmaa (vikatilaan joutumista) se ei kuitenkaan estänyt eikä minulla ollut sen parempaa käsitystä sulautetun järjestelmän toiminnasta, joten sen ratkaiseminen jäi osaavampien tehtäväksi. Kyseinen ohjelmistovirhe ja sen selvittäminen veivät kuitenkin ajallisesti useamman päivän skriptausaikaa, kun sen mahdollisia syitä selvitettiin, sitä replikoitiin ja sitä yritettiin turhaan ratkaista bisneslogiikan tasolla.

Tässä vaiheessa refaktaroin myös muut skriptit ottamaan huomioon STO:n, mutta myös jossain määrin käyttämään samoja funktioita ja rakenteita. Skriptien parissa oli johdon asettamien tiukkojen aikataulujen takia työskennellyt parhaimmillaan 3–4 ohjelmistokehittäjää, joilla kaikilla oli hiukan erilainen tyyli skriptien kirjoittamisen suhteen. Siksi olikin hyvä yhdenmukaistaa skriptejä samaan muuttujat-apufunktiot-pääfunktio-skriptivaiheet malliin, jotta ne kaikki olisivat samalla tavalla yhtä helposti luettavissa uusille käyttäjille ja tarjoaisivat samalla jotain mallia omien skriptien kirjoittamisen suhteen.

4.2.4 Kolmas refaktorointi – moduulin noodien käynnistäminen ilman moduulin käynnistämistä

Kolmas refaktorointi koski moduulin käynnistämistä. Alun perin kaikki skriptejä kirjoittaneet ohjelmistokehittäjät olivat lähteneet siitä, että moduulin noodit käynnistetään samalla, kun moduuli käynnistetään Running-tilaan. Tämä ei olisi kuitenkaan tarpeellista, koska tarvittavat moottorit voitaisiin käynnistää erikseen skriptien avulla. Moduulin käynnistäminen säätöprosessin ajaksi voisi olla jopa vaarallista, koska se saattaisi säädön aikana reagoida muualta tulleisiin käskyihin. Esimerkiksi jos laatikko menisi vahingossa kiinni, antaisi se robotille käskyn mennä lukemaan laatikossa mahdollisesti olevan koeputkitarjottimen ja sen telineiden viivakoodit. Jos STO ei jostain syystä toimisi, voisi moduulin sisälle jostain syystä kurottautunut säätöskriptien ajaja jäädä tällöin puristuksiin robotin ja moduulin seinän väliin.

Tämä ratkaistiin helposti lisäämällä skriptiin erillinen käynnistyskomento tarvittaville noodeille. Lisäksi käyttäjää ohjeistettiin käynnistämään moduuli Ready for startup -tilaan Running-tilan sijasta. Tämäkin on lopulta vain tilapäisratkaisu, koska tulevaisuudessa IMT:tä olisi tarkoitus käyttää erillisessä huoltomoodissa (service mode) sen sijaan, että moduuleja käynnistettäisiin tai pysäytettäisiin tiettyyn tilaan ennen säätötoimenpiteiden aloittamista.

4.2.5 Neljäs refaktorointi – skriptien korjaaminen pilotteja varten

Tähän asti skriptejä oli kirjoitettu ja testattu pitkälti prototyypimoduuleilla. Kun ensimmäinen pilottilaite saapui ja skriptejä testattiin niillä hieman perusteellisemmin, kävi ilmi, että skripteissä oli vielä muutamia puutteita ja ongelmia:

- Jotkut skriptit oli rakennettu prototyyppien tietokannan perusteella: jos tietokanta-arvot muuttuisivat, eivät skriptit enää toimisi tarkoitetulla tavalla.

- Tietokannassa oli väärät poikkeama-arvot koeputkitelineille ja ”tarjottimille” (tray): nämä arvot oli asetettu siten, että ne toimivat vain prototyypeissä.

- Z-arvo koeputkien poimimiselle puuttui: vain yläarvo tallennettiin, mutta se ei riittäisi, koska putken poimimiskohtaa ei vielä määritelty ohjelmistollisesti.

- Z-akseli oli 30 mm alempana prototyypeissä: tämä tarkoitti sitä, että jotkin kovakoodatut oletusarvot jäisivät piloteissa 30 mm ylemmäksi kuin mitä oli tarkoitus.

Laatikkojen säätöskriptissä laskettiin kalibrointityökalun paikka tietokanta-arvon perusteella. Jos tietokannassa olisi jostain syystä väärä arvo, esimerkiksi virheen takia, robotti hakeutuisi aivan muualle, mikä saattaisi hämmentää käyttäjää. Tämän ratkaisin siten, että mittasin robotin kotianturipositioiden perusteella suurpiirteisen XY-sijainnin säätötyökalun paikalle, jonka arvon kovakoodasin skriptiin.

IOS-moduulin kiekkojen säätäminen perustui myös tietokanta-arvoille jopa siinä määrin, että skriptin saattoi ajaa läpi onnistuneesti vain kerran, mikäli tietokannasta löytyvät arvot olivat tietynlaiset. Arvojen muututtua skripti ei enää toimisi halutulla tavalla, vaan jos tietokannassa olisi jo oikeat arvot tai jos käyttäjä olisi vahingossa päivittänyt sinne vääränlaiset arvot, kiekot eivät alustuisi halutulla

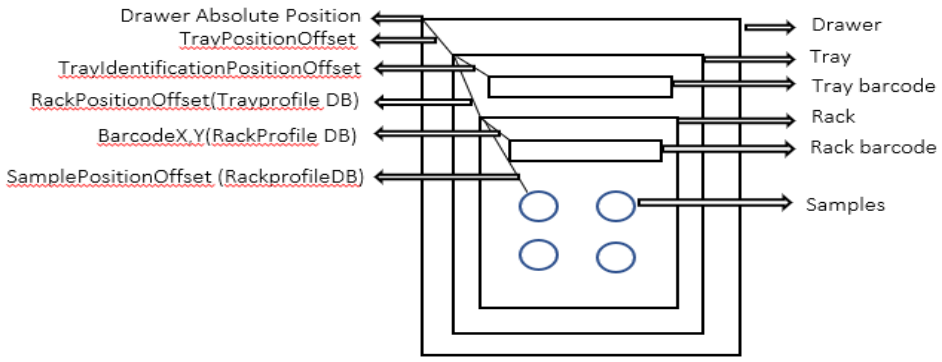
tavalla skriptin ajon jälkeen. Skripti ei myöskään ottanut kotipositioon ajamista huomioon, ellei alustusskriptiä ollut ajettu ennen säätämistä. Siksi lisäsinkin tähän skriptiin erikseen kotipositioon ajamisen sekä kaavan poikkeama-arvon laskemiseksi:

```
var newAdjustmentValue = 360 - ((diskPosition - homeValue + 360) %
360) + 90;
```

Esimerkkikoodi 7. IOS-moduulin kiekkojen laskentakaava.

Kyseisessä koodinpätkässä käyttäjän asettamasta arvosta vähennetään kiekon arvo kotipositiossa ja lisätään siihen 360, jotta saadaan varmasti positiivinen arvo. Tämän jälkeen arvo jaetaan modulo-operaattorilla 360, jolloin jakojäännöksestä tulee kiekon uusi säätöarvo. Myöhemmin kävi ilmi, että lähdekoodin toimintatavan takia arvoon pitää lisätä vielä 90 IOS-moduulin kiekkojen ollessa kyseessä.

Väärät tietokanta-arvot puolestaan aiheuttivat sen, että mentäessä poimimaan koeputkea se nostettaisiin aivan väärästä kohtaa poikkeama-arvojen laskutavan takia: tietokannassa olleet offset-arvot laskettaisiin yhteen, ja tästä muodostuisi paikka, josta poimijarobotti yrittäisi poimia koeputkea tai johon se yrittäisi laittaa sellaista. Tämä oli merkitsevää lähinnä laatikkojen säätämisen kannalta, koska poiminta-arvo saatiin ynnäämällä TrayPositionOffset-, RackPositionOffset- ja SamplePositionOffset-arvot, jotka määrittyivät sen perusteella, mitä koeputkitelinettä, -tarjotinta ja -paikkaa käytettiin (kuva 16). Arvot toimivat kyllä prototyypeissä, koska ne oli ”puukotettu” niitä varten oikeaan muotoon, joten tietokanta-arvot oli korjattava pilotteja varten kullekin tarjotin- ja telintyypille siten, että ne perustuivat teknisiin piirustuksiin.

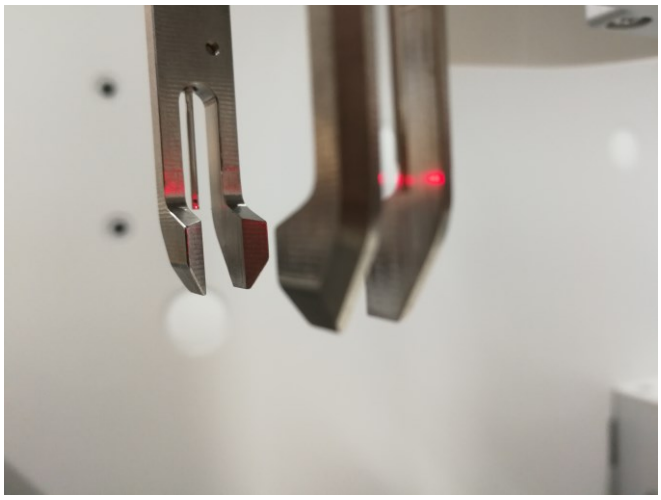


Kuva 16. Poikkeama-arvojen määrittyminen laatikoissa.

