LAPIN AMK
Lapland University of Applied Sciences

3D-TULOSTIMEN AUTOMATISOINTI

Juho Kujala

Opinnäytetyö

Tieto- ja viestintätekniikka
Insinööri (AMK)

2023

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Tieto- ja viestintätekniikka
Insinööri (AMK)

Tekijä	Juho Kujala	**Vuosi**	2023
Ohjaaja	Anssi Ylinampa		
Toimeksiantaja	Lapland Robotics -hanke		
Työn nimi	3D-tulostimen automatisointi		
Sivumäärä	50		

3D-tulostimen automatisoinnista Dobot Magician -työpöytärobottikäsivarrella oli ollut puhetta Lapin ammattikorkeakoulun IOT-laboratoriossa, mutta mitään konkreettista ei ikinä ole sovittu. Itsellä ajatus kuitenkin oli jäänyt mieleen, joten päätin tehdä aiheesta opinnäytetyön Lapland Robotics -hankkeeseen. Samalla saataisiin hyötyä irti IOT-laboratorioon hankituista työpöytärobottikäsivarsista ja projektin onnistuessa uusia mahdollisuuksia 3D-tulostamiseen IOT-laboratoriossa. Tavoitteeksi projektissa muodostui prototyypin kehittäminen, prototyyppi todentaisi konseptin toimivuuden. Isoimpia kysymyksiä oli oikean teknologian valinta, mikä mahdollistaisi Magicianin helpon integroinnin 3D-tulostimen kanssa. Toisena ratkottavana kysymyksenä olisi Magicianin soveltuvuus tehtävään.

Opinnäytetyön tekeminen alkoi Magicianin tehtävään soveltuvuuden todentamisella erinäisten testien ja prototyyppityökalujen suunnitelulla. Ongelmana oli, kuinka valmiit 3D-tulosteet saadaan pois uusien tieltä. Melko aikaisessa vaiheessa pidättäydyin ratkaisussa, jossa Magician nostaa PEI-päällysteisen tulostusalustan magneettiselta 3D-tulostimen pediltä. Tämän jälkeen aloin myös keskittyä projektin ohjelmointiosuuteen, jossa tulisi toteuttaa järjestelmä 3D-tulostimen automatisoinnille sekä integroida Magician osaksi järjestelmää. Puntaroin eri 3D-hallintaohjelmistoja sekä tulostimen emolevyn laiteohjelmistoja, näistä valitsin Marlinin emolevylle ja OctoPrintin 3D-tulostimen hallintaan. OctoPrinttiin vaikutti helpoimmilta lähteä toteuttamaan automaatiojärjestelmää tämän pitkälle viedyn lisäosajärjestelmän ansiosta.

Opinnäytetyöstä syntynyt prototyyppi, johon kuuluivat OctoPrint-lisäosa sekä Magicianin ja 3D-tulostimen yhteistyön mahdollistavat työkalut, onnistui mielestäni ihan hyvin. Prototyyppijärjestelmän OctoPrint lisäosa kattoi suurimman osan ominaisuuksista mitä suunnittelin ja Magicianin integraatio lisäosan kanssa onnistui aika hyvin. Prototyyppi osoittaa, että automaatiojärjestelmästä voisi saada toimivan kokonaisuuden, jos projektin veisi loppuun.

Avainsanat 3D-tulostus, automaatio, Dobot Magician, OctoPrint

LAPIN AMK
Lapland University of Applied Sciences

Abstract of the thesis

Degree Programme in Information
and Communication Technology
Bachelor of Engineering

Author	Juho Kujala	**Year**	2023
Supervisor	Anssi Ylinampa		
Commissioned by	Lapland Robotics project		
Title	Automating 3D-printer		
Number of pages	50		

The aim of this thesis study was to create a prototype automated 3D printer system using a Dobot Magician desktop robotic arm. The goal of the prototype was to prove the viability of the automated 3D printer using a Dobot Magician desktop robotic arm. One of the biggest questions was choosing the right technology, which would enable easy integration of Magi-cian with a 3D printer. Another question to be resolved would be the magician's suitability for the task.

The work was started with finding out about the Magicians suitability for the task by performing different kinds of tests and designing prototype tools. The problem to solve was how to remove finished 3D prints from the printer bed so that a new print job could begin. From a very early point, a solution where Magician would lift a flexible PEI-coated build surface from the 3D printer bed was chosen. After this, the focus was on the programming side of the project, where the system for automating the 3D printer would be developed, and also to integrate Magician as part of the system. Different 3D printer firmware and 3D printer control software were evaluated, and from these, were chosen Marlin firmware for the 3D printer motherboard and OctoPrint for the control of the 3D printer. OctoPrint seemed the easiest tool for implementing an automation system thanks to OctoPrint's advanced add-on system.

A prototype born from this thesis study, which includes the OctoPrint add-on and the tools that enable 3D printer and magicians to cooperate, was reasonably successful. The OctoPrint add-on covered most of the planned features and the Magicians integration with the add-on was successful. The prototype shows that the automation system could be turned into a functional system if the project would be completed.

Keywords 3D printing, automation, Dobot Magician, OctoPrint

SISÄLLYS

KÄYTETYT LYHENTEET JA TERMIT

3D-tulostus	valmistusmetodi, jossa kappale muodostuu useista päällekkäisistä kerroksista
Jaettu kirjasto	kollaasi luokkia/funktioita, joita voi käyttää ohjelman suorituksen aikana
Laiteohjelmisto	ohjelmisto, joka hallitsee fyysistä laitteistoa
Makro	sarja komentoja, jotka suoritetaan tietystä käskystä
MCU	mikrokontrolleri
OctoPrint	ohjelmisto 3D-tulostimen hallintaan
Portattu	ohjelmiston kääntäminen toiselle suoritinarkkitehtuurille
Slicer	ohjelmistoa, joka käsittelee 3D-mallin 3D-tulostamista varten
Thread	pala koodia, jonka prosessori suorittaa vuorollaan

1 JOHDANTO

Lapland Robotics -hanke on Lapin ammattikorkeakoulun ja Lapin yliopiston yhteystyössä toteuttama hanke. Hankkeen tavoitteena on tuoda robotiikka ja digitaaliset kaksoset osaksi lappilaista yhteiskuntaa sekä nostaa alueen innovaatiokyvykkyyttä. Hankkeessa selvitetään robotiikan, tekoälyn ja digitaalisten kaksosten soveltumista eri teknologia sekä muilla aloilla. Hankkeesta syntyy avointa tietoa, prototyyppejä miehittämättömistä roboteista ja 5G-ympäristöistä. (Lapland Robotics 2020.) Opinnäytetyö on tehty Lapland Robotics hankkeen nimiin ja tavoitteena on 3D-tulostimen automatisointi. Työ mahdollistaisi 3D-tulosteiden tuottamisen sarjassa ilman että käyttäjä joutuu operoimaan tulostinta tulostus operaation aikana. Mahdollisena lisänä työn tulos mahdollistaisi etänä tulostamisen.

Järjestelmä koostuu OctoPrint 3D-tulostushallintaohjelmistolle kehitettävästä lisäosasta, Dobot Magician -työpöytärobottikäsivarresta sekä 3D-tulostetuista komponenteista, jotka mahdollistavat robottikäsivarren ja 3D-tulostimen yhteistyön. OctoPrint on yksi kolmesta suositusta tulostuskäyttöliittymästä, mutta käytännössä ainoa, jossa on tuki kehittää kolmannen osapuolen lisäosia helposti. OctoPrintin etuna on myös kattava dokumentaatio lisäosa kehitysprosestista. Käyttäjä voi asentaa tai kehittää itse lisäosia lisätäkseen käyttöliittymään uusia ominaisuuksia. Dobot Magician on Dobot -yhtiön valmistama työpöytärobottikäsivarsi opetuskäyttöön. Magician robottikäsivartta voi käskyttää sarjaportin kautta joko valmistajan omalla ohjelmistolla tai käytännössä millä tahansa ohjelmointikielellä, tämä mahdollistaa kolmannen osapuolen kehittämisen Magician robottikäsivarrella.

Opinnäytetyö koostuu täten kolmesta osasta, OctoPrint-lisäosan kehittämisestä, Dobot Magician robottikäsivarren ja 3D-tulostimen yhteistyön suunnitellusta ja tähän tarvittavien komponenttien CAD-mallinnuksesta ja 3D-tulostamisesta. Tässä raportissa käydään kukin osa-alue läpi ja millaisia haasteita kehitystyössä nousee esille. Isoin kysymys on komponenttien soveltuvuus tehtävään. Työssä käytössä oleva 3D-tulostin on halvimmasta päästä eikä Magician robottikäsivarsi ole mikään teollisuusrobotti, vaikkakin on kyvykäs toiminnoiltaan omassa kategoriassaan.

Lopulta kyseessä on prototyyppi, sillä jos järjestelmästä haluaa oikeasti luotettavan, vaatisi tämä paljon työtä ja suunnittelua. Halvat 3D-tulostimet ovat luonteeltaan epäluotettavia ja vaikka tulostin olisi hyvin säädetty, toistuvat näissä samat vikatilanteet usein.

2 3D-TULOSTUSOHJELMISTO

2.1 3D-tulostinlaiteohjelmisto

Laiteohjelmisto on ohjelmisto, joka toimii laitteiston ja ohjelmiston välissä tarjoten laitteistoabstraktiota, jota ylemmän tason ohjelmisto voi käyttää hyväkseen. 3D-tulostimen laiteohjelman tehtävä on kääntää slicer-ohjelmiston 3D-mallista tuottama G-koodi signaaleiksi, jotka aktivoivat ja deaktivoivat 3D-tulostimen komponentteja kuten askelmoottoriohjaimia, tuulettimia, lämmityselementtejä ynnä muuta sellaista. (Florian 2023a.)

Esimerkkinä lineaarinenliikekomento G1 X50 Y20, aluksi laiteohjelmisto tekee tarvittavat laskutoimitukset, jotta saadaan määritettyä kuinka kaukana tulostimen kelkat ovat g-koodi käskyssä annetusta asennosta. Tämän jälkeen mikrokontrolleri lähettää askelmoottoriohjailmelle askelsignaaleja, jotka ajavat ohjaimeen kytkettyä askelmoottoria. (Florian 2023b.)

Laiteohjelmiston valintaan vaikuttavat 3D-tulostimelta haettavat ominaisuudet, käytössä oleva emolevy sekä budjetti. Käytössä oleva emolevy vaikuttaa valintaan siten että laiteohjelmiston tulee olla yhteensopiva emolevyn mikroprosessorin sekä kytkentäkaavion kanssa. Budjetti koska laiteohjelmisto sitouttaa jossain määrin käyttäjän tiettyyn laitteistokonfiguraatioon. Loppujen lopuksi kaikki suositut laiteohjelmistot tuottavat korkealaatuisia 3D-tulosteita, jos laiteohjelmisto on konfiguroitu tulostimelle oikein.

Ilmaisia avoimen lähdekoodin laiteohjelmistoja löytyy iso määrä ja suuriosa näistä perustuu Marlin laiteohjelmistoon. Esittelen tässä muutaman suositun ja aktiivisesti ylläpidetyn laiteohjelmiston. Esittelen myös muutamia ominaisuuksia laiteohjelmisto kohtaisesti, joista voi olla hyötyä projektissa tai jos ominaisuus on laiteohjelmistolle uniikki.

2.1.1 Marlin

Marlin on alkujaan Sprinter ja grbl laiteohjelmistoissa johdettu avoimen lähdekoodin 3D-tulostin laiteohjelmisto RepRap perheen 3D-tulostimille, vuonna 2011 projekti irtaantui erilliseksi projektiksi. (Marlin 2023.) Marlin ja tästä johdetut muut laiteohjelmistot on yleisin käytössä oleva 3D-tulostus laiteohjelmisto. Harrastelijoiden lisäksi myös tunnetut kaupalliset 3D-tulostimet käyttävät Marlinia, esimerkkinä Prusa ja Ultimaker. Marlinin suosio perustuu laiteohjelmiston luotettavuuteen, kattavaan emolevytukeen sekä kykyyn toimia 8 ja 32-bittisillä emolevyillä laitteistoabstraktiokerroksen (HAL) ansiosta. (Marlin 2023.)

Alla olevassa luettelossa lueteltu muutamia ominaisuuksia, joista voisi hyötyä projektissa.

- laaja G-koodi toteutus yli 150 komennolla
- automaattinen PID-kontrolleri lämmityselementille, lämmityksen automaattinen katkaisu lämpötilan karatessa
- G-koodi makrotuki, jossa makrot määritellään joko laiteohjelmiston konfigurointi vaiheessa tai makrot syötetään tulostimelle G-koodi komennolla (Marlin 2023).

