OULUN AMMATTIKORKEAKOULU

Timi Partala

FOIL ROLL MARK DETECTOR -LAITTEEN TESTAUS

FOIL ROLL MARK DETECTOR -LAITTEEN TESTAUS

Timi Partala
Opinnäytetyö
Syksy 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, laite- ja tuotesuunnittelu

Tekijä: Timi Partala
Opinnäytetyön nimi: Foil Roll Mark Detector -laitteen testaus
Työn ohjaaja: Olli Himanka
Työn valmistumislukukausi ja -vuosi: Syksy 2023 Sivumäärä: 23

Opinnäytetyön aiheena oli testata Foil Roll Mark Detectorin toiminta yksikkö-, integraatio- sekä järjestelmätestauksen osilta. Foil Roll Mark Detector on yrityksen ensimmäinen laite, joka mittaa ja tarkastelee alumiinifolion pintaa virheiden varalta. Tällaisen virheen löytäminen ehkäisee folion repeämisen jatkojalostuksessa, mikä säästää asiakkaalta rahaa ja aikaa. Opinnäytetyössä kerrotaan testausmenetelmistä ja testauksessa huomatuista ongelmista. Testattavia asioita yksikkötestauksen lisäksi ovat muun muassa moottorin, kaapeleiden, DSP-kortin ja käyttöliittymän testit. Työssä myös esitellään ohjelmia, joita projektissa sekä testauksessa käytetään.

Opinnäytetyön tuloksena saatiin testattua suurin osa yksikkötesteistä ja integraatiotesteistä sekä osa järjestelmätesteistä. Testaukset jatkuvat edelleen opinnäytetyön jälkeen.

Asiasanat: yksikkötestaus, integraatiotestaus, järjestelmätestaus, Linux, SQL

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Device and Product Design

Author(s): Timi Partala
Title of thesis: Foil Roll Mark Detector testing
Supervisor(s): Olli Himanka
Term and year when the thesis was submitted: Autumn 2023
Number of pages: 23

The topic of the thesis was to test the operation of the Foil Roll Mark Detector in terms of unit testing, integration testing, and system testing. The Foil Roll Mark Detector will detect errors from aluminium foil before the error causes the foil to break. The thesis discusses these testing methods and the issues observed during testing. In addition to unit testing, the things tested include the motor, cables, DSP card, and user interface, among others. The thesis also introduces the programs used in the project and testing.

As a result of the thesis, the majority of unit and integration tests were successfully conducted, along with some system tests. Testing will continue beyond the completion of the thesis.

Keywords: unit testing, integration testing, system testing, Linux, SQL

SISÄLLYS

SANASTO

DSP-kortti	Digitaalinen signaaliprosessorikortti
FRMD	Foil Roll Mark Detector -laite
Git	Versionhallintajärjestelmä
GNU GPL -lisenssi	Avoimen lähdekoodin lisenssi
I/O-kortti	Input/Output -kortti
Kernel	Käyttöjärjestelmän ydinosa
Qt	Kehitysympäristö erilaisten alustojen sovellusten luontiin
Roll Mark / valssimerkki	Tasaisesti löytyvä virhe foliosta
Root-käyttäjä	Linuxissa pääkäyttäjä, jolla on kaikki oikeudet
Sarjaväyläkortti	PCIe-kortti, joka käyttää rs232-sarjaväylää
Shell-skripti	Komentoriviskripti, joka ajaa useampia komentoja kerralla
Ubuntu 22.04	Linux Debian -pohjainen käyttöjärjestelmä
Yksikkötesti	Ohjelman testausmenetelmä, jossa ohjelmakoodi testataan osissa eli yksiköissä ohjelman toimivuuden varmistamiseksi

1 JOHDANTO

Opinnäytetyön aiheena on testata Foil Rollmark Detectorin (FRMD) toiminta käyttäen erilaisia testausmenetelmiä. FRMD on yrityksen ensimmäinen laite, joka mittaa alumiinifolion pinnasta virheitä, jotka olisivat liian vaikeita ja aikaavieviä havaita ilman laitetta. Folion pinnassa olevat virheet johtuvat useimmiten tehtaan työrullassa olevasta rikkoumasta, joka tuottaa folion pintaan toistuvan virheen, jota laite sitten etsii.

Laite asennetaan tehtaaseen alumiinifolion ala- tai yläpuolelle, jossa sensori liikkuu johteessa servon avulla edestakaisin ottaen folion pinnasta näytteitä. Sensorin keräämästä datasta saadaan selville mahdolliset toistuvat virheet eli valssimerkit folion pinnasta, jotka sitten ilmoitetaan käyttöliittymässä käyttäjälle vakavuusasteen mukaan. Asiakas voi tällöin halutessaan pysäyttää tuotantolinjan ja tarkastella virhettä tarkemmin itse ja arvioida, onko virhe liian iso folion jatkojalostettavuuden kannalta.

Projektissa on tehty tähän asti käyttöliittymää laitteelle sekä koodattu tarvittavat toiminnallisuudet. Koodi on testattu yksikkötesteillä, mutta muut tulevaisuudessa koodattavat asiat, komponentit ja järjestelmä pitää vielä testata erikseen.

Testausmenetelminä käytetään muun muassa yksikkötestaamista, jolla testataan ohjelman toiminta pienissä osissa ja katsotaan, että ohjelma toimii oikein. Lisäksi tehdään integraatiotestausta, esimerkiksi moottoriohjaimen sarjaliikenneväylän testaamista, sekä järjestelmätestausta. Projektin suuruuden vuoksi testaaminen on jaettu useiden henkilöiden kanssa, jolloin osa testausmenetelmistä saattaa puuttua tekstistä.

Työn tilaajana toimii SR-Instruments Oy, joka valmistaa teräs- sekä alumiiniteollisuuteen mitta- ja laaduntarkastuslaitteita Oulussa. Yrityksen tuotteet auttavat teräs- ja alumiiniteollisuuden yrityksiä huomaamaan virheet tuottamissaan metalleissa ennen niiden jatkojalostamista, jolloin asiakkaille koituu vähemmän kuluja. Olen ollut SR-Instrumentsilla töissä vuoden 2022 kesästä alkaen, jolloin myös tämä projekti aloitettiin. Nyt on enää jäljellä laitteen testaaminen, josta teen tämän opinnäytetyön.

2 KÄYTETYT OHJELMAT JA TYÖKALUT

Ohjelmat, joita projektissa käytetään, on valittu yksinkertaisuuden, toimivuuden sekä hyvien kokemusten pohjalta. Näitä ohjelmia ja työkaluja yhdistää myös avoin lähdekoodi, jolloin ne ovat ilmaisia ja niihin on helppo löytää mahdollisia apuja ongelmien saapuessa.

2.1 Qt-ohjelmointiympäristö

Qt on alustariippumaton ohjelmointiympäristö, jolla voi tehdä graafisia käyttöliittymiä. Qt tukee useita kääntäjiä, muun muassa GCC:tä ja MinGW:tä. Qt:ssä on myös Qt Quick, joka sisältää deklaratiivisen skriptikielen QML:n, joka sallii JavaScriptin käyttämisen Qt:ssä. Qt:n helppo Designer-työkalu mahdollistaa nopean graafisen käyttöliittymän teon ohjelmaan, johon voi yhdistää C++:lla tehtyjä UI-elementtejä kätevästi. Qt tukee myös SQL-tietokannan käytön sekä useiden prosessorisäikeiden ajon ohjelmassa. (1.)

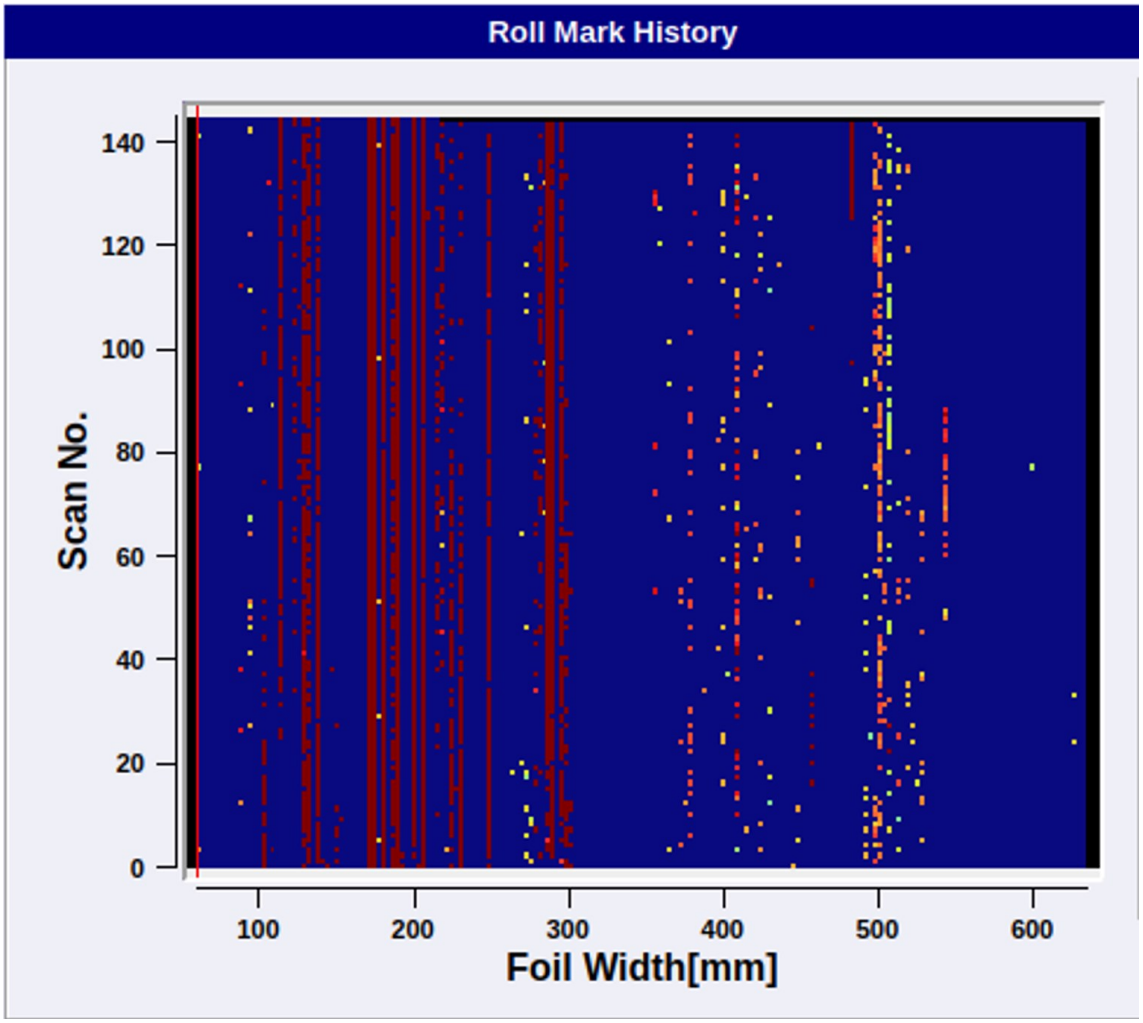
2.2 Ubuntu 22.04

Ubuntu on ilmainen Debian-pohjainen Linux-käyttöjärjestelmä, jonka on kehittänyt englantilainen yritys Canonical. Se on myös kevyempi ja nopeampi kuin Microsoftin Windows-käyttöjärjestelmät, mikä tekee siitä paremman vaihtoehdon. Ubuntusta on tehty kolme eri versiota, Desktop, Server ja Core, joista yleisin on Desktop eli niin sanottu normaali käyttöjärjestelmä.

Projektissa käytetään Linuxia yhteensopivuusongelmien vähentämiseksi, joita Windows-käyttöjärjestelmällä tulisi mahdollisesti muun muassa digitaalisen signaaliprosessorin (DSP) datankeruun kanssa. Tällöin tarvittaisiin erillinen Linux-kone, jonka kautta se voisi keskustella mittapään kanssa. Lisäksi projektissa käytetään aiemmin Linuxille tehtyjä koodeja muista projekteista, joita voidaan käyttää hyödyksi tuotteessa.

2.3 Qwt eli Qt Widgets for technical Applications

Qwt on avoimen lähdekoodin kirjasto Qt:lle, jossa on tehty valmiiksi erilaisia widgettejä sekä graafisen käyttöliittymän komponentteja. Qwt antaa mahdollisuuden tehdä muun muassa moniulotteisia kaavioita sekä lämpökarttoja, jotka ovat tärkeitä datan visualisoinnissa (kuva 1). (2.)



KUVA 1. Esimerkki Qwt:n lämpökartasta. Valssimerkin vakavuus vaaleasta tumman punaiseen. (Kuvakaappaus otettu FRMD-ohjelmasta.)

2.4 MariaDB ja DBeaver

MariaDB on general public license (GNU GPL) -lisensoitu yhteisön kehittämä MySQL:ään pohjautuva relaatiotietokantajärjestelmä. MariaDB:tä kehittää sama yhteisö, kuin MySQL:ää. Yhteisö yrittää pitää sen mahdollisimman yhteensopivana MySQL:n kanssa.

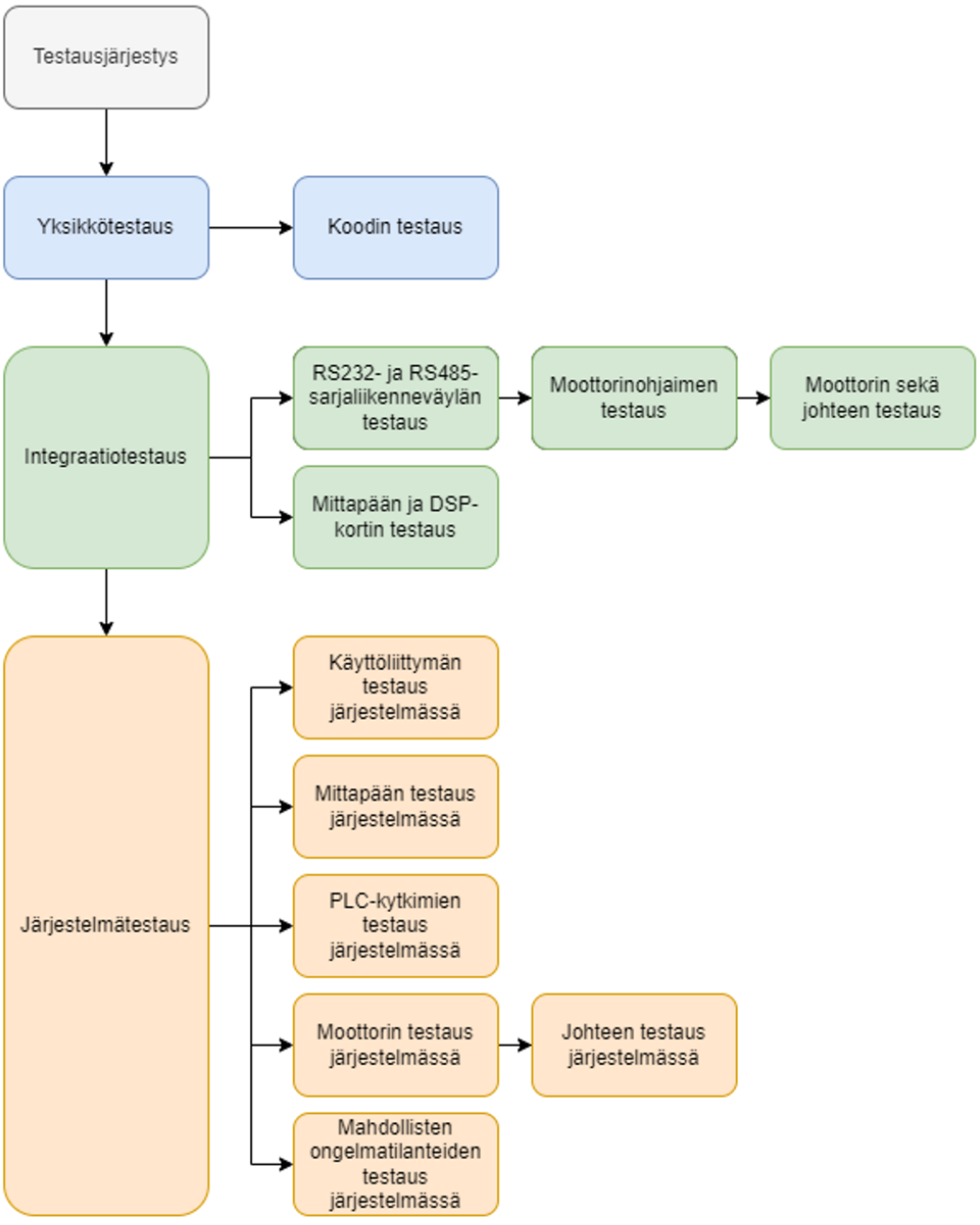
DBeaver CE on ilmainen SQL-tietokannan työkalu, jolla voi tehdä SQL-kyselyitä graafisessa käyttöliittymässä. Tämä helpottaa tietokantaparametrien asettamista ja muokkaamista. DBeaver on myös saatavilla Linuxille, Windowsille sekä MacOS:lle.

2.5 Oracle VM VirtualBox

Oracle VM VirtualBox on avoimen lähdekoodin virtualisointiohjelma, joka mahdollistaa useiden virtuaalikäyttöjärjestelmien ajon samanaikaisesti yhdellä tietokoneella. Se on tärkeä testaajille, jotka tarvitsevat eri käyttöjärjestelmiä eivätkä tarvitse erillistä konetta. Projektissa VM VirtualBoxia käytetään moottoriohjaimen testaukseen virtuaalisella Windows 10 -käyttöjärjestelmällä, koska testiohjelmaa ei ole saatavilla Linuxille.

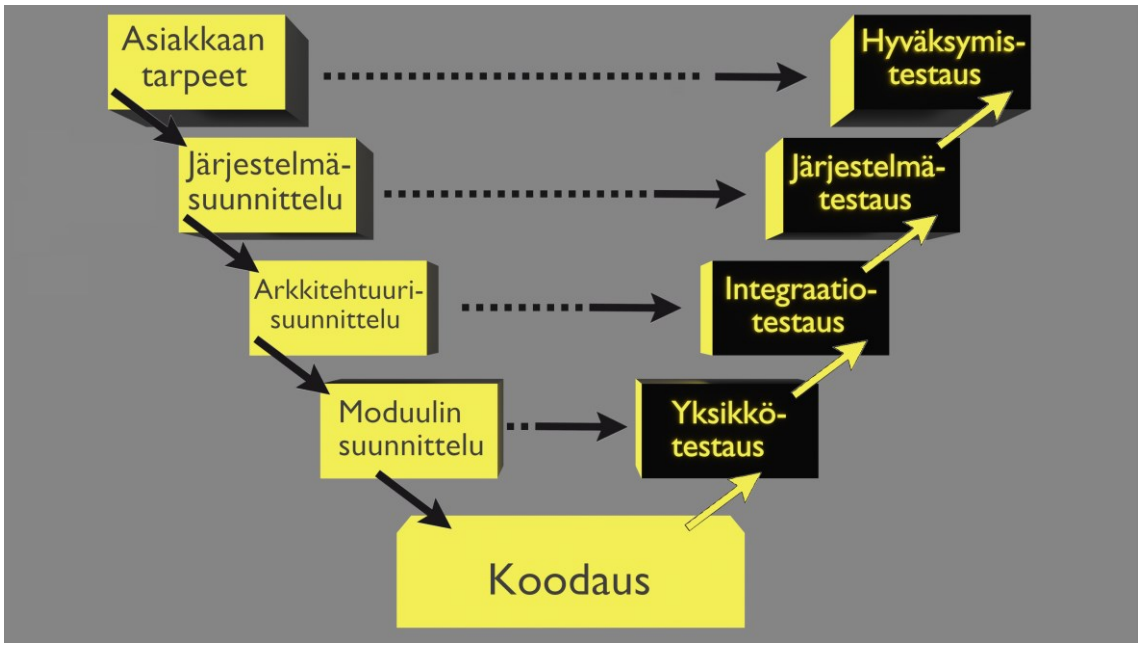
3 TESTAUKSEN TOTEUTUS

Laitteen testausta toteutettiin sitä mukaan, kun saatiin tarvittavat tiedot ja komponentit kasaan. Testattavat osuudet aloitettiin yksikkötesteistä ja lopetettiin järjestelmätestaukseen (kuva 2).



KUVA 2. Testauksen järjestys opinnäytetyössä.

Laitteen testaus on yksi keskeisimmistä projektin vaiheista. On tärkeää testata laite ja ohjelma hyvin ennen laitteen lähettämistä eteenpäin. Testausta tapahtuu koko projektin läpi, jolloin koko ohjelma pysyy aina toimivana, kun ohjelmaan lisätään tai siitä poistetaan asioita. Tähän soveltuu hyvin V-malli (kuva 3), joka on yksi ohjelmistokehityksen elinkaarimalleista (englanniksi Software Development life Cycle, SDLC). Se on edistyneempi muoto vesiputousmallista ja siinä testataan jokainen vaihe sitä mukaan, kun projektissa päästään eteenpäin. Tämä takaa toimivan koodin jokaisessa vaiheessa, toisin kuin vesiputousmallissa, jossa koodin testaus tehdään vasta, kun koodi on valmis. (3; 4; 5; 6.)



KUVA 3. V-malli ohjelmistokehityksen elinkaarimalleista.

3.1 Yksikkötestaus

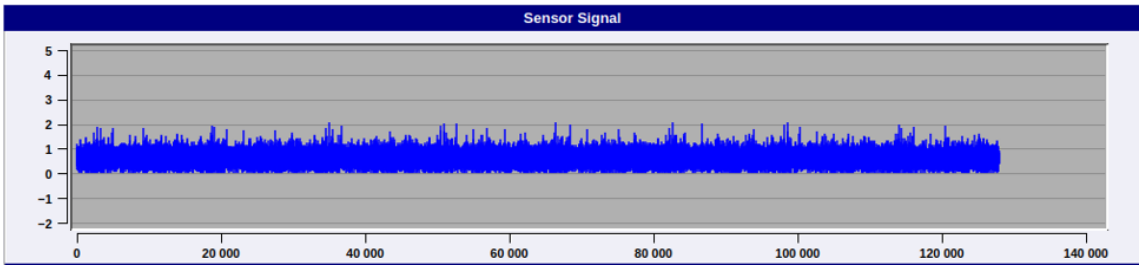
Ohjelman testaaminen aloitetaan ensimmäisenä yleensä yksikkötesteillä, mikä on alhaisin testimuoto. Yksikkötesteillä testataan tiettyjä osa-alueita koodista, jolloin koodit saadaan testattua toimiviksi, ennen kuin ne yhdistetään isoksi kokonaisuudeksi. Yksikkötestaaminen ei kuitenkaan korvaa oikeiden komponenttien kanssa testaamista. Vaikka koodi toimisikin testeissä, se ei välttämättä toimi komponenttien kanssa halutulla tavalla, mikä huomataan myöhemmin järjestelmätestauksessa.

Yksikkötestien tekeminen aloitetaan usein jo ennen kuin varsinaista koodia on kirjoitettu. Tällöin testiin kirjoitetaan, miten testattava osuus pitäisi toimia. Tämän jälkeen aletaan itse koodia

kirjoittamaan. Näin varmistetaan, että ohjelman koodi toimii halutulla tavalla, eikä yksikkötestikoodia kirjoiteta ohjelman kannalta. Usein testit voidaan myös kirjoittaa jälkikäteen, kun tiedetään, että ohjelma toimii oikein tai halutaan varmistaa toiminnallisuus. Tällä tavoin voidaan esimerkiksi parannella testattavaa osuutta ja katsoa, että toiminnallisuus pysyy samana. Yksinkertaisuudessaan testi voi olla vain muutamia rivejä (kuva 4), kun testataan vaikka visuaalisia elementtejä ohjelmasta. Kuvassa neljä annetaan ”updateSensorSignalChartData” -metodille argumenteiksi iso yksiulotteinen taulukko sekä datan määrä kokonaislukuna. Tämä simuloi oikeaa määrää dataa, joka tulee näkymään päänäkymässä Sensor Signal -kaaviossa (kuva 5).

```
TEST_F(TestMainWindow, testSensorSignalChart)
{
    m_mainWindow->updateSensorSignalChartData(dataPoints, 127900);
    QApplication::processEvents();
    QThread::msleep(2000);
}
```

KUVA 4. Yksikkötesti kuvitteellisesta sensorin datasta.

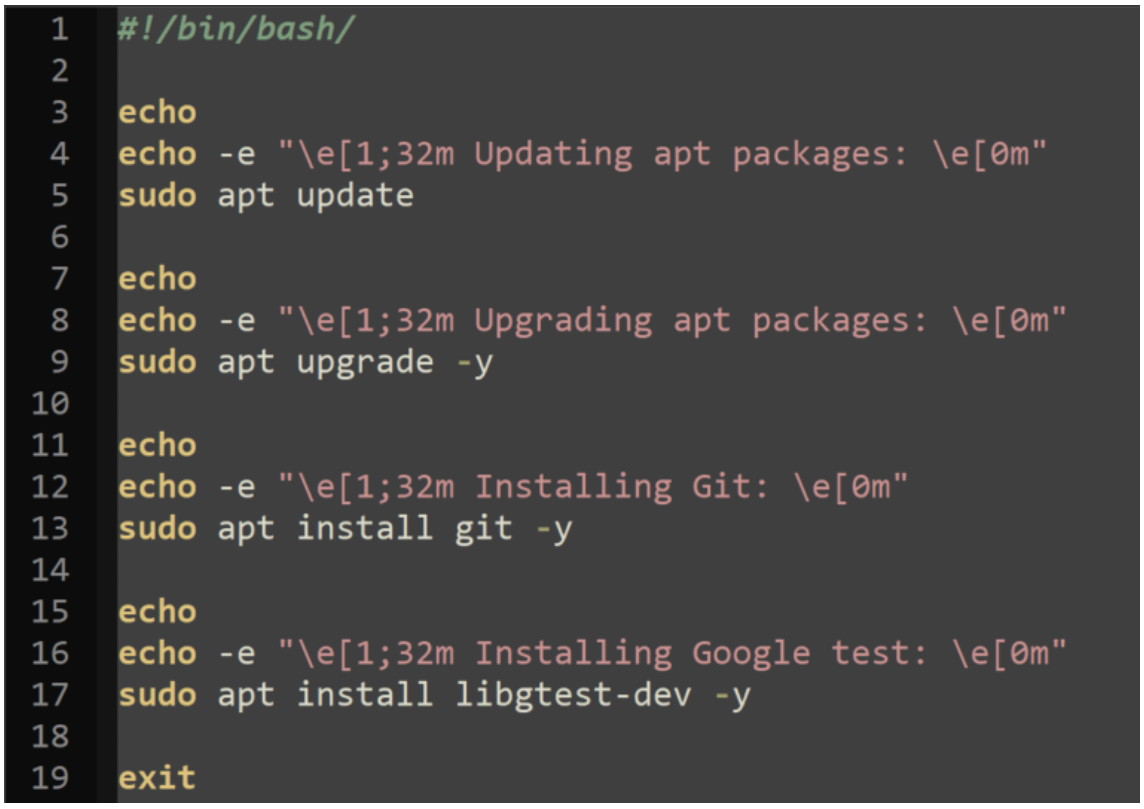


KUVA 5. Yksikkötestin tekemää dataa Sensor Signal -kaaviossa.

3.2 Tietokoneen valmistelu

Käytössä oli uusi Ubuntu 22.04 käyttöjärjestelmällä varusteltu tietokone. Tietokoneeseen asennettin tarvittavat ohjelmat ja päivitykset, jonka jälkeen siihen asennettiin tarvittavat PCIe-kortit sekä niihin sopivat ajurit. Ajureiden asentaminen tapahtui Root-käyttäjänä, jotta asennustiedostojen kääntäminen onnistuisi oikein.

Tietokonetta testattaessa tehtiin myös shell-skripti (kuva 6) asennetuista ohjelmista ja ajureista. Skripti hoitaa tulevat asennukset automaattisesti, jolloin seuraavien koneiden valmisteluun ei kulu niin paljon työaikaa.



```
#!/bin/bash/

echo
echo -e "\e[1;32m Updating apt packages: \e[0m"
sudo apt update

echo
echo -e "\e[1;32m Upgrading apt packages: \e[0m"
sudo apt upgrade -y

echo
echo -e "\e[1;32m Installing Git: \e[0m"
sudo apt install git -y

echo
echo -e "\e[1;32m Installing Google test: \e[0m"
sudo apt install libgtest-dev -y

exit
```

KUVA 6. Esimerkki Bash-skriptistä, joka päivittää tietokoneen ja asentaa Gitin sekä Google-testin.

3.3 RS485- ja RS232-sarjaliikenneväylien testaus

Integraatiotestaus on toinen testausvaihe yksikkötestauksen jälkeen. Integraatiotestauksessa on ideana testata datan kulkeminen komponenttien välillä, testata niiden toiminta yhdessä ja arvioida niiden välinen sopivuus sekä mahdolliset ongelmat. Integraatiotestauksessa huomataan mahdolliset virheet ajoissa ja ne on helpompi korjata siinä vaiheessa, ennen kuin päästään järjestelmätestaukseen. (7.)

Tietokoneen valmistelun jälkeen alkoi testaaminen sarjaväyläkortin kanssa, jonka kautta ohjattiin moottorinohjainta. Tarkoitus oli käyttää joko RS485- tai RS232-sarjaliikenneväylää. RS485 käyttää differentiaalista sarjaväylää, joka tarkoittaa sitä, että se siirtää dataa kahden johdon avulla ja siihen voi yhdistää jopa 32 laitetta. RS232 käyttää vain yhtä johtoa datan siirtämisessä ja sen voi yhdistää vain yhteen laitteeseen kerralla. RS485 voi siirtää dataa vain yhteen suuntaan kerralla, kun taas RS232 voi siirtää ja vastaanottaa dataa samanaikaisesti. RS485:n etuna on, että se pystyy siirtämään dataa pidemmälle matkalle ja on vähemmän herkkä häiriölle. RS485-kaapelille ei ole standardisoitua liitintä, jolloin sen joutui tekemään itse, jonka jälkeen voitiin kokeilla yhteyttä

moottorinohjaimeen. RS485-kaapeliin tulee vain kolme johtoa, jotka ovat maadoitusjohto sekä kaksi datajohtoa. Pinnijärjestykset löytyvät moottorinohjaimen ja PCIe-kortin käyttöohjeista.

RS485-sarjaväylää testattiin ensin tietokoneella terminaalin kautta käyttämällä Minicom-ohjelmaa, jonka kautta pystyi lähettämään moottorinohjaimeen viestin, jonka pitäisi vastata ”<” toimiessaan oikein. Testin edetessä huomattiin, ettei RS485:n kautta tule mitään vastausta, joten siirryttiin testaamaan RS232-johtoa.

RS485 vaihdettiin standardisoituun RS232:een ja kokeiltiin lähettää viesti jälleen Minicomin kautta ilman onnistumista. Huomattiin, että RS232 tarvitsee nollamodeemin, jossa vastaanottava- ja lähettäjäjohdot käännetään päittäin, jolloin johdot menevät TxD:stä TxD:hen ja RxD:stä RxD:hen. Johtojen vaihdon jälkeen saatiin vastaus Minicomista ja todettiin RS232-johdon toimivan oikein. (8.)

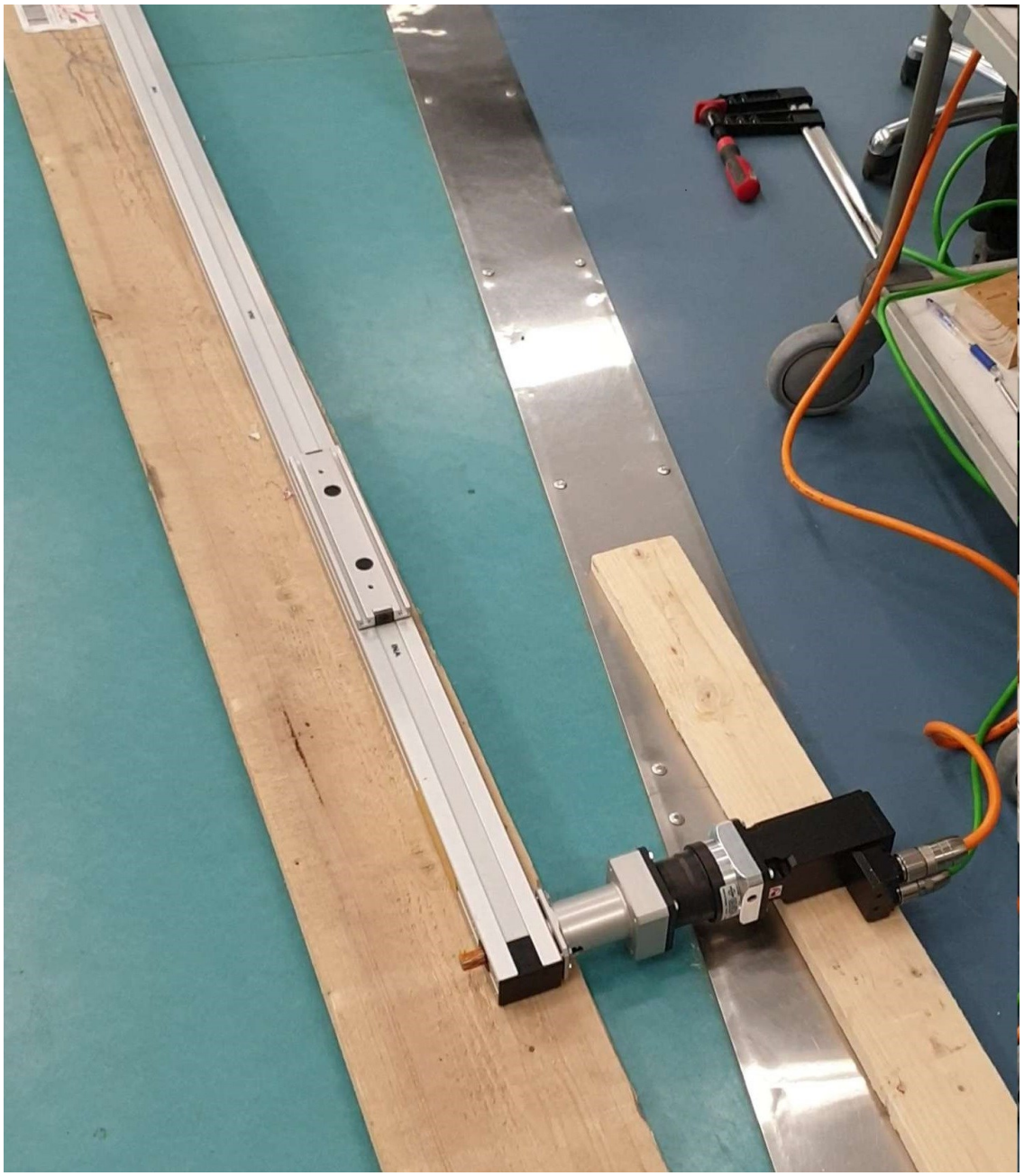
3.4 Moottorinohjaimen ja moottorin testaus

Molempien sarjaliikenneväylien testauksen jälkeen siirryttiin testaamaan moottorinohjainta valmistajan omalla ohjelmalla, joka on saatavilla vain Windows-käyttöjärjestelmille. Ohjelma onnistuttiin asentamaan erilliselle Windows 10 -käyttöjärjestelmällä varustetulle tietokoneelle, jonka jälkeen siirryttiin asentamaan Ubuntuun Windows 10 -virtuaalikone. Virtuaalikone ei ole ideaalinen vaihtoehto, mutta sillä pystyttiin testaamaan suoraan yhteyttä moottorinohjaimeen, jolloin ei tarvinnut olla erillistä Windows-konetta sen testaamiseen. Virtuaalikoneen alustamisen jälkeen ohjelma tunnisti moottorinohjaimen sekä pystyi ohjaamaan moottoria edestakaisin, kuten haluttiin. Moottoria oli tarkoitus ohjata myöhemmin suoraan koodista tietyillä komennoilla, jotka löytyivät laitteen käyttöohjeista.

Moottorinohjaimelle voitiin antaa myös komentoja Linuxissa muun muassa Minicomin kautta sekä Windowsilla, esimerkiksi Tera Termilla. Testauksen edetessä huomattiin, ettei mikään manuaalissa olevista komennoista toiminut, joten siirryttiin takaisin käyttämään valmistajan omaa ohjelmaa ja alettiin seuraamaan sarjaliikennettä. Tällä tavoin saatiin tietoon komennot, joita ohjelma käyttää moottorin liikuttamiseen. Sarjaliikenteen tarkastelun ja komentojen muokkausten jälkeen moottori saatiin ohjattua halutulla tavalla suoraan terminaalin kautta ja myöhemmin koodista.

Moottorin testausten jälkeen moottoriin kiinnitettiin vaihteisto**,** joka kiinnitettiin johteeseen (kuva 7). Vaihteiston suhde on 1:8, jolloin moottorinohjain piti ohjelmoida oikein vastaamaan kolmen millimetrin liikkuvuutta vaihteiston jälkeen. Moottoria ajettiin eteenpäin, jotta saatiin testattua johteen liikkuvuus ja tarkastettua hihnan liikkuma matka millimetreissä. Tarkoitus oli liikkua kerralla noin kolmen millimetrin mittainen matka. Tämä testattiin liikuttamalla johdetta useita kertoja kolmen millimetrin sarjoissa, jonka jälkeen matka mitattiin. Mittaus tehtiin myös kolmen millimetrin välein, jolloin katsottiin, että liikkuvuus oli suunnilleen oikea.

Moottorilla on oma vääntömomentti, joka oli asetettu mahdollisimman pieneksi, ettei johde vahingossa rikkoisi mitään, jos se ylittäisi maksimikohdan tai törmäisi johonkin. Tätä testattiin laittamalla esine johteen eteen ja yritettiin pysäyttää hihnan liikkuminen mahdollisimman kevyesti ja katsottiin, oliko vääntömomenttia liikaa tai liian vähän. (8.)



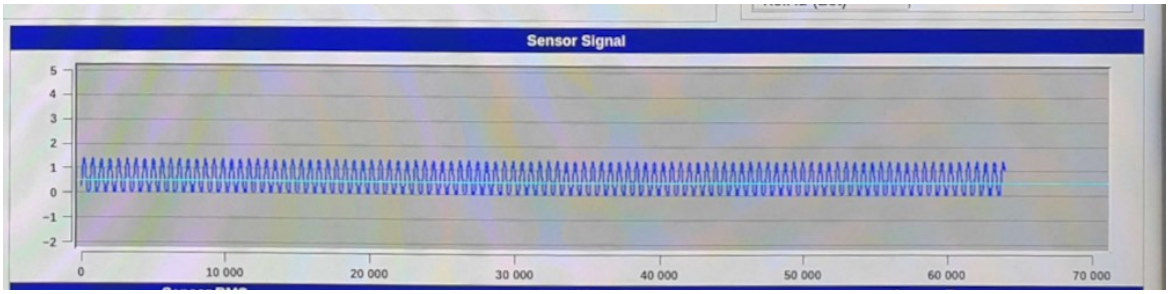
KUVA 7. Moottori vaihteiston kanssa kiinnitettynä johteeseen.

3.5 DSP-kortin ja mittapään testaus

DSP eli digitaalinen signaaliprosessori on prosessori, joka on suunniteltu digitaalista signaalinkäsittelyä varten. Se pystyy suorittamaan digitaalisen signaalinkäsittelyn tehtäviä nopeammin kuin yleiset prosessorit ja kuluttaa vähemmän virtaa. Digitaaliset signaaliprosessorit ovat kuitenkin huonoja epäsäännöllisen ohjelmakoodin ajamisessa, jonka vuoksi ne eivät sovellu esimerkiksi nykyisten käyttöjärjestelmien ajoon.

DSP-korttia testattiin ensin antamalla sille siniaaltoa funktiogeneraattorista ja toisesta funktiogeneraattorista annettiin ratanopeus hertseinä. Tämän jälkeen DSP-kortilla yritettiin saada lähettämään siniaaltodata Ethernet-johtoa pitkin tietokoneeseen, jossa käyttöliittymä piirsi siniaaltodatan näytölle (kuva 8).

DSP-kortin testauksen jälkeen siirryttiin testaamaan mittapäätä. Mittapään testaamiseen tarvittiin ratanopeus, jotta DSP osaa ottaa oikean määrän näytteitä lähetettävää dataa varten. Ratanopeus voitiin ottaa joko funktiogeneraattorista tai sitten enkooderista, joka mittasi rummun pyörimisnopeutta ja muunsi ne pulsseiksi. Mittapäässä oli sensori, joka mittasi laserin valoa kimmottuaan ensin alumiinirummun pinnasta, minkä jälkeen data siirtyi ensin DSP-kortille ja sieltä tietokoneelle.



KUVA 8. Siniaalto dataa DSP-kortilta.

3.6 Järjestelmätestaus

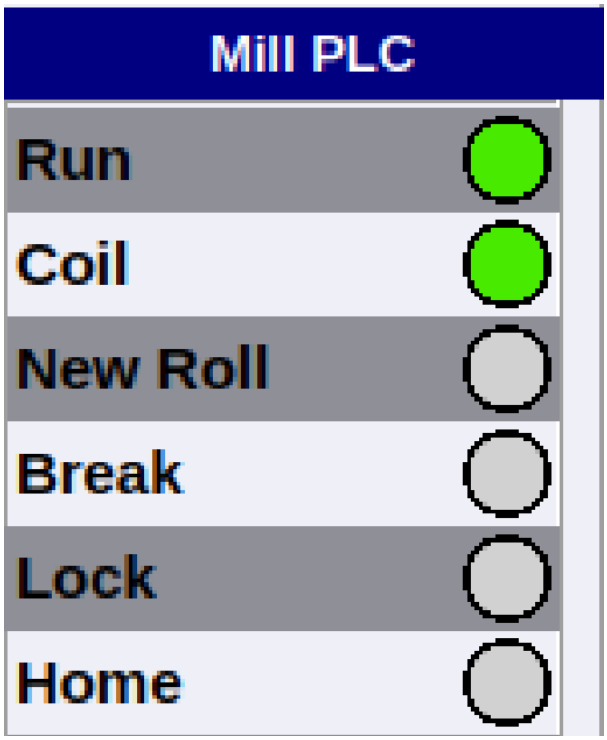
Järjestelmätestaus on kolmas testausvaihe integraatiotestauksen jälkeen. Siinä on tarkoitus testata laitteen toiminta alusta loppuun samanlaisessa tai lähes samanlaisessa ympäristössä kuin oikeassa tilanteessa asiakkaan luona. Järjestelmätestauksessa katsotaan, että kaikki toiminnallisuudet toimivat halutulla sekä vaaditulla tavalla.

Tärkeimmät järjestelmätestit projektissa olivat päänäkymän graafisen käyttöliittymän testaus, PLC-signaalien toiminnan testaus sekä moottorin ja mittapään vakaustestaus oikeanlaisissa olosuhteissa.

Graafisen käyttöliittymän testauksessa testattiin käyttöliittymän käyttäjäkokemusta sekä mahdollisia ongelmatilanteita, joita käyttäjä voisi saada aikaan, esimerkiksi painelemalla nappeja ja sekoittamalla muita asioita. Käyttöliittymässä voidaan myös lähentää ja loitontaa kaavioita, jotka

tulee testata myös, ettei käyttäjä jää jumiin liian lähennettyyn tai loitonnettuun kuvaan. Kaavojen tyhjentämisen ja laittamisen pitää toimia halutulla tavalla.

PLC-signaalien testaukseen oli rakennettu virtakytkinlaatikko. Kytkimiä painelemalla saatiin simuloitua tehtaasta tulevia PLC-signaaleja. PLC:ltä tulevat signaalit kertovat laitteelle, mitä pitää tehdä tietyn signaalin ollessa päällä. Esimerkiksi, kun RUN ja COIL (kuva 9) signaalit tulevat, ohjelma käynnistää normaalin ajotilan, jossa datankeruu alkaa ja servo liikuttaa mittapäätä eteenpäin kolmen millimetrin välein. Järjestelmätestauksessa testattiin jokainen mahdollinen PLC-signaaliyhdistelmä ja katsottiin, että ohjelma toimii oikein. Myös tämä laite antaa PLC-signaaleja ulostuloina, jotka piti testata, katsomalla I/O-kortista ledit sekä varmistamalla yleismittarilla ulostulojännite.

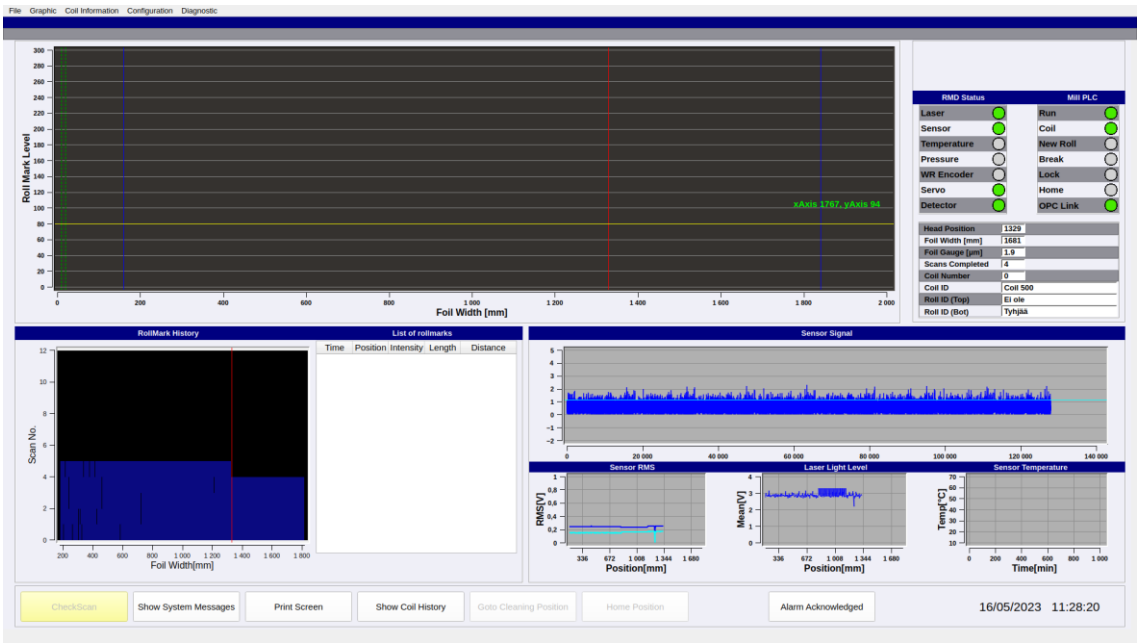


KUVA 9. PLC-signaalit päänäkymässä.

Vaikka moottori ja mittapää oli testattu aiemmin, tulee ne testata samalla järjestelmätestauksessa vakauden varmistamiseksi. Tämä tapahtui samalla, kun ohjelma oli normaalissa ajotilassa ja tarkasteltiin näiden toimintoja virheiden varalta. Testausta tehtiin myös ottamalla moottorista sekä mittapäästä sähköt pois ja katsottiin, että ohjelma reagoi halutulla tavalla. Sähköt laitettiin myös takaisin, jolloin ohjelman pitäisi osata jatkaa normaalisti ilman, että jouduttaisiin käyttämään ohjelmaa kiinni ja käynnistämään se uudestaan.

3.7 Käyttöliittymän esittely

Ohjelman käyttöliittymä perustuu suurimmaksi osaksi erilaisiin kaavioihin ja numeroarvoihin, joista käyttäjä saa selville tarvittavat tiedot ohjelman toiminnasta sekä mahdollisista ohjelmiston virheistä. Testauksen edetessä saatiin dataa, joka piirtyi näihin kaavioihin, jotka auttoivat laitteen testauksessa. Käyttöliittymän (kuva 10) päänäkymässä näkyy isossa tumman värisessä kaaviossa punaisella viivalla mittapään paikka ja mahdolliset valssimerkit erilaisina väreinä vakavuusasteen mukaan. Oikealla puolella näkyy laitteen sekä PLC:n indikaattorit, jotka kertovat käyttäjälle laitteen tilan ja kunnon. Vasemmalla alhaalla ”Roll Mark History” -kaaviossa näkyvät mahdolliset valssimerkit aiemmilta kierroksilta eri väreinä vakavuusasteen mukaan. Keskellä alhaalla ”Roll Mark List” -listassa valkoisella pohjalla näkyvät valssimerkkien tiedot listana. Oikealla alhaalla näkyy sensorin data erilaisissa muodoissa, sekä lämpötilat.



KUVA 10. Käyttöliittymän päänäkymä.

4 TULOKSET

Työn tavoitteena oli testata Foil Rollmark Detectorin toiminta koodin sekä fyysisien komponenttien puolesta. Testaaminen jatkui koko opinnäytetyön ajan ja jatkuu edelleen pitkälle tulevaisuuteen, joten laitteen testaamisen tulokset ovat hieman vajavaiset.

Suurin osa laitteen fyysisistä komponenteista saatiin testattua sekä niiden toiminnallisuus varmistettua. Mittapään sensorin signaalin vahvistuksen testaaminen jäi vielä testaamatta laitepulan takia. Myös kokonaisuuden testaaminen jäi tekemättä osien puutteen takia. Siinä moottori ohjaisi mittapäätä alumiinirummun alapuolella ja sensori ottaisi arvoja rummusta, kuten oikeassa tilanteessa asiakkaan luona.

Koodin yksikkötestaus oli yksi keskeisimmistä testausmuodoista, jota tehtiin jo pitkään ennen muiden asioiden testausta ja sillä saatiin testattua suurin osa koodista. Koodia tulee edelleen lisää, joten senkin testaus jatkuu vielä opinnäytetyön jälkeen.

Testaaminen on saatu projektissa niin pitkälle kuin näillä tiedoilla ja laitteilla on mahdollista. Uuden tietokoneen alustuksen yhteydessä tehtiin shell-skripti, joka myös todettiin toimivaksi uusien koneiden myötä ja jonka todettiin helpottavan koneiden valmiiksi laittamista huomattavasti.

5 POHDINTA

Opinnäytetyön aiheena oli testata Foil Roll Mark Detectorin toiminta erilaisilla testeillä. Opinnäytetyössä tehdyt testaukset onnistuivat hyvin, vaikka ovatkin vajaat kokonaisuuteen nähden. Integraatio- ja järjestelmätestauksen olisi voinut aloittaa aikaisemmin, jos tarvittavat komponentit olisivat olleet aikaisemmin saatavilla, jolloin niiden testaaminen ei olisi jäänyt näin vajaaksi. Markkinapulan takia komponenttien tilaaminen olisi voitu hoitaa ehkä aiemmin, riippuen siitä olisiko ollut vielä tietoa, mitä pitää tilata.

Yksikkö- , integraatio- ja järjestelmätestaus jatkuvat vielä pitkälle opinnäytetyön jälkeen, jolloin saadaan testattua kaikki asiat halutuilla tavoilla. Seuraavaksi rakennetaan testipenkki, joka tulee vastaamaan asiakkaan tehtaan luona olevaa järjestelmää. Tällöin voidaan varmistaa, että laite toimii halutulla tavalla.

Tulevaisuudessa testaukset tehdään kokonaisuutena järjestelmätestauksessa, kun on saatu testipenkki kasaan. Testaukseen voisi tehdä listan, jossa olisi asiat, mitkä pitää testata erikseen, jotta varmistetaan laitteen kokonaisuuden toimivuus. Tällainen voisi olla ”Factory Acceptance Testing” eli FAT-dokumentti, joka pitää tehdä asiakkaalle todisteeksi laitteen toimivuudesta.

LÄHTEET

1. Qt 2023. About Qt. Hakupäivä 16.4.2023. https://wiki.qt.io/About_Qt

2. Qwt 2023. About Qwt. Hakupäivä 16.4.2023. https://qwt.sourceforge.io/

3. Javatpoint 2023. Software development life cycle. Hakupäivä 17.5.2023. https://www.javatpoint.com/software-development-life-cycle

4. Javatpoint 2023. Levels of testing. Hakupäivä 17.5.2023. https://www.javatpoint.com/levels-of-testing

5. GeeksforGeegs 2023. Software engineering sdlc v-model. Hakupäivä 18.5.2023. https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

6. Javatpoint 2023. V-model. Hakupäivä 19.5.2023 https://www.javatpoint.com/v-model

7. Zaptest 2023. Mitä on integraatiotestaus? Syväsukellus tyyppeihin, prosessiin ja toteutukseen. Hakupäivä 17.5.2023. https://www.zaptest.com/fi/mita-on-integraatiotestaus-syvasukellus-tyyppeihin-prosessiin-ja-toteutukseen

8. Etechnophiles 2023. Rs232 and rs485 differences. Hakupäivä 9.4.2023. https://www.etechnophiles.com/rs232-vs-rs485-difference/