Viimeisimpänä piti korjata skriptien Z-arvot joidenkin kovakoodattujen arvojen osalta. Protolaboratorioon saapuneissa IOS:n pilottimalleissa Z-akseli oli 30 milliä korkeammalla kuin prototyypeissä, joiden pohjalta skriptit oli tehty. Tämän takia ajettaessa Z-akselin suuntaisesti lähelle kalibrointityökalua, jotta käyttäjä voisi tarkistaa ja tarvittaessa korjata robotin XY-paikkaa, robotti jäi liian ylös, jolloin oikean XY-paikan varmistaminen muuttui paljon vaikeammaksi. Tämä oli helppo korjata kyseistä arvoa vaihtamalla, koska pilottilaitteet ovat ominaisuuksiltaan mahdollisimman lähellä loppukäyttäjän käyttämiä laitteita, joten arvojen saattoi olettaa olevan pysyviä tästä lähin. Jos näin ei olisi, niin arvon voisi helposti korjata skriptissä.

Toinen Z-arvoon liittynyt ongelma oli poimimiskorkeuden puuttuminen. Oikeiden tietokanta-arvojen perusteella koeputken poimimispaikaksi telineistä tulisi telineen korkeuden (42) ja tarjottimen pohjan paksuuden (6,7) perusteella 48,2, mikä olisi siis telineen yläosa ja siten aivan liian matalalla. Lisäksi robotin sensori oli 10,5 millimetrin päässä leukojen kärjestä, joten jos tietokannassa ei olisi rajoituksia Z-arvon minimille, menisivät kärjet tällaisessa tapauksessa telineen lävitse noin 10 millimetrin verran. Lähdekoodissa ei ole tällä hetkellä määritelty putken poimintakorkeutta, ja kyseisen ominaisuuden implementoiva ”tarina” (story) olisi edessä vasta kuukausien päästä, joten ainoa ratkaisu tähän oli lisätä tilapäinen poikkeama-arvo poimintakorkeudelle, joka oli noin 10 mm.

Robotin sensorin aiheuttama 10 millimetrin heitto voitiin puolestaan vähentää suoraan kalibrointityökalun korkeudesta, jota käytettiin laatikkojen koordinaattien säätämiseen tarkoitetussa skriptissä. Access-paikkojen säätämisessä vastaavaksi poikkeama-arvoksi tuli testauksen tuloksena sen sijaan -13, koska siinä käytettiin vain Z-akselin huippuarvoa, joten putki piti tosiasiassa poimia hieman alempaa. Putki piti myös syöttää tarpeeksi pohjaan kuljettimissa, koska muuten riskinä oli etenkin luotettavuusajojen (reliability run) aikana, että putki ”kiipeäisi” ulos kuljettimesta tai telinepaikastaan, jos se poimittaisiin toistuvien järjestelmään syöttöjen ja sieltä pois ottamisien aikana joka kerta liian ylhäältä.



Kuva 17. Poimijarobotin sensorin sijainti.

4.2.6 Viides refaktorointi – apufunktioiden järjestely

Koska skriptit alkoivat olla pitkiä jo pelkkien apufunktioiden määrän myötä, ne oli hyvä sijoittaa johonkin toiseen tiedostoon sieltä ladattavaksi, jotta varsinaiset säätöskriptitiedostot pysyisivät mahdollisimman luettavina eikä samoja apufunktioita tarvitsisi kopioida joka skriptiin erikseen. Tätä varten loinkin erillisen HelperFunctions.csx-apufunktiotiedoston HelperScripts-kansioon varsinaisten säätöskriptien käytettäväksi. Tähän kansioon voisi myöhemmin sijoittaa myös muita skriptejä, jotka olisivat muokattavissa ja ladattavissa erikseen, kunhan IMT:n GUI:ta kehitettäisiin tarpeeksi eteenpäin.

Aluksi ideana oli ladata kyseinen skripti suoraan käyttöön lähdekoodin puolella, koska kyseessä on yleisskripti, joka ladattaisiin oletusarvoisesti kaikkiin skripteihin, mutta tästä koituisi mahdollisesti vain muutamia ongelmia. Apufunktioiden lataaminen olisi täysin piilotettu loppukäyttäjältä, joten hän ei olisi välttämättä edes tietoinen apufunktioiden olemassaolosta. Käyttäjä voisi jatkossa myös vahingossa poistaa kyseisen skriptin, mutta ohjelma viittaisi edelleen koodissa samaan polkuun, jolloin skriptejä ei voitaisi ajaa siitä koituvan virheilmoituksen takia.

```
var scriptCode = $"#load \"/App/HelperScripts/HelperFunc-
tions.csx\"\n" + mainScript;
```

Esimerkkikoodi 8. Skripti ladataan lähdekoodin puolella yhdistämällä pääskriptin alkuun load-komento polkuineen.

Ainoa hyöty tästä olisi vain se, ettei apufunktioita tarvitsisi muistaa ladata erikseen skriptien alussa.

Loin myös uusia yhdistelmäkomentoja säätöskriptien käytettäväksi. Tällä tavalla esimerkiksi moottorien käynnistäminen ja kotipositioon ajaminen skriptissä typistyi huomattavasti helppolukuisempaan ja helpommin asetettavissa olevaan muotoon:

```
await StartMotorsInPositionMode(XMotor);
await HomeMotorNode(XMotor, homingMethod);
```

Esimerkkikoodi 9. Tässä esimerkissä käynnistetään X-akselin moottori ja ajetaan se kotipositioon, jonka komento tarvitsee vielä erillisen homingMethod parametrin, joka on eri robotin ja kiekkojen moottoreilla. Jos loputkin robotin ja IOS:n kiekkojen moottorit halutaan alustaa samalla tavalla, täytyy käyttäjän toistaa samat komennot skriptissään vielä kuusi kertaa.

```
await StartAndHomeMotors(nodes)
```

Esimerkkikoodi 10. Tässä käynnistys ja kotipositioon ajaminen on hoidettu yhdellä funktiokutsulla, jolle tarvittavat noodit voidaan syöttää listana. Varsinaiset komennot parametreineen on abstraktoitu toisessa skriptitiedostossa sijaitsevaan StartAndHomeMotors-funktioon.

Tässä vaiheessa lisäsin myös moottorien käynnistämisen ja kotipositioon ajamisen kaikkiin skripteihin oletusarvoisesti, koska ne täytyi kuitenkin tehdä skriptien alussa, eikä ollut mielekästä ajaa näitä komentoja erikseen erillisellä alustusskriptillä, mikä oli alun perin tarkoituksena. Erillinen alustusskripti menetti näin pitkälti alkuperäisen merkityksensä, mutta sitä saattoi silti edelleen käyttää testaamaan noodien alustusta.

4.3 AIM-moduulin kiekkojen säätäminen

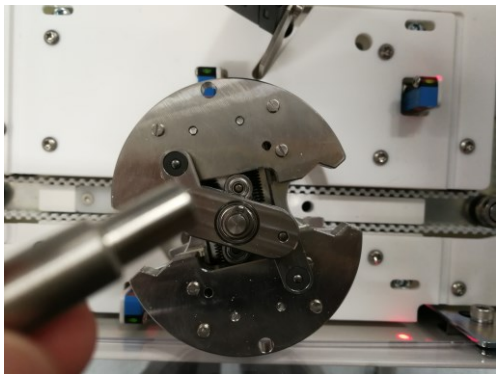
AIM-moduuli on huomattavasti yksinkertaisempi laite kuin IOS-moduuli. XYZR-akseleilla liikkuvaa robottia ei ole, ja kaikki säädettävät osat ovat kiekkoja. Suurin kysymysmerkki oli alussa se, mitä kaikkea pitäisi edes säätää. Näytteiden aspirointiin tarkoitettu imukiekko (aspiration disk) ja kiireelliset STAT-näytteet jonon ohitse reitittävä prioriteettikiekko (priority disk) tarvitsivat kyllä säätämistä, mutta moduulin takalinjalla toimiva kääntökiekko ja koeputket viivakoodin lukemisen ajaksi pysäyttävä mekanismi olivat vielä tässä vaiheessa kysymysmerkkejä. Toiveena oli, että ne toimisivat tarpeeksi hyvin oletusarvoilla, jottei niitä tarvitsisi säätää erikseen. Tämän takia päätettiin jäädä seuraamaan niiden toimintaa piloteissa ja ryhtyä kirjoittamaan säätöskriptiä niiden osalta vasta, jos sellainen osoittautuisi tarpeelliseksi.

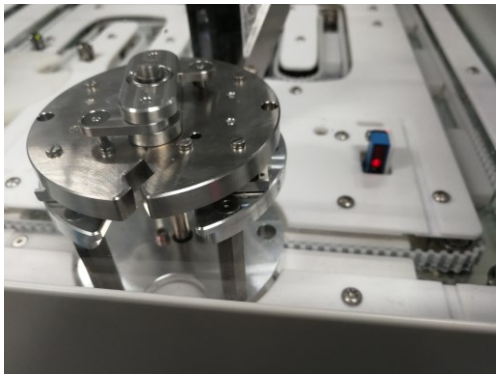
4.3.1 Imukiekon säätäminen

AIM-moduulin säätöskripti tehtiin vasta IOS-skriptien valmistumisen jälkeen, joten skriptaaminen oli verrattain helppo ja suoraviivainen prosessi IOS:n skriptausprosessiin verrattuna, etenkin kun säädettävänä oli vain kiekkoja, jotka säädettiin samaan tapaan samalla kaavalla kuin IOS-moduulin kiekotkin.

Imukiekon säätämisessä käytettiin moduulin mukana tullutta kalibrointityökalua, jonka avulla saatiin kiekon oikea paikka linjaamalla imukiekosta ja moduulin kuljetinhihnojen välistä löytyvät reiät kohdilleen ja lukitsemalla ne paikoilleen kalibrointityökalulla (kuva 18 ja 19). Kun imukiekko on saatu paikoilleen, tallennettaisiin sen paikka-arvo ja poikkeama-arvo laskettaisiin kotipaikka-arvon

perusteella samalla kaavalla kuin IOS-moduulin kiekotkin sillä erotuksella, ettei arvoon tarvitse lisätä 90 astetta.





Kuva 18 ja 19. AIM-moduulin kalibrointityökalu ja reiät imukiekossa ja kuljetinhihnojen välissä.

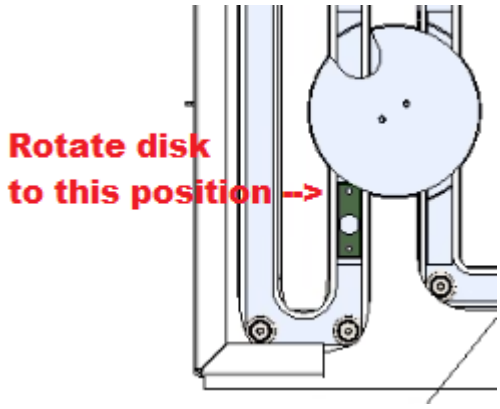
4.3.2 Prioriteettikiekon säätäminen

Prioriteettikiekko säädettiin samaan tapaan kuin muutkin kiekot, mutta siinä ei vain käytettäisi kalibrointityökalua. Käyttäjän pitäisi tietää, mihin kohtaan kiekko pitäisi kääntää, jotta saataisiin oikea säätöarvo. Asento, johon käyttäjän pitäisi liikuttaa kiekko, on tietokannassa nimellä ”IncomingFromInputBuffer”. Tämä ei kerro loppukäyttäjälle välttämättä yhtään mitään, joten käyttäjää kehotettiin hänelle osoitetussa tekstikehotteessa kääntämään kiekko ”vasempaan alakulmaan”.

```
await InvokeConfirmPromptAsync($"Säädä prioriteettikiekkoa pyörittä-
mällä se IncomingFromInputBuffer-asentoon (kiekon \"vasen alakulma\"
moduulin etupuolelta katsottaessa).");
```

Esimerkkikoodi 11. Esimerkki tekstikehotteella annettavasta ohjeistuksesta.

Jatkossa paras vaihtoehto olisi käyttää kuvaa käyttäjän ohjeistamiseksi, koska se olisi kaikkein yksiselitteisin ja selkein tapa (kuva 20).



Kuva 20. Esimerkki kuvan käyttämisestä ohjeistuksen tukena.

Prioriteettikiekko käännettäisiin säädön yhteydessä manuaalisesti ja silmämääräisesti IncomingFromInputBuffer-asentoon. Tarvittaessa tai halutessa siinä voitaisiin käyttää apuna tavallista viivoitinta, jotta kiekon hahlot saataisiin osumaan kohdilleen.

Mahdolliseksi ongelman lähteeksi prioriteettikiekossa osoittautui se, että vaikka kiekko säädetäänkin ”vasempaan alakulmaan”, niin alustuessaan oikein se osoittaa kuitenkin ”oikeaan alakulmaan” kohti imukiekkoa. Tämä saattaa aiheuttaa hämmennystä käyttäjässä, joka saattaa olettaa tehneensä virheen tai skriptin toimivan virheellisesti, minkä takia olisikin hyvä tähdentää tekstitse tai kuvallisesti se, minkälaiseen asentoon prioriteettikiekon pitäisi päätyä alustuessaan oikein.

4.3.3 AIM-säätöskriptin refaktoroinnit

AIM-moduulin säätöskriptin kirjoittaminen aloitettiin noin 6 viikkoa IOS-moduulin skriptien tekemisen aloittamisesta, joten useimmilta refaktoroinneilta säästyttiin kokemuksen myötä. Lisäksi moduuli oli huomattavasti yksinkertaisempi kuin IOS-moduuli eikä monimutkaisista XYZR-liikkeistä tarvinnut huolehtia.

Moduulin säätäminen ilman sen käynnistämistä sekä STO-signaalin huomioon ottaminen olivat merkittävimpiä seikkoja, joiden myötä skriptin rakennetta päädyttiin organisoimaan uudestaan. Tämä oli korjattu nopeasti siten, että noodit

käynnistettiin skriptin kautta moduulin alussa ja aina tarvittaessa STO-signaalin aktivoituessa.

Skripti oli myös rakennettu aluksi yhden ison funktion varaan, jonka sisällä säätöprosessi tapahtui, mikä teki skriptistä ”kankeahkon” ja vaikeaselkoisen, etenkin kun molemmat kiekot säädettiin hieman eri tavalla ja ne ohjeistivat käyttäjää eri kehotteilla. Lopullisessa versiossa säätöprosessi oli pilkottu kahteen erilliseen säätöfunktioon, joka käytti hyväksi HelperScript.csx-tiedostoon sijoitettuja apufunktioita, jolloin säätöskriptistä tuli varsin ytimekkään ja selkeän oloinen.

5 Skriptiprosessin ongelmakohtia

5.1 Ilmenneitä ongelmia

Skriptejä jouduttiin muokkaamaan erinäisistä syistä useaan otteeseen, mikä viivästytti skriptausprosessia melkoisesti, etenkin kun skriptejä piti vielä monesti testata ja hioa ajaen niitä fyysisillä moduuleilla. Osa ongelmista olisi ollut vältettävissä, kun taas osa johtui ulkoisista syistä eikä niille voitu oikein mitään. Erityisesti seuraavat seikat nousivat esille skriptausprosessin aikana ongelmien juurisyinä:

- 3–4 ohjelmistokehittäjän osallistuminen liian kiireellisesti aikataulutettuun kehitystyöhön
- mekaaniset ongelmat laitteiden kanssa
- ohjelmistolliset ongelmat laitteiden kanssa
- AIM- ja IOS-pilottien myöhästyminen ja alkuvaikeudet niiden konfiguroimisessa
- iterointiprosessin hitaus.

5.2 Ongelmien syitä, seurauksia ja ratkaisuehdotuksia

5.2.1 Liian kunnianhimoinen aikataulutus

Kuten sanottu, kaikki skriptit olisi pitänyt saada valmiiksi jo 6 viikon aikana. Tällä tavoiteltiin sitä, että moduulien säätö voitaisiin jättää laitteita valmistavan alihankkijan työntekijöille, jotta ohjelmistokehittäjät ja V&V-insinöörit voisivat keskittyä jatkossa muuhun R&D-työhön moduulien rutiininomaisen säätämisen sijaan. Tässä oltiin kuitenkin aivan liian optimistisia oikeastaan kaiken suhteen.

Kiireellisen aikataulutuksen takia skriptien tekemiseen resursoitiin 3–4 ohjelmistokehittäjää. Tämän seurauksena skriptit kirjoitettiin 3–4 eri tavalla. Töitä tehtiin rinnakkain, joten joitakin asioita tehtiin turhaan useaan otteeseen, kuten esimerkiksi robotin XYZ-tunnistus, joka oli aluksi käytössä kahtena erilaisena funktiona access-paikkojen ja laatikkojen säätöskripteissä. Toisaalta huomattiin myös, että toisten tekemiä funktioita ja komentoja saattoi toisinaan käyttää omassakin skriptissä, jolloin säästyi ylimääräiseltä työltä. Lisäksi koin itse aloittelevana ohjelmistokehittäjänä hyödylliseksi muiden kokeneempien ohjelmistokehittäjien rinnalla työskentelemisen.

Kirjavan ja kiireisen työn tuloksena skriptejä jouduttiin korjaamaan, refaktoroimaan ja yhdenmukaistamaan myöhemmin. Jos ne olisivat olleet 1–2 ohjelmistokehittäjän käsialaa, ne olisivat voineet olla alusta lähtien paljon yhdenmukaisempia ja siten paljon helpommin työstettävissä jatkossa, kun joillakin ohjelmoijilla olisi ollut jo syvempää tietoa skriptausprosessista ja siinä käytettävistä komennoista. Esimerkiksi access-paikkojen ja laatikkojen säätöön liittyvät skriptit hyödynsivät samoja funktioita ja toiminnallisuuksia, kuten myös IOS-moduulin ja AIM-moduulin kiekkojen säätöskriptit. Lisäksi olisi voinut olla eduksi, jos jollakin olisi ollut selkeä ”omistajuus” (ownership) kaikista skripteistä heti alusta lähtien, ja olisi sitä myöten keskittynyt kehittämään pelkästään niitä sekä IMT-työkalua. Hieman rauhallisemmin edettäessä olisi todennäköisesti tullut paljon parempaa jälkeä ja joiltakin refaktoroinneilta olisi voitu välttyä.