2.1.2 Klipper

Klipper on Kevin O’Connor vuonna 2016 alkunsa saanut 3D-tulostin laiteohjelmisto. Klipper poikkeaa perinteisistä laiteohjelmistoista yhdellä merkittävällä tavalla. Perinteisesti laiteohjelmisto pyörii kokonaisuudessa 3D-tulostimen emolevyllä, tarvitsee Klipper toimiakseen 3D-tulostus emolevyn lisäksi toisen tietokoneen, tavanomaisesti RPI 2, 3 tai 4. Klipperissä 3D-tulostimen emolevy ja RPI jakavat työn siten että RPI:n korkeaa laskentatehoa käytetään hyödyksi G-koodin prosessointiin ja täten 3D-tulostimen kontolle ei jää muuta kuin suorittaa käskyt, jotka RPI antaa. Tämä työnjako mahdollistaa 8-bittisten alhaisen laskentatehon emolevyjen suorittamaan jopa kymmenen kertaa enemmän askelmoottoriaskelia kuin esimerkiksi Marlinilla. (Haley 2022.)

Klipperi tukee kaikkia ominaisuuksia, joita nykypäivän 3D-tulostin laiteohjelmistolta voi odottaa. Klipperi tukee 3D-tulostimia, joissa on useampi emolevy, esimerkiksi kolmen emolevyn kokoonpano, jossa ekstruuderi, lämmityselementit sekä askelmoottoriohjaimet saisivat kaikki omat kontrollerinsa. Ylimääräisen emolevyjen lisääminen ei vaadi käyttäjältä muuta kuin pari riviä tekstiä konfiguraatiotiedostoon. Kaikki tulostimen konfigurointi tehdään ihmis- luettavaan konfiguraatiotiedostoon, eli muutoksia tehdessä emolevyn laiteohjelmistoa ei tarvitse kääntää emolevylle uudestaan. Mahdollisuus tehdä omia makroja eli, käyttäjä voi luoda uusia G-koodikomentoja ja määrittää nämä tulostimen konfiguraatiotiedostossa. Makro-Komennot ovat ohjelmoitavia eli käyttäjä voi esimerkiksi käyttää tulostimen tilaa hyödyksi makroissa. Input Shaping ehkäisee tulostimen värinästä johtuvaa tulostuslaadun heikkenemistä, ilmiötä kutsutaan nimellä ringing. Sisäänrakennettu monipuolinen JSON-pohjainen applikaatiorajapinta perus G-koodi rajapinnan lisäksi, jota ohjelmistokehittäjät voivat käyttää hyödyksi työssään. (Klipper 2023.)

2.1.3 RepRapFirmware

RepRapFirmware on Duet3D:en julkaisema ja ylläpitämä laiteohjelmisto 32-bittisille Duet3d-perheen emolevyille, mutta laiteohjelmiston avoimen lähdekoodin myötä laiteohjelmisto on myös portattu useille LPC sekä STM32-pohjaisille emolevyille. Laiteohjelmisto on monikäyttöinen. Se ei ole rajoittunut ainoastaan 3D-tulostukseen, vaan tukee myös CNC-koneita, kaiverruskoneita sekä laserleikkureita. (RepRap 2020.) Erona muihin 3D-tulostuslaiteohjelmistoihin, RepRapFirmwaren perusperiaatteena on ”G-koodia joka paikassa” eli kaikki laiteohjelmiston tuetut toiminnot ovat käytettävissä G-koodi komennoilla. 3D-tulostimen konfiguraation voi siis suorittaa käyttämällä puhtaasti G-koodia ilman erillisiä komentoja. G-koodi joka paikassa periaate helpottaa myös tulostimen konfiguroimista sekä käyttöä sillä laiteohjelmiston reaktio on sama, tuleepa komennot mistä lähteestä hyvänsä. Komennot voivat peräisin joko käyttäjältä, G-koodi makrosta tai 3D-tulostettavasta tiedostosta. (Duet3D 2023a.)

RepRapFirmware sisältää suuren määrän ominaisuuksia, mutta tässä muutama projektin kannalta kiinnostavaa. G-koodi makrotuki yhdistettynä monipuolisen G-

koodi toteutuksen kanssa, joka tukee ehto- ja toistolauseita. (Duet3D 2023a.) Makrotuki helpottaa 3D-tulostimen automatisointia. CAN-väyläpohjainen järjestelmä mahdollistaa lisäkorttien ja työkalujen lisäämisen. (Duet3D 2023b.) Duet3D tarjoaa ison valikoiman CAN-väyläisiä lisäkortteja, joilla automatisoidun 3D-tulostus linjaston toteutus helpottuisi. Emolevyyn on mahdollista liittää korttitietokone kuten Raspberry PI, joka lisää järjestelmän kyvykkyyttä. Korttitietokone mahdollistaa lisäosien toteuttamisen, mikäli RepRapFirmwaren oman verkkokäyttöliittymän ominaisuudet eivät riitä. (Duet3D 2023c.)

Erityisen kiinnostava näistä on mahdollisuus omien lisäosien toteuttamiseen. Emolevyyn kiinnitetylle korttitietokoneelle asennetaan Duet Software Framework (DSF), joka on kollaasi erinäisiä ohjelmistoja emolevyn hallintaan. Omia lisäosia kehittäessä voi käyttää hyödyksi Duet Software Frameworkin tarjoamaa REST-rajapintaa, jolla voi hallita tulostinta.

2.2 3D-tulostimenhallinta

3D-tulostinta voi kontrolloida monesta eri lähteestä. Ensimmäinen vaihtoehto on, että tulostin on täysin standalone ja hallinta tapahtuu kokonaan 3D-tulostimeen liitetyltä näytöltä, jota emolevyn laiteohjelmisto ajaa (Kuvio 1). Toinen vaihtoehto olisi emolevystä erillinen tietokone, joka isännöi hallintaohjelmistoa, joka lähettää emolevylle G-koodia tavallisimmin sarjaliikenteenä.

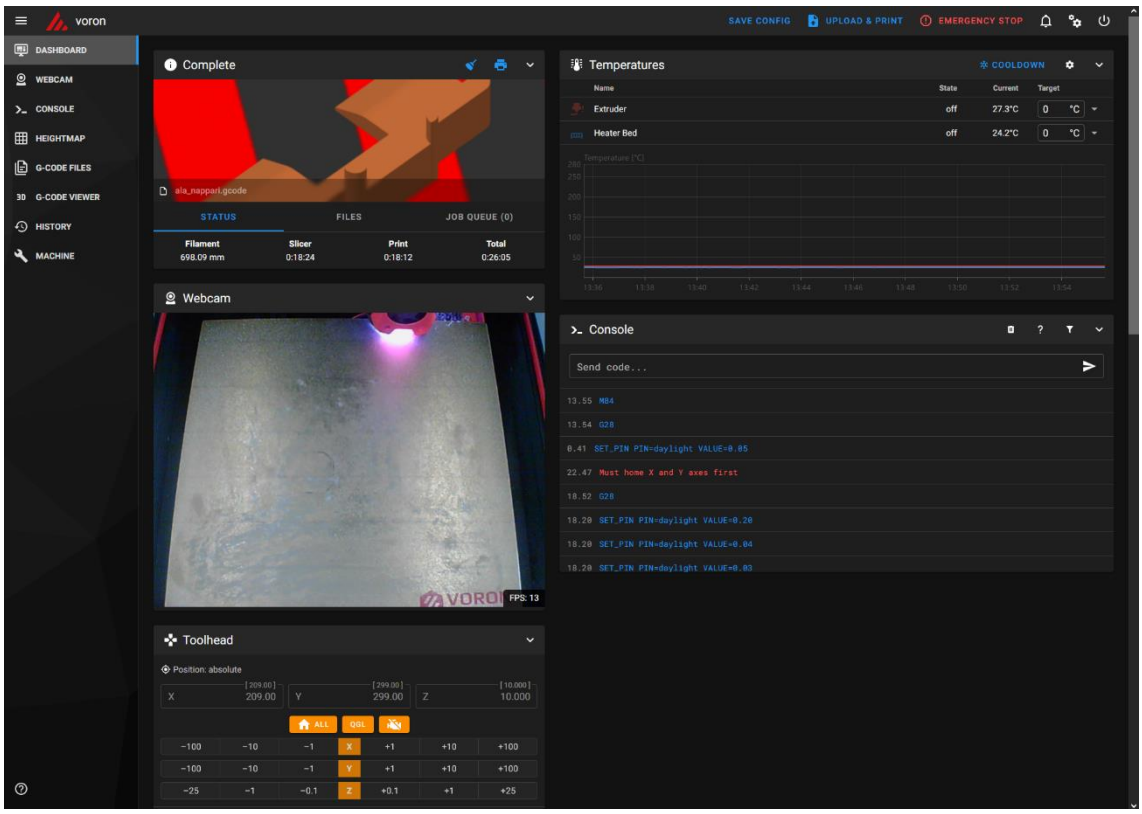
Kuvio 1. Kirjoittajan Voron 2.4R2 hallintapaneeli

Näitä hallintaohjelmistoja on erilaisia. On geneerisiä, joilla voi hallita useimpia laiteohjelmistoja, esimerkkinä OctoPrint ja Pronterface, sitten on tiettyihin laiteohjelmistoihin räätälöityjä ohjelmistoja, esimerkkinä Mainsail ja Fluid Klipperille sekä RepRapFirmwarelle Duet Web Control. 3D-tulostimen hallinnan siirtäminen toiselle tietokoneelle antaa mahdollisuuden luoda ominaisuuksiltaan kyvykkäämmän tulostimen hallintaympäristön kuin yksinään tulostimelta hallittaessa. Tässä projektissa vaatimuksena olisi Magicianin helppo integrointi 3D-tulostimen kanssa, joten hallintaohjelmiston tulisi mahdollistaa tämä.

2.2.1 Mainsail/Fluid

Mainsail on Klipperille räätälöity avoimen lähdekoodin web-selainpohjainen hallintapaneeli, joka toimii yhteistyössä muiden Klipperin ympärille toteutettujen projektien kanssa (Kuvio 2). Yksi näistä projekteista on Moonraker Klipper API, johon Mainsailin toiminta pitkälti nojautuu. Moonraker on Python3-pohjainen verkkopalvelin, joka isännöi API-rajapintaa, jota asiakasohjelmisto voi käyttää Klipper-laiteohjelmiston hallintaan. (Moonraker. 2023.)

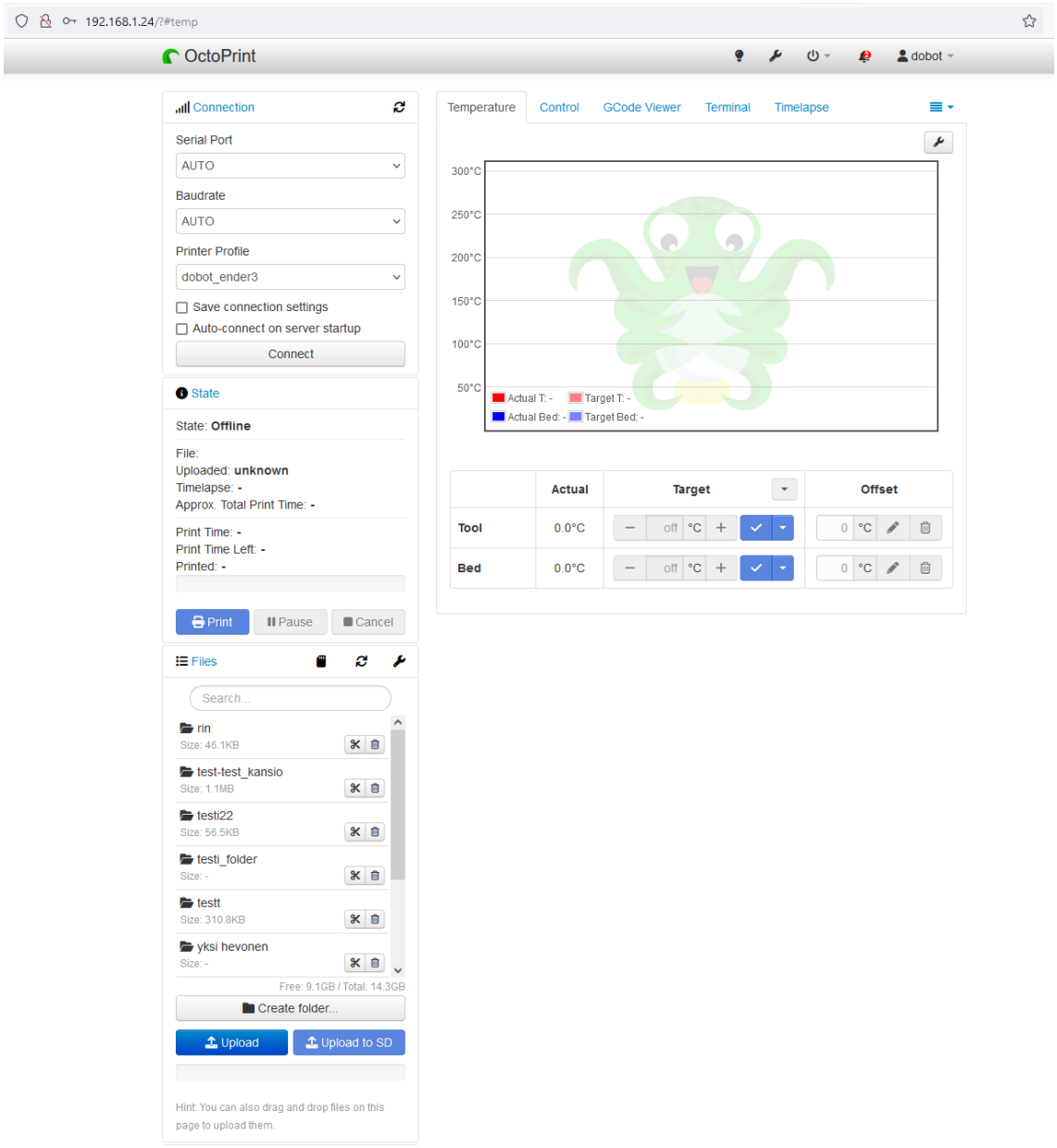
Mainsail ei sisällä natiivia lisäosatukea kuten esimerkiksi OctoPrint, mutta avoimen lähdekoodin projektina yhteisö on luonut useanlaisia lisäosia sekä laajennuksia, jotka on integroitu Mainsailiin. Mainsail dokumentaatio sisältää myös osuuden Mainsail-projektin kehitysympäristön asentamiseksi. Näillä tiedoilla Magicianin integroiminen Mainsailin kanssa on toteutettavissa.



Kuvio 2. Kirjoittajan Voron2.4 R2 Mainsail-käyttöliittymä

2.2.2 OctoPrint

OctoPrint on avoimen lähdekoodin ohjelmisto 3D-tulostimien hallinnointiin. OctoPrint oli ensimmäisiä etähallintaohjelmistoja 3D-tulostimille ja olikin iso tekijä konseptin tekemisestä suosituksi (Kuvio 3). OctoPrint kykenee hallinnoimaan suurinta osaa kuluttajamarkkinoilla olevista 3D-tulostimista ja tukea voi aina laajentaa OctoPrintin kattavalla lisäosajärjestelmällä. (OctoPrint. 2023.)

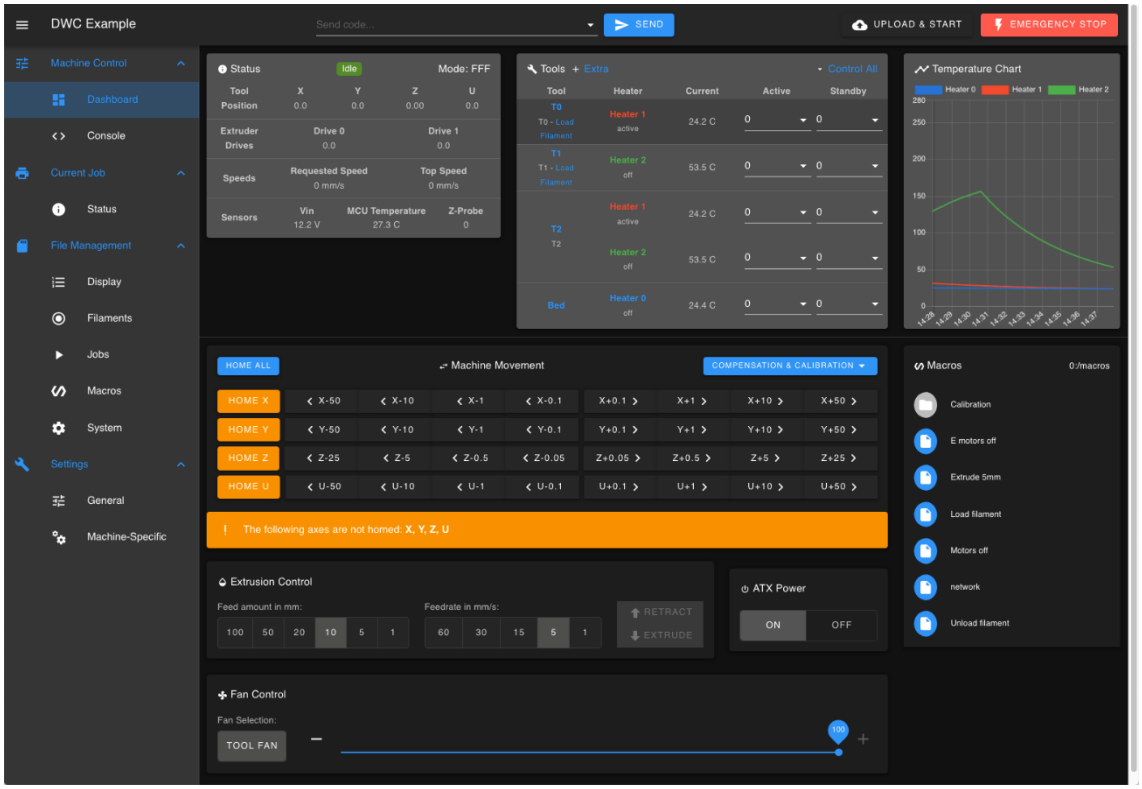


Kuvio 3. Kirjoittajan OctoPrint-käyttöliittymä.

Lisäosajärjestelmä onkin ominaisuus, josta OctoPrint parhaiten tunnetaan. OctoPrint on suunniteltu siten, että lisäosien asentaminen, kehittäminen ja ylläpitäminen olisi helppoa ja sujuvaa. OctoPrint-lisäosarakenteen tulee seurata muutamia tiettyjä käytäntöjä, jotta lisäosan asennusvaiheessa OctoPrint tunnistaa lisäosan osaksi järjestelmää. Pitkälle viedyn lisäosajärjestelmän lisäksi OctoPrint on hyvin dokumentoitu. Dokumentointi kattaa laajasta OctoPrintin sisäisen toiminnan sekä lisäosien kehitysprosessin.

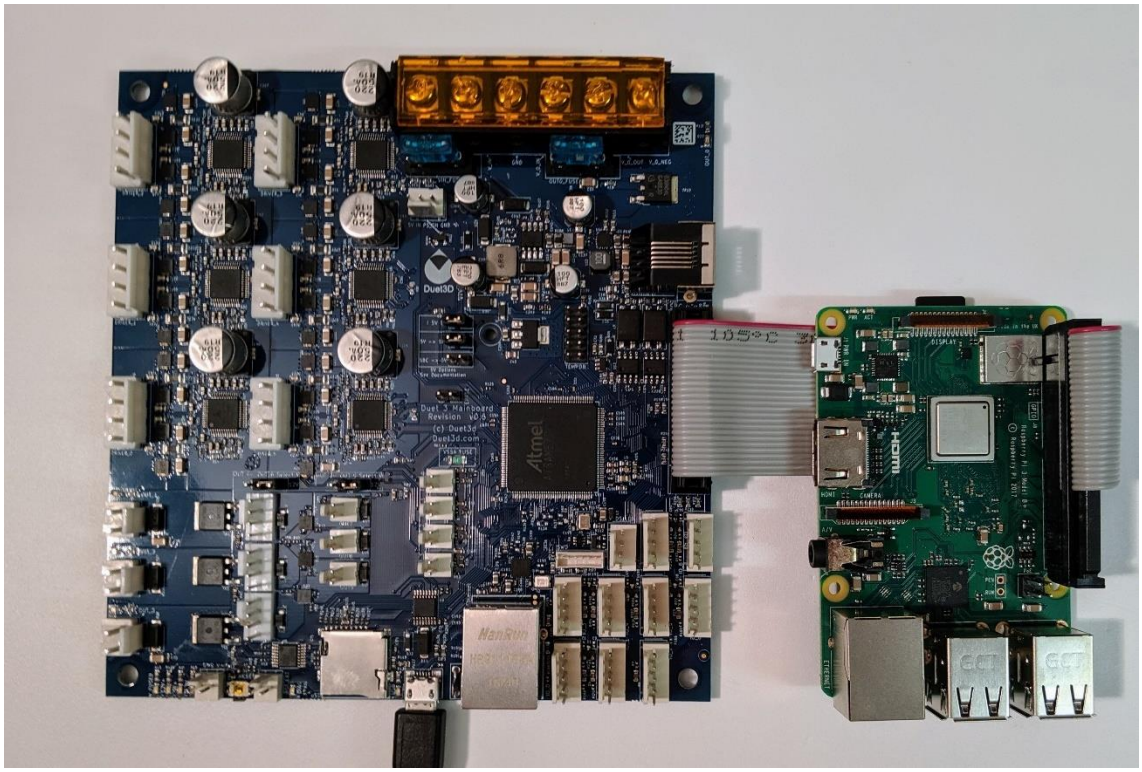
2.2.3 Duet Web Control

Duet Web Control on avoimen lähdekoodin web-selainpohjainen hallintapaneeli Duet3D-perheen emolevyille (Kuvio 4), jotka käyttävät RepRapFirmwarea. Kolmannen osapuolen lisäosat ovat olleet tuettuaja DSF/DWC -versioista 3.3 eteenpäin, lisäosia on kahdenlaisia, DWC ja SBC. (Duet3D 2023d.)



Kuvio 4. DW- käyttöliittymä (Duet3D 2023e)

DWC-lisäosat lisäävät toiminnallisuutta web-selaimessa hallintapaneelin puolella, SBC-lisäosat vaativat Duet3D-emolevyn olevan SBC-konfiguraatiossa, eli liitettynä erilliseen korttitietokoneeseen (Kuvio 5). SBC-lisäosat ovat hyödyllisiä, jos lisäosa vaati enemmän toiminnallisuutta mitä DWC-lisäosa kykenee tarjoamaan, esimerkkinä Magicianin integroiminen 3D-tulostimen kanssa. Lisäosa voi käyttää hyödyksi kumpaakin komponenttia samanaikaisesti, eli DWC tarjoaa hallintapaneelin lisäosan käyttöliittymän ja SBC-lisäosa voi suorittaa korttitietokoneella omia tehtäviään.



Kuvio 5. SBC-konfiguraatio (Duet3D 2023f)

2.2.4 Oman hallinta ohjelmiston kehittäminen

Yhtenä vaihtoehtona olisi oman hallintaohjelmiston kehittäminen, joka on räätälöity Magicianin ympärille. Se miten hallintaohjelma käskyt 3D-tulostimelle lähettää riippuu siitä mitä laiteohjelmistoa käytetään. Klipperin kohdalla käytettäisiin Moonraker API-rajapintaa. RepRapFirmwaren kohdalla joko emolevyn API-rajapintaa tai jos emolevy on SBC konfiguraatiossa, käytettäisiin DSF REST-rajapintaa. Marlin laiteohjelmiston kohdalla käskyt lähetettään suoraan emolevyn sarjaporttiin tai käyttämällä OctoPrint API-rajapintaa.

Oman hallintaohjelmiston kehittäminen olisi tietenkin enemmän työtä ja suurin osa yllä mainittujen hallintaohjelmistojen ominaisuuksista tulisi puuttumaan. Toisaalta Magicianin integrointi voisi olla helpompaa sillä kehitystä ei tarvitse tehdä yllä mainittujen hallintaohjelmistojen ehdoilla.

2.3 Yhteenveto

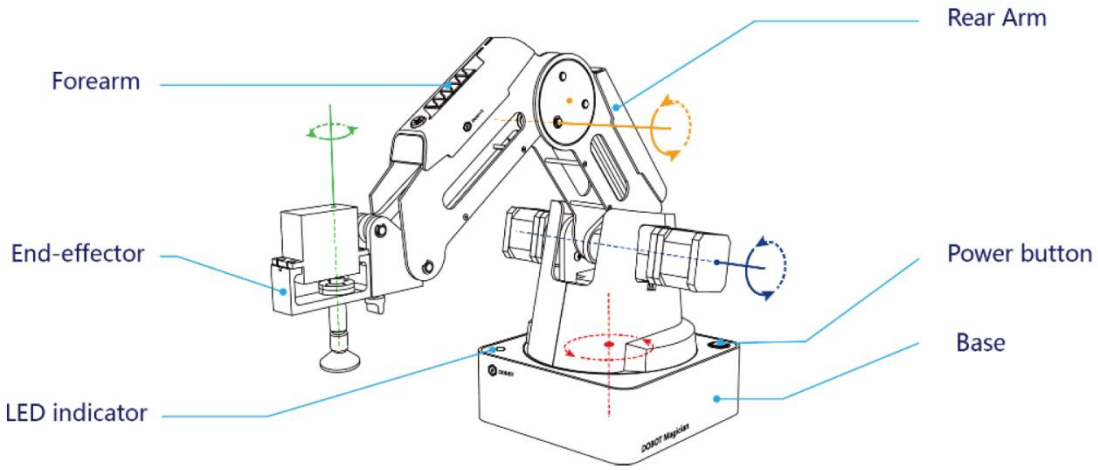
Projektiin valikoitui OctoPrint-hallintaohjelmisto sekä Marlin-laiteohjelmisto, valinta perustuu projektissa käytössä olevaan rautaan sekä 3D-tulostin hallintaohjelmistolta vaadittaviin ominaisuuksiin. Päädyin Marlin-laiteohjelmistoon parista syystä. Ensimmäinen syy on koulun IOT-laboratorion halvat Ender3 Pro 3D-tulostimet, joiden 8-bittiset emolevyt pelaavat hyvin yhteen Marlinin kanssa. Toisena syynä on Marlinin ja OcroPrintin hyvä yhteensopivuus. Klipperi olisi myös ollut vaihtoehto, mutta tämä ei ole yhtä hyvin yhteensopiva OctoPrintin kanssa mitä Marlin. RepRapFirmware olisi vaatinut kokonaan uuden emolevyn ostamisen firmiksen 32-bittisyyden myötä.

Hallintaohjelmistoksi valitsin OctoPrintin sen lisäosajärjestelmän vuoksi, muutkin mainitsemani hallintaohjelmistot mahdollistavat omien ominaisuuksia lisäämisen osaksi hallintaohjelmistoa, mutta eivät yhtä tehokkaasti ja helposti kuin OctoPrint.

3 MAGICIAN END-EFFECTORIN SUUNNITTELU

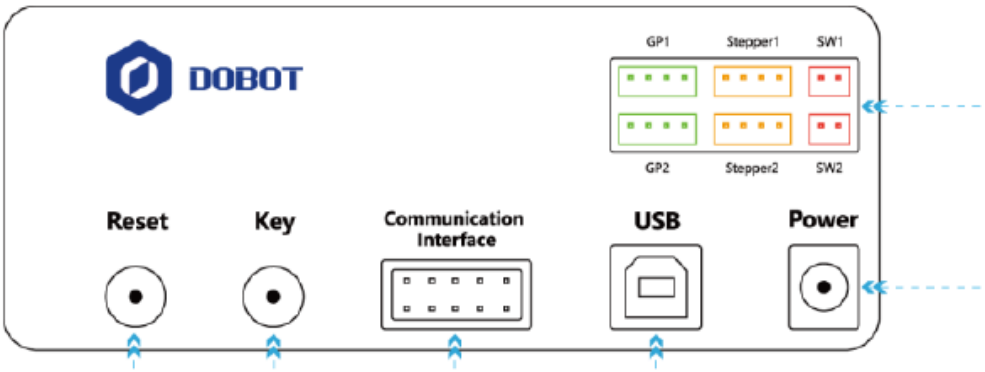
3.1 Lyhyesti Dobot Magicianista

Dobot Magician on monikäyttöinen työpöytärobottikäsivarsi, usealla erilaisella end-effectorilla, eli työkalu robotin käsivarren päässä. Työkaluja on muun muassa 3D-tulostus hotend, laserkaiverrin sekä kynä piirtämistä ja kirjoittamista varten (Kuvio 6).



Kuvio 6. Dobot Magician (Shenzhen Yuejianf Technology Co., Ltd. 2020)

Magician tukee myös kolmannen osapuolen työkalujen kehittämistä robotin takaa löytyvän IO-paneelin avulla (Kuvio 7) (Shenzhen Yuejianf Technology Co., Ltd. 2020). IO-paneelista löytyy sarjaportti, kaksi multipleksattua GPIO-porttia, joita voi käyttää joko PWM tai ADC tilassa sekä Kaksi askelmoottori ja lisävirtaporttia.



Kuvio 7. Dobot IO-paneeli (Shenzhen Yuejianf Technology Co., Ltd. 2020)

Dobot Magician käyttää nivel ja karteesi koordinaattijärjestelmiä. Nivelkoordinaatiossa koordinaatit määräytyvät nivelten J1, J3, J3 ja J4 perusteella, kun taasen karteesikoordinaatissa koordinaatit perustuvat Magicianin alustaan määritettyyn nollapisteeseen (Kuvio 8) (Shenzhen Yuejianf Technology Co., Ltd. 2020.)

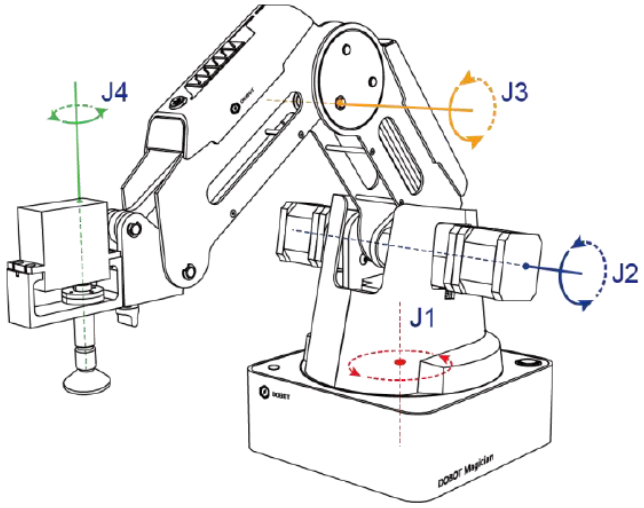


Figure 3.4 Joint coordinate system

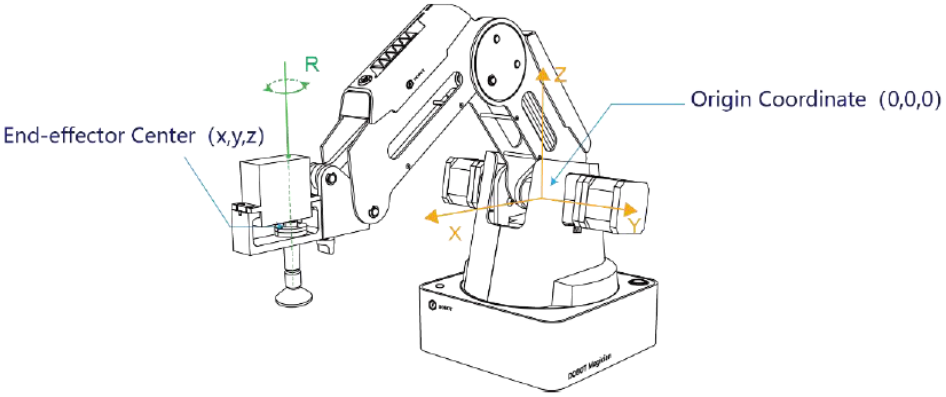


Figure 3.5 Cartesian coordinate system

Kuvio 8. Magician koordinaatisto (Shenzhen Yuejianf Technology Co., Ltd. 2020)

Magicianille liikekomentoja syötettäessä voi valita kolmesta eri liikeratatyypistä. Jogging liikeratatyypissä Magiciania juoksutetaan valittua akselia pitkin joko karteesi- tai nivel koordinaatistossa. PTP-liikeratatyyppi tukee useampaa variaatiota,

MOVJ, MOVL ja JUMP. PTP-komennon liikerata riippuu valitusta PTP-variaatiosta. MOVJ-variaatiossa Magicianin nivelet ajavat itsensä määritettyyn kulmaan annetun liikekomennon suorittamiksi välittämättä liikkeen liikeradasta (Kuvio 9), kun taasen MOVL-variaatiossa liikkeen liikerata on suora viiva lähtöpisteestä kohdepisteeseen (Kuvio 9). JUMP-variaatiossa Magician liikkuu aluksi määritetyn korkeuden verran ylös, ajaa tämän jälkeen kohdepisteen yläpuolelle ja lopuksi liikkuu alaspäin kohdepisteeseen (Kuvio 9). ARC-liikekomennon liikerata on kaari, joka määritellään kolmen pisteen avulla (Kuvio 9) (Shenzhen Yuejianf Technology Co., Ltd. 2020).

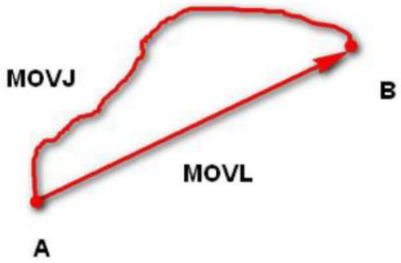


Figure 3.6 MOVL/MOVJ mode

Figure 3.7 JUMP mode

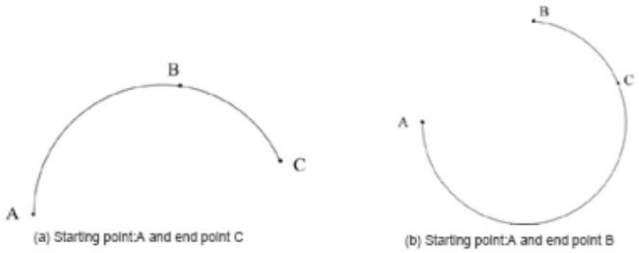


Figure 3.8 ARC mode

Kuvio 9. liikeratatyypit (Shenzhen Yuejianf Technology Co., Ltd. 2020)

3.2 3D-tulostimen ja Dobot Magiciannin yhteistyö

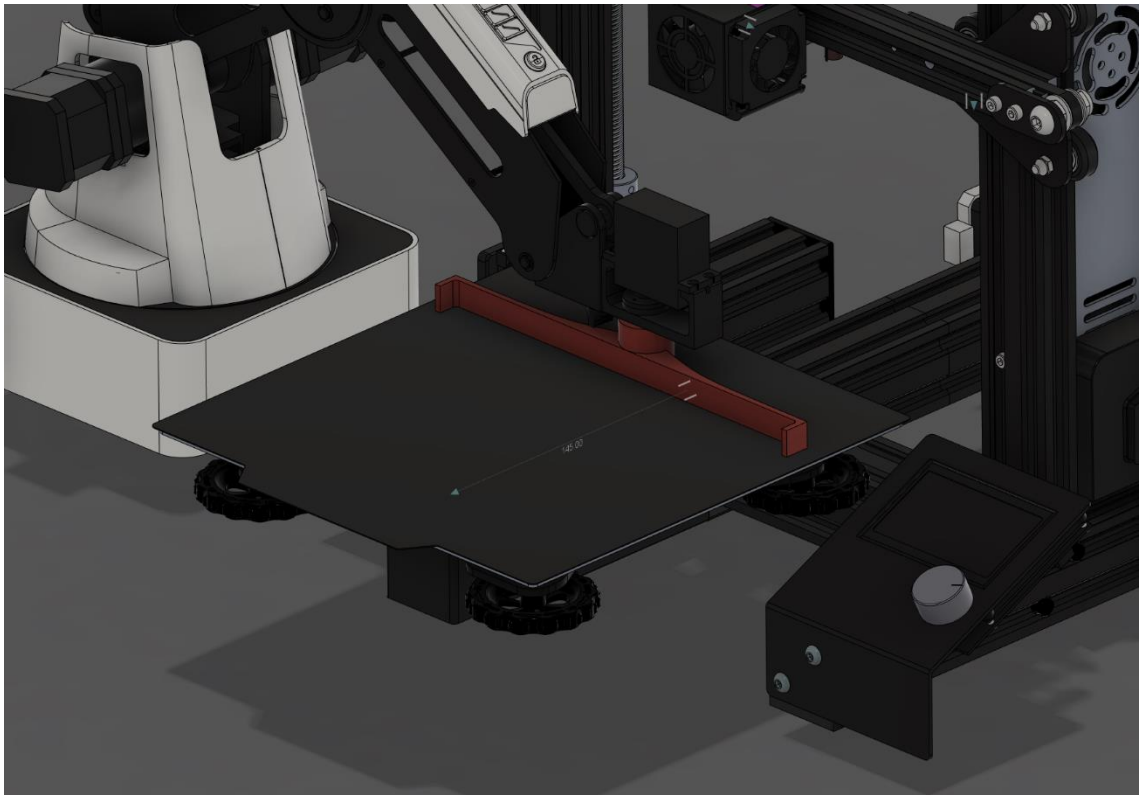
3.2.1 3D-tulostusten poistaminen tulostimen pediltä

Automaatiojärjestelmässä Dobot Magicianin tehtävä on poistaa valmiit tulosteet 3D-tulostimen pediltä, jotta uusi tulostus tulostusjonosta voi alkaa. Projekti alkoi suunnittelemalla, kuinka Magician voisi poistaa valmiit tulosteet tulostusalustalta. Testaamisessa käytin avukseni Magicianin valmistajan DobotStudio -hallintaohjelmistoa, jossa on helppo esimerkiksi luoda erilaisia liikeratoja työkalun testaamiseksi. (Kuvio 10).

Kuvio 10. DobotStudio on työkalu Magicianin hallintaan

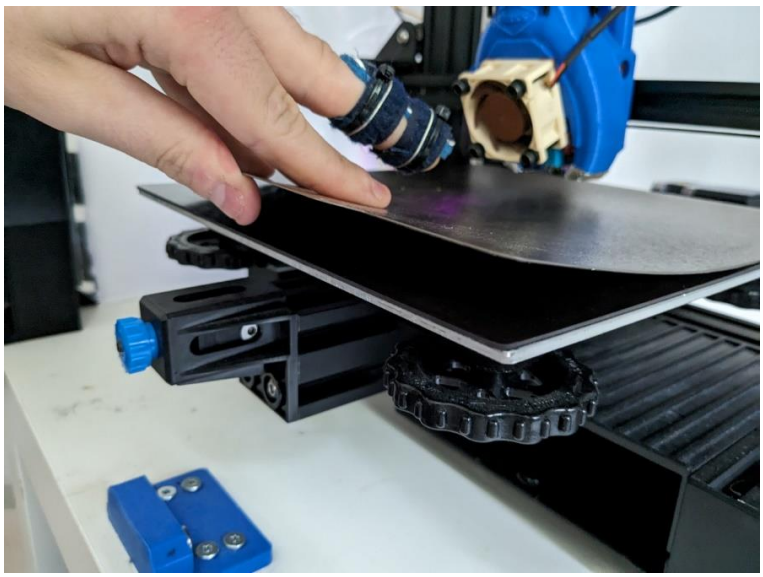
Ensimmäinen ajatus oli, että Magician yksinkertaisesti pyyhkäisee valmiit tulosteet 3D-tulostimen pediltä, mutta luovuin tästä ideasta varhaisessa vaiheessa parista syystä (Kuvio 11). Tuloste voi olla todella tiukasti kiinni tulostusalustasta. Yleensä alustan jäähdyttyä tuloste irtoaa vaivatta, mutta tämä ei ole taattua ja

tuloste pitää irrottaa tulostusalustasta voimaa käyttämällä. Toisena syynä on yhden kerroksen korkuiset tulostussuoritukset, kuten kärjen puhdistusvedot alustan reunaan tulostuksen alussa tai slicerin luomat brimit tulostuksen ympärillä, näitä olisi todella vaikea saada luotettavasti irti tämän tyylisellä työkalulla.



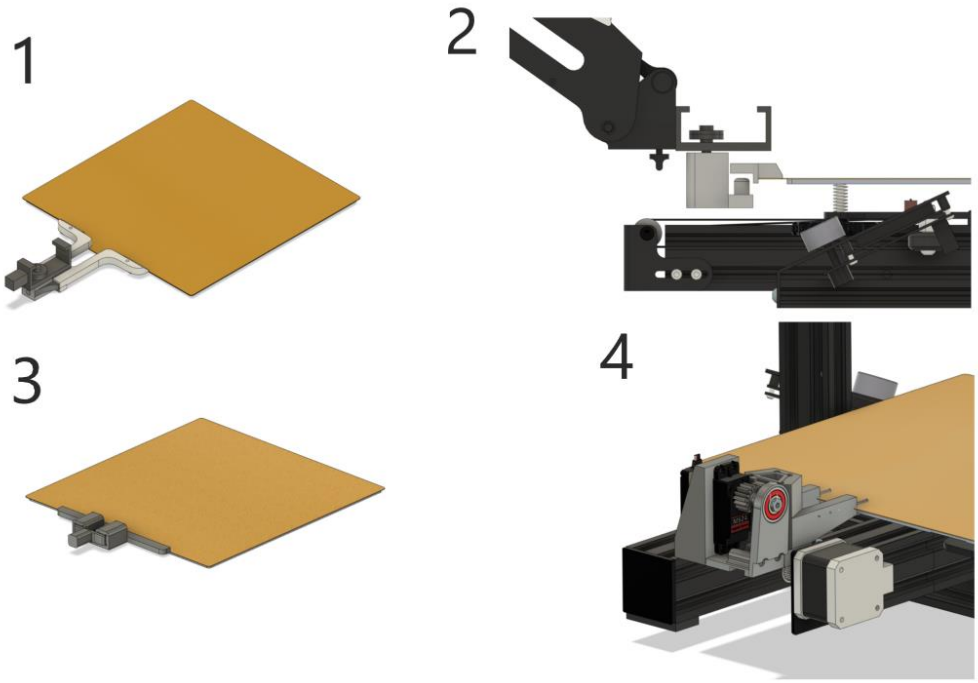
Kuvio 11. Prototyyppityökalu, joka ei nähnyt elämää CAD:in ulkopuolella

Idea johon lopulta päädyin on tulosalustan poistaminen kokonaisuudessaan tulostuksen päätteeksi ja tämän mahdollistaa magneettinen peti ja joustava tulostusalusta (Kuvio 12). IOT-laboratorion tulostimissa on alkujaan lasiset tulostusalustat, joten tilaukseen laitettiin Crealityn PEI-päällysteinen joustava tulostusalusta.



Kuvio 12. PEI-päällysteinen joustava tulostusalusta

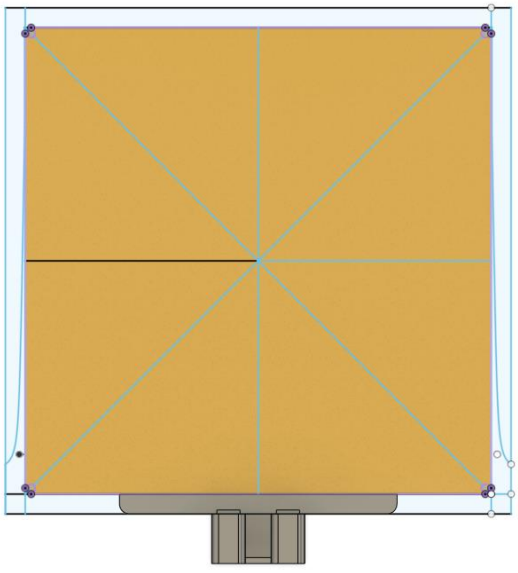
Ideana on, että Magicianin käsivarren päähän tulee työkalu (end-effector), joka nostaisi tulostusalustan magneettiselta pediltä, tulostusalustassa olisi vastakappale, josta työkalu saisi otteen. Kuviossa 13 on erilaisia iteraatioita työkalusta projektin aikana, osa iteraatioista hyödynsi Magicianin neljättä akselia eli servomoottoria käsivarren päässä, mutta luovuin tästä asioiden yksinkertaistamiseksi.



Kuvio 13. Nostotyökalu prototyyppejä

Ongelmaksi tässä ratkaisussa nousi Magicianin heikko nostovoima, eli Magicianilla ei ollut riittävästi nostovoimaa ja robotin askelmoottorit menettivät askelia satunnaisesti. Ratkaisuksi tähän ideoin auttavaa kättä eli 3D-tulostimen taakse asetettavaa servoa (Kuvio 13) joka auttaisi nostotyössä, servoa ajettaisiin Magicianin IO-paneelista (Kuvio 7). Parempi ratkaisu oli kuitenkin ostaa ohuempi magneetti tarralevy tulostimen petiin, tämän jälkeen Magician suoritti tulostusalustan noston vaivatta. Nostotyökalun lisäksi järjestelmä koostuisi kahdesta hyllystä, jossa ensimmäiseen ladotaan valmiit tulosteet ja toisesta nostetaan uusi tulostusalusta 3D-tulostimen pedille (Kuvio 14).

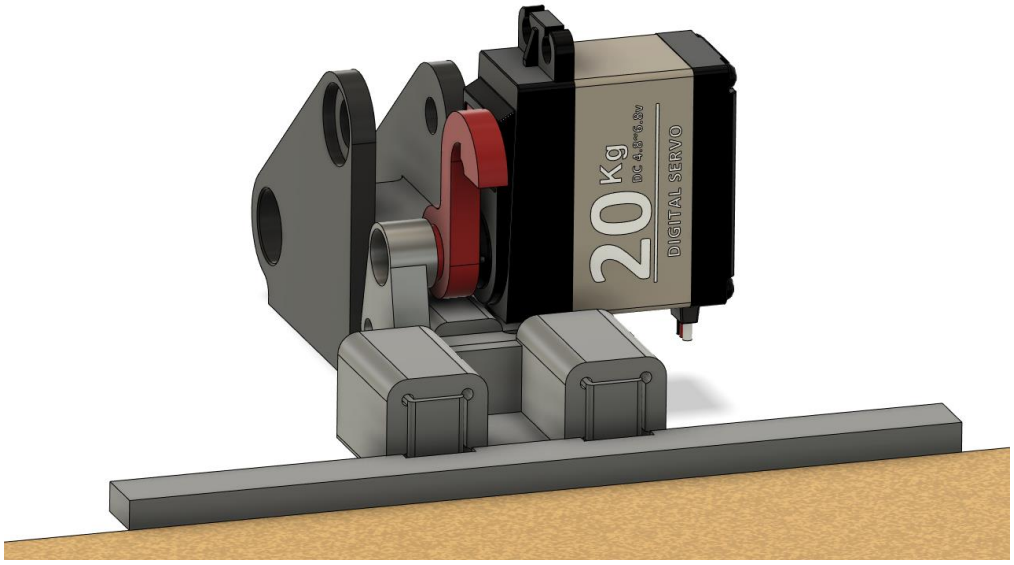
Tulostusalustojen tulisi asettua hyllyyn joka kerta identtiseen asentoon. Yksi idea oli, että hyllyntason seinät kapenevat loppua kohden, jolloin seinät ohjaavat tulostusalustan aina samaan asentoon Magicianin työntäessä alustaa hyllyyn. 3D-tulostin ideasta pari iteraatiota, jossa testailin eri toleransseja tulostusalustan ja seinien etäisyydessä. Hyvältä toleranssilta tuntui 0,15 millimetriä kumminkin puolin seiniä, tulostusalusta liukui hyvin sisään ja pysyi paikoillaan.



Kuvio 14. Hyllyjen protoilua

Tulostusalustojen latomisesta ja nostelusta hyllyihin ilmeni tarve siihen, että nostotyökalu kykenisi myös vetämään tulostusalustaa, johon sen hetken työkaluite-

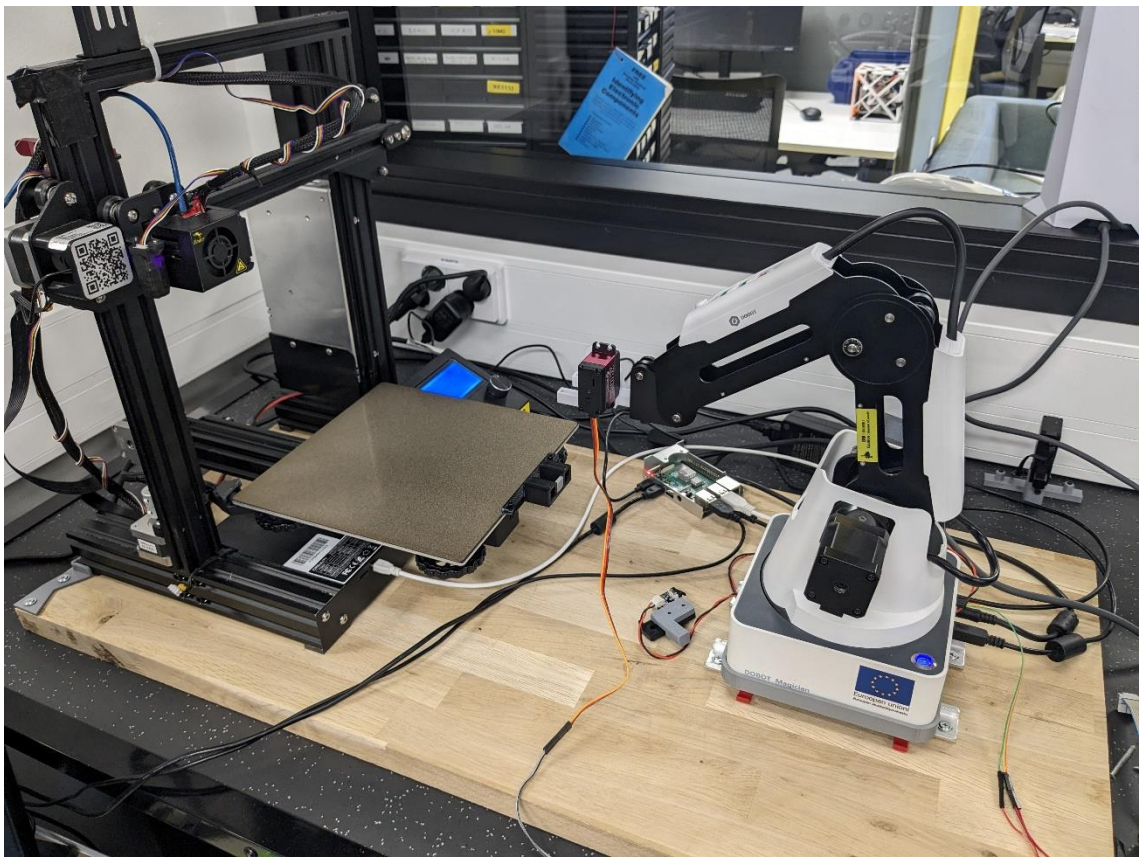
raatio ei kyennyt (Kuvio 13, työkalu 3). Lisäsin nostotyökaluun servon, joka laskee koukun. Se nappaa vastakappaleesta kiinni vetämisen mahdollistamiseksi (Kuvio 15).



Kuvio 15. Uusin versio nostotyökalusta ja tämän vastakappaleesta

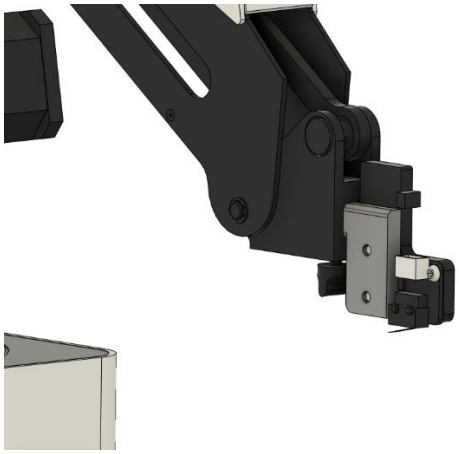
3.2.2 Testialusta

Testialustan tehtävänä on koeajaa suunniteltuja työkaluja, alusta koostuu puulevystä, Magician robottikäsivarresta sekä 3D-tulostimesta (Kuvio 16). Magician ja 3D-tulostin ovat sijoitettu kohtisuoraan toisiinsa nähden nostotyökalun ja tulostusalustan vastakappaleen kohdistamisen helpottamiseksi. Magician ja 3D-tulostin ovat kiinteästi kiinni alustassa, jotta näiden sijainti toisiinsa nähden ei pääse vahingossa muuttumaan. Tämä helpottaa testaamista, sillä nostotyökalun ja tämän vastakappaleen kohdistaminen on nykyisin tehtävä käsin. Mikäli jomman kumman sijainti muuttuu, ei aiemmat Magicianille annetut liikekomennot enää osu kohdalleen.



Kuvio 16. Testialusta

Myöhemmin projektissa suunnittelin kokeilumielessä proben, joka auttaisi nostotyökalun ja tämän vastakappaleen kohdentamisessa Magicianin Z-akselilla, sillä vastakappaleen korkeus muuttuu, jos 3D-tulostimen petiä säädetään. Proben toimista tarkemmin OctoPrint-lisäosa kappaleessa, jossa käyn proben toimintaperiaatteen tarkemmin läpi. (Kuvio 17).



Kuvio 17. Mikrokytkin probe

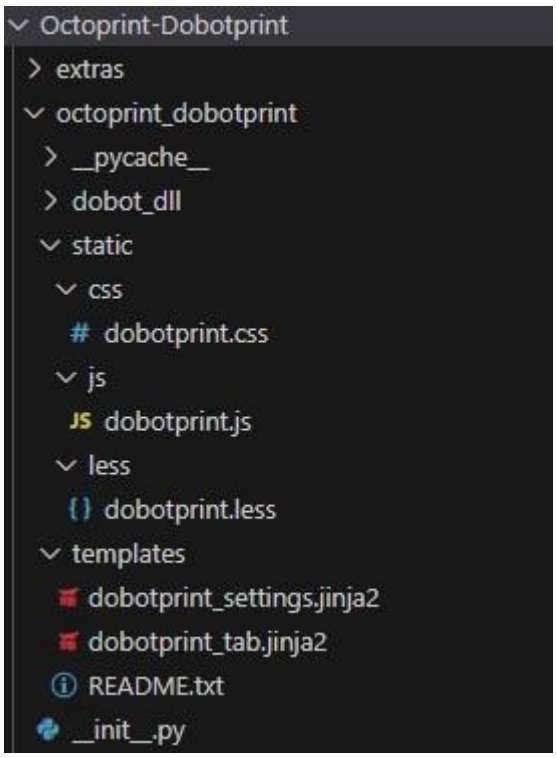
4 OCTOPRINT-LISÄOSA

OctoPrint-lisäosa on Python-paketti, joka päällimmäisellä tasolla määrittelee itselleen Control Propeties eli metadataa. Tämän avulla OctoPrint initialisoi lisäosan osaksi OctoPrint lisäosa-alijärjestelmää, esimerkiksi *__plugin_load__* property initialisoi luokan johon lisäosan toiminnallisuus on toteutettu (Kuvio 18). Lisäosa voi liittää itsenä osaksi järjestelmää parilla eri tapaa, mutta projektin lisäosa nojautui ainoastaan yhteen näistä, niin sanottuihin Mixins. Mixins ovat erilaisia pohjaluokkia, joita lisäosa voi periä sekä ylikirjoittaa, täten lisäten toiminnallisuutta OctoPrintiin. Esimerkiksi kuviossa 18 lisäosa perii StartupPlugin pohjaluokan, josta voidaan ylikirjoittaa *on_after_startup* metodin, jota kutsutaan OctoPrintin käynnistyessä.

```
class DobotprintPlugin(
    octoprint.plugin.StartupPlugin,
    octoprint.plugin.SettingsPlugin,
    octoprint.plugin.AssetPlugin,
    octoprint.plugin.TemplatePlugin,
    octoprint.plugin.EventHandlerPlugin,
    octoprint.plugin.SimpleApiPlugin
):
    #Lisäosan toteutetaan tähän luokkaan...
    def on_after_startup(self):
        self._dobot_printer = DobotPrinter(
        self._printer, self._logger, self._settings, self._fire_event)

__plugin_name__ = "Dobotprint Plugin"

__plugin_pythoncompat__ = ">=3,<4"  # Only Python 3

def __plugin_load__():
    global __plugin_implementation__
    __plugin_implementation__ = DobotprintPlugin()

    global __plugin_hooks__
    __plugin_hooks__ = {
        "octoprint.plugin.softwareupdate.check_config": __plugin_implementation__.get_update_information
    }
```

Kuvio 18. Lisäosan metadataa ja Mixins-alaluokka.

Lisäosan toiminta tulee pitkälti nojautumaan Mixins-pohjaluokkiin, sillä ne antavat pääsyn OctoPrintin tilaan sekä mahdollistavat OctoPrintin hallinnan lisäosasta. OctoPrint mahdollistaa myös käyttöliittymän helpon toteuttamisen lisäosalle. OctoPrint-lisäosan kansiorakenteen tulee olla tietynlainen, jotta järjestelmä osaa initialisoida lisäosan (Kuvio 19).



Kuvio 19. OctoPrint-lisäosan kansiorakenne

OctoPrint lisää kuviossa 18 näkyvät templetes, css ja js kansioiden sisällön dynaamisesti osaksi OctoPrintin käyttöliittymää, huomioitavaa on jinja2 templeten nimeämis- käytäntö. Nimi määrittelee mihin templete OctoPrintin käyttöliittymässä sijoitetaan, esimerkiksi *_tab.jinja2* päätteinen templete lisätään käyttöliittymän tab-osioon. (kuvio 20).

Kuvio 20. Lisäosan tab-välilehti.

OctoPrint myös automaattisesti sisällyttää OctoPrint JavaScript clientin sekä Knockout JavaScript kirjastot lisäosan JavaScriptiin, näitä käytetään hyödyksi toiminnallisuuden toteuttamisessa. Esimerkkinä Octoprint.files property (Kuvio 21) antaa kätevästi tehdä http-pyyntöjä backendiin ja vastauksena tulee tietoa OctoPrintin tiedosto ja kansio rakenteesta, Knockout js puolestaan mahdollistaa

MVVM-suunnittelumallin käyttämisen käyttöliittymässä. OctoPrintin omat toiminnallisuudet on toteutettu samoilla perusluokilla ja Knockout js viewmodeleilla millä lisäosa rakennetaan, eli lisäosa on hyvin tiiviisti integroitu OctoPrintin kanssa.

```
OctoPrint.files.list(true).done(function (response) {
  //tee jotain vastauksella...
});
```

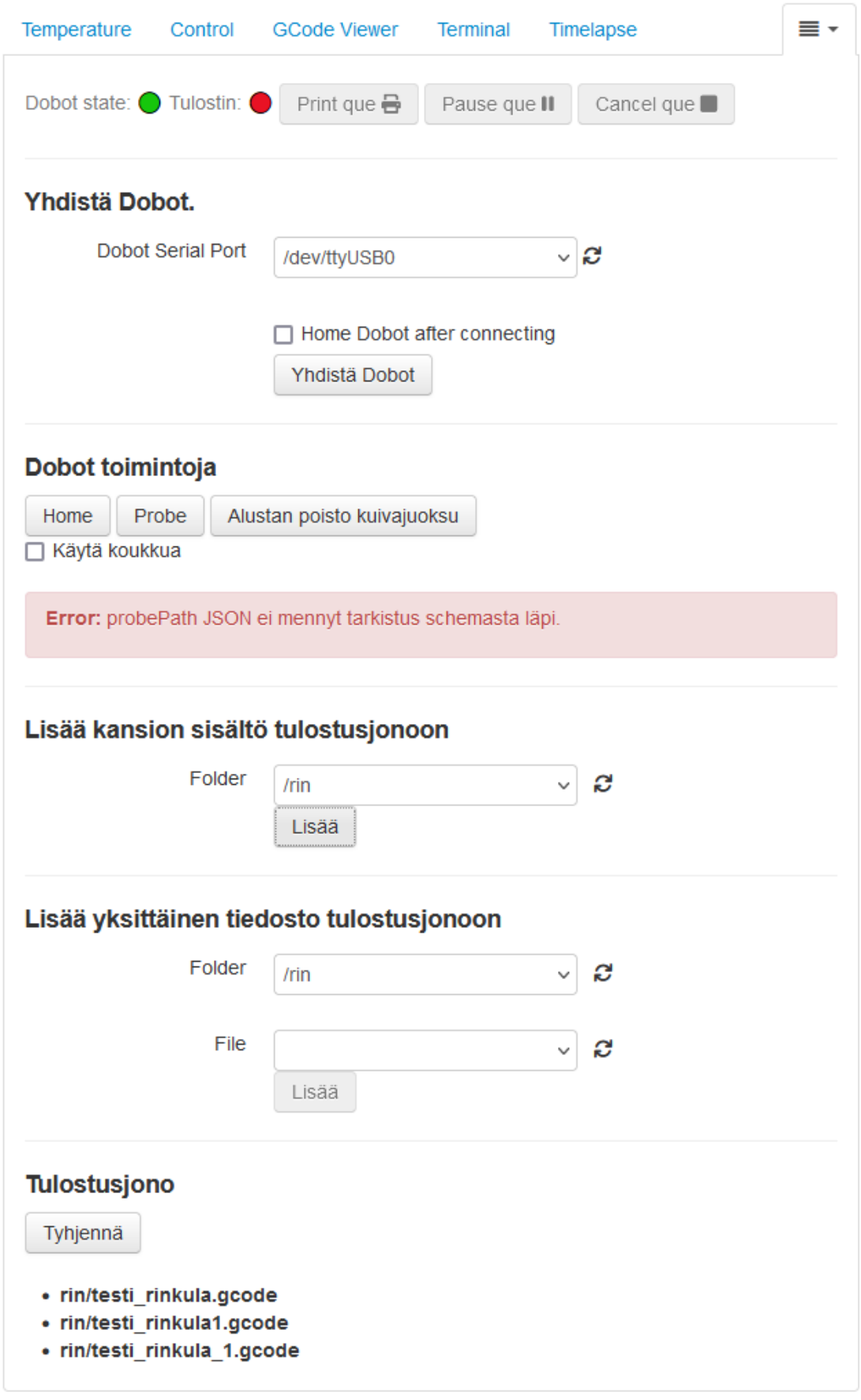
Kuvio 21. Injektoitu JavaScripti

4.1 Frontend

Käyttöliittymä sisältää seuraavat ominaisuudet:

- yhteyden muodostaminen Magicianin kanssa
- Magicianin kotiutus komennon ajaminen
- G-koodi tiedostojen lisääminen tulostusjonoon sekä jo jonossa olevien tiedostojen listaaminen
- tulostusjonon käynnistäminen, pysäyttäminen sekä peruuttaminen
- Magicianin asetusten säätäminen

Lisäosan logiikka tapahtuu backendissä Python-koodin puolella, frontend ainoastaan lähettää käyttäjän syötteen backendille sekä esittää tietoa lisäosan statuksesta, kuten onko Magician yhdistetty, tulostusjonon sisällön sekä mahdolliset lisäosan virheviestit. Lisäosan käyttöliittymä koostuu kahdenlaisesta templetestä. Tab (kuvio 22) josta lisäosan päätoiminnallisuus löytyy sekä settings (Kuvio 25) jossa voi määrittää lisäosa sekä Magician kohtaisia asetuksia.



Kuvio 22. Lisäosan päänäkymä

4.1.1 Lisäosan päänäkymä

Käyttäjän syötteen lähettäminen backendille on toteutettu implementoimalla OctoPrint SimpleApiPlugin pohjaluokka lisäosaan (Kuvio 23), tämä mahdollistaa yksinkertaisten http-pyyntöjen lähettämisen back sekä frontendin välillä.

```
class DobotprintPlugin(
    octoprint.plugin.StartupPlugin,
    octoprint.plugin.SettingsPlugin,
    octoprint.plugin.AssetPlugin,
    octoprint.plugin.TemplatePlugin,
    octoprint.plugin.EventHandlerPlugin,
    octoprint.plugin.SimpleApiPlugin
):

  #SimpleApiPlugin pohjaluokan metodi
  def on_api_command(self, command, data):
        if command == "disconnectDobot":
            #Suorita koodia...

        elif command == "connectDobot":
            #Suorita koodia...

        elif command == "dobotFilePrint":

            self._logger.info("print file : " +
                              data["folder"] + "/" + data["file"])
            # filut juuresta
            if data["folder"] is "/":
                self._dobot_printer.add_print_to_queue(data["file"])
            # filut tietystä kansiosta
            else:
                self._dobot_printer.add_print_to_queue(
                    data["folder"] + "/" + data["file"])

            self.update_frontend()

        elif command == "dobotFolderPrint":
            #Suorita koodia...

        #ja niin edelleen...
```

Kuvio 23. SimpleApiPlugin ohjelmallinen toteutus

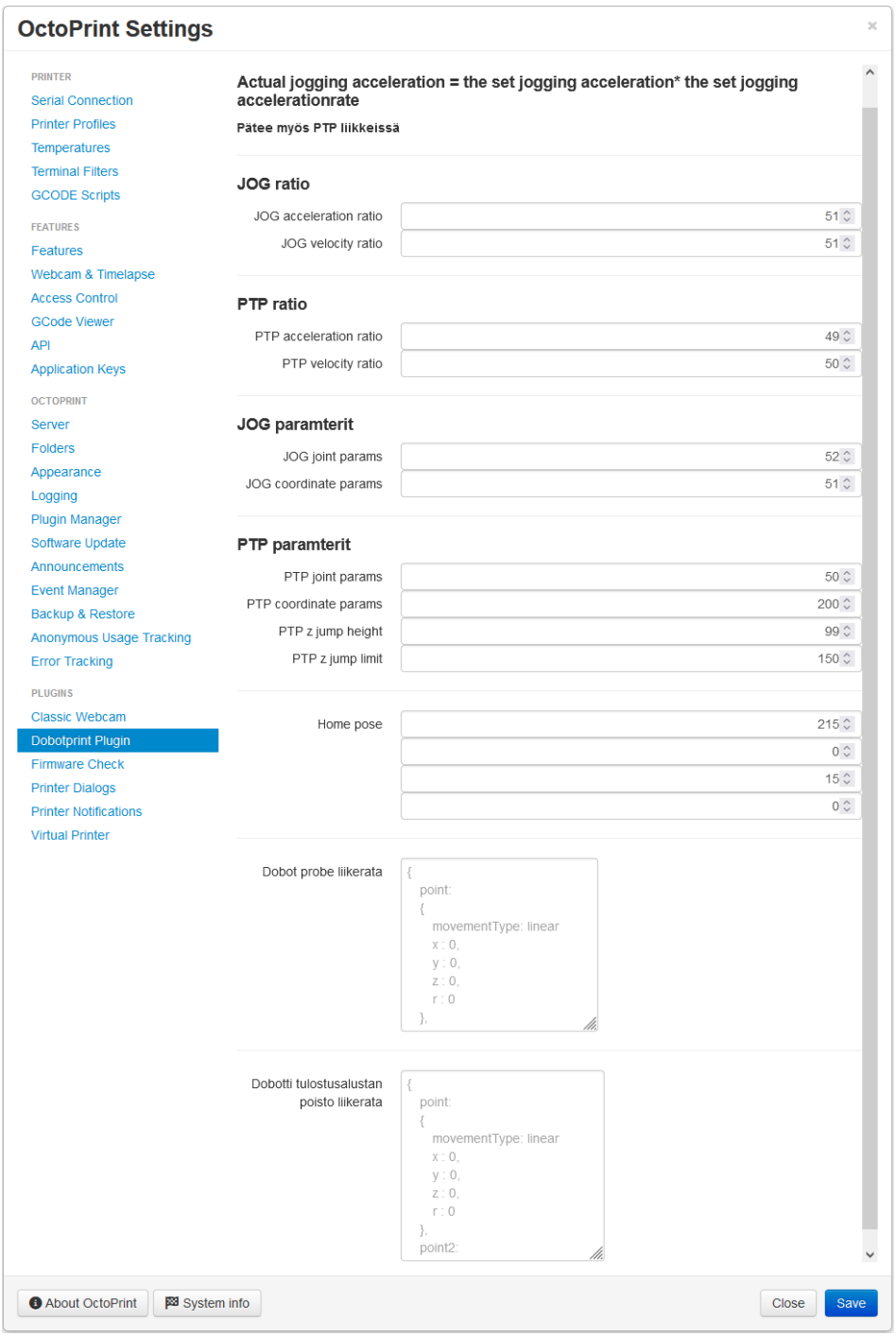
Esimerkkinä syötteen lähettämisestä tiedoston lisääminen tulostusjonoon. Http-pyynnön tekeminen frontendissä tapahtuu OctoPrint-client kirjastolla (Kuvio 24), http-pyynnössä parametrina annetaan tiedostonimi, joka halutaan lisättäväksi tulostusjonoon. Kaikki käyttäjän tekemät toiminnot kuviossa 22 näkyvässä käyttöliittymässä välitetään tällä tavoin backendille.

```
//select valikoiden valittu arvo
self.dobotprint = {
  file: ko.observable(''),
  folder: ko.observable(''),
  staticFolder: ko.observable(''),
  serialport: ko.observable(''),
  homeDobotOnStartUp: ko.observable(false),
  useServoHook: ko.observable(false),
  isPaused: ko.observable(false)
};

self.dobotFilePrint = function () {
    if (self.dobotprint.file() && self.dobotprint.folder()) {
        console.log(self.dobotprint.file() + " filu")
        file = { file: self.dobotprint.file(), folder: self.dobotprint.folder() };
        OctoPrint.simpleApiCommand(plugin_identifier, "dobotFilePrint", file);
    }
};
```

Kuvio 24. http-pyynnön lähettäminen backendille OctoPrint-client kirjastolla

4.1.2 Lisäosan asetukset

OctoPrint tekee lisäosan asetuspaneelin toteuttamisesta helppoa sillä se mahdollistaa lisätä omia dialogeja OctoPrintistä löytyvään asetukset osioon. Asetuspaneelista voi säätää Magicianin parametreja, kuten liikkeiden nopeuden, kotiutuskomennon asennon sekä kaksi tekstikenttää, johon annetaan Magicianille liikekomentoja JSON-formaatissa (Kuvio 25).



Kuvio 25. Lisäosan asetukset

OctoPrint lähettää käyttäjän antamat syötteet backendille, jossa järjestelmä kirjoittaa asetukset config.yaml nimiseen tiedostoon. Esimerkkinä kuviossa 25 näkyvä JOG ratio kentän asetuksen tallentaminen. Huomioitavaa on data-binding attribuutti kuviossa 26 ja kuinka se määrittää nimen asetukselle config.yaml tiedostossa.

```
<form class="form-horizontal">
    <h4>JOG ratio</h4>
    <div class="control-group">
        <label class="control-label">{{ _('JOG acceleration ratio') }}</label>
        <div class="controls">
            <input type="number" class="input-block-level" min="10" max="100"
                data-bind="value: settings.plugins.dobotprint.jogAccelereation">
        </div>
        <label class="control-label">{{ _('JOG velocity ratio') }}</label>
        <div class="controls">
            <input type="number" class="input-block-level" min="10" max="100"
                data-bind="value: settings.plugins.dobotprint.jogVelocity">
        </div>
    </div>
</form>
```

```
dobotprint:
  jogAccelereation: '15'
  jogVelocity: '15'
  bedRemovePath: |-
    {
      "point": {
        "movementType": "linear",
        "x": 200,
        "y": 0,
        "z": 0,
        "r": 0
      },
      "point2": {
        "movementType": "linear",
        "x": 265,
        "y": 0,
        "z": 0,
        "r": 0
      },
      "point3": {
```

Kuvio 26. dobotprint_settings.jijna2 templete sekä config.yaml tiedosto

Lisäosa saa asetukset käyttöön implementoimalla OctoPrint SettingPlugin pohjaluokan. Kuviossa 27 näkyvä metodi mahdollistaa asetusten käyttöön panon käyttäjän tallentaessa uusia asetuksia. Pohjaluokka paljastaa *_settings protetyn*, josta asetuksiin pääse käsiksi avain-arvo -pareilla.

```
class DobotprintPlugin(
    octoprint.plugin.StartupPlugin,
    octoprint.plugin.SettingsPlugin,
    octoprint.plugin.AssetPlugin,
    octoprint.plugin.TemplatePlugin,
    octoprint.plugin.EventHandlerPlugin,
    octoprint.plugin.SimpleApiPlugin
):

    def on_settings_save(self, data):
        octoprint.plugin.SettingsPlugin.on_settings_save(self, data)
        self._logger.info(self._settings.get(["jogVelocity"]))
        self._dobot_printer.apply_dobot_parameters()
```

Kuvio 27. Metodi, jota kutsutaan asetusten tallentamisen yhteydessä

4.2 Backend

4.2.1 Magicianin ajaminen lisäosassa

Magicianin valmistaja tarjoaa kolmannen osapuolen kehittäjille esimerkkejä Magicianin hallintaan useilla eri ohjelmointikielillä verkkosivuillaan, mukaan lukien Python. Lisäosassa hyödynnetään valmistajan esimerkistä C++ jaettua kirjastoa, joka toteuttaa protokollan, jota Magician ymmärtää sekä Python wrapperia, joka

helpottaa jaetun kirjaston käyttöä. Valmistajan Python-esimerkki ei kuitenkaan toiminut sellaisenaan kohdejärjestelmässä koska esimerkin mukana tulleet jaetut kirjastot on käännetty x86-suoritinarkkitehtuurille.

Valmistaja tarjoaa jaetun kirjaston lähdekoodin eli lähdekoodi voidaan kääntää RPI:n ARM-prosessorille. Uudelleen käännämme jaetun kirjaston kohdejärjestelmässä (Kuvio 28).

```
pi@dobot:~ $ cd magiciandll-master/
pi@dobot:~/magiciandll-master $ ls -la
total 36
drwxr-xr-x  3 pi pi  4096 Oct  5 21:31 .
drwxr-xr-x 21 pi pi  4096 Oct  5 21:31 ..
-rw-r--r--  1 pi pi   935 Oct  5 21:32 DobotDll.pro
-rw-r--r--  1 pi pi 11371 Nov 23  2020 LICENSE
-rw-r--r--  1 pi pi   510 Nov 23  2020 README.md
drwxr-xr-x  4 pi pi  4096 Oct  5 21:31 src
-rw-r--r--  1 pi pi   970 Nov 23  2020 version.rc
pi@dobot:~/magiciandll-master $
```

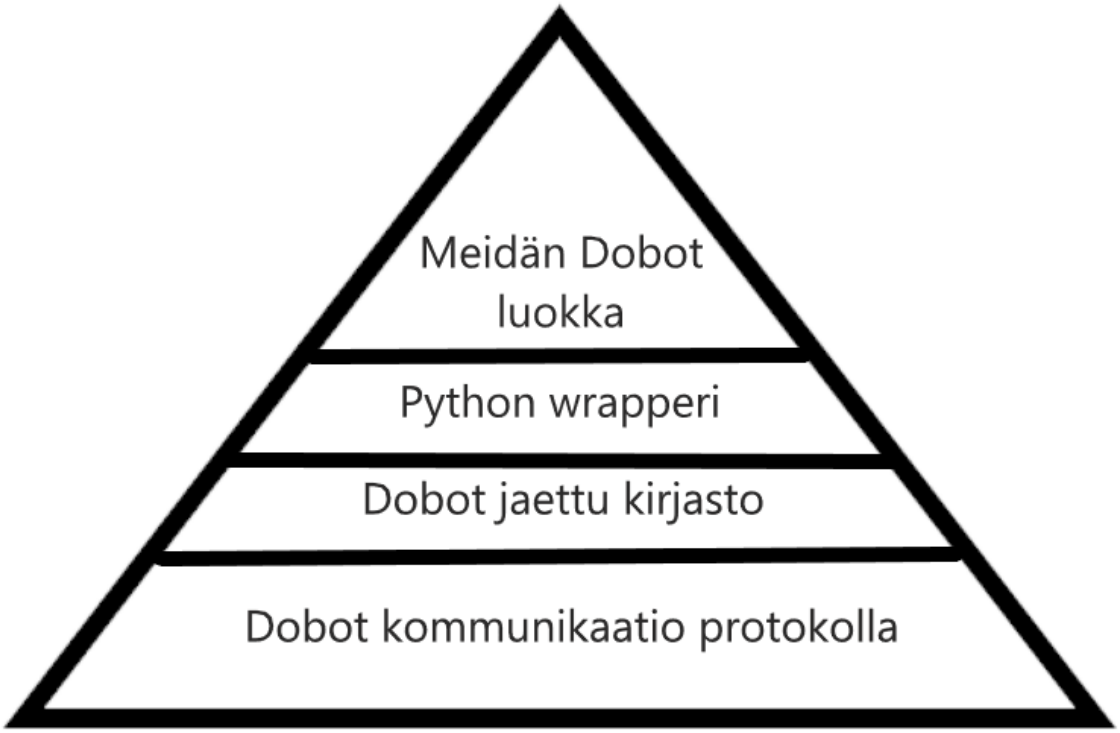
Kuvio 28. Jaetun kirjaston lähdekoodit

Lähdekoodit käännetään QT-työkaluilla, eli ensimmäiseksi asennetaan työkalut ja vaadittavat kirjastot, jonka jälkeen generoidaan qmake-työkalulla make-tiedosto ja ajetaan tämä (Kuvio 29). Tuloksena saadaan ARM-arkkitehtuurille yhteensopiva jaettu kirjasto. Esimerkin jaettu kirjasto korvataan uudelleen käännetyllä versiolla ja esimerkki toimii nyt RPI:llä.

```
pi@dobot:~ $ sudo apt-get install -y libqt5serialport5 libqt5serialport5-dev &&
sudo apt-get install -y qt5-default qt4-dev-tools

pi@dobot:~ $ qmake -o makerfile DobotDll.pro
Info: creating stash file /home/pi/magiciandll-master/.qmake.stash
pi@dobot:~ $ make
pi@dobot:~/magiciandll-master $ cd ..
pi@dobot:~ $ cd output2
pi@dobot:~/output2 $ ls -la
total 380
drwxr-xr-x  2 pi pi   4096 Oct  7 13:58 .
drwxr-xr-x 21 pi pi   4096 Oct  7 13:51 ..
lrwxrwxrwx  1 pi pi     20 Oct  7 13:58 libDobotDll.so -> libDobotDll.so.1.0.0
lrwxrwxrwx  1 pi pi     20 Oct  7 13:58 libDobotDll.so.1 -> libDobotDll.so.1.0.0
lrwxrwxrwx  1 pi pi     20 Oct  7 13:58 libDobotDll.so.1.0 -> libDobotDll.so.1.0.0
-rwxr-xr-x  1 pi pi 377204 Oct  7 13:58 libDobotDll.so.1.0.0
pi@dobot:~/output2 $
```

Kuvio 29. Lähdekoodin kääntäminen

Esimerkistä saatu jaettu kirjasto ja Python wrapperi periaatteessa riittää projektin tarpeisiin, eli Magicianin hallinta lisäosasta on mahdollista. Tein vielä yhden kerroksen tähän eli Python-luokan, johon toteutan omaa toiminnalisuutta käyttämällä esimerkin mukana tullutta Python wrapperia. kuviossa 30 näkyy kaikki eri komponentit Magicianin hallintaan.



Kuvio 30. Magicianin käskyttämisen komponentit

Otetaan esimerkkinä lineaarinen PTP-liikekomento, kuviossa 31 näkee kuinka, käytämme Dobot luokkaa komennon lähettämiseksi Magicianille. Ensimäiseksi Dobot-luokka tulee initialisoida ja parametrina annetaan valmistajan Python esimerkistä saatu Python wrapperi.

```
from . import DobotDllType as dType #Dobot Python wrapperi
from .dobotti import Dobot #Meidän Dobot luokka

self._dobot = Dobot(dType) #Dobot luokka ottaa parametrina Dobot Python wrapperiin
connect_result = self._dobot.connect(serialport) #Yhdistetään Dobottiin, parametrina sarjaportti
                                                 #johon Magician on  liitettynä
result = self._dobot.ptpLinearCommand(True, x, y, z, r) #Liikekomento ottaa parametrina tavoite koordinaatit
```

Kuvio 31. Liikekomennon lähettäminen Magicianille

Metodia (Kuvio 32) voi käyttää kahdella tavalla, ensimmäinen parametri määrittää jääkö koodi odottamaan komennon suorittautumista vai jatketaanko muun koodin suorittamista heti kun komento on saatu lähetettyä Magicianille. Magician pitää indeksiä suoritetuista komennoista ja hyödynnämme tätä edellä mainitun toiminnallisuuden toteuttamiseksi.

```
def ptpLinearCommand(self, blocking, x, y, z, r):
    cmdResult = []
    if self.execute_dobot_commands is True and self.dobot_connected is True:
        if blocking:
            cmdResult = self.dType.SetPTPCmd(
                self.api, self.dType.PTPMode.PTPMOVLXYZMode, x, y, z, r, isQueued=1)
            if cmdResult[1] == 0:  # jos dobot kommunikaatio onnistui
                while self.execute_dobot_commands is True:
                    currentIndex = self.dType.GetQueuedCmdCurrentIndex(self.api)
                    if currentIndex[1] == 0:  # jos dobot kommunikaatio onnistui
                        if cmdResult[0] > currentIndex[0]:# tarkistetaan onko Magician suorittanut komennon.
                            self.dType.dSleep(5)
                            continue
                        else:
                            break
                    else:
                        break
        else:
            cmdResult = self.dType.SetPTPCmd(
                self.api, self.dType.PTPMode.PTPMOVLXYZMode, x, y, z, r, isQueued=1)
        return cmdResult
    return cmdResult
```

Kuvio 32. Dobot luokan *ptpLinearCommand* metodi

Magicianille lähetetyt komennot (Kuvio 33) palauttavat oman indeksinsä ja Magicinilta voi pyytää nykyistä indeksiä eli nykyistä indeksiä voidaan verrata liikekomennon indeksiin ja odottaa tai edetä koodissa tämän mukaan. Tämä toimii koska varsinainen indeksi korottuu vasta, kun Magician on saanut suoritettua komennon loppuun ja annettu komento palauttaa tämän tulevan indeksin.

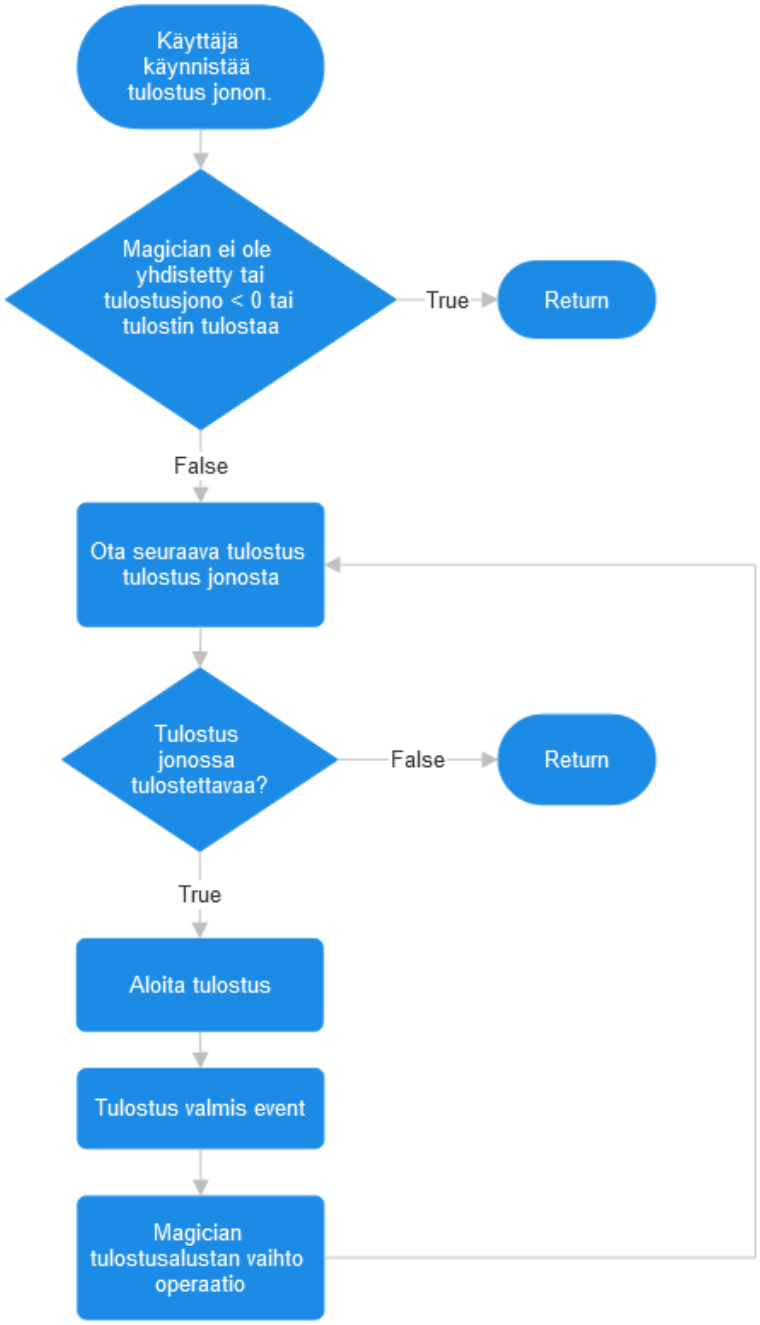
```python
def SetPTPCmd(api, ptpMode, x, y, z, rHead, isQueued=0):
    cmd = PTPCmd()
    cmd.ptpMode = ptpMode
    cmd.x = x
    cmd.y = y
    cmd.z = z
    cmd.rHead = rHead
    queuedCmdIndex = c_uint64(0)
    result = 0
    timeOutCount = 0

    while (timeOutCount < 3):
        result = api.SetPTPCmd(byref(cmd), isQueued, byref(queuedCmdIndex))
        if result != DobotCommunicate.DobotCommunicate_NoError:
            dSleep(5)
            timeOutCount += 1
            continue
        else:
            break

    return [queuedCmdIndex.value, result] #Liikkeen indeksi sekä komennon onnistumisen tulos
```

Kuvio 33. Esimerkki koodien Python wrapper *PTP-komento* funktio

Ajatuksena Dobot-luokassa on myös se, että luokan voi periä. Luokka kattaa kaiken toiminnallisuuden mitä standalone Magicianilla voi tehdä. Jos Magicianiin liitetään valmistajan tarjoamia lisälaitteita, kuten värianturin tai lineaariraide, niin luotaisiin uusi luokka, joka perii Dobot-luokan ja sitten toteuttaa lisälaitekohtaiset toiminnallisuudet omassa luokassaan.

4.2.2 Tulostuksen automatisointi

Logiikka tulostuksen automatisoinnin takana on melko yksinkertainen. Automatisoitu tulostusoperaatio lähtee liikkeelle käyttäjän syötteestä, jonka jälkeen tulostusjonon suorittaminen pyörii OctoPrint *'PrintDone'* eventin ympärillä, jonon tyhjentymiseen asti. Automatisoinnin prosessi on esitetty kuviossa 34, kaaviossa on esitetty erilaisia reunaehtoja ja prosessin eteneminen.



Kuvio 34. Tulostusjonon vuokaavio

Tulostusjonon suorittaminen lähtee liikkeelle käyttäjän syötteestä (kuvio 35). Metodissa käytettävä *_dobot_printer property* on ilmentymä DobotPrinter luokasta, joka sisältää kaiken tulostusjonon suorittamiseen liittyvän.

```
class DobotprintPlugin(
    octoprint.plugin.StartupPlugin,
    octoprint.plugin.SettingsPlugin,
    octoprint.plugin.AssetPlugin,
    octoprint.plugin.TemplatePlugin,
    octoprint.plugin.EventHandlerPlugin,
    octoprint.plugin.SimpleApiPlugin
):
    def on_api_command(self, command, data):

        if command == "jokukomento":
            #Pyy skribulaa....

        elif command == "startPrintQueue":
            if data["useCurrentJobFile"] is True:
                    self._dobot_printer.add_print_to_queue(self._printer.get_current_job()["file"]["path"])

            self._logger.info("aloita tulostusjonon vääntäminen")
            self._dobot_printer.useServoHook = data["useServoHook"]
            self._dobot_printer.start_print_queue()
            self.update_frontend()

        elif command == "stopPrintQueue":
            self._logger.info("Pysäytä tulostusjono")
            self._dobot_printer.stop_print_queue()

        elif command == "pausePrintQueue":
            self._logger.info("pause print queue")
            self._printer.pause_print()

        #jne...
```

Kuvