SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Timo Syrjälä

HIL-testauslaitteiston automatisointi

Opinnäytetyö
Kevät 2022
SeAMK Tekniikka
Automaatiotekniikan tutkinto-ohjelma, ylempi AMK

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Tutkinto-ohjelma: Automaatiotekniikka, ylempi AMK

Tekijä: Timo Syrjälä

Työn nimi: HIL-testauslaitteiston automatisointi

Ohjaaja: Niko Ristimäki

Vuosi: 2022 Sivumäärä: 54 Liitteiden määrä: 2

Tämän opinnäytetyön tavoitteena oli tutkia HIL-testilaitteiston automatisoitua käyttöä. Opinnäytetyön toimeksiantaja on Epec Oy. Työssä suunniteltiin ja toteuteutettiin tarkoitusta varten tehdylle esimerkkisovellukselle automaattiset HIL-testit sekä toteutettiin näille jatkuvan integroinnin sekä toimituksen julkaisuputki. Toteutuksen päätarkoitus oli tutkia automaattisen testauksen mahdollisuuksia HIL-testauslaitteistoa käytettäessä. Tutkimuksessa pyrittiin käyttämään mahdollisuuksien mukaan avoimen lähdekoodin työkaluja. Lisäksi pyrittiin laatimaan toteutus niin, että se olisi mahdollisimman helposti siirrettävissä eri projekteihin.

Opinnäytteessä käsitellään erilaisia testien suunnittelutekniikoita sekä perusteet HIL-testauksesta, jatkuvasta integroinnista ja jatkuvasta toimituksesta. Työssä esitellään käytetyt työkalut, joita ovat muun muassa National Instrumentsin realiaikatestaustyökalut, Azure DevOps, CODESYS ja Python-ohjelmointikieli.

Opinnäytetyön tuloksena saatiin luotua Azure DevOps -julkaisuputki, jossa suoritetaan aiemmin suunnitellut automaattiset HIL-testit luodulle sovellukselle, ja julkaistaan niistä raportit DevOpsiin.

[1] Asiasanat: CI/CD, Julkaisuputki, Testauslaitteet, Testausmenetelmät

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Degree programme: Automation engineering

Author: Timo Syrjälä

Title of thesis: Automating HIL testing

Supervisor(s): Niko Ristimäki

Year: 2022 Number of pages: 54 Number of appendices: 2

The thesis was commissioned by Epec Oy. The purpose of the thesis was to study the automatic usage of HIL test equipment acquired to the company. A set of automated HIL-tests were designed and implemented with an example application and a CI/CD pipeline was then built for the execution of the tests.

The main goal was to investigate the possibilities to utilize automated testing when using HIL-testing equipment. Open-source tools were preferred in the study. In addition, the priority was that the implementation could be easily used also in other projects.

The thesis studied different test design techniques and the basics of HIL testing, continuous integration, and continuous delivery. The tools used for building the system, such as National Instrument's real time testing tools, Azure DevOps, CODESYS and Python programming language were also introduced in the thesis.

As the result of the thesis, Azure DevOps pipeline was created where the earlier designed automatic HIL-tests are executed with the created application and the results are published in the pipeline.

[1] Keywords: CI/CD, Pipeline, Testing equipment, Testing methods

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

CI	Continuous Integration – jatkuva integrointi
CD	Continuous Delivery – jatkuva toimitus
HIL-testaus	Hardware In the Loop, testaustapa jossa fyysinen laite on korvattu simuloidulla laitteella
Mustalaatikkotestaus	Määrittelyyn perustuva testaustekniikka
PXI	PCI eXtensions for Instrumentation, modulaarinen instrumentaatio-alusta

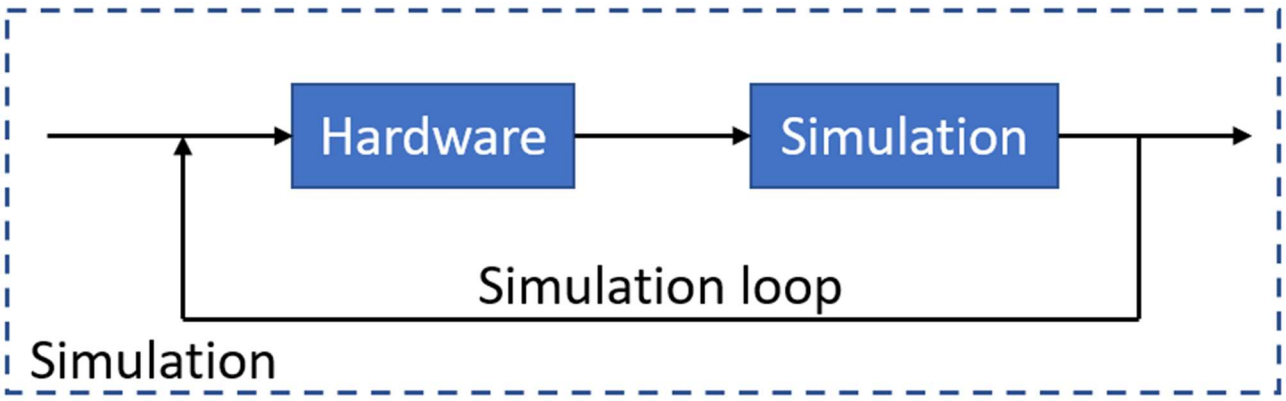
1 Johdanto

Tulevaisuuden maailmassa valmistajat haluavat edelleen leikata laitteiden ja järjestelmien kehittämiseen kuluvaa aikaa ollakseen kilpailukykyisiä markkinoilla. Tämä korostuu erityisesti itse laitteiden sekä niiden ohjauksen sähköistyessä. (Joshi 2020, 13.)

Laitteet ja järjestelmät ovat tulleet yhä monimutkaisemmiksi ja niiden ohjauksessa käytettyjen ohjainyksiköiden määrä on noussut. Esimerkiksi alemman keskitason autojen 30– 50 ohjausyksiköstä ollaan siirtymässä kalliimpien autojen 80 - 100 ohjausyksikköön per ajoneuvo. Myös ohjainyksiköissä olevan ohjelmiston määrä on lisääntynyt ja monimutkaistunut. Modernissa autossa on nykyään noin 100 miljoonaa koodiriviä. Tämä on lähes 6 kertaa enemmän kuin Boeing 787 Dreamliner -lentokoneessa. (Joshi 2020, 13.)

Noin 90 % ajoneuvoteollisuusalueen uusista innovaatioista koskee elektroniikkaa. Valmistajat haluavat lisätä ajoneuvoihin uusia ominaisuuksia, kasvattaa niiden autonomisuutta ja mahdollistaa paremmat yhteydet eri verkkoihin. Tämän kehityksen johdosta laitteiden monimutkaisuuden oletetaan vain kasvavan tulevaisuudessa. (Joshi 2020, 13.)

Tähän kehityksen tarpeeseen vastaa osaltaan HIL (Hardware In Loop) -simulaatio. HIL-simulaatiossa osa järjestelmästä on toteutettu oikeilla fyysisillä laitteille, kun taas loppuosa on toteutettu fyysisesti tai matemaattisesti simuloiduilla malleilla. Tämä toimintatapa antaa mahdollisuuden yhdistää simulaation nopeus sekä kokonainen laitteisto simuloimalla se. HIL-simulaatio antaa mahdollisuuden rakentaa testipenkkejä, joissa testit voidaan suorittaa automaattisesti, ja ne pystytään saamaan myös toistettaviksi.



Kuva 1. HIL-simulaatio (perustuu Joshi 2020, 14)

HIL-testauksella saavutettavia etuja ovat testaukseen kuluvan ajan pieneneminen, testauksen alkaminen mahdollisimman aikaisessa vaiheessa, testien toistettavuus ja variointimahdollisuus sekä testikattavuuden kasvattaminen. Testauksen käytettävä aika pienenee, koska HIL-testaus tarjoaa realistisen ympäristön jossa eri signaalien tyyppi ei eroa oikeasta laitteesta. Testit, jotka muuten jouduttaisiin suorittamaan oikealla laitteella, voidaan siirtää toteutettavaksi HIL-testissä.

Testaus voidaan aloittaa mahdollisimman aikaisin, vaikka kaikkia järjestelmän osia ei vielä olisi olemassa. Puuttuvat laitteet voidaan simuloida, ja tämä tarjoaa myös lisämahdollisuuksia tutkia kehitettävää järjestelmää. Testien toistettavuus paranee, koska ne voidaan toistaa samanlaisina joka kerta. Tämä luo myös mahdollisuuden automatisoituihin testeihin, joiden tulokset ovat vertailukelpoisia eri testiajoista. Erilaisia muuttujia voidaan myös lisätä testeihin, jolloin saadaan selville, miten järjestelmä toimii muuttuvissa tilanteissa. Testikattavuus paranee, koska testejä voidaan automatisoituina suorittaa enemmän. Myös testit voivat kattaa laajemmin erilaisia toimintaolosuhteita. Automatisointi vapauttaa testaajat manuaalisten testien suorittamisesta ja mahdollistaa keskittymisen tulosten analysointiin, uusien testien luomiseen sekä epäonnistuneiden testien korjaamiseen. (Joshi 2020, 21.)

1.1 Työn tausta

Työn taustana on aiemmin suoritettu uuden testerin hankinta. Johtuen ajan sekä henkilöresurssien puuttumisesta, laitteiston käyttöönottoa ja käyttöä ei ole päästy

suunnittelemaan ja toteuttamaan. Nykyinen käytössä oleva vanha testauslaitteisto ei enää vastaa uusien ohjausyksiköiden kehityksessä esiin nousevia vaatimuksia.

Uusi laitteisto mahdollistaa laajemman testausmenetelmien kehittämisen verratuna yrityksessä käytössä oleviin menetelmiin. Uudella laitteistolla on mahdollista rakentaa kokonaisia järjestelmiä, joissa kaikki toimilaitteet on simuloitu, ja testauksen kohteet eli ohjausyksiköt on kytketty laitteistoon.

Uutta laitteistoa halutaan käyttää laajemmin yrityksen sisällä kuin nykyistä testerilaitteistoa. Järjestelmän tarjoamia ominaisuuksia ja käyttöä on tarkoitus tutkia toteuttamalla yrityksessä olevan koulutusprojektin sisältö testilaitteistolla. Tästä työstä saatavilla tuloksilla sekä kehitettävällä prosessilla järjestelmä pystytään ottamaan käyttöön helpommin myös muissa testausprojekteissa. Toteutuksen yhteydessä on tarkoitus myös tutkia ja suunnitella testauksen automatisointia.

Työstä saatava kirjallinen materiaali sekä prosessin aikana opittava toimintatapa luovat pohjan soveltaa laitteistoa myös muille, tämänhetkisille ja tuleville laitteille.

1.2 Työn tavoite

Työssä on tarkoitus suunnitella ja toteuttaa testauslaitteiston käyttöönotto koskien ohjelmistoa. Työ ei ota kantaa mahdollisiin fyysisiin kytkentöihin tulevaisuutta varten, poislukien pakolliset kytkennät esimerkkijärjestelmää varten. Työ ei myöskään käsitä testijärjestelmän rakentamista muille kuin myöhemmissä luvuissa esiteltäville ohjainlaitteille.

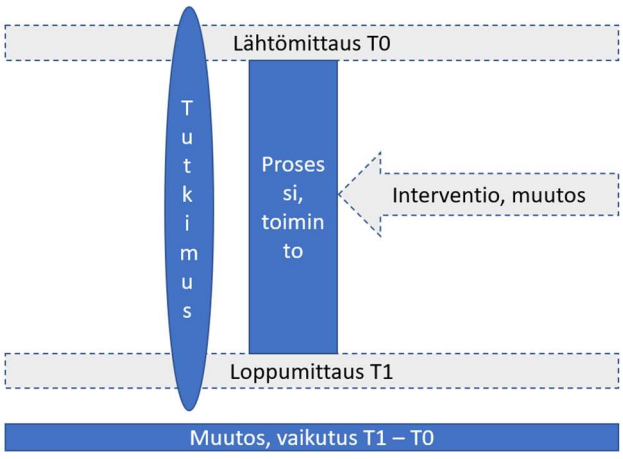
Työssä pyritään tutkimaan testauksen ja raportoinnin automatisointia yrityksessä jo olevilla työkaluilla sekä ohjelmistoilla. Yrityksessä on siis olemassa jo automatisointia mahdollistavia ohjelmistoja, mutta ne vaativat tutkimista ja sovitustyötä, että ne saadaan palvelemaan uutta laitteistoa.

1.3 Työn rakenne

Ensimmäisessä luvussa on esitetty tutkimusongelma sekä sen rajaus ja työn tavoite. Samassa luvussa on myös selitetty tutkimusongelma sekä yrityksen historia. Luvussa 2 esitellään työssä käytetty teoriapohja sekä käytettävä testauslaitteisto. Luvussa 3 on kerrottu työssä käytetyt työkalut sekä testattavan ohjausjärjestelmän muodostavat Epec-ohjausyksiköt. Neljännessä luvussa on esitelty automaattisten testien luonti järjestelmälle. Viimeisessä luvussa on yhteenveto työn tuloksista sekä pohdintaa niiden soveltamisesta. Lisäksi on esitelty tulevaisuuden parannuskohteita.

1.4 Tutkimusmenetelmä

Tämä opinnäytetyö suoritetaan kehittämistutkimuksena. Kehittämistutkimuksessa on taustalla ilmiö tai prosessi, jota halutaan ymmärtää sekä parantaa. Edellä mainittu kohde kehittämiselle voidaan nähdä ongelman muodossa joka, halutaan ratkaista altistamalla se ulkoiselle interventiolle. Kuviossa 2 on kuvattu kehittämistutkimuksen malli. Mallista voidaan havaita tutkimuksen ja intervention suhde haluttuun muutoskohteeseen. (Kananen 2012, 13.)



Kuva 2. Kehittämistutkimuksen malli (perustuu Kananen 2012, 45)

Kehittämistutkimuksella haetaan kirjallisen tuotoksen lisäksi myös käytännössä toimivia ratkaisuja. On vaarana, että tutkimusosa työstä voi jäädä kokonaan yrityksen päivittäisen normaalin kehittämistyön jalkoihin. (Kananen 2012, 43.)

Kun parannuksista kommunikoidaan laajemmalle yleisölle, kyseessä voidaan katsoa olevan tutkimusluonteinen työ. Tämä dokumentointi mahdollistaa työn esittämisen myös isommalle joukolle, mikä altistaa sen ulkopuoliselle keskustelulle ja arvinoinnille. (Kananen 2012, 44.)

Kehittämistutkimus ei käsitä ainoastaan ilmiön kuvailua, vaan sillä on tavoitteena löytää konkreettinen parannus asioihin. Pelkkä vaihtoehtojen löytäminen ei ole riittävää, koska kehitetyt vaihtoehdot täytyy myös testata ja siten todeta niiden sopivuus ja toimivuus ongelman ratkaisuun. (Kananen 2012, 44.)

Kehittämistutkimus sopii tähän työhön hyvin, koska työssä tutkitaan ja pyritään käyttöönottamaan uusi testilaitteisto sekä luomaan malli tai toimintapa sen käyttöön kohdeyrityksessä. Tutkimusmenetelminä käytetään kirjalliseen materiaaliin tutustumista, omatoimista selvitystyötä testilaitteistosta sekä keskusteluja eteen tulevista haasteista työn aikana. Teknisessä osassa keskitytään laitteiston käytön suunnitteluun ja sen toteustukseen.

1.5 Yritysesittely

Epec on vuonna 1978 Seinäjoelle perustettu yritys. Yrityksen perusti Veikko Rintamäki nimellä E-P Elektroniikka. Vuonna 1989 yrityksen nimi vaihtui nykyiseen muotoonsa, Epec Oy. Perustamisensa jälkeen yritys alkoi valmistaa erilaisia elektroniikkatuotteita, joita olivat esimerkiksi suuret ulkoilmanäytöt. Yritys teki myös elektroniikkasuunnittelua sekä myi tietokoneita. Vuonna 2004 Epecistä tuli osa Ponsse Oyj konsernia, metsäkoneyhtiön ostettua Epecin. (Epec, [viitattu 5.2.2021].)

Nykyään Epec toimii elektroniikka-alalla suunnitellen ja valmistaen liikkuvien työkoneiden koneenohjausjärjestelmiä sekä niihin tarvittavia ohjelmistoja. Yritys on laajentanut tarjoamaansa myös sähköisiin voimalinjoihin sekä kokonaisten järjestelmien suunnitteluun pelkkien ohjauskomponenttien sijaan. (Epec, [viitattu 5.2.2021].)

Epec suunnittelee ja valmistaa sulautettuja näyttöjä ja ohjausyksiköitä. Näytöissä ja ohjausyksiköissä käytetään standardisoitua CAN-väylää. Yhtiön tuotteet on suunniteltu haastaviin olosuhteisiin ja ne on valmistettu alumiinista ja muovista. Tuotteiden tärinän kestävyys on jopa 100 G:tä, ja vesi- sekä pölytiiveydeltään laitteet ovat IP67-luokkaisia. Koska tuotteet on suunniteltu haastaviin ajoneuvo-olosuhteisiin, tuotteet kestävät käyttöä laajalla käyttölämpötila-alueella. (Epec, [viitattu 5.2.2021].)

Epecin asiakaskunta koostuu emoyhtiön lisäksi muun muassa kotimaisista ja ulkomaisista maanrakennus-, maatalous- ja metsäkoneiden valmistajista. Esimerkkejä asiakkaista ovat Ponsse, Sandvik Mining and Construction ja Metso Minerals. (Epec, [viitattu 5.2.2021].)

Yhtiön pääkonttori sijaitsee Seinäjoella, missä on myös yrityksen tehdas. Tehtaassa valmistetaan kaikki yrityksen suunnittelemat tuotteet. Samoissa tiloissa sijaitsevat myös projektipalvelut, huolto, tuotekehitys ja asiakastuki. Yrityksellä on toimipisteet myös Tampereella, Turussa sekä Kiinan Shanghaissa. Shanghain toimipisteessä sijaitsee tekninen tuki lähinnä Aasian markkinoita varten. Tampereella on tuotekehitysyksikkö, joka sijaitsee samoissa tiloissa Ponssen tuotekehitysyksikön kanssa. Turussa on myös tuotekehitysyksikkö joka sijaitsee asiakkaan tiloissa. (Epec, [viitattu 5.2.2021].)

2 HIL- ja mustalaatikkotestaus

Tässä luvussa esitellään työssä käytetyt testien suunnittelu- ja testaustekniikat.

2.1 Mustalaatikkotestaus

Mustalaatikko (Black box) on määrittelyyn perustuvaa testausta. Tällaisessa testaustavassa ohjelmisto tai järjestelmä mielletään mustaksi laatikoksi, jossa on sisäänmenoja sekä ulostuloja, mutta testaajalla ei ole näkymää mitä ohjelmistossa tai järjestelmässä sisäisesti tapahtuu. Testaaja siis keskittyy siihen mitä järjestelmä tai ohjelmisto tekee, eikä siihen miten järjestelmä sen tekee. (Black, Van Veenendaal & Graham 2009, 79.)

Mustalaatikkotestausta voi olla toiminnallista (functional) tai ei-toiminnallista (non-functional). Toiminnallinen testaus tarkastelee mitä järjestelmä tekee ja ei-funktionaalinen testaus tarkastelee miten hyvin järjestelmä nuo asiat tekee. Ei-toiminnallisia eli laadullisia ominaisuuksia voivat olla esimerkiksi suorituskyky, käytettävyys ja ylläpidettävyys. (Black, Van Veenendaal & Graham 2009, 79.)

2.2 HIL-testaus

Sulautettuja järjestelmiä kehitettäessä tulee eteen tilanteita, joissa järjestelmän täydellinen testaus on mahdotonta, esimerkiksi sen fyysisen koon tai työturvallisuuden takia. Tällaisissa tapauksissa ”hardware-in-the-loop” (HIL) -simulaatio järjestelmälle ja ympäristölle mahdollistaa kehityksen aikaisen testaamisen ilman turvallisuuden heikkenemistä. (Pont& Short 2008, 2.)

2.3 HIL-testauksen historia

HIL-teknologia on ollut käytössä 1950-luvulta saakka puolustus- ja ilmailuteollisuudessa. Huolimatta teknologian korkeasta hinnasta tuolloin, puolustus- ja ilmailuteollisuus käytti HIL-teknologiaa hyväkseen pääosin hengenvaarallisten ja erittäin kalliiden prototyyppijärjestelmien takia. Autoteollisuus ei vielä tuohon aikaan ollut kiinnostunut HIL-teknologiasta sen hinnan ja ajoneuvoteollisuuden käyttämien yksinkertaisimpien

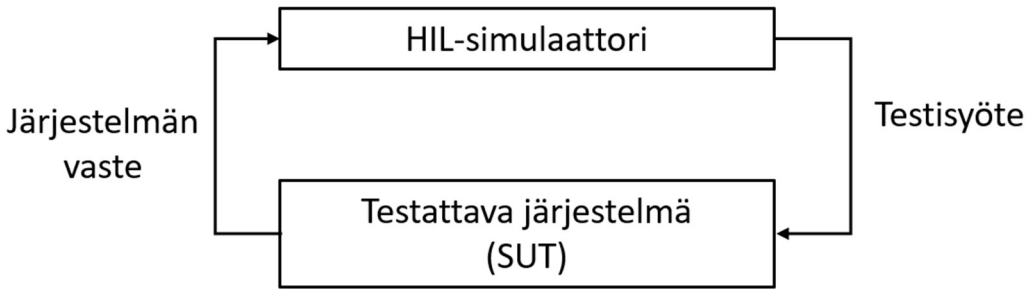
ohjausjärjestelmien takia. Ajan kuluessa myös ajoneuvoteollisuudessa alettiin lisätä monimutkaisempia sulautettuja järjestelmiä ajoneuvoihin. Tiukentuneet vaatimukset päästöissä lisäsivät myös erilaisen ohjausjärjestelmien tarvetta. Näistä vaatimuksista aiheutunut monimutkaisuuden lisääntyminen sysäsi myös ajonuvoteollisuuden tutkimaan parempia testausjärjestelmiä sekä tapoja. Tästä johtuen HIL-testaus on ollut käytössä 90-luvulta lähtien. (Allen ym. 2004, 1.)

Nykyisin HIL-testaus on käytössä seuraavilla aloilla:

- Toiminnalliseen turvallisuuteen liittyvä tuotekehitys, jossa testaus oikealla laitteistolla on monimutkaista ja kallista.

- Teollisuudenalat, joissa testaus voi vaurioittaa kalliita laitteistoja.

- Teollisuudenalat, joissa testaus oikeassa toimintaympäristössä on vaarallista (autonomiset ajoneuvot) tai mahdotonta (avaruusalukset). (Brayanov& Stoyanova 2019, 70.)



Kuva 3. HIL-testauksen periaate. (perustuu Joshi 2020, 17)

2.4 Avoimen ja suljetun silmukan HIL-testaus

HIL-testauksessa voidaan erottaa kaksi erilaista tapaa- avoimen ja suljetun silmukan testaus. Avoimen silmukan testauksessa HIL-simulaattorin luoma simulaatiodata ei ole riippuvainen testattavan järjestelmän tuottamasta edellisestä vasteesta. Testin vasteena muodostuva data vain tallennetaan myöhempää arviointia varten, sillä ei ole vaikutusta

luotavaan testisyötteeseen. Avoimen silmukan testauksessa ei tarvitse luoda testisyötteenä olevaa simulaatiodataa realiajassa. (Schlager, Obermaisser & Elmenreich 2007, 2.)

Suljetun silmukan testauksessa testattavan järjestelmän tuottama vaste vaikuttaa suoraan simulaatioon luotavaan vasteeseen. Tästä johtuen simulaation vaste on luotava realiajassa. HIL-simulaation käyttö parantaa testausta, koska siten voidaan simuloida laitteisto joka ei olisi saatavissa pelkässä ohjelmistolla toteutettavassa simulaatiossa. Tämän simulaation käyttö edellyttää sopivien rajapintojen löytämistä simulaattorin ja sulautetun järjestelmän välille. (Schlager ym. 2007, 2.)

Toisin kuin pelkkä ohjelmistolla tehtävä simulaatio, HIL-simulaatiossa käytetään oikeita fyysisiä ohjainlaitteita. Tästä johtuen HIL-simulaation käyttö vaatii myös oikean testausympäristön rakentamisen, millä voidaan jäljitellä laitteiden fyysistä ympäristöä. Myös osa ohjainlaitteista voidaan simuloida, jos halutaan testata vain tiettyä fyysistä ohjainlaitetta osana isompaa kokonaisuutta. (Schlager ym. 2007, 2.)

HIL-simulaatiolla saavutetaan muutamia etuja sulautettujen järjestelmien kehityksessä. Näitä ovat esimerkiksi:

- Prototyyppien aikainen testaus tulee mahdolliseksi simuloitujen ympäristöjen avulla.
- Simuloitu ympäristö voidaan luoda halutunlaiseksi.
- Testauksen valvonta on mahdollista, koska HIL-järjestelmällä voidaan saada näkyviin testausarvot, jotka muuten eivät olisi nähtävissä.
- Kun ympäristö on saatu pystytettyä, siinä voidaan suorittaa suuri määrä erilaisia testejä.
- On mahdollista suorittaa testejä, jotka muuten voisivat olla esimerkiksi hengenvaarallisia. (Schlager ym. 2006, 1.)

2.5 Erilaisia markkinoilla olevia simulaattoreita

Markkinoilla on eri tasoisia HIL-simulaattoreita. Pienimmillään tällainen HIL-simulaattori on yhden ohjauslaitteen testaukseen tarkoitettu. Isoimmat simulaattorit on rakennettu laajojen sekä hajautettujen reaaliaikajärjestelmien testaamiseen. Tällainen hajautettu järjestelmä voi olla esimerkiksi lentokoneen autopilotti. (Schlager ym. 2007, 3.)

2.5.1 Keskitetty HIL-simulaattori

Modulaarinen komponentteihin perustuva HIL-simulaattori käyttää yhtä laitetta, joka konfiguroidaan tarjoamaan kaikki tarvittavat rajapinnat testattavaan laitteeseen. Simulaattori voidaan varustaa erilaisilla modulaarisilla I/O- ja prosessorikorteilla, joilla se pystytään räätälöimään tarvittavaa simulaatiota varten. I/O-rajapinta käsittää analogi- ja digitaaliliityntöjä, CAN-, PWM- sekä liikkeenohjausta. (Schlager ym. 2007, 161.)

2.5.2 Hajautettu HIL-simulaattori

Hajautettu HIL-simulaattori koostuu erillisistä toistensa kanssa yhteydessä olevista solmuista, jotka suorittavat hajautettua simulaatiomallia. Jokainen solmuista voidaan varustaa sovelluksen vaatimalla liityntärajapinnalla. (Schlager ym. 2007, 161.)

2.6 Testitapausten suunnittelutekniikat

Tässä luvussa luodaan katsaus erilaisiin testitapausten suunnittelutekniikoihin. Luvussa käsitellään ekvivalenssiluokitus, raja-arvotestaus, luokittelupuumenetelmä sekä luokittelupuumenetelmä sulautetuillejärjestelmille.

2.6.1 Ekvivalenssiluokitus

Ekvivalenssiluokitus on samantyyppinen testitapausten suunnittelutekniikka kuin myöhemmässä luvussa esiteltävä luokittelupuumenetelmä. Se mahdollistaa monimutkaisempien järjestelmien testauksen, koska siinä testattavaa järjestelmää voidaan tarkastella useammalta kuin yhdeltä näkökannalta. (Hass 2008, 179.)

Tekniikassa mahdolliset syötteet jaetaan eri luokkiin. Samassa luokassa olevat arvot toimivat kaikki samalla tavalla testattavan järjestelmän suhteen. Voidaan olettaa, että testaamalla yksi arvo luokasta, saman luokan muut arvot antavat saman tuloksen. Kun syötteet jaetaan luokkiin, yleensä saadaan ainakin kaksi luokkaa: validit arvot ja epävalidit arvot. Epävalidien luokka sisältää arvot, jotka testattavan järjestelmän tulisi hylätä tai järjestelmän toimintaa ei voida ennustaa. Testitapaukset tulisi suunnitella molemmille, toimiville ja ei-toimiville testitapauksille. (Hass 2008, 153.)

2.6.2 Raja-arvotestaus

Raja-arvotestaus perustuu luokkien välisten rajojen tunnistamiseen ja testaukseen. Raja-arvotestausta voidaankin pitää ekvivalenssiluokittelun jatkeena. (Black ym. 2009, 83.)

Invalid	Valid		Invalid
0	1	99	100

Kuva 4. Esimerkki raja-arvoista (perustuu Black ym. 2009, 83.)

Raja-arvotestauksessa otetaan pienin ja suurin arvo oikeista syötteistä sekä viimeinen ja ensimmäinen arvo epäkelvoista syötteistä. Kuvion neljä tapauksessa nämä kelvot arvot ovat 1 ja 99 sekä epäkelvot arvot 0 ja 100. Esimerkistä saadaan siten muodostettua kolme ekvivalenssiluokitustestiä sekä neljä raja-arvotestiä. (Black ym. 2009, 83.)

2.6.3 Luokittelupuumenetelmä

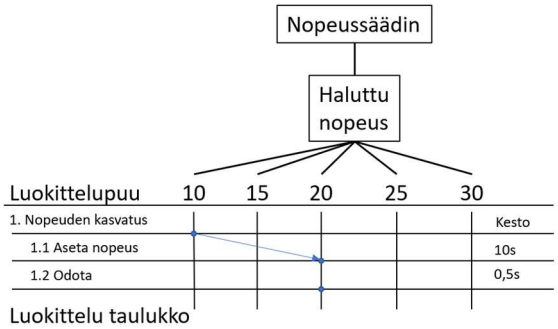
Luokittelupuumenetelmä on systemaattinen tapa, jolla voidaan suunnitella mustalaatikkotestauksessa käytettäviä testitapauksia. Se on epämuodollinen testien suunnittelutekniikka. Notenboomin ja Broekmanin (2003, 144) mukaan luokittelupuumenetelmän kehittivät Grochtmann ja Grimm vuonna 1993.

Luokittelupuumenetelmä on epämuodollinen suunnittelutekniikka, joka luo vain yleisluontoiset säännöt testitapaukselle, ja antaa testaajille enemmän vapautta sekä testien suunnitteluun että suoritukseen. Tällaisen tekniikan käyttö perustuu testaajan luovuudelle ja

tuntumalle testattavasta järjestelmästä sekä sen heikkouksista. Epämuodollinen tekniikka vaatii yleensä tarkkaa soveluusalueen tietämystä ja osaamista. Suunnittelutekniikka on vähemmän riippuvainen lähtötietojen laadukkuudesta. Tästä johtuen sen haittapuolena on huonompi etenemisen seuranta lähtötietoihin perustuen. (Broekman & Notenboom 2003, 120.)

2.6.4 Luokittelupuumenetelmä sulautetuille järjestelmille

Luokittelupuumenetelmä sulautetuille järjestelmille (CTM/ES) on notaatiolaajennus tavalliselle luokittelupuumenetelmälle. Laajennuksessa määritellään yksittäinen testisyöte testiaskeleina. Solmukohdat, joita kutsutaan luokitteluiksi, määrittävät testisignaalin. Sallitut arvot jokaiselle signaalille määritellään ja arvot jaetaan ekvivalenssiluokkiin. Peräkkäiset solmukohdat muodostavat testiaskeleet. Jokainen askel määrittää testisyötteen tietylle aikavälille. Aikavälit on listattu taulukon oikeaan reunaan. Askeleen alku- ja loppupisteet ovat niin sanottuja synkronointipisteitä, koska ne synkronoivat syötteen signaalin jokaisen askeleen alussa ja lopussa. Syötteen arvot saadaan luokasta johon se sisältyy. Synkronisointipisteiden välinen signaalinmuoto määritellään erilaisella viivan muodolla (ramppi, siniaalto). (Broekman & Notenboom 2003, 242.)



Kuva 5. CTM/ES-testitapaus sulautetuille järjestelmille.

2.7 Jatkuva Integrointi

Jatkuva Integraatio on ohjelmiston kehittämistapa, jossa ohjelmistoa integroidaan jatkuvasti kehityksen aikana, toisin kuin projekteissa, joissa yksittäisten ohjelmoijien tai tiimien tuottama lähdekoodi integroidaan vasta päivien tai kuukausien jälkeen. Integroinnin

viivästyttäminen kasvattaa virheiden sekä konfliktien vakavuutta ja määrää, tämä voi aiheutua koodien yhdistämisessä. Hyvä jatkuvan integroinin tapa edellyttää ohjelmoijilta lähdekoodien päivittämistä versionhallintaan päivittäin. Versionhallinnan päivittämisen lisäksi järjestelmä tulisi kääntää ja testata, että voidaan varmistua ohjelmiston toimivuudesta jokaisen versionhallinnan päivityksen jälkeen. Myös muut kehittäjät pystyvät siten varmistumaan, että versionhallinnassa on toimiva versio ohjelmistosta, johon he pystyvät lisäämään omat muutoksensa. (Laukkanen, Itkonen & Lassenius 2017, 56.)

Tyypillisesti CI-palvelinta käytetään valvomaan uusia muutoksia versionhallinnasta. Kun CI-palvelin havaitsee muutoksia lähdekooditiedostoissa, se kääntää ohjelmiston ja suorittaa automaattiset testit. Jos käännös tai testi epäonnistuu, kehittäjälle, jonka muutokset aiheuttivat ongelmia, lähetetään viesti virheestä. Viestissä kehotetaan korjaamaan virheet tai palaamaan takaisin toimivaan versioon kyseisestä tiedostosta, jolloin järjestelmä pysyy toimintakuntoisena. (Laukkanen ym. 2017, 56.)

2.8 Jatkuva toimitus

Jatkuva julkaisu on ohjelmiston kehittämistapa, jossa ohjelmisto voidaan julkaista tuotantoon milloin tahansa. Tämä voidaan saada aikaan optimoimalla kehitysprosessia sekä automatisoimalla käännös-, testi- ja julkaisuprosessia. Jatkuva julkaisu laajentaa jatkuvaa integrointia siten, että ohjelmistoa testataan jatkuvasti ja se on siten jatkuvasti julkaistavassa tilassa. Julkaisuprosessin täytyy myös olla automatisoitu. (Laukkanen ym. 2017, 56.)

Jatkuvat ohjelmiston kehitystavat tarjoavat hyötyjä, kuten nopea palaute ohjelmistokehitys-prosessista sekä asiakkailta. Usein tapahtuvat julkaisut saavat aikaan parempaa asiakastyytyväisyyttä sekä parantavat ohjelmiston laatua. (Laukkanen ym. 2017, 56.)

2.9 Epec ATE -järjestelmä

Epec ATE -järjestelmä koostuu National Instrumentsin valmistamasta PXI-rungosta johon on kalustettu ohjain, ja SLSC-rungosta, johon on kiinnitetty RMX-10051 tehonjakolaite ja RMX-4125 teholähteestä, näillä toteutetaan tehonsyöttö testilaitteistolle.

2.9.1 PXI - PCI eXtensions for Instrumentation

PXI-runkoa voi ajatella ikäänkuin työpöytätietokoneen kotelona. PXI-runkoon on kiinnitetty ohjain, joka vastaa työpöytätietokoneessa emolevyä. PXI-rungossa on tilaa myös laajennusmoduuleille joita voi ajatella työpöytätietokoneen laajennuskortteina, kuten esimerkiksi verkkokortti. PXI on yleinen standardi joten siihen hankitut osat eivät ole valmistajakohtaisia. Runko on kompakti kooltaan, tästä johtuen sen lämmönsiirtokyky ei ole hyvä. Rungossa oleva ohjain käyttää Linux-realiaikakäyttöjärjestelmää ja sitä voidaan ohjata tietokoneelle asennettavalla Veristand-ohjelmistolla. (What is PXI?, [viitattu 1.2.2022].)

Kuva 6. PXI-järjestelmä (What is PXI?, [viitattu 1.2.2022].)

2.9.2 SLSC - Switch Load and Signal Conditioning

Samassa yhteydessä on hankittu myös SLSC-laajennusrunko sekä siihen erillisiä I/O-moduleja. SLSC-rungon etuna edellisessä luvussa mainittuun PXI-runkoon on mahdollisuus käyttää suurempaa IO-määrää, sekä parempi tuuletus, joka mahdollistaa suurempien virtamäärien käyttämisen testissä. Laajennusrunko ei sisällä itsenäistä ohjauslogiikkaa, vaan sitä ohjataan Ethernet-väylän kautta PXI-rungon kontrollerilla.

SLSC-runkoon on mahdollista kiinnittää National Instrumentsin valmistamia laajennuskortteja. National Instrumentsillä on myös ulkopuolisia partnereita, jotka valmistavat laajennuskortteja. (SLSC, [viitattu 1.2.2022].)

Kuva 7. SLSC- ja PXI-rungot kalustettuina (SLSC, [viitattu 1.2.2022].)

2.9.3 Tehonsyöttö

Järjestelmän tehonsyöttö hoidetaan RMX-10051-tehonjakolaitteella sekä ohjelmoitavalla RMX-4125-teholähteellä.

RMX-10051 on kolmivaiheinen 16 A:n tehonjakolaite. Laite jakaa tehoa liitimiensä kautta useammalle eri rungolle, esimerkiksi PXI- ja SLSC-rungolle tai teholähteelle. Laitteessa on peräkkäinen käynnistys, hätäkatkaisuominaisuus sekä sisäänrakennetut varokkeet. (RMX-10051, [viitattu 2.2.2022].)

Kuva 8. RMX-10051-tehonjakolaite (RMX-10051, [viitattu 2.2.2022].)

RMX-4125-teholähde mahdollistaa 0 – 80 V tasajännitteen syötön suurimmillaan 56 A:n virralla. Suurin teholähteen tuottama teho on 1500 W. Teholähdettä on mahdollista ohjata USB-, verkko- tai RS232-yhteyttä käyttäen. Laitteen etupanelissa on kytkimet paikallista käyttöä varten. (RMX-4125, [viitattu 2.2.2022].)

Kuva 9. RMX-4125 Teholähde (RMX-4125, [viitattu 2.2.2022].)

3 Käytetyt työkalut

Tässä luvussa esitellään testilaitteiston luomiseen ja käyttöön vaadittavat työkalut. Työssä käytettävät työkalut sisältävät erilaisia ohjelmistoja, sekä fyysisiä testilaitteistoja.

3.1 Testona

Testona on työkalu mustalaatikkotestien suunnitteluun. Ohjelma tukee kaikkia määrittelyihin perustuvia testimetodeja, kuten esimerkiksi luokittelupuu-, ekvivalenssiluokittelu- ja raja-arvotestausta. Ohjelmalla on mahdollista testitapausten systemaattinen generointi luokittelupuusta. (Testona, [viitattu 23.1.2022].)

3.2 Robot Framework

Robot Framework on yleiskäyttöinen avoimen lähdekoodin kehys testiautomaation ja robotisoidun prosessiautomaation luomiseen. Se on laajennettavissa erilaisilla kirjastoilla. Robot Framework on kehitetty Python-ohjelmointikieltä käyttäen. Sitä voidaan kuitenkin ohjelmoida käyttäen useita eri ohjelmointikieliä, kuten esimerkiksi Java. Robot Frameworkin käyttö perustuu avainsanojen luomiseen ja käyttöön. Avainsanoja käyttämällä voidaan ottaa käyttöön Robot Frameworkin kirjastojen sisältämiä funktioita. Robot Framework sisältää useita sisäisiä ohjelmakirjastoja perustoiminnallisuuksien suorittamiseen. Toimintoja on mahdollista laajentaa erikseen asennettavilla kirjastoilla. Tällaisia toimintoja ovat esimerkiksi verkkosivujen, tietokantojen tai mobiiliapplikaatioiden testaamiseen luodut kirjastot. Käyttäjä voi luoda myös omia kirjastojaan erilaisiin testaustarkoituksiin. Robot Frameworkin kehitti Pekka Klärck diplomityössään Nokia Networksille. (Robot Framework [viitattu 24.11.2021].)

3.3 CODESYS-ohjelmointiympäristö

Epec-ohjainlaitteissa on käytössä CODESYS 3.5 -ohjelmointiympäristö. CODESYS on valmistajasta riippumaton IEC 61131-1 -standardin mukainen ohjelmistoympäristö ohjausjärjestelmien suunnitteluun. Yhtiö on perustettu vuonna 1994 nimellä 3S-Smart Software Solutions GmbH. Yhtiön nimi muuttui muotoon CODESYS GmbH vuonna 2020.

CODESYS-ympäristöä käytetään lähes kaikilla automaatioteollisuuden aloilla, tehtaista koneenrakennukseen. (CODESYS, [viitattu 15.1.2021].)

3.4 Azure DevOps Server

Azure Devops koostuu eri palveluista kehittäjille, ne mahdollistavat eri tiimien yhteistyön lähdekoodin kehityksessä. Näitä palveluita ovat esimerkiksi versionhallinta ja erilaiset CI/CD-palvelut. (Microsoft, [viitattu 30.1.2022].)

3.5 Azure Artifacts

Azure Artifacts on Azure DevOps Serverin käyttämä paketinhallintatyökalu. Sen avulla voidaan jakaa eri tyyppisiä paketteja julkisista tai yksityisistä lähteistä. Paketit tallentuvat järjestelmään ja näitä on mahdollista käyttää esimerkiksi julkaisuputkessa. (Microsoft, [viitattu 12.11.2021].)

3.6 Azure Pipelines

Azure Pipelines on palvelu, jolla on mahdollista luoda käännös- tai julkaisuputkia. Julkaisuputkessa voidaan käyttää useita ohjelmointikieliä ja eri projektityyppejä. Julkaisuputkella on mahdollista toteuttaa jatkuva integrointi (CI) sekä jatkuva toimitus/julkaisu (CD). Tekemällä muutoksia versionhallintaan voidaan muutoksista kääntää uusi suoritettava ohjelmistopaketti (jatkuva integrointi). Julkaisuputkella voidaan ottaa edellisessä kohdassa integroitu muutokset sisältävä julkaisupaketti, ladata se testijärjestelmään, suorittaa automaattiset testit sekä tallentaa testiraportit julkaisuputkeen. Käännös- ja julkaisuputki voidaan suorittaa eri tietokoneissa, jolloin julkaisuputki poimii käännösputken tuotoksena olevat artifaktit ja käyttää niitä omassa suorituksessaan. (Microsoft, [viitattu 11.11.2021].)

3.7 Agentti

Agentti on ’työntekijä’, joka suorittaa julkaisuputkeen määritettyjä erilaisia tehtäviä. Tällainen tehtävä voi olla esimerkiksi sovelluksien tai firmware-ohjelmistojen päivitys

testijärjestelmässä oleviin ohjauslaitteisiin. Jokaisen julkaisuputken suoritukseen tarvitaan vähintään yksi agentti. Agentteja on kahden tyyppisiä: Microsoftin isännöimiä tai itse isännöitäviä. Microsoftin isännöimän agentin ylläpito ja päivitykset hoidetaan käyttäjän puolesta. Julkaisuputki suoritetaan aina uudessa virtuaalikoneessa, joka tuhotaan käytön jälkeen. Käytettäessä Microsoftin isännöimää agenttia täytyy myös ottaa huomioon siitä tulevat kustannukset. Microsoft tarjoaa kokeilupaketteja, joilla pääsee tutustumaan palveluun, mutta laajemmassa käytössä kannattaa miettiä kumman tyyppinen agentti tulee edullisimmaksi käyttää. Itse isännöitävä agentti mahdollistaa käyttäjälle agentin ominaisuuksien hallinnan sekä mahdollistaa ohjelmien asennuksen. Tällaista agenttia voi käyttää Azure-julkaisuputkessa sekä Azure DevOps Serverissä. Useamman agentin käyttöä samassa koneessa ei suositella, koska sillä voi olla haitallisia vaikutuksia julkaisuputken suorituskykyyn. (Microsoft [viitattu 24.11.2021].)

3.8 Python-ohjelmointikieli

Python on helposti opittava ja tehokas ohjelmointikieli. Pythonissa on mahdollista käyttää olio-ohjelmointia ja rakentaa erilaisia tietorakenteita. Kieli on tulkattu ohjelmointikieli, joka tarkoittaa, että ohjelman suoritusta varten käynnistetään erillinen komentotulkki, joka lukee Python-kielisen lähdekoodin, ja toteuttaa siinä olevat käskyt. (Korpela 2001.)

Vaikka Pythonissa on runsaasti valmiita työkaluja yleiseen käyttöön, Pythoniin on mahdollista asentaa kolmannen osapuolen valmistamia moduleja. Yleinen paikka löytää tällaisia moduleita on Python Package Index (PyPi). (Python Package Index [viitattu 23.1.2022].)

3.9 PyVISA

PyVISA on Python-moduuli joka on tarkoitettu VISA-väylää tukevien mittalaitteiden käyttöön. Moduulin on kehittänyt Matthieu Dartiailh, joka myöskin ylläpitää sitä. (PyVISA [viitattu 27.1.2022].)

VISA on lyhenne sanoista Virtual Instrument Software Architecture. VISA on teollisuuden käyttämä kommunikointirajapintastandardi testaus- ja mittauslaitteille. Sitä kutsutaan joskus

myös kommunikointiajuriksi. VISA mahdollistaa ohjelmien suunnittelun ja valmistuksen väyläriippumattomiksi. Käyttämällä VISA-rajapintaa voidaan kommunikoida monen eri rajapinnan yli, esimerkiksi USB- tai Ethernet-rajapinta. (VISA, [viitattu 25.1.2022].)

3.10 VeriStand Python

VeriStand Python on moduuli, joka sisältää rajapinnan VeriStand-järjestelmän käyttöön. Moduulin on alunperin kehittänyt ja sitä tukee National Instruments. Rajapinta sisältää funktiot VeriStand-järjestelmän kanssa kommunikointiin. Rajapinnan avulla on mahdollista suorittaa testejä kahdessa eri tilassa: 1) Deterministinen tila, jossa testisekvenssi ladataan VeriStand-järjestelmään, ja testit suoritetaan realiajassa. 2) Python-tila, jossa testi-PC kommunikoi laitteiston kanssa yhteyskäytävää käyttäen. Python-tila emuloi deterministisen tilan toimintaa. Tämä moodi on hyödyllinen, jos halutaan käyttää hyödyksi koko Pythonin tarjoamaa ekosysteemiä. (Veristand Python, [viitattu 27.1.2022].)

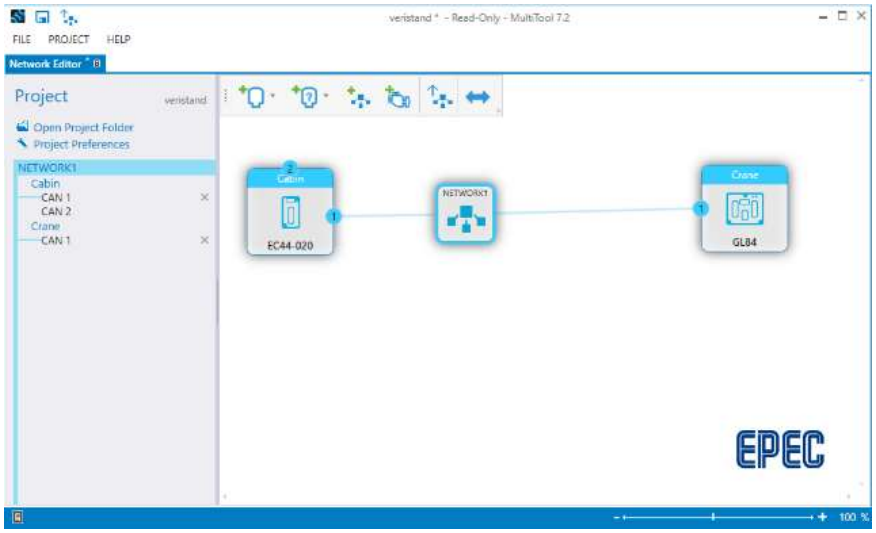
3.11 VeriStand

VeriStand-sovellusohjelmalla on mahdollista konfiguroida I/O-kanavia, kerätä dataa, luoda syötteitä ja kommunikoida National Instrumentsin realiaikatestilaitteistojen kanssa. Ohjelmistosta on tarjolla kolmea eri hintaista lisenssiä: a) VeriStand Development System, joka sisältää eniten ominaisuuksia. Tällä lisenssillä on mahdollista luoda testiympäristöjä sekä suorittaa testejä. b) Toinen lisenssityyppi on VeriStand Operator System -lisenssi, joka mahdollistaa testien suorittamisen. c) Kolmantena on VeriStand PC Development System -lisenssi, joka mahdollistaa testiympäristöjen luonnin. (VeriStand, [viitattu 26.1.2022].)

3.12 Epec SDK -ohjelmistojakelu

Epec SDK -jakelu sisältää tarvittavat sovelluskirjastot ja apuohjelmat jotka vaaditaan sovelluksen luomiseen Epec-ohjauslaitteille. Epec SDK -jakelussa olevalla Multitool-sovelluksella on mahdollista suunnitella yhden tai useamman Epec-ohjausyksikön järjestelmä sekä niiden tarvitsemat asetukset. Multitool-sovelluksesta on mahdollista luoda CODESYS-ohjelmointiympäristössä käytettävä koodipohja, jonka avulla käyttäjä voi

rakentaa oman sovelluksensa. Koodipohja sisältää sovelluksessa tarvittavat alustukset sekä muuttujat ja siinä on käytössä Epecin PLCopen-kirjastot. (Epec, [viitattu 19.1.2022].)



Kuva 10. Epec Multitool.

3.13 Epec EC44

Epec EC44 on CODESYS 3.5 -ohjelmoitava laite. Laitteessa on 32 bittinen suoritin, 1 megatavu SRAM-muistia sekä 1024 kilotavua tilaa käyttäjän sovellukselle. EC44-laitteessa on 2 CAN-väylää, sen kotelo on IP69-suojausluokkaa lisäksi siinä on status led. Kotelossa on 46-pinninen lukittava Leavyseal-liitin I/O- sekä CAN-liityntöjä varten. Liittimessä on 16 sisään tulevaa liityntää sekä 16 ulos menevää liityntää. Multitool-ohjelmisto sisältää tuen EC44-ohjainlaitteen ohjelmointiin. (Epec, [viitattu 27.1.2022].)

Kuva 11. Epec EC44 -ohjainlaite (Epec, [viitattu 27.1.2022]).

3.14 Epec GL84

Epec GL84 on standardin CiA 301 v4.2 mukaan toimiva CANOpen-laite eli niin sanottu CANOpen slave. Laitteessa on 3 lukittavaa liitintä: kaksi kappaletta 46 pinnistä sekä yksi 21 pinninen liitin. Kotelo on IP69K-suojausluokituksen omaava. Laitteessa käytettävä prosessori on 32 bittinen 200 Mhz:n kellotaajudella toimiva ja siinä on 2 CAN-liityntää. Liittimiin on tuotu yhteensä 34 kpl ulostuloa sekä 35 kpl sisääntuloa. Multitool-ohjelmistolla voidaan ohjelmoida isäntälaite lataamaan asetukset slave-laitteelle. (Epec, [viitattu 18.1.2022].)

Kuva 12. Epec GL84 CANOpen slave (Epec, [viitattu 18.1.2022]).

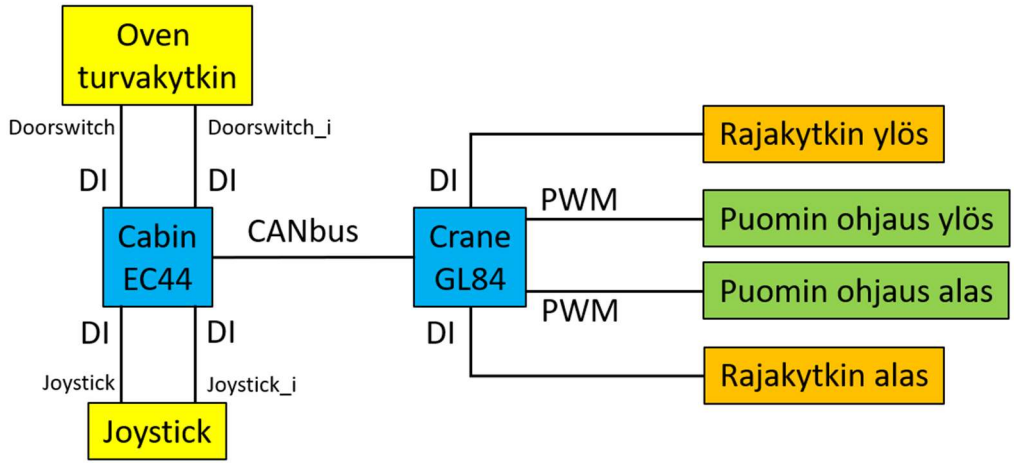
4 Automaattisten testien luonti järjestelmälle

Testattava järjestelmä on osa Epecin uusille asiakkaille tarjottavasta koulutuspaketista. Koulutusprojektissa rakennetaan ohjausjärjestelmä kuvassa 13 esitetylle nosturiautolle. Tässä työssä toteutetaan osa tuosta kokonaisesta koulutusprojektista. Toteutettava ohjausjärjestelmän arkkitehtuuri poikkeaa koulutusprojektissa esitettävästä arkkitehtuurista, jossa on kaksi ohjelmoitavaa laitetta. Tässä työssä toisen ohjelmoitavan yksikön korvaa CANOpen slave -laite sekä Cabin-laitteeseen on toteutettu turvatoiminto, joka on ovikytkimen tilan tarkkailu (Pärnänen 2020).

Kuva 13. Epec-koulutusprojekti uusille asiakkaille (Epec, [viitattu 6.9.2020].)

4.1 Epec-ohjainlaitteet

Testattava järjestelmä koostuu kahdesta eri ohjausyksiköstä joilla on arkkitehtuurissa omat tehtävänsä, ne on kuvattu kuvassa 14. EC44 – Cabin on ohjelmoitava laite joka toimii CAN-väylän sekä GL84 CANopen slave -laitteen isäntänä. EC44-laitteeseen on kytketty turvatoimintona oleva kahdennettu turvakytkin (Doorswitch ja Doorswitch_i digitaalisignaalit) sekä kahdennettu joystick-signaali (Joystick ja Joystick_i analogiasignaali). EC44-ohjainlaite on CAN-väylällä yhdistetty GL84 CANopen slave -laitteeseen. GL84-slaveen on kytketty puominohjauksen ylä- ja alasuunnan rajakytkimet sekä ylä- ja alasuunnan venttiilien ohjaus. GL84-slave saa asetuksensa ja ohjauksensa EC44-laitteelta.



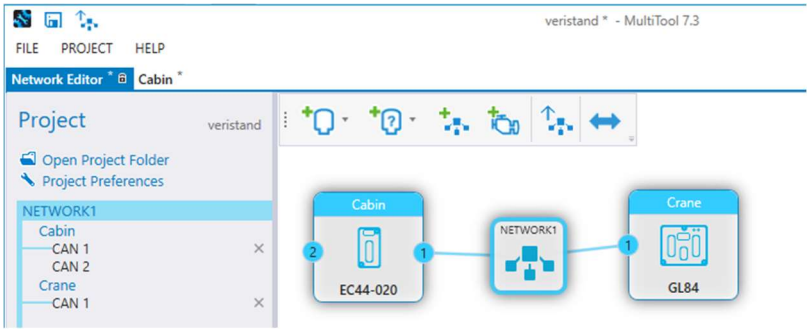
Kuva 14. Testattavan järjestelmän arkkitehtuuri.

4.2 Testattava järjestelmä

EC44-laitteen sovellusohjelmisto koostuu Epec Multitool -ohjelmistolla luodusta koodipohjasta CANopen ja I/O-alustuksineen. Tähän koodipohjaan on kirjoitettu sovellus käyttäen Epec-sovelluskirjastoja.

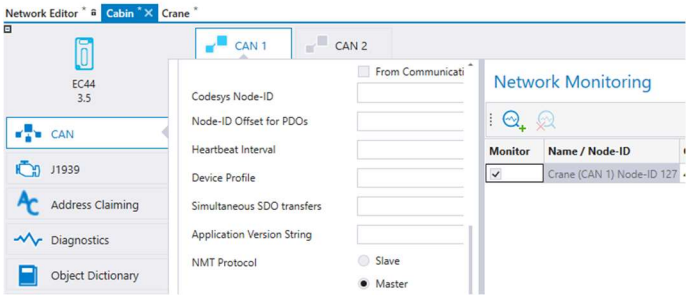
4.2.1 Ohjausjärjestelmän luonti Multitool-sovelluksella

Multitool-pääikkunassa on projektiin lisätty EC44-020-ohjelmoitava laite nimeltään Cabin sekä GL84 CANopen slave -laite nimeltään Crane. Cabin-laitteen CAN1 sekä Crane-laitteen CAN1-liitynnät on yhdistetty samaan Network1-väylään.



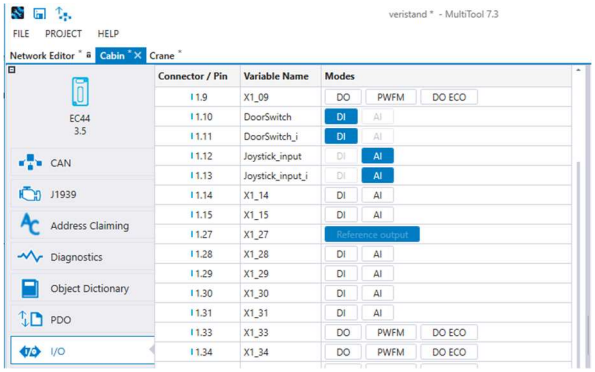
Kuva 15. Multitool network editor -näkymä.

Kun yksiköt on lisätty projektiin, tehdään niille tarvittavat asetukset. Cabin-yksikölle on tarpeellista määritellä CAN-, I/O- sekä PDO-asetukset. EC44-yksikkö määritetään NMT-masteriksi väylään. Tällöin se huolehtii väylän sekä yksiköiden tilan valvonnasta.



Kuva 16. Cabin-laitteen CANopen-asetukset.

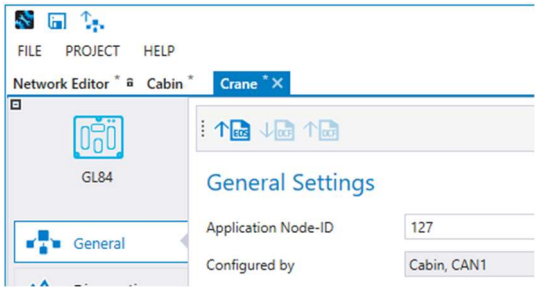
I/O-asetukset EC44-laitteelle sisältävät sisääntuloiksi määritellyt digitaalisisääntulonastat X1.10 DoorSwitch ja X1.11 DoorSwitch_i sekä analogiasisääntulonastat X1.12 Joystick_input ja X1.13 Joystick_input_i.



Kuva 17. Cabin-laitteen IO-asetukset.

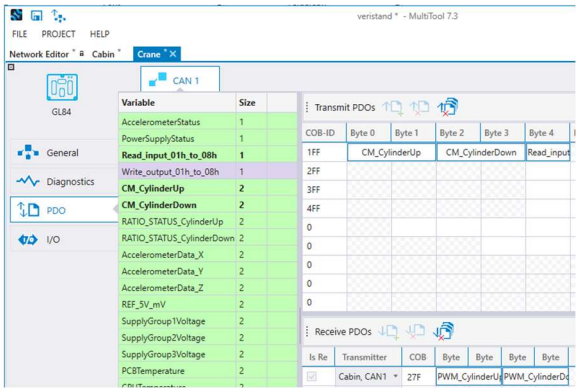
Crane-laitteessa vastaavasti X1.1 ja 1.2 on määritetty LimitSwitchHigh ja LimitSwitchLow. Nastoilla X1.39 ja 1.40 toteutetaan virtaohjaus ylä- sekä alasuuntaan

Crane-laitteen General-välilehdellä määritellään tämän laitteen konfiguroiva ohjelmoitava laite väylässä. Valinnaksi asetetaan Cabin-laite.



Kuva 18. Crane-laitteen CAN-asetukset.

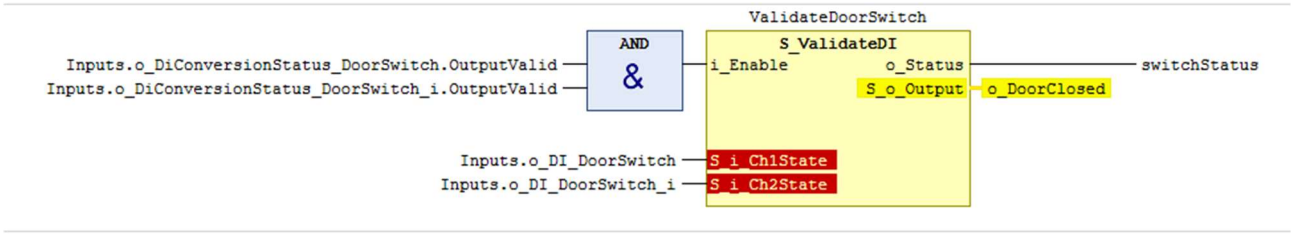
Crane-laitteen PDO-asetuksissa määritellään laitteen lähettämät ja vastaanottamat PDO-viestit väylältä. Laite vastaanottaa väylätä Cabin-laitteen ohjaaman virtapyynnön ja lähettää väylälle virranmittaustiedon ylä- ja alasuunnan ohjauksesta sekä rajakytkimien tilatiedon.



Kuva 19. Crane-laitteen PDO-asetukset.

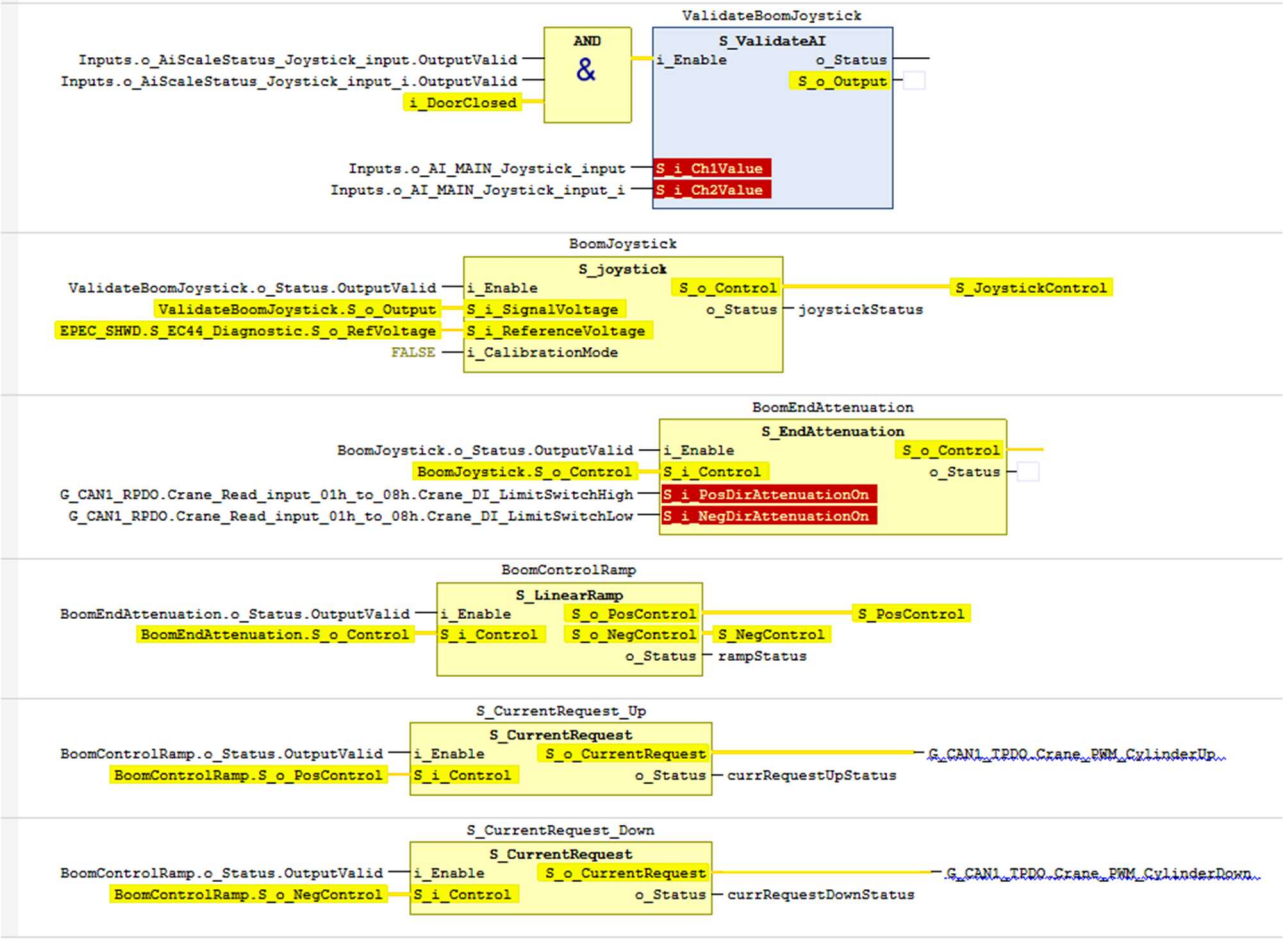
4.2.2 CODESYS-sovellus

Testissä käytettävä sovellus koostuu kahdesta eri lohkosta: SafeCabinDoor, jossa toteutetaan laitteiston turvatoiminto, joka on oven tilan tarkkailu, sekä BoomControl, jolla muodostetaan virtaohjaus venttiilille. Lohkot ovat toteutettu käyttäen Epecin kehittämiä sovelluskirjastoja.



Kuva 20. SafeCabinDoor

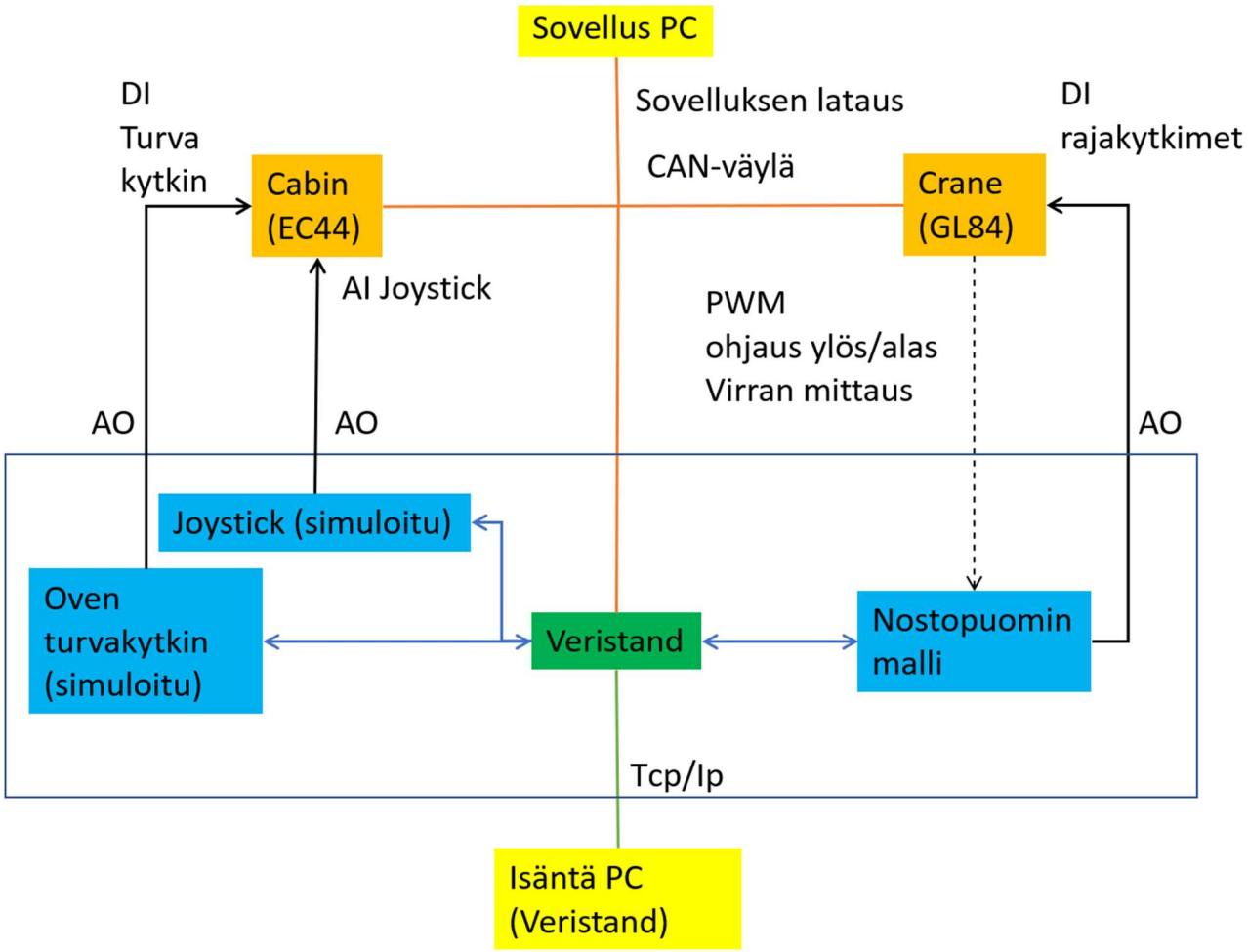
SafeCabinDoor-ohjelmassa varmistetaan turvatoimintona olevan kahdennetun oven kytkimen signaalien tila sekä itse oven tilatieto. (Epec, [viitattu 15.1.2022].)



Kuva 21. BoomControl-ohjelma.

BoomControl-ohjelmassa muodostetaan virtapyynti väylälle lähetettäväksi. Joystick-signaalien tila sekä oven tilatieto validoidaan ValidateBoomJoystick-lohkossa. Niiden ollessa kunnossa muodostetaan joystickin molemmista kanavista ohjaustieto BoomJoystick-lohkossa. BoomEndAttenuation-lohkossa ohjaussignaaliin muodostetaan päätyvaimennus Crane-laitteeseen liitetettyjen rajakytkimien perusteella. BoomControlRamp-lohko muodostaa ohjausrampin ylä- tai alasuuntaan ohjauksen perusteella. S_CurrentRequest_Up- ja S_CurrentRequest_Down-lohkot muodostavat lopullisen väylälle lähetettävän virtapyynnin. Nämä virtapyynnit sijoitetaan TPDO-muuttujiin, jotka on luotu Multitool-projektissa. (Epec, [viitattu 16.1.2022].)

4.3 HIL-testausjärjestelmä



Kuva 22. Arkkitehtuurikuva, jossa näkyy testattava järjestelmä sekä Veristand-laitteisto.

4.3.1 HIL-testilaitteiston toiminta

Kuvassa 22 on esitetty rakennettu HIL-testilaitekytkentä. Kytkentä koostuu Epec-ohjauslaitteista, Veristand-testauslaitteistosta sekä kahdesta tietokoneesta. Epec-ohjauslaitteissa suoritetaan edellisessä luvussa kuvattua sovellusta, jolla luetaan Veristandin simuloimaa joystickin toimintaa. Tämän ohjauksen sekä simuloidun oven turvakytkimen perusteella ohjataan Epec GL84 CANopen-orjalaitetta, joka taas ohjaa PWM-signaalilla siihen kytkettyä venttiilin karaa. Tällä hetkellä laitteistossa ei ole valmiutta virran mittaamiseen, joten venttiiliä vastaa ulkoinen vastus, johon PWM-signaali ohjataan. Puomimallin ohjaukseen käytetään GL84-laitteen omaa virranmittausta, joka lähetään väylälle. Isäntä-PC:llä suoritetaan gateway-palvelinta, jolla pystytään ohjaamaan testilaitteiston toimintaa.

4.4 HIL-testisekvenssien luonti ja ajaminen Robot Frameworkia käyttäen

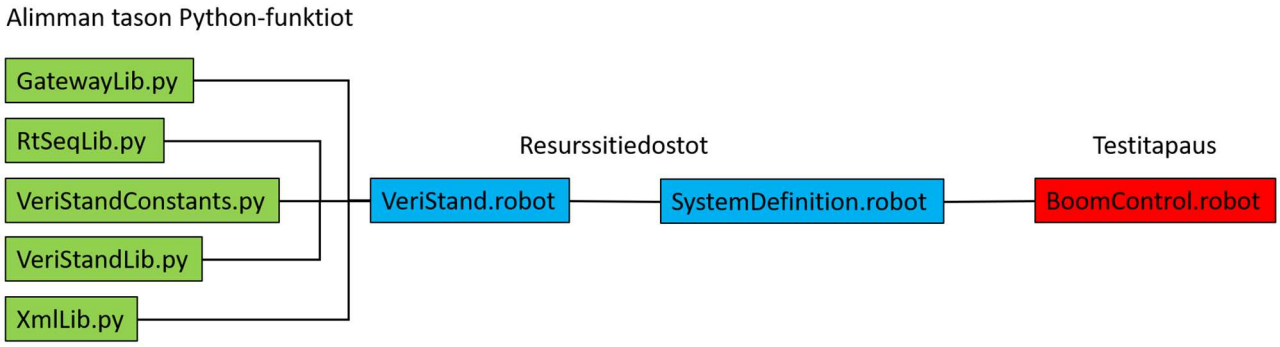
Tässä luvussa esitetään tapa suorittaa automaattisesti HIL-testisekvenssejä. Luvussa käydään lävitse Robot Framework -testitapauksen suoritukseen vaadittujen tiedostojen luonti, sekä esitetään miten testitapaus koostuu näistä luoduista tiedostoista.

4.4.1 Robot Frameworkin käyttämiseen luodut tiedostot

Testisekvenssien suorittamiseen käytetään National Instrumentsin julkaisemaa niveristand-Python-kirjastoa. Tällä kirjastolla voidaan käyttää Veristand-ohjelmiston rajapinnan tarjoamia funktioita.

Python-kirjaston tarjoamat funktiot on kirjoitettu viiteen erilliseen luokkaan, jotka tuodaan Robot Frameworkiin jokainen omana objektinaan. Luokkien tarjoamista metodeista on laadittu Robot Frameworkin käyttämiä resurssitiedostoja. VeriStand.robot-tiedosto sisältää avainsanat järjestelmämäärityksen lataamiseen testeriin, yhteyskäytävän käynnistämiseen sekä kanavien arvojen muuttamiseen sekä vikatilojen luomiseen eri kanaviin. SystemDefinition.robot sisältää testissä tarvittavien Veristand-tiedostojen sijainnit sekä avainsanat, joilla asetetaan järjestelmä valmiiksi testejä varten sekä sammutetaan jännitteet suoritetun testin jälkeen.

BoomControl.robot on itse testitapaus joka on koottu edellisessä luvussa esitellyistä resurssitiedostoista. Käyttämällä resurssitiedostoja tällä tavoin, voi testaaja luoda testitapaukset vain Robot Framework -avainsanoja käyttämällä sekä niiden vaatimalla syntaksilla.



Kuva 23. Python-funktioiden käyttö testitapausten luontiin resurssitiedostojen avulla.

4.5 Testitapausten ja syötteiden suunnittelu järjestelmälle käyttäen luokittelupuumenetelmää

Tässä luvussa esitetään esimerkki testitapausten sekä tarvittavien testisyötteiden suunnitelusta testattavalle järjestelmälle luokittelupuumenetelmää apuna käyttäen.

4.5.1 Testitapausten suunnittelu HIL-testaukseen

Luokittelupuumenetelmä tunnistaa testattavan järjestelmän ominaisuudet. Tällaiset ominaisuudet ovat sellaisia, jotka voivat vaikuttaa järjestelmän toiminnallisuuteen tai turvallisuuteen. (Broekman & Notenboom 2003, 145.)

Tunnistettujen ominaisuuksien arvoalueet jaetaan erillisiin luokkiin käyttäen ekvivalenssiluokittelua. Tämä tarkoittaa arvoalueiden muodostamista siten, että yhden arvon kyseisestä luokasta voidaan olettaa vastaavan kaikkia luokan mahdollisia arvoja. Testitapaukset muodostetaan yhdistämällä eri ominaisuuksien luokkia. (Broekman & Notenboom 2003, 146.)

Ideaalitilanteessa testitapausten suunnittelun perusteena on toiminnalliset vaatimukset. Aina kuitenkaan vaatimuksia ei ole saatavilla tai ne ovat puutteellisia tai vanhentuneita. Luokittelupuumenetelmän selkeä etu on, että tässäkin tilanteessa sitä voidaan soveltaa testien suunnitteluun. Testaaja, jolla on perustason ymmärrys testattavasta järjestelmästä, lisää puuttuvat tiedot. Jos vaatimukset puuttuvat kokonaan, testaajalla täytyy olla syvällinen tuntemus testattavasta järjestelmästä ja sen toiminnasta. (Broekman & Notenboom 2003, 145.)

Luokittelupuumenetelmään on myös kehitetty laajennus erityisesti sulautettujen järjestelmien signaalien määrittelyyn (Conrad 2001, 2.)

4.5.2 Testiobjektin määrittäminen

Testien suunnitelu alkaa toiminnallisen ongelman tai niin sanotun testiobjektin määrittämisellä. Tässä tutkimuksessa tuo testiobjekti on puominohjaus.

Testiobjektin odotettu toiminta on puomin liike joystickillä tehtävän ohjauksen mukaan. Lähestyttäessä liikkeen ääriasentoja puomissa on päätyvaimennus.

Järjestelmässä on myös turvatoimintona oven turvakytkin, joka asettaa järjestelmän turvalliseen tilaan, jos ovi avataan kesken liikkeen. Oven täytyy olla myös suljettuna, että puomin liikettä voidaan alkaa ohjata.

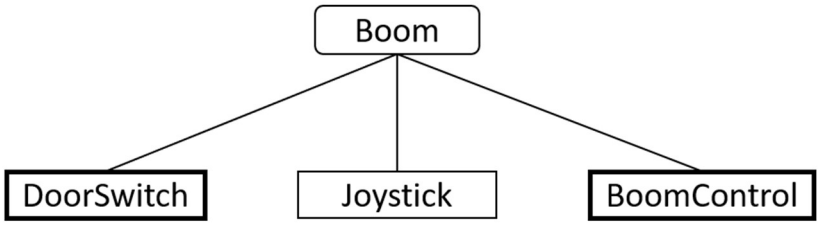
4.5.3 Testiobjektin ominaisuuksien määrittäminen

Testiin liittyvä ominaisuus on ominaisuus, joka vaikuttaa järjestelmän toimintaan (Büchner 2019, 8). Tässä tapauksessa näitä ominaisuuksia ovat:

- Joystick, jolla ohjataan puomin suunta.
- DoorSwitch, joka on järjestelmän turvatoiminto.

- BoomControl esittää järjestelmän tuottamaa ohjaus signaalia. (Broekman & Notenboom 2003, 145.)

Ominaisuudet voidaan ilmaista graafisesti puumaisessa muodossa, kuten kuvassa 24.



Kuva 24. Testattavan järjestelmän ominaisuudet (Broekman & Notenboom 2003, 146.)

Puumaisessa esityksessä eri haarat ovat siis luokitteluja, jotka luovat perustan niiden alle tuleville luokille. Kuvassa 24 on esitetty järjestelmästä laadittu luokittelupuu. Ylimpänä on testiobjekti Boom. Kuvasta puuttuu vielä itse luokittelun alaiset luokat, koska ne määritetään myöhemmässä vaiheessa. (Broekman & Notenboom 2003, 146.)

DoorSwitch ja BoomControl ovat yllä olevassa puussa yhdistelmätyyppisiä solmuja, nämä on esitetty paksummilla reunuksilla olevilla neliöillä.

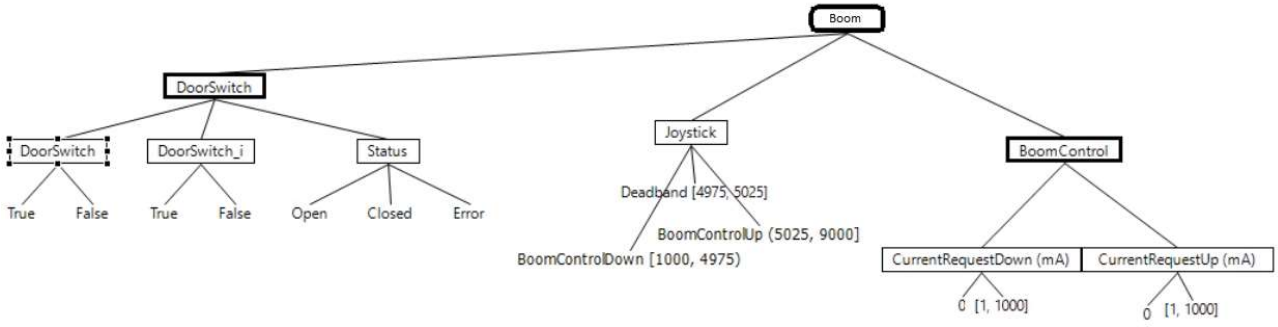
Yhdistelmä on näkymätön tämän vuorottelun osalta ja tarkoittaa, että kahden eri luokittelun sisältämiä arvoja voidaan yhdistää testitapauksessa eivätkä ne sulje toisiaan pois (Büchner 2019, 25).

BoomControl yhdistelmäsolmu siis koostuu CurrentRequestDown- ja CurrentRequestUp-luokitteluista, jotka taas voidaan jakaa eri luokkiin ekvivalenssiluokittelun avulla. Ovikytkin on myös samantyyppinen luokittelu, joka koostuu kytkinsignaalin (*DoorSwitch*) ja sen invertoidun signaalin luokittelusta (*DoorSwitch_i*). (Broekman & Notenboom 2003, 146.)

4.5.4 Testiobjektin ominaisuuksien arvoalueiden jako luokkiin

Kun testiin liittyvät ominaisuudet on löydetty, määritellään niiden saamat mahdolliset arvot. Kyseiset arvot jaetaan eri luokkiin ekvivalenssiluokittelun avulla. Tämä tarkoittaa, että yhtä

luokkaa kohti valitaan yksi arvo, jonka voidaan olettaa vastaavan kaikkia samassa luokassa olevia arvoja. (Broekman & Notenboom 2003, 146.)



Kuva 25. Testiobjektin ominaisuudet jaettuna ekvivalenssiluokkiin

- DoorSwitch- ja DoorSwitch_i-luokittelujen saamat arvot on jaettu true- ja false-luokkiin. Statusluokittelu kuvaa oven tilaa ja se voi saada arvot luokista auki, kiinni tai virhe.

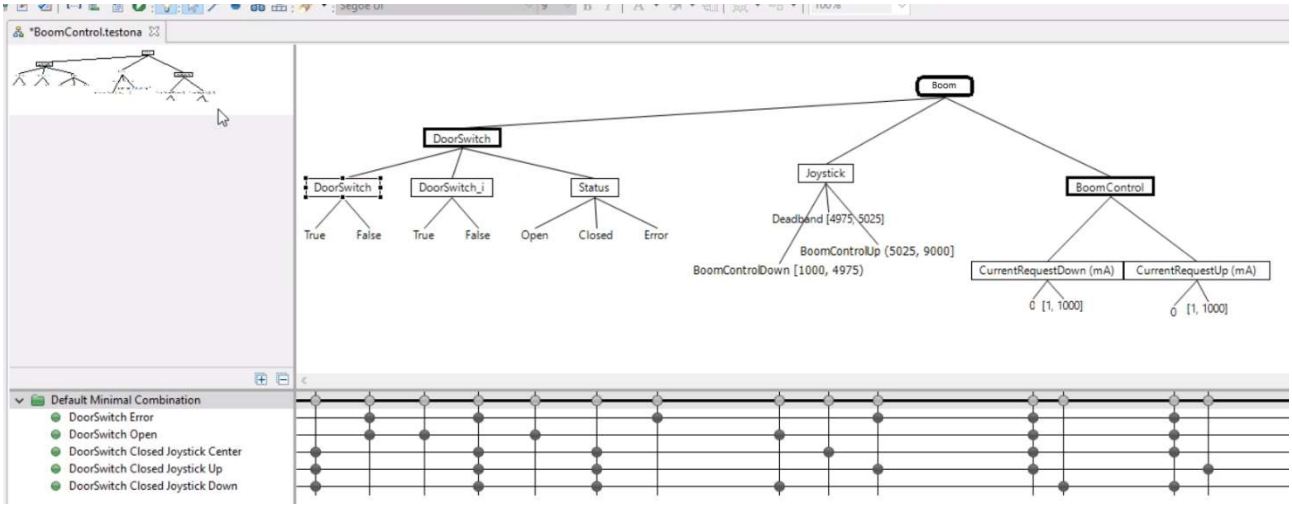
- Joystick-luokittelu on jaettu luokkiin a) BoomControlDown, joka vastaa signaalin jännite-arvoja 1 V – 4.975 V, b) Deadband-luokka, eli keskitetty joystick vastaa jännitearvoja 4,975 V – 5,025 V ja c) BoomControlUp-luokka vastaa arvoja 5,025 V – 9 V.

- BoomControl-yhdistelmäsolmu ja sen alla olevat luokittelut CurrentRequestDown ja CurrentRequestUp kuvaavat järjestelmän tuottamaa vastetta eri sisääntulevien signaalien kombinaatioilla.

4.5.5 Testitapausten määrittely laaditusta luokittelupuusta

Edellisissä luvuissa esitetty luokittelupuu on piirretty kuvassa 26 Testona-ohjelmalla ja siitä on johdettu testitapaukset minimaalista kombinaatiota käyttäen. Minimaalinen kombinaatio tarkoittaa, että jokaisen luokan arvoja käytetään testitapauksessa vähintään yhden kerran (Kruse 2014, 10).

Luokittelupuusta voidaan johtaa testitapaukset määrittelemällä riippuvuudet eri luokkien välillä (Kruse& Zeppetzauer 2014, 1587).

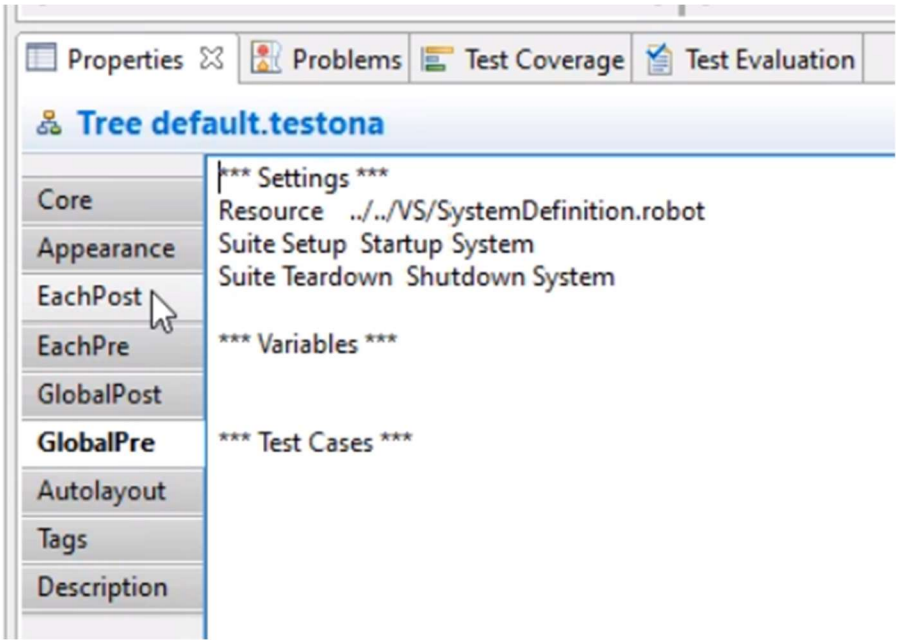


Kuva 26. Testitapaukset luokittelupuusta

Kun testitapaukset on luotu ohjelmassa, niistä voidaan tuottaa testitapauspohjat, jotka ovat Robot Framework -muotoisia. Testitapaukset eivät sisällä itse testiaskeleita, mutta ne sisältävät rungon sekä testidatan, jolla saadaan itse askeleet helposti tehtyä valmiita avainsanoja käyttäen. Liitteessä 1 on esitetty ohjelmalla luotu testitapauksen pohja, sekä liitteessä 2 on esitetty valmis testitapaus.

4.6 Luokittelupuumenetelmällä luodun testipohjan käyttö

Edellisessä luvussa on esitetty luokittelupuumenetelmän käyttö testitapauksien suunnitteluun sekä testisyötteiden luomiseen. Tuomalla luodut testitapaukset Testona-ohjelmasta tietyillä asetuksilla saadaan Robot Framework -yhteensopiva pohja, johon voidaan rakentaa edellisessä luvuissa luoduista resurssitiedostojen sisältämistä avainsanoista testitapauksia.



Kuva 27. Testona export -asetukset.

4.6.1 Testien suorittaminen Robot Frameworkia käyttäen

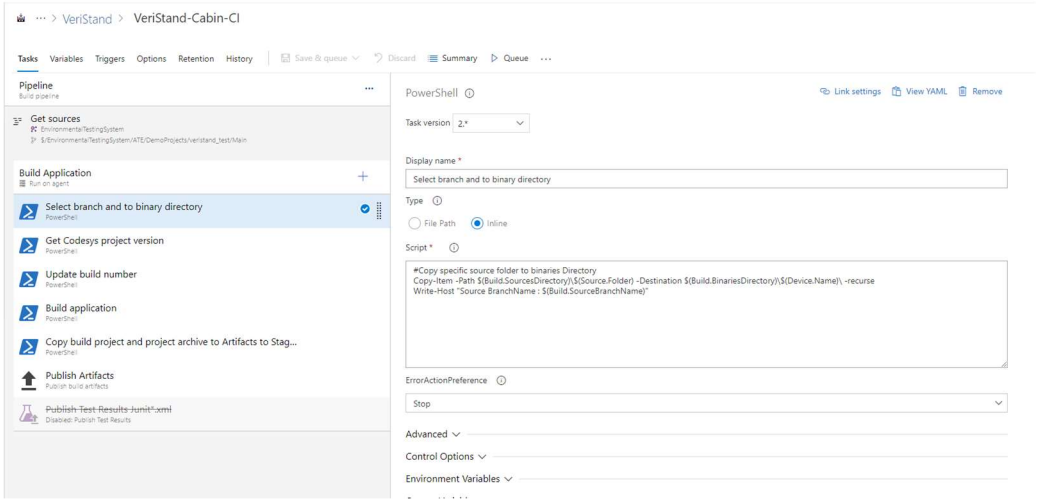
Edellisessä luvussa saadaan luotua testitapauksen runko syötteineen. Tähän runkoon voidaan laatia luvussa 4.7.1 kuvatuista resurssitiedostoista suoritettavia testitapauksia. Resurssitiedostot sisältävät tarpeelliset avainsanat eri testiaskeleiden suorittamiseen. Testiajon tuloksena on xUnit-muotoinen raportti, joka voidaan liittää kyseisen testin tuloksiin automaattisesti. Seuraavassa luvussa käydään tarkemmin läpi testien suoritusta ja raportointia.

4.7 CI/CD-putki testien suorittamista varten

Epec oy:ssä on käytössä Azure DevOps Server -palvelin. ADO-palvelinta on käytetty aiemmin eri projekteissa kääntämiseen sekä testien suorittamiseen. Tästä johtuen se oli myöskin luonnollisin vaihtoehto tässä työssä luotujen automatisoitujen testien ja CI/CD-putken toteuttamiseen.

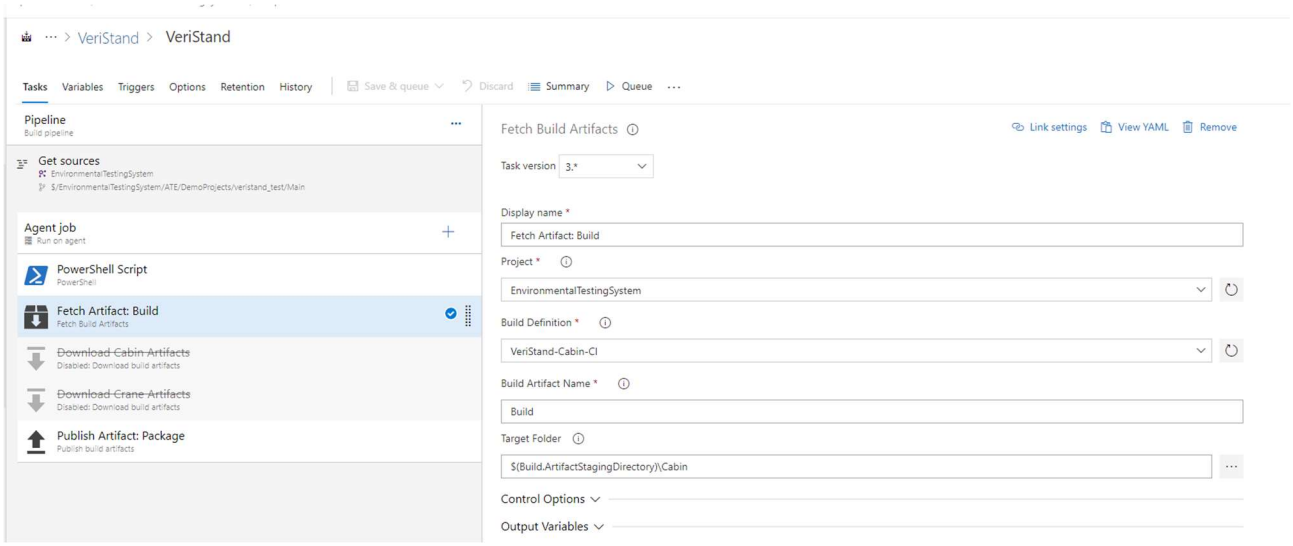
4.7.1 Käännösputket

Ensimmäisessä vaiheessa käännetään testattava CODESYS-sovellus suoritettavaksi binääri-tiedostoksi, joka voidaan ladata ohjauslaitteeseen. Sovelluksen käännös laukaistaan automaattisesti käyntiin, kun versionhallintaan tallennetaan uusi versio. Käännös suoritetaan CODESYS-ohjelmiston avulla Pythonia käyttäen. Kun käännös on saatu suoritettua, julkaistaan sen tuloksena syntyvät artifaktit palvelimelle, mistä seuraava käännösvaihe pystyy poimimaan ne.



Kuva 28. VeriStand-Cabin-CI-käännösputki ja sen eri tehtävät

Kun VeriStand-Cabin-CI-käännös on saatu suoritettua, julkaistaan sen tuottamat käännösartifaktit Azure DevOps -palvelimelle. Seuraavassa vaiheessa VeriStand-putkessa poimitaan äskeiset artifaktit sekä versionhallintaan tallennetut Robot Framework -skriptit sekä sovellus- ja firmwareohjelmistojen päivitykseen käytettävät Python-skriptit että testilaitteiston teholähteen ohjaukseen käytettävä Python-skripti. Näistä tiedostoista muodostetaan artifakti, joka voidaan poimia julkaisuputkessa.



Kuva 29. VeriStand-julkaisuputki.

4.7.2 Julkaisuputki

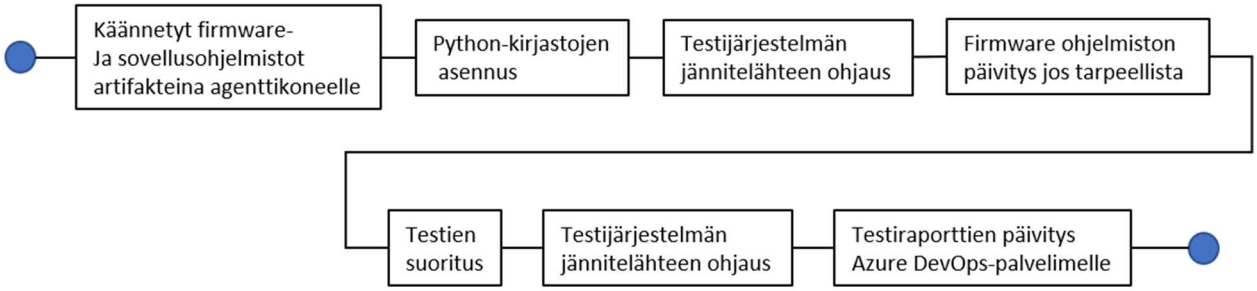
Julkaisuputki koostuu erilaisista tehtävistä, jotka sisältävät käyttäjän määrittelemiä komentoja. Nämä tehtävät voivat olla esimerkiksi Python-kielellä kirjoitettuja skriptejä. Julkaisuputki tullaan suorittamaan HIL-testeihin määritetyllä tietokoneella, joka on yhteydessä testerilaitteistoon. Tällä koneella on myös VeriStand-ohjelmisto asennettuna, jolloin se pystyy käyttämään VeriStand-yhteyskäytävä sovellusta. Yhteyskäytävä ja lisenssi vaaditaan yhteyden muodostamiseen testerilaitteistoon.

Aluksi käännösartifaktit noudetaan agenttikoneelle. Kun tiedostot on siirretty agenttikoneen hakemistossa, se voi suorittaa putkeen määritetyt tehtävät. Tämän jälkeen asennetaan agenttikoneelle testien suoritukseen tarvittavat Python-kirjastot.

Seuraavaksi ohjataan testerin virtalähde päälle siihen tehtyä Python-skriptiä käyttäen. Kun laitteet löytyvät väylältä, niiltä voidaan kysyä firmware- sekä sovellusversiot. Jos artifakteina on saatu uudemmat versiot kuin vastausten perusteella laitteilla on, voidaan ohjelmistot päivittää omalla skriptillään.

Ohjelmiston päivityksen jälkeen suoritetaan testit Robot Frameworkia ja muodostettua testiohjelmaa käyttäen. Suoritettavat testiohjelmat ovat versionhallinnassa, josta ne poimitaan artifaktina agenttikoneelle.

Kun testit on suoritettu, ohjataan testerilaitteiston jännitelähde pois päältä. Viimeisessä tehtävässä päivitetään testiraportit suoritetuista testeistä ja hyväksytään julkaisu.



Kuva 30. Julkaisuputken eri tehtävät.

5 Yhteenveto ja pohdinta

Työn aiheena oli tutkia yrityksessä olevan testilaitteiston käyttöä sekä miettiä testauksen automatisoinnin mahdollisuuksia. Työssä tutustuttiin myös testitapausten ja -datan suunnitteluun luokittelupuumenetelmän avulla. Tarkoituksena oli saada luotua eräänlainen testikehys, jolla voidaan toteuttaa erilaisten järjestelmien automaattinen testaus ilman että kaikkea tarvitsee suunnitella jokaisella kerralla alusta asti uudelleen.

Kun työn aiheeksi oli valikoitunut testilaitteiston käytön tutkiminen, oli seuraava asia miettiä mikä olisi tyypillinen käyttötapaus Epec-ohjauslaitteille. Keskustelemalla asiakastuen kanssa sellaiseksi valikoitui venttiilinohjaus.

Testattavaan järjestelmään päätettiin myös lisätä turvatoiminnoksi ohjaamon oven tarkkailu. Järjestelmäksi muodostui EC44-ohjauslaite, jolla luettiin ovikytkimen tila sekä puomia ohjaavan joystickin asento. Ohjaus ylä- tai alasuuntaan toteutetaan Epec GL84 CANOpen slave -laitteella. Tämä laite ohjaa venttiililohkoa proportionaalisella virtaohjauksella.

Järjestelmälle haluttiin heti alusta asti kehittää automaattiset testit sekä tarvittavat muut toiminnot CI/CD-putkea käyttäen. Työssä tutkittiin myös mahdollisuutta toteuttaa putki avoimen lähdekoodin työkaluja käyttäen. Työ aloitettiin tutkimalla testauksen ja jatkuvan integroinnin teoriaa. Tämän jälkeen tutustuttiin testijärjestelmään sekä sille mahdollisesti löytyviin rajapintoihin. Testirajapinnaksi valikoitui Veristand Python -kirjasto, jonka kehitystä National Instruments tukee. Tässä vaiheessa tutkittiin rajapinnan käyttöä sekä sen käyttöämisen mahdollisuutta Robot Frameworkin kanssa automatisoinnin toteuttamiseksi.

Testien suunnitteluun käytettiin luokittelupuumenetelmää, jolla saadaan tunnistettua testikohteen toimintaan vaikuttavat eri ominaisuudet, sekä jaettua ominaisuuksiin vaikuttavat suureet erillisiin ekvivalenssiluokkiin. Testit suunniteltiin erillistä tarkoitukseen suunniteltua ohjelmaa käyttäen. Ohjelmalla laadittiin luokittelupuu, josta pystyttiin automaattisesti luomaan testitapaukset syötteineen. Nämä luodut testitapaukset muodostavat testipohjan Robot Frameworkiin, missä varsinaiset avainsanoihin perustuvat testitapaukset määritellään. Seuraavaksi toteutettiin testattavan järjestelmän johdotus ja kokoaminen sekä tehtiin siihen yksinkertainen testattava sovellus CODESYS-ohjelmointiympäristöä käyttäen.

CI/CD-toteutusta varten järjestelmälle oli luotava oma julkaisuputki. Julkaisuputki luotiin jo käytössä olevalle Azure DevOps -palvelimelle. Palvelimella suoritetaan automaattisen testin kaikki vaiheet. Vaiheisiin kuuluu viisi eri osiota: a) firmware- ja sovellusohjelmistojen käännettyjen versioiden haku, b) tarvittavien Python-kirjastojen asennus, c) testijärjestelmän jännitelähteen ohjaus siihen kehitetyllä Python-skriptillä, d) firmware- ja sovellusohjelmistojen päivitys, joihin molempiin myös suunniteltiin oma Python-skriptinsä ja e) itse testien suoritus sekä testiraporttien kerääminen ja tuloksien julkistaminen. Nämä eri vaiheet suoritetaan niin sanotulla agentti-koneella. Agentti on ohjelmisto, joka asennetaan valitulle koneelle, joka sitten pystyy ottamaan komentoja vastaan ADO-palvelimelta sen suorittaessa putkessa olevia tehtäviä.

Lopputuloksena saatiin ratkaisu, jossa agenttikoneelle on tarpeellista määritellä ainoastaan VeriStand-yhteyskäytävä toimintaan. Kaikki muut ohjelmistot toimitetaan koneelle putken alkuvaiheissa. Tästä on se hyöty, että agentti-koneena voidaan periaatteessa käyttää mitä konetta tahansa, jolla on pääsy samaan sisäverkkoon, kuin missä testilaitteisto sijaitsee. Myöskään testien suorittamiseen tai muokkaamiseen ei vaadita kalliita erillisiä ohjelmistolisenssejä, vaan niitä pystyvät muokkaamaan kaikki kehitysprojektin jäsenet. Uudella testerilaitteistolla pystytään testaamaan useamman ohjauslaitteen kokonaisuuksia. Lisäksi käyttämällä erillisiä viansyöttökortteja pystytään jo kehitysvaiheessa saamaan esille vikoja, jotka voisivat vaatia erillisen mekaniikan käyttöä. Uuden laitteiston käyttämä alusta on realiaikainen ja tarjoaa siten paremman suorituskyvyn sekä myöskin paremman laajennettavuuden. Alusta mahdollistaa myös erillisten mallien käytön testattavan järjestelmän simulointiin.

Tässä työssä on tutkittu vain testerilaitteiston ohjelmistopuolta testien automatisoinnin kannalta. Työssä ei ole otettu kantaa tulevaisuuden tarpeisiin koskien testilaitteiston laajentamista uusilla moduuleilla tai niiden hankinnan suunnittelua. Kun uusien laajennusmodulien tarve on tunnistettu ja niiden käyttätapa on suunniteltu, voidaan testien automatisointia suunnitella edelleen.

Yhtenä kehitysmahdollisuutena voidaan pitää Asam XIL -rajapinnan käyttöä testien suoritukseen. Toisena kehitysmahdollisuutena voidaan pitää tulevaisuudessa käyttöön tulevia virtualisointimahdollisuuksia. Tässä mahdollisuutena on ohjauslaitteen ohjelmistolla

tuotetun niin sanotun kaksosen liittäminen HIL-testausjärjestelmän osaksi. Tällöin pystytään osa järjestelmästä simuloimaan ja osa on yhteydessä HIL-testausjärjestelmään. Simuloitu osa voi olla laite, jota ei vielä ole olemassa fyysisenä laitteena. Kolmantena kehitysmahdollisuutena on erillisen liitinrajapinnan käyttö. Liitinrajapintaa käyttämällä voidaan testattavaa järjestelmää vaihtaa nopeasti. Kun testattava järjestelmä on vaihdettu, voidaan uusi liityntäkonfiguraatio ladata testerille testiputkessa.

LÄHTEET

Allen, J., Mashesh, B., Nabi, S.& Rzemien, K. 2004. An Overview of Hardware-In-the-Loop Testing Systems at Visteon. [Verkkojulkaisu]. [Viitattu 25.4.2021]. Saatavana ResearchGate -tietokannasta. Vaatii käyttöoikeuden.

Black, R., Van Veenendaal, E. & Graham, D. 2009. Foundations of software testing: ISTQB Certification. Third edition. Andover: Cengage Learning

Brayanov, N.& Stoyanova, 2019. A. Review of hardware-in-the-loop - a hundred years progress in the pseudo-real testing. Electrotechnica & Electronica 54. [Verkkojulkaisu]. [Viitattu 19.4.2021]. Saatavana ResearchGate -tietokannasta. Vaatii käyttöoikeuden.

Broekman, B. & Notenboom, E. 2003. Testing embedded software. Harlow: Pearson Education limited

Büchner, F. 2019. Test Case Design Using the Classification Tree Method TESSY White Paper [Verkkojulkaisu]. [Viitattu 15.3.2021]. Saatavana: https://www.hitex.com/fileadmin/documents/tools/dynamic/tessy/WP-TESSY-Test-Case-Design-Using-CTM.pdf

Codesys, Ei päiväystä. Codesys GmbH. [Verkkosivu]. Codesys GmbH. [Viitattu 15.1.2021] Saatavana: https://store.codesys.com/en/codesys.html

Conrad, M. 2001. An Extension of the Classification-Tree Method for Embedded Systems for the Description of Events. [Verkkojulkaisu]. [Viitattu 21.9.2021]. Saatavana ResearchGate-tietokannasta. Vaatii käyttöoikeuden.

Conrad, M. 2004. Systematic Approach to Testing Automotive Control Software. [Verkkojulkaisu]. [Viitattu 20.9.2021]. Saatavana ResearchGate-tietokannasta. Vaatii käyttöoikeuden.

Epec. Ei päiväystä. Epec programming and libraries manual: How to validate 2 channel DI. [Verkkosivu]. Epec Oy. [Viitattu 15.1.2022]. Saatavana: https://extranet.epec.fi/Public/Manuals/EPEC_Programming_And_Libraries/epecprogrammingandlibrariesmanual_man000538.htm?#t=projecttopics%2Ftopic100376.htm&rhsearch=validate&rhhlterm=validate&rhsyns=%20

Epec. Ei päiväystä. Epec programming and libraries manual: How to validate 2 channel joystick. [Verkkosivu]. Epec Oy. [Viitattu 16.1.2022]. Saatavana: https://extranet.epec.fi/Public/Manuals/EPEC_Programming_And_Libraries/epecprogrammingandlibrariesmanual_man000538.htm?#t=projecttopics%2Ftopic100384.htm&rhsearch=validate&rhhlterm=validate&rhsyns=%20

Epec. Ei päiväystä. Epec company presentation. [Verkkojulkaisu]. Epec Oy. [Viitattu 5.2.2021]. Saatavilla: Vain yrityksen sisäisessä käytössä.

Epec. 24.5.2018. Epec training project. [Verkkojulkaisu]. Epec Oy. [Viitattu 6.9.2020]. Saatavilla: Vain yrityksen sisäisessä käytössä.

Epec. Ei päiväystä. EPEC EC44 CONTROL UNIT. [Verkkosivu]. Epec Oy. [Viitattu 27.1.2022]. Saatavana: https://epec.fi/products/control-system-products-ec44/

Epec. Ei päiväystä. EPEC GL84 SLAVE UNIT. [Verkkosivu]. Epec Oy. [Viitattu 18.1.2022]. Saatavana: https://epec.fi/products/control-system-products-gl84/

Epec. Ei päiväystä. Software Development Environment. [Verkkosivu]. Epec Oy. [Viitattu 19.1.2022]. Saatavana: https://epec.fi/epec-oy-products/application-development-environment/

Hass, A., M., J. 2008. Guide to advanced software testing. Norwood: Artech House, Inc.

Joshi, A. 2020. Automotive Applications of Hardware-in-the-Loop (HIL) Simulation. Warrendale: SAE International

Kananen, J. 2012. Kehittämistutkimus opinnäytetyönä: Jyväskylän ammattikorkeakoulu.

Korpela, J. 2001. Perustietoa Python ohjelmointikielestä. [Verkkosivu]. [Viitattu 22.1.2022] Saatavana: https://jkorpela.fi/python/

Kruse, P. 2014. Enhanced Test Case Generation with the Classification Tree Method. [Verkkojulkaisu]. [Viitattu 31.1.2022]. Saatavana: https://d-nb.info/1060717921/34

Kruse, P.& Zeppetzauer U. 2014. Automating Test Case Design within the Classification Tree Editor. Proceedings of the 2014 Federated Conference on Computer Science and Information Systems. [Verkkojulkaisu]. [Viitattu 31.1.2022]. Saatavana: https://annals-csis.org/Volume_2/pliks/263.pdf

Laukkanen, E., Itkonen, J., & Lassenius, C. 2017. Problems, Causes and Solutions When Adopting Continuous Delivery - A Systematic Literature Review. [Verkkojulkaisu]. [Viitattu 19.4.2021]. Saatavana ResearchGate -tietokannasta. Vaatii käyttöoikeuden.

Microsoft. Ei päiväystä. What is Azure DevOps? [Verkkojulkaisu]. Microsoft Corp. [Viitattu 30.1.2022]. Saatavana: https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

Microsoft. Ei päiväystä. What is Azure Pipelines? [Verkkojulkaisu]. Microsoft Corp. [Viitattu 11.11.2021]. Saatavana: https://docs.microsoft.com/en-us/azure/devops/pipelines/get-started/what-is-azure-pipelines?view=azure-devops

Microsoft. Ei päiväystä. What is Azure Boards? [Verkkojulkaisu]. Microsoft Corp. [Viitattu 10.11.2021]. Saatavana: https://docs.microsoft.com/en-us/azure/devops/boards/get-started/what-is-azure-boards?view=azure-devops&tabs=agile-proc

Microsoft. Ei päiväystä. About agents & agent pools. [Verkkojulkaisu]. Microsoft Corp. [Viitattu 24.11.2021]. Saatavana: https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents?view=azure-devops&viewFallbackFrom=azdevops&tabs=browse

Microsoft. Ei päiväystä. Azure Artifacts. [Verkkosivu]. Microsoft Corp. [Viitattu 12.11.2021]. Saatavana: https://azure.microsoft.com/en-us/services/devops/artifacts

Python Package Index. Ei päiväystä. Find, install and publish Python packages with the Python Package Index. [Verkkosivu]. [Viitattu 23.1.2022] Saatavana: https://www.python.org/about/

PyVISA. Ei päiväystä. PyVISA: Control your instruments with Python. [Verkkosivu]. [Viitattu 27.1.2022]. Saatavana: https://pyvisa.readthedocs.io/en/latest/

Pont, M., J. & Short, M. 2008. Assessment of High-Integrity Embedded Automotive Control Systems using Hardware in the Loop Simulation. [Verkkolehtiartikkeli]. Journal of Systems and Software 81. [Viitattu 11.3.2021]. Saatavana ResearchGate -tietokannasta. Vaatii käyttöoikeuden.

Pärnänen, E. 2020. Customer service manager. Epec Oy. Keskustelu. 1.9.2020.

Robot Framework. Ei päiväystä. Robot Framework. [Verkkosivu]. [Viitattu 27.1.2022]. Saatavana: https://robotframework.org/

RMX-4125. Ei päiväystä. RMX-4125 Programmable Power Supply Device. [Verkkosivu]. [Viitattu 2.2.2022]. Saatavana: https://www.ni.com/fi-fi/support/model.rmx-4125.html

RMX-10051. Ei päiväystä. RMX-10051 Power Distribution Device. [Verkkosivu]. [Viitattu 2.2.2022]. Saatavana: https://www.ni.com/fi-fi/support/model.rmx-10051.html

Schlager, M., Obermaisser, R.& Elmenreich, W. 2007. A Framework for Hardware-in-the-Loop Testing of an Integrated Architecture. [Verkkojulkaisu]. [Viitattu 17.3.2021]. Saatavana ResearchGate -tietokannasta. Vaatii käyttöoikeuden.

Schlager, M., Elmenreich, W.& Wenzel, I. 2006. Interface Design for Hardware-in-the-Loop Simulation [Verkkojulkaisu]. Proceedings of the 2006 IEEE International Symposium on Industrial Electronics. [Verkkolehtiartikkeli]. [Viitattu 11.3.2021]. Saatavana ResearchGate -tietokannasta. Vaatii käyttöoikeuden.

SLSC. 2022. Save Time and Maximize Reuse in HIL Testing with the SLSC Extension for PXI and CompactRIO. [Verkkosivu]. [Viitattu 1.2.2022]. Saatavana: https://www.ni.com/fi-fi/innovations/white-papers/18/save-time-and-maximize-reuse-in-hil-testing-with-the-slsc-extens.html

Testona. Ei päiväystä. Test design using the classification tree method. [Verkkosivu]. [Viitattu 23.1.2022] Saatavana: https://products.expleogroup.com/testona/classification-tree-method/

VeriStand. Ei päiväystä. What Is VeriStand? [Verkkosivu]. [Viitattu 26.1.2022] Saatavana: https://www.ni.com/fi-fi/shop/data-acquisition-and-control/application-software-for-data-acquisition-and-control-category/what-is-veristand.html

Veristand Python. Ei päiväystä. Veristand Python. [Verkkosivu]. National Instruments. [Viitattu 27.1.2022]. Saatavana: https://niveristand-python.readthedocs.io/en/latest/

VISA. Ei päiväystä. What Is VISA? [Verkkosivu]. [Viitattu 25.1.2022]. Saatavana: https://www.tek.com/en/support/faqs/what-visa

What is PXI? Ei päiväystä. What is PXI? [Verkkosivu]. [Viitattu 1.2.2022] Saatavana: https://www.ni.com/fi-fi/shop/pxi.html

LIITTEET

Liite 1: Testona-ohjelmalla luotu testitapauspohja

Liite 2: Testitapauspohja johon on määritetty Robot Framework -testiaskeleet

Liite 1: Testona-ohjelmalla luotu testitapauspohja

```
*** Settings ***
Resource    ../../VS/SystemDefinition.robot
Suite Setup
Suite Teardown

*** Variables ***


*** Test Cases ***
DoorSwitch Error (ChA = ChB = False)
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = False
     ...                    DoorSwitch_i = False
     ...                    DoorSwitch.Out = Error
     ...                    Joystick_Voltage = BoomControlUp (5025, 9000]
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = 0


DoorSwitch Open (BoomControlDown)
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = False
     ...                    DoorSwitch_i = True
     ...                    DoorSwitch.Out = Open
     ...                    Joystick_Voltage = BoomControlDown [1000, 4975)
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = 0


DoorSwitch Closed Joystick Center
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = True
     ...                    DoorSwitch_i = False
     ...                    DoorSwitch.Out = Closed
     ...                    Joystick_Voltage = Deadband [4975, 5025]
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = 0
```

DoorSwitch Closed Joystick Up
[Tags] ready
[Setup] Startup System
[Teardown] Shutdown System
[Documentation] Generated from Testona
... DoorSwitch = True
... DoorSwitch_i = False
... DoorSwitch.Out = Closed
... Joystick_Voltage = BoomControlUp (5025, 9000]
... BoomControl.CurrentRequestDown = 0
... BoomControl.CurrentRequestUp = [1, 1000]

DoorSwitch Closed Joystick Down
[Tags] ready
[Setup] Startup System
[Teardown] Shutdown System
[Documentation] Generated from Testona
... DoorSwitch = True
... DoorSwitch_i = False
... DoorSwitch.Out = Closed
... Joystick_Voltage = BoomControlDown [1000, 4975)
... BoomControl.CurrentRequestDown = [1, 1000]
... BoomControl.CurrentRequestUp = 0

DoorSwitch Error (ChA = ChB = True)
[Tags] ready
[Setup] Startup System
[Teardown] Shutdown System
[Documentation] Generated from Testona
... DoorSwitch = True
... DoorSwitch_i = True
... DoorSwitch.Out = Error
... Joystick_Voltage = BoomControlDown [1000, 4975)
... BoomControl.CurrentRequestDown = 0
... BoomControl.CurrentRequestUp = 0

DoorSwitch Open (BoomControlUp)
[Tags] ready koe
[Setup] Startup System
[Teardown] Shutdown System
[Documentation] Generated from Testona
... DoorSwitch = False
... DoorSwitch_i = True
... DoorSwitch.Out = Open
... Joystick_Voltage = BoomControlUp (5025, 9000]

```
...         BoomControl.CurrentRequestDown = 0
...         BoomControl.CurrentRequestUp = 0
```

Liite 2: Testitapauspohja johon on määritetty Robot Framework -testiaskeleet

```
*** Settings ***
Resource    ../../VS/SystemDefinition.robot
Suite Setup
Suite Teardown

*** Variables ***


*** Test Cases ***
DoorSwitch Error (ChA = ChB = False)
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = False
     ...                    DoorSwitch_i = False
     ...                    DoorSwitch.Out = Error
     ...                    Joystick_Voltage = BoomControlUp (5025, 9000]
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = 0
     Set  Joystick_input  8
     Sleep    1s
     Set  DoorSwitch  0
     Set channel fault    DoorSwitch_i    0
     Sleep    0.1s
     Test    CR_CylinderDown   0
    Test    CR_CylinderUp     0
    Test    CylinderDown_Duty    0
    Test    CylinderUp_Duty    0


DoorSwitch Open (BoomControlDown)
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = False
     ...                    DoorSwitch_i = True
     ...                    DoorSwitch.Out = Open
     ...                    Joystick_Voltage = BoomControlDown [1000, 4975)
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = 0
     Set  Joystick_input  2
     Sleep    1s
     Set  DoorSwitch  0
     Sleep    0.1s
     Test    CR_CylinderDown   0
```

```
    Test    CR_CylinderUp     0
    Test    CylinderDown_Duty    0
    Test    CylinderUp_Duty    0

DoorSwitch Closed Joystick Center
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = True
     ...                    DoorSwitch_i = False
     ...                    DoorSwitch.Out = Closed
     ...                    Joystick_Voltage = Deadband [4975, 5025]
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = 0
     Set  Joystick_input  5
     Sleep    1s
     Test    CR_CylinderDown   0
    Test    CR_CylinderUp     0
    Test    CylinderDown_Duty    0
    Test    CylinderUp_Duty    0

DoorSwitch Closed Joystick Up
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = True
     ...                    DoorSwitch_i = False
     ...                    DoorSwitch.Out = Closed
     ...                    Joystick_Voltage = BoomControlUp (5025, 9000]
     ...                    BoomControl.CurrentRequestDown = 0
     ...                    BoomControl.CurrentRequestUp = [1, 1000]
    Set  Joystick_input  8
     Sleep    1s
     Test    CR_CylinderDown   0
    Test In Limits    CR_CylinderUp    1    1000
    Test    CylinderDown_Duty    0
    Test in limits   CylinderUp_Duty    1   100

DoorSwitch Closed Joystick Down
    [Tags]  ready
    [Setup]  Startup System
    [Teardown]  Shutdown System
     [Documentation]  Generated from Testona
     ...                    DoorSwitch = True
     ...                    DoorSwitch_i = False
     ...                    DoorSwitch.Out = Closed
     ...                    Joystick_Voltage = BoomControlDown [1000, 4975)
```

```
    ...                     BoomControl.CurrentRequestDown = [1, 1000]
    ...                     BoomControl.CurrentRequestUp = 0
    Set  Joystick_input  2
    Sleep    1s
   Test In Limits    CR_CylinderDown    1    1000
    Test    CR_CylinderUp   0
    Test in limits   CylinderDown_Duty    1   100
   Test    CylinderUp_Duty    0

DoorSwitch Error (ChA = ChB = True)
   [Tags]  ready
   [Setup]  Startup System
   [Teardown]  Shutdown System
    [Documentation]  Generated from Testona
    ...                     DoorSwitch = True
    ...                     DoorSwitch_i = True
    ...                     DoorSwitch.Out = Error
    ...                     Joystick_Voltage = BoomControlDown [1000, 4975)
    ...                     BoomControl.CurrentRequestDown = 0
    ...                     BoomControl.CurrentRequestUp = 0
    Set  Joystick_input  2
    Set  DoorSwitch  5
   Sleep    1s
    Set channel fault    DoorSwitch_i    5
    Sleep    0.1s
    Test    CR_CylinderDown   0
   Test    CR_CylinderUp     0
   Test    CylinderDown_Duty    0
   Test    CylinderUp_Duty    0

DoorSwitch Open (BoomControlUp)
   [Tags]  ready  koe
   [Setup]  Startup System
   [Teardown]  Shutdown System
    [Documentation]  Generated from Testona
    ...                     DoorSwitch = False
    ...                     DoorSwitch_i = True
    ...                     DoorSwitch.Out = Open
    ...                     Joystick_Voltage = BoomControlUp (5025, 9000]
    ...                     BoomControl.CurrentRequestDown = 0
    ...                     BoomControl.CurrentRequestUp = 0
    Set  Joystick_input  8
    Sleep    1s
    Set  DoorSwitch  0
    Sleep    0.1s
    Test    CR_CylinderDown   0
   Test    CR_CylinderUp     0
   Test    CylinderDown_Duty    0
   Test    CylinderUp_Duty    0
```