5.2.2 Mekaaniset ongelmat

Säätöprosessien aikana poimijarobotti vääntyi kelvottomaksi kahteen otteeseen: kerran alussa sattuneen lapsuksen takia ja toisen kerran epähuomiossa ”multitaskauksen” yhteydessä, jolloin robotti pääsi ajamaan kotipositiossa suoraan X-akselin suuntaisesti ollessaan syvällä laatikossa, jolloin se osui laatikon metallireunaan. Vaikka molemmilla kerroilla vika saatiin korjattua melko nopeasti, ensimmäisellä kerralla meni verrattain paljon aikaa sen tajuamiseen, että heitot robotin leukojen XY-arvoissa johtuivat vääntyneestä, mutta pikakorjauksena suoristetusta, robotista itsestään eikä esimerkiksi skriptikoodista. Lisäksi muutamissa yhteyksissä jokin osa ei ole toiminut halutulla tavalla, kuten esimerkiksi putkentunnistinsensori ei ole virhekytkennän takia toiminut lainkaan, tai moduulien välillä on ollut mekaanisia eroja, kuten prototyyppien ja pilottien Z-akselin sijainti.

Joskus saattoi käydä niin, että mekaniikka- tai sähkötiimi vaihtoi moduulista jonkin osan toiseen, minkä jälkeen skriptit toimivat eri tavalla kuin aikaisemmin. Tällaisista muutoksista ei aina tieto kulkenut eteenpäin, mutta nyttemmin moduulien tilaa ja vikoja seurataan tarkemmin ja paljon järjestelmällisemmin aamukokouksien yhteydessä.

Kaiken kaikkiaan mekaaniset ongelmat ovat melko tavanomaisia, ja niiden tunnistamiseen auttaa loppujen lopuksi vain kokemus laitteista ja säätöprosessista sekä laitteiden ja ohjelmakoodin välisestä vuorovaikutuksesta. Aloittelevana ohjelmistokehittäjänä oma reaktioni vain oli, että vika lienee taas minun koodissani, ja asiaan oli vaikea saada varmistusta, koska kehitin jotain sellaista, minkä parissa kukaan muu ei ollut tekemisissä.

5.2.3 Ongelmat IMT:n kanssa

Saman ohjelmistokehitysprojektin parissa työskentelee yhteensä noin 20 ohjelmistokehittäjää kolmessa eri tiimissä. Tämän takia saattaa käydä niin, että ohjelmisto ja skriptit saattavat käyttäytyä yhtäkkiä eri tavalla kuin aikaisemmin

lähdekoodiin tehtyjen muutosten takia. Alussa monen ohjelmistokehittäjän työskentely aiheutti sen, että samoihin komentoihin tehtyjen muutosten myötä, esimerkiksi kokonaislukutyypin (int) vaihtaminen merkkijonotyypiksi (string), omat skriptit eivät enää toimineetkaan, jos niihin ei ollut tehty samoja muutoksia. Skriptit olivat erityisen alttiita tälle, koska skriptejä itsessään ei testattu mitenkään, jolloin muutoksia tehnyt ohjelmistokehittäjä tuli toisinaan korjanneeksi vain IMT-solutionissa olleet yksikkötestit, jolloin skriptit itsessään eivät enää toimisi muutettujen komentojen osalta.

Lisäksi IMT:ssä on ScriptInterfaceEvaluator-luokka, jonka tarkoituksena on tarkastaa skriptit virheiden varalta ennen skriptien ajamista. Tästä on ollut kuitenkin vain lähinnä päänvaivaa, koska ScriptInterfaceEvaluator testasi skriptikomennot omasta luokastaan löytyviä funktioita vasten, jolloin tietyt toimenpiteet, kuten CancellationToken-rakenne, try/catch-virheenkäsittely ja JSON-deserialisointi kävivät skripteissä mahdottomiksi operaatioiksi, koska evaluaattori tulkitsi tällaiset rakenteet virheiksi eikä näin ollen antanut edes ladata skriptiä ajettavaksi. Jos skriptievaluaattorin kommentoi pois käytöstä, niin se antoi ladata tällaisia rakenteita sisältävät skriptit, mutta IMT antoi edelleen samat virheilmoitukset oikeiden skriptausvirheiden osalta virheellistä skriptiä ajettaessa. Näin ollen skriptievaluaattori vaikutti pikemminkin haittaavan kehitystyötä.

Skriptievaluaattorin tarpeellisuudesta käytiin aikanaan keskustelua, ja se päätettiin lopulta jättää koodiin, koska V&V-insinöörit voisivat alkaa jossain vaiheessa tekemään sen pohjalta RF-testejä. Näin ei kuitenkaan ole viimeisempien tietojen mukaan vielä tehty, joten koko evaluaattorin voisi ottaa kokonaan pois lähdekoodista kehitystyötä häiritsemästä ja pohtia tarpeen vaatiessa muita tapoja skriptien testaamiseksi.

5.2.4 Ohjelmistovirheet sulautetussa järjestelmässä

Erityisen paljon ongelmia tuotti firmware-tason ohjelmistovirheet sulatetun järjestelmän lähdekoodissa. Kyseinen virhe aiheutti sen, että kun kiekkoihin tai R-akselin moottoriin katkaistiin virta ja ne liikkuivat, joko itsestään (”hytkyminen”)

tai käyttäjän suorittaman toimenpiteen takia (kiekkojen liikuttaminen säätövaiheessa), niin laitettaessa moottorit takaisin päälle ja annettaessa liikkumiskäsky, käsky ei mennyt ikinä perille, vaan moottori jäi vikatilaan. Tällöin se piti käynnistää vielä uudemman kerran erikseen.

Kiekkojen osalta ongelma saatiin kierrettyä organisoimalla skripti siten, ettei säädettäessä kiekkoja tarvinnut liikuttaa skriptikomennoin eikä niihin tarvinnut katkaista erikseen virtaa, jolloin niitä olisi voinut liikuttaa manuaalisesti. Esimerkiksi aikaisemmin IOS-moduulin kiekot säädettiin siten, että moduuli oli käynnissä, jolloin kiekkoihin oli katkaistava virta. Virhe ilmeni useimmiten, kun kiekkoja yritti taas käynnistää, mikäli yritti jatkaa moduulin käyttöä normaalisti. Kun myöhemmin säätöprosessia muutettiin siten, että moduuleja ei enää käynnistetty kään, olivat kiekot jo valmiiksi ilman virtaa, jolloin ne pystyi kääntämään manuaalisesti haluttuun asentoon ilman, että virhe ilmenisi.

Ohjelmistovirhe vaikutti erityisen vaikeasti R-akselin toimintaa, koska R-akseli toisinaan liikahti liikaa ”hytkyessään” robotin liikkuessa XYZ-akseleita pitkin, jolloin se joutui edellä mainittuun vikatilaan, mutta toisinaan vikatilaa ei ilmennyt, koska R-akseli ei ollut liikahtanut tarpeeksi. Tämän takia ongelman juurisyyn selvittäminen lykkääntyi pitkäksi aikaa, koska ilmeisesti prototyypeissä R-akseli ei hytkynyt jostain syystä liikaa, vaikka niissäkin ilmeni toisinaan sama virhe. Pilottien kohdalla virhe ilmeni jo niin usein, ettei skriptejä voinut enää ajaa luotettavasti alusta loppuun, etenkään neljän säädettävän laatikon osalta. Tilapäiseksi ratkaisuksi muodostui se, että R-akselin moottori käynnistettiin ja virta siihen katkaistiin kahteen otteeseen, mikä vaikutti tilapäisratkaisuna toimivan joten kuten, muttei kuitenkaan täysin johdonmukaisesti joka kerta.

Ongelmaksi paljastui lopulta firmware-tasolla tehtävä paikkavertailu, jossa erillisen käynnistämiskäskyn yhteydessä vertailtiin moottorin positiota käynnistämistä edeltäneeseen positioon. Jos moottorin positio oli eri, kuten R-akselin moottorin positio saattoi hytkynnän takia olla, joutui moottori vikatilaan.

Tällaiset ohjelmistovirheet eivät olleet korjattavissa skriptiä tai edes bisneslogiikkaa muuttamalla, vaan niiden syitä piti hakea sulautetulta tasolta lähtien, mihin oma osaamiseni ei yksinkertaisesti riittänyt, koska en tuntenut sulautettua lähdekoodia ollenkaan, vaan osasin etsiä vikaa vain lähinnä skriptejä ajamalla sekä bisneslogiikkatason lokitiedostoja lukemalla. Tällöin ainoaksi ratkaisuksi muodostui tilapäisratkaisujen (ns. workaround) tekeminen, mikä ei kuitenkaan olisi pidemmän päälle kestävä ratkaisu.

5.2.5 Ohjelmistovirheet skriptikoodissa

Toinen ohjelmistovirhe liittyi myös robotin R-akseliin, mutta tällä kertaa syy piili tekemässäni skriptikomennossa, joka liittyi nopeuksien asettamiseen. Annettaessa R-akselin moottorin säätönopeudeksi 20, moottori alkoi käyttäytyä arveluttavasti, jos sille antoi skriptissä varmuuden vuoksi käynnistämiskäskyn ”turhaan”, jos käyttäjä ei ollutkaan liikuttanut R-akselia. Ilmeisesti R-akselin moottorilla oli firmware-tasolla erikoiskäsittely liian alhaisille nopeuksille, ja jos tuo nopeus oli 20 tai alle, moottori menisi vikatilaan. Esimerkiksi 21 oli jo täysin käypä arvo R-akselin moottorillekin. Tästä virheestä teki erityisen vaikean se, että R-akselin suhteen oli jo ollut muitakin ongelmia sekä se, että saman säätönopeuden asettaminen ei tuntunut aiheuttavan virheitä XYZ-akselien moottorien kohdalla.

Tämä ohjelmistovirhe ratkaistiin siten, ettei R-akselin moottorille enää asetettu säätönopeutta, koska firmware toimi kuitenkin halutulla tavalla eikä R-akseli tarvinnut muutenkaan alhaisempaa säätönopeutta, koska sen kohdalla ei ollut samanlaista törmäysriskiä kuin XYZ-akselien kohdalla. Lisäksi XYZ-akselien moottorien nopeuksia nostettiin yhdellä yksiköllä arvoon 21 varmuuden vuoksi.

5.2.6 Alkuvaikeudet pilottimoduulien kanssa

Skriptit oli vielä testattava piloteilla, mikä ei ollut mahdollista, ennen kuin pilotit saapuisivat. Tämän jälkeen pilotit piti vielä konfiguroida erikseen, minkä jälkeen säätöprosessin saattoi vasta aloittaa. Kaiken kaikkiaan pilottimoduulien pystytys

ja niiden konfigurointi söi ensimmäisten moduulien osalta aikaa muutaman viikon verran, mutta jatkossa moduulien pystyttäminen ja konfiguroiminen oli jo paljon nopeampaa ja rutiininomaisempaa touhua.

En itse hoitanut pilottimoduulien pystytystä tai konfigurointia, vaan keskityin yksinomaan IMT:llä tehtäviin säätöihin. Pystyin kyllä käyttämään prototyyppejä, mutta skriptejä oli tärkeää päästä testaamaan myös pilottimoduulien kohdalla. AIM-moduulin kohdalla pilotti ja prototyyppi erosivat toisistaan jopa niin merkittävästi, ettei ollut järkevää aloittaa skriptien kirjoittamista prototyyppimoduuleilla.

Pilottimoduuleista paljastui vielä puutteita, joiden takia jotkin ominaisuudet eivät toimineet. STO-signaali toimi kyllä AIM-moduulissa, mutta IOS-moduulissa se ei toiminut, koska STO-signaalin laukaisevat magneetit olivat sisällä sellaisessa palkissa, jonka materiaali häiritsi signaalia siinä määrin, ettei se toiminut lainkaan. Tämä oli huono juttu STO-signaalin testaamisen kannalta, mutta onneksi sitä pystyi simuloimaan irrottamalla oikeasta piirilevystä ”hyppyliittimen”, mikä laukaisisi STO-signaalin.

Viimeisimpänä ongelmana on ollut muutamien pilottien sensori- ja kuituvahvistinongelmat, jotka todennäköisesti johtuvat väärinkytketyistä kaapeleista. Tämänkin ongelman toteamisessa meni ainakin yksi työpäivä, kun luulin vian olevan vain omissa skripteissäni, vaikka poimijarobotin sensori ei vain tunnistanut kalibrointityökalun reunoja, ja robotin leukojen välissä oleva sensori sekä kuituvahvistimen näyttö olivat pimeinä. Tähän mennessä olin säätänyt ja testannut skriptejä jo kahdella pilotilla onnistuneesti, enkä osannut odottaa tällaisia virheitä tulevilta pilottimoduuleilta.

Kuljetusvaikeuksille ja alkukankeudelle ei voi mitään, mutta sensorien tilojen lukeminen (XYZ-minimi- ja -maksimisensorit, laatikot) olisi kyllä jo testattavissa pienen skriptin avulla. Testiskriptien luonti on periaatteessa tuotannon tehtävä, mutta jatkossa voisi luoda pienimuotoisen testiskriptin moduulien vastaanottoa varten. Tällöin moduulia voisi testata samalla tavalla kuin se tullaan

todennäköisesti tuotannon puolellakin testaamaan. Samalla voisi tutustua paremmin sensorien lukemiseen, mitä ei ole nykyisissä säätöskripteissä tarvinnut tehdä.

5.2.7 Ongelmat iterointiprosessissa

Säätöskriptien luonteen takia niiden toiminnallisuutta ei voi oikein testata esimerkiksi RF-testeillä, koska säätötilannetta, kuten kalibrointityökalun paikkaa ja muita olosuhteita ei ole kovin mielekästä mallintaa testimielessä. RF:n Docker-konteilla saattoi korkeintaan testata, antaisiko IMT virheilmoituksen, jos skriptissä olisi virhe. Tällöin ainoaksi vaihtoehdoksi skriptien toiminnallisuuden testaamiseksi jäi niiden testaaminen itse moduuleilla.

Skriptausprojektin alussa IMT:tä ei ollut vielä konfiguroitu toimimaan moduuleilla, joten testasin omaa ohjelmistokoodiani ja skriptejä lokaalisti IDE:n välityksellä ajamalla IMT:tä Visual Studion kautta. Tässä oli etuna se, että saatoin tehdä nopeasti muutoksia pelkkiin skripteihin tai IMT:n puolen koodiin, mikäli jompikumpi niistä muuttuisi, mutta LAM:n koodi pysyisi samana. Saatoin olla IDE:n välityksellä suoraan yhteydessä testausmoduulissa olevaan LAM-tietokoneeseen ajaen selaimessa localhostin kautta IMT-konttia, jonka kautta skriptiä saattoi ajaa moduulissa, mikä nopeutti iterointia, kun Docker-kuvia (image) ei tarvinnut luoda erikseen LAM-konttia varten.

Ensimmäisen skriptin valmistumisen jälkeen noin 2–3 viikon kuluttua projektin aloittamisesta kuitenkin konfiguroin IMT:n toimimaan moduuleissa omassa Docker-kontissaan, ja aloin ajamaan skriptejä ensisijaisesti tätä kautta, koska oli myös tärkeää testata GUI-puolta siten, miten IMT:tä käytettäisiin normaalistikin. Jatkossa jouduin myös muuttamaan LAM-puolen koodia jonkin verran uusien komentojen luonnin myötä, joten jouduin joka tapauksessa luomaan ja lataamaan Docker-kuvia testausmoduulille useamman kerran. Lisäksi lokaali testaus kävi oikeastaan mahdottomaksi siinä vaiheessa, kun skripteihin piti ladata erikseen useampia aputiedostoja, jotta tarvittavat noodit, yleisfunktiot ja moduulikohtaiset funktiot toimisivat, koska näiden tiedostojen oli sijaittava Docker-

kontissa toimiakseen. Docker-kontti toimii omana eristettynä ympäristönään (Poulton 2020: 75) eikä sille voi näin ollen antaa latauspoluksi toisella koneella lokaalisti sijaitsevaa tiedostoa SSH-yhteyden avulla ilman merkittävää lisäsäätöä ja -vaivaa.

```
#load "/App/HelperScripts/IOSNodes.csx"
#load "/App/HelperScripts/HelperFunctions.csx"
#load "/App/HelperScripts/IOSHelperFunctions.csx"
```

Esimerkkikoodi 12. Access-paikkojen säätöskriptin kolme ensimmäistä riviä viimeisimmässä skriptiversiossa. Load-komennolla ladataan Docker-kontin sisälle kopioidut apuskriptiedostot pääskriptin käytettäväksi. Jos näihin skripteihin ja IMT:hen tulee muutoksia, pitää niitä varten luoda uusi IMT Docker-kuva ja ladata ne moduulille. Jos haluaa kokeilla pelkkiä skriptimuutoksia, niin muutetut skriptit voi kopioida testimielessä nopeasti polkuun SSH-yhteyden avulla.

Muuttujien ja apufunktioiden sijoittaminen pois yksittäisistä säätöskripteistä tekee niistä toki paljon luettavampia ja selkeämpiä, mutta virheenetsintä muuttuu vaikeammaksi, kun skriptien rakenne on hajautettu useampaan skriptitiedostoon. Tällöin mahdolliset virheet käyvät myös useimmiten ilmi vasta silloin, kun skriptiä yrittää ajaa IMT:llä, jolloin käyttäjä saa virheilmoituksen. Tämän jälkeen skripti on korjattava ja uusi Docker-kuva on luotava sekä ladattava moduulille tai korjattu skripti on kopioitava Docker-kontin polkuun, jotta sen voisi testata virheiden varalta.

5.3 Tilanne nyt ja tulevaisuuden kehityssuuntia

Tätä kirjoittaessani skriptejä ei ole vieläkään luovutettu alihankkijalle, ja näyttää siltä, että tulemme säätämään ainakin joitakin pilotteja ja moduuleja omassa toimipisteessämme. Mielestäni tämä on hyvä asia, koska näin saamme enemmän kokemusta säätöprosessista ja skriptien toiminnasta, minkä myötä voimme toivon mukaan kehittää säätöskriptit sellaiselle tasolle, että ne ottavat kaikki mahdolliset tilanteet huomioon tavalla tai toisella. Tämä olisi hyödyllistä niin nykyisten kuin tulevienkin säätöskriptien kannalta.

Jatkoa ajatellen ScriptEvaluator olisi hyvä poistaa, jotta skripteihin saataisiin lisää try/catch-tyyppistä virheenhallintaa. Tällä hetkellä sitä ei oikeastaan ole tai

sitä on yritetty tehdä pelkillä tekstikehotteilla, jotka käyttäjä voi vahingossa ohittaa, jolloin skripti jatkuisi normaalisti ja tietokantaan päätyisi pahimmassa tapauksessa vääriä arvoja.

Tekstikehotteiden oheen olisi myös hyvä saada tarvittaessa kuvatuki, jotta vaikeammin selitettävissä olevat asiat avautuisivat paremmin käyttäjille. Tällä hetkellä kyseiselle ominaisuudelle ei ainakaan vaikuta olevan mitään omaa tarinaa työlistalla (backlog).

Tuotannon kanssa käytyjen keskustelujen perusteella olisi myös hyvä varmistua, että IMT:n rajapinnasta löytyvät kaikki tarpeelliset komennot, joita heidän tarvitsee käyttää moduulien testauksessa. Tällä hetkellä komentoja on lisätty alun jälkeen lähinnä sitä mukaa, kuin niitä on tarvittu skriptejä varten.

6 Yhteenvetoa

Tässä projektissa luodut skriptit ovat käyttövalmiita, mutta vielä ei voida täysin varmasti sanoa, kuinka luotettavasti skriptit tulevat toimimaan asiakaskäytössä eri moduulien kanssa. Skriptejä on muutettu, korjattu ja refaktoroitu niin moneen otteeseen eri syistä, että nyt kun ne alkavat vakiintua, niin niitä olisi hyvä vain ajaa muutamilla moduuleilla sen toteamiseksi, että ne ajavat asiansa nykyisellään, ja toimivat mahdollisesti siinäkin tapauksessa, että moduulin arvot poikkeavat tavallista enemmän, mutta kuitenkin sallituissa rajoissa, siitä, mitä on pidettävä hyväksyttävänä.

Suuri kysymysmerkki on myös se, kuinka skriptejä ennestään tuntemattomat käyttäjät tulevat käyttäytymään säätöprosessin aikana ja miten tämä vaikuttaa skriptien ajoon. Aletaanko tekstikehotteiden yli ”hyppimään” siten, että joitakin kehotteissa mainittuja toimenpiteitä jätetään tekemättä tai ne tehdään eri järjestyksessä kuin on tarkoitus? Miten skriptejä ja niiden toimintaperiaatteita tuntematon käyttäjä menettelee virhetilanteessa? Minkälaisia virhetilanteita hän saattaa edes kohdata, ja miten hän tulee niihin reagoimaan? Tällaisiin tilanteisiin on

vaikea eläytyä maallikon tavoin, kun on työskennellyt skriptien parissa neljän kuukauden verran.

IMT-projekti ja sen skriptien kehittäminen tulee kestämään vielä joitakin vuosia uusien moduulien kehittämisen ja kasvavien vaatimusten myötä. Siksi olisikin hyvä olla valmis ja vakaa pohja, jolle on turvallista rakentaa projektin tulevaisuutta.

Henkilökohtaisella tasolla voin todeta projektin olleen hyvin opettavainen. Opin projektin aikana tuntemaan moduulien toimintaa paremmin sekä käyttämään monipuolisesti tarvittavia työkaluja ja teknologioita. Organisatoriselta puolelta taas sain hyvän käsityksen siitä, kuinka monella eri sidosryhmällä oli intressejä IMT:n suhteen ja miten nämä intressit poikkesivat toisistaan tai oletetuista intresseistä. Hienoin asia on ehkä kuitenkin ollut se, että on saanut oman pienen vastuualueen tai ”omistajuuden” IMT:stä ja sen skripteistä, mikä on motivoinut kehittämään ja ideoimaan molempia eteenpäin sekä pitänyt oman työn mielekkäänä. Näistä lähtökohdista onkin hyvä jatkaa projektia sekä oman ammatillisen osaamisen kehittämistä.

Lähteet

Acharyya, Swagata. 2024. Cleaner Code: Tackling Arrowhead Anti-Pattern. Verkkoaineisto. Medium. Comviva MFS Engineering Tech Blog. <https://medium.com/@dfs.techblog/cleaner-code-tackling-arrowhead-anti-pattern-238d5ce91390>. 23.1.2024. Luettu 10.4.2024.

Brown, William J.; Malveau, Raphael C.; McCormick III, Hays W.; Mowbray, Thomas J. 1998. AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. Wiley.

Cimen, Baris. 2021. 10 Most Common Anti-Patterns Every Software Engineer Must Avoid! Verkkoaineisto. Medium. <https://bariscimen.medium.com/10-most-common-anti-patterns-every-software-engineer-must-avoid-182091438c2b>. 16.5.2021. Luettu 10.4.2024.

Fowler, Martin & Beck, Kent. 2018. Refactoring: Improving the Design of Existing Code. Pearson.

Main, Jeff. 2020. Designing for User Error. <https://www.inclusionhub.com/articles/designing-for-user-error>. 10.11.2020. Luettu 10.4.2024.

Martin, Robert C. 2009. Clean Code: A Handbook of Agile Software Craftsmanship. Prentice Hall.

Michaelis, Mark. 2016. Essential .NET - C# Scripting. Verkkoaineisto. MSDN Magazine. <https://learn.microsoft.com/en-us/archive/msdn-magazine/2016/january/essential-net-csharp-scripting>. Tammikuu 2016. Luettu 10.4.2024.

Poulton, Nigel. 2020. Docker Deep Dive: Zero to Docker in a Single Book. E-kirja. Amazon.com.

Vogel, Peter. 2021. Making Your Life Easier with C# Scripting. Verkkoaineisto. <https://visualstudiomagazine.com/articles/2021/06/14/csharp-scripting.aspx>. 14.6.2021. Luettu 30.3.2024.

IOSRobotAccessPositionAdjustmentScript.csx – ensimmäinen versio

```
using System;
using System.Threading;
using System.Threading.Tasks;

async Task<int> ZAxisTubeDetectionHeight(int ZAxisMotor, int ZTubeDetec-
tHeight, int travelHeight)
{
    tubeDetectorSensorState = await GetSensorState(ZAxisMotor, 6);
    int tiltSwitch = await GetSensorState(ZAxisMotor, 7);
    //decrement until jig is detected
    while (tubeDetectorSensorState == 0)
    {
        tubeDetectorSensorState = await GetSensorState(ZAxisMotor, 6);
        await MotorRotateToPosition(ZAxisMotor, ZTubeDetectHeight);
        --ZTubeDetectHeight;
        tiltSwitch = await GetSensorState(ZAxisMotor, 7);
        if (tiltSwitch == 1){
            await MotorRotateToPosition(ZAxisMotor, travelHeight);
            await InvokeConfirmPromptAsync($"Tilt switch activated!");
        }
    }
    return ZTubeDetectHeight;
}

async Task<int> SetAxisBorder (int axisNode, int ZAxisMotor, int motor-
Value, bool firstBorder)
{
    tubeDetectorSensorState = await GetSensorState(ZAxisMotor, 6);
    //decrement/increment until jig is not detected anymore
    while (tubeDetectorSensorState == 0)
    {
        await MotorRotateToPosition(axisNode, motorValue);
        if (firstBorder){
            --motorValue;
        }else{
            ++motorValue;
        }
        tubeDetectorSensorState = await GetSensorState(ZAxisMotor, 6);
    }
    return motorValue;
}

async Task<int> GetBorders(int axisMotor, int ZAxisMotor )
{
```

```
    //Set starting positions
    int initialPosition = await GetMotorPositionAsInt(axisMotor);
    int getBorder1 = initialPosition + 10;
    int getBorder2 = initialPosition - 10;

    //Set 1st border
    bool firstBorder = true;
    await MotorRotateToPosition(axisMotor, getBorder1);
    tubeDetectorSensorState = await GetSensorState(ZAxisMotor, 6);
    getBorder1 = await SetAxisBorder(axisMotor, ZAxisMotor, getBorder1,
firstBorder);

    //Set 2nd border
    firstBorder = false;
    await MotorRotateToPosition(axisMotor, getBorder2);
    tubeDetectorSensorState = await GetSensorState(ZAxisMotor, 6);
    getBorder2 = await SetAxisBorder(axisMotor, ZAxisMotor, getBorder2,
firstBorder);

    int finalValue = (getBorder1 + getBorder2) / 2;

    await MotorRotateToPosition(axisMotor, finalValue);

    return finalValue;
}

//Set nodes and other values
int XAxisMotor = 71;
int YAxisMotor = 72;
int ZAxisMotor = 73; //also tube detector sensor (switchnum 6) and tilt
switch (switchnum 7)
int RAxisAndGripper = 100;

int outputAccessDiskMotor = 65;
int inputAccessDiskMotor = 60;

int inputRobotCoordinate = 4;
int outputRobotCoordinate = 5;

int tubeDetectorSensorState;
var travelHeight = Convert.ToInt32(await ReadModuleAdjustment("Specimen-
RobotTravelHeight"));

//Set motor speeds
// await SetMotorAdjustmentSpeed(XAxisMotor);
// await SetMotorAdjustmentSpeed(YAxisMotor);
// await SetMotorAdjustmentSpeed(ZAxisMotor);
```

```
async Task SetAccessPosition(int accessPosition, int robotCoordinate)
{
//Make sure access disk is in the right position
await InvokeConfirmPromptAsync($"Rotating disk, make sure input and out-
put access disks are clear.");
await MotorRotateToPosition(accessPosition, 90);

//Get values from database
int ZAxisInitialValue = travelHeight;
int XAxisInitialValue = await GetAdjustmentCoordinate(robotCoordinate,
0);
int YAxisInitialValue = await GetAdjustmentCoordinate(robotCoordinate,
1);
int RAxisInitialValue = await GetAdjustmentCoordinate(robotCoordinate,
3);

await InvokeConfirmPromptAsync($"Insert adjustment jig to output access
position and confirm action.");
await InvokeConfirmPromptAsync($"NB!!! Motor is going to drive  output
access position, so take cover! Confirm action.");

//Go to access output position at travel height
await MotorRotateToPosition(ZAxisMotor, ZAxisInitialValue);
await MotorRotateToPosition(XAxisMotor, XAxisInitialValue);
await MotorRotateToPosition(YAxisMotor, YAxisInitialValue);
await MotorRotateToPosition(RAxisAndGripper, RAxisInitialValue);

//Open gripper
await ControlGripper(RAxisAndGripper, false);

//Set Z closer to jig
int loweredZ = 210;
await MotorRotateToPosition(ZAxisMotor, loweredZ);

//Release X/Y/R motors
await ReleaseMotor(XAxisMotor);
await ReleaseMotor(YAxisMotor);
await InvokeConfirmPromptAsync($"Make sure gripper is on top of the jig
and move it on X and Y axes manually if neccessary. Confirm action.");

//Set X/Y motors on
await StartMotorInPositionMode(XAxisMotor);
await StartMotorInPositionMode(YAxisMotor);

//Set Z
await InvokeConfirmPromptAsync($"NB!!! Gripper is going to drive down to
sensor, so take cover! Confirm action.");
```

```
int finalZ = (int)(await ZAxisTubeDetectionHeight(ZAxisMotor, loweredZ,
travelHeight));

//Move R-axis 90 degrees to move on X-axis
int RAxisXPos = 90; //98 is better?
await MotorRotateToPosition(RAxisAndGripper, RAxisXPos);

//Set X borders
int finalX = await GetBorders(XAxisMotor, ZAxisMotor);

//Move R-axis 0 degrees to move on Y-axis
int RAxisYPos = 0;
await MotorRotateToPosition(RAxisAndGripper, RAxisYPos);

//Set Y
int finalY = await GetBorders(YAxisMotor, ZAxisMotor);

//Save X/Y/Z/R to database
await InvokeConfirmPromptAsync($"Final values: X({finalX}), Y({finalY}),
Z({finalZ}), R({RAxisInitialValue})");
await UpdateModuleCoordinateAdjustment(robotCoordinate, finalX, finalY,
finalZ, RAxisInitialValue);

if(robotCoordinate == 4){
    await InvokeConfirmPromptAsync($"InputRobotCoordinate axis positions
saved!");
}else if (robotCoordinate == 5){
    await InvokeConfirmPromptAsync($"OutputRobotCoordinate axis positions
saved!");
}

//Set disk back to accepting carriers position
await InvokeConfirmPromptAsync($"Disk resetting back to former position,
please take the jig away.");
await MotorRotateToPosition(accessPosition, 0);
}

await SetAccessPosition(outputAccessDiskMotor, outputRobotCoordinate);
await SetAccessPosition(inputAccessDiskMotor, inputRobotCoordinate);

//Restore motor speeds
// await SetMotorSpeedToConfiguration(XAxisMotor, "SpecimenRobotXAxisMo-
tor");
// await SetMotorSpeedToConfiguration(YAxisMotor, "SpecimenRobotYAxisMo-
tor");
// await SetMotorSpeedToConfiguration(ZAxisMotor, "SpecimenRobotZAxisMo-
tor");
```

```
await InvokeConfirmPromptAsync($"Script finished.");
```

IOSRobotAccessPositionAdjustmentScript.csx – viimeisin versio

```
#load "/App/HelperScripts/IOSNodes.csx"
#load "/App/HelperScripts/HelperFunctions.csx"
#load "/App/HelperScripts/IOSHelperFunctions.csx"


//Set constants and variables - aseta vakiot ja muuttujat
const string inputRobotCoordinate = "InputAccessDiskRobotCoordinate";
const string outputRobotCoordinate = "OutputAccessDiskRobotCoordinate";
const int loweredZ = 180;
const int driveZToDepth = 140;
const int pickupOffset = 13;  //TODO: get rid of pickupOffset once pickup
height is calculated properly elsewhere


//Helper functions - apufunktiot
//Detect Y axis center with one border - selvitä Y-akselin keskipiste
yhdellä reunalla
async Task<float> DetectYAxisCenter(float initialValue)
{
    int offset = 20;
    int yBorderOffset = 4;
    var yBorder = initialValue;
    var RAxisYPos = 0;

    await MoveR(RAxisYPos);
    await MoveY(initialValue - offset);
    MoveY(initialValue);
    yBorder = await DetectTubeDetectorSensorStateChange(IOSSpecimenRobo-
tYAxisMotor);
    float approximateAxisCenter = yBorder + yBorderOffset;
    await MoveY(approximateAxisCenter);

    await InvokeConfirmPromptAsync($"Make sure Y axis is centered to the
calibration tool and press \"Confirm\".\n\nVarmista, että Y-akselin
keskipiste on  kalibrointityökalun keskellä ja paina \"Confirm\"!");
    await StartMotorsInPositionMode(IOSRobotMotorNodes);
    float axisCenter = await GetMotorPosition(IOSSpecimenRobotYAxisMo-
tor);
    return axisCenter;
}

//Main function - pääfunktio
async Task SetAccessPosition(string robotCoordinate, string accessPosi-
tionName)
{
```

```
    //Get values from database - hae arvot tietokannasta
    float XAxisInitialValue = await ReadModuleCoordinateAdjustment(ro-
botCoordinate, "X");
    float YAxisInitialValue = await ReadModuleCoordinateAdjustment(ro-
botCoordinate, "Y");
    float ZAxisInitialValue = await ReadModuleCoordinateAdjustment(ro-
botCoordinate, "Z");
    float RAxisInitialValue = await ReadModuleCoordinateAdjustment(ro-
botCoordinate, "Phi");
    await InvokeConfirmPromptAsync($"Initial database values for {access-
PositionName}: X({XAxisInitialValue}), Y({YAxisInitialValue}), Z({ZAxi-
sInitialValue}), R({RAxisInitialValue}).\n\nAlkuperäiset tietokanta-arvot
{accessPositionName}-positiolle: X({XAxisInitialValue}), Y({YAxi-
sInitialValue}), Z({ZAxisInitialValue}), R({RAxisInitialValue}).");

    //Go to access output position - mene access-positioon
    await MoveXYZR(XAxisInitialValue, YAxisInitialValue, loweredZ, 0);

    //Make sure the gripper is on top of the calibration tool - varmista,
että tarttuja on kalibrointityökalun päällä
    await InvokeConfirmPromptAsync($"Make sure gripper is on top of the
calibration tool, and open the IOS doors to move the robot on X and Y
axes manually if neccessary.\nRemember to close the IOS doors before
pressing \"Confirm\".\n\nVarmista, että tarttuja on kalibrointityökalun
päällä ja avaa IOS-moduulin ovet siirtääksesi robottia X- ja Y-akseleilla
manuaalisesti tarvittaessa.\nMuista sulkea IOS-moduulin ovet ennen kuin
painat \"Confirm\".");
    await StartMotorsInPositionMode(IOSRobotMotorNodes);

    //Set Z - aseta Z
    await InvokeConfirmPromptAsync($"Warning!!! Robot is going to drive
down along the Z axis!\n\nVaroitus!!! Robotti ajaa alas Z-akselia
pitkin!");
    float finalZ = await DetectZHeight(driveZToDepth);
    await InvokeConfirmPromptAsync($"{accessPositionName} Z-axis top
point: {finalZ}\n\n{accessPositionName} Z-akselin yläpiste: {finalZ}");
    finalZ -= pickupOffset; //TODO: get rid of pickupOffset once pickup
height is calculated properly elsewhere

    //Set X - aseta X
    float finalX = await DetectAxisCenter(IOSSpecimenRobotXAxisMotor,
XAxisInitialValue);
    await InvokeConfirmPromptAsync($"{accessPositionName} X-axis center
point: {finalX}\n\n{accessPositionName} X-akselin keskipiste: {finalX}");

    //Set Y - aseta Y
    float finalY = await DetectYAxisCenter(YAxisInitialValue);
```

```
    await InvokeConfirmPromptAsync($"{accessPositionName} Y-axis center point: {finalY}\n\n{accessPositionName} Y-akselin keskipiste: {finalY}");

    //Raise the robot to travel height korkeudelle - nosta robotti travel height-korkeudelle
    await MoveZ(IOSSpecimenRobotTravelHeight);

    //Save values to DB - tallenna arvot tietokantaan
    await InvokeConfirmPromptAsync($"Final {accessPositionName} values: X({finalX}), Y({finalY}), Z({finalZ}), R({RAxisInitialValue}), do you want to save these values to the database?\n\nLopulliset {accessPosition-Name} arvot: X({finalX}), Y({finalY}), Z({finalZ}), R({RAxi-sInitialValue}), haluatko tallentaa nämä arvot tietokantaan?");
    await UpdateModuleCoordinateAdjustment(robotCoordinate, finalX, fi-nalY, finalZ, RAxisInitialValue);
    await InvokeConfirmPromptAsync($"Values saved to the database.\n\nAr-vot tallennettu tietokantaan.");
}


//Script steps - skriptin vaiheet
await InvokeConfirmPromptAsync($"Start/restart the IOS module into Ready for startup state. Press \"Confirm\" when done.\n\nKäynnistä/uudelleen-käynnistä IOS-moduuli käynnistysvalmiuteen (Ready for startup -tila). Paina \"Confirm\", kun olet valmis.");

//Start and home motor nodes - käynnistä ja aja moottorit kotipositioon
await StartAndHomeMotors(IOSRobotMotorNodes);
await TestGripper();

//Insert the calibration tool(s) - aseta kalibrointityökalu(t)
await InvokeConfirmPromptAsync($"Open the IOS module doors and rotate output access disk (left) and/or input access disk (right) to access po-sition.\n\nAvaa IOS-moduulin ovet ja kierrä output access disk (vasen) ja/tai input access disk (oikea) access-positioon.");
await InvokeConfirmPromptAsync($"Insert calibration tool(s) to access po-sition(s).\n\nAseta kalibrointityökalu(t) access-positioihin.");
await InvokeConfirmPromptAsync($"Close the IOS module doors.\n\nSulje IOS-moduulin ovet.");
await StartMotorsInPositionMode(IOSRobotMotorNodes);

//Adjust access positions (you can comment out one of the SetAccessPosi-tion function calls with "//" if you only need to adjust one position)
//Säädä access-positiot (voit kommentoida toisen SetAccessPosition-funk-tion pois "//"-merkeillä, jos haluat säätää vain yhden position)
await SetAccessPosition(outputRobotCoordinate, "outputAccessDisk");
await SetAccessPosition(inputRobotCoordinate, "inputAccessDisk");
```

```
await InvokeConfirmPromptAsync($"Open the IOS doors and remove the cali-
bration tool(s) from access positions.\n\nAvaa IOS-moduulin ovet ja
poista kalibrointityökalu(t) access-positioista.");
await InvokeConfirmPromptAsync($"Close the IOS module doors and start the
module for the new values to initialize.\n\nSulje IOS-moduulin ovet ja
käynnistä moduuli uusien arvojen alustamiseksi.");
```

AIMAdjustmentScript.csx – ensimmäinen versio

```
//Set variables
const int aspirationAccessDiskMotor = 62;
const int priorityDiskMotor = 65;
const int diskMotorHomingMethod = 33;
const string aspirationDiskAdjustmentKey = "AspirationAccessDiskMotor";
const string priorityDiskAdjustmentKey = "PriorityDiskMotor";

//Script functions
async Task AdjustDisk(int diskMotor, string adjustmentKey)
{
    //Home motor
    await InvokeConfirmPromptAsync($"Homing {adjustmentKey}.");
    await HomeMotor(diskMotor, diskMotorHomingMethod);
    var homeValue = await GetMotorPosition(diskMotor);
    await InvokeConfirmPromptAsync($"Current positional
{nameof(homeValue)}: {homeValue}.");

    //Read disk adjustment
    var adjustmentValue = await ReadDiskAdjustment(adjustmentKey);
    await InvokeConfirmPromptAsync($"Current {adjustmentKey} offset
value: {adjustmentValue}.");

    //Release and restart motor for adjustment
    await ReleaseMotor(diskMotor);

    switch (diskMotor)
    {
        case aspirationAccessDiskMotor:
            await InvokeConfirmPromptAsync($"Please turn {adjustmentKey}
gripper to access position and insert calibration tool to the gripper
hole so that it connects with the hole in the conveyor belt and confirm
action.");
            break;
        case priorityDiskMotor:
            await InvokeConfirmPromptAsync($"Adjust {adjustmentKey} by
rotating the disk to IncomingFromInputBuffer position and confirm ac-
tion.");
            break;
    }

    //Start motor
    await StartMotorInPositionMode(diskMotor);

    // Read disk position
    var diskPosition = await GetMotorPosition(diskMotor);
```

```
    //Update adjustments
    await InvokeConfirmPromptAsync($"{adjustmentKey} position: {diskPosi-
tion}.");
    adjustmentValue = -(diskPosition - homeValue);

    await UpdateDiskAdjustment(adjustmentKey, adjustmentValue);
    await InvokeConfirmPromptAsync($"New {adjustmentKey} offset value:
{adjustmentValue} saved to the database. \nPlease restart the AIM module
or fast stop and start for changes to take effect.");

    //Remove the calibration tool
    if (diskMotor == aspirationAccessDiskMotor)
    {
        await InvokeConfirmPromptAsync($"Please remove the calibration
tool from the {adjustmentKey} gripper hole.");
    }
}

//Script steps - comment/uncomment to call desired function(s)
await InvokeConfirmPromptAsync($"Please start the AIM module into running
state and stop it orderly so that it ends up in stopped state.");

await AdjustDisk(aspirationAccessDiskMotor, aspirationDiskAdjustmentKey);
await AdjustDisk(priorityDiskMotor, priorityDiskAdjustmentKey);
```

AIMAdjustmentScript.csx – viimeisin versio

```
#load "/App/HelperScripts/AIMNodes.csx"
#load "/App/HelperScripts/HelperFunctions.csx"


//Script functions - skriptin funktiot
//Adjust aspiration disk - säädä imukiekko
async Task AdjustAspirationDisk(float homeValue)
{
    await InvokeConfirmPromptAsync($"Turn aspiration access disk to ac-
cess position and insert calibration tool to the aspiration disk hole so
that it connects with the hole in the conveyor belt.\n\nKäännä imukiekko
access-asentoon ja aseta kalibrointityökalu imukiekon reikään niin, että
se yhdistyy kuljetushihnan reikään.");
    var aspirationDiskPosition = await GetMotorPosition(AIMAspirationAc-
cessDiskMotor);
    await CalculateAndUpdateOffset(aspirationDiskPosition, homeValue,
"AspirationAccessDiskMotor");
    await InvokeConfirmPromptAsync($"Remove the calibration tool from the
aspiration disk hole.\n\nPoista kalibrointityökalu imukiekon reiästä.");
}

//Adjust priority disk - säädä prioriteettikiekko
async Task AdjustPriorityDisk(float homeValue)
{
    await InvokeConfirmPromptAsync($"Adjust by rotating the priority disk
to IncomingFromInputBuffer position (\"lower left corner\" of the disk
when viewing the module from the front).\n\nSäädä prioriteettikiekkoa
pyörittämällä se IncomingFromInputBuffer-asentoon (kiekon \"vasen ala-
kulma\" moduulin etupuolelta katsottaessa).");
    var priorityDiskPosition = await GetMotorPosition(AIMPriorityDiskMo-
tor);
    await CalculateAndUpdateOffset(priorityDiskPosition, homeValue, "Pri-
orityDiskMotor");
}


//Script steps - skriptin vaiheet
await InvokeConfirmPromptAsync($"Start/restart the AIM module into Ready
for startup state. Press \"Confirm\" when done.\n\nKäynnistä/uudelleen-
käynnistä AIM-moduuli käynnistysvalmiuteen (Ready for startup -tila).
Paina \"Confirm\", kun olet valmis.");
await StartAndHomeMotors(AIMDiskMotors);

var aspirationDiskHomeValue = await GetMotorPosition(AIMAspirationAc-
cessDiskMotor);
```

```
var priorityDiskHomeValue = await GetMotorPosition(AIMPriorityDiskMotor);

await InvokeConfirmPromptAsync($"Remove the module cover from the AIM
module.\n\nPoista kansi AIM-moduulista.");

//Adjust disks (you can comment out/in the desired function(s)) - säädä
kiekot (voit kommentoida halutut funktiot pois/päälle)
await AdjustAspirationDisk(aspirationDiskHomeValue);
await AdjustPriorityDisk(priorityDiskHomeValue);

await InvokeConfirmPromptAsync($"Put the module cover back on and start
the AIM module. The disks should initialize correctly according to new
values.\n\nLaita kansi takaisin paikoilleen ja käynnistä AIM-moduuli.
Kiekkojen pitäisi asettua nyt uusien arvojen mukaisesti.");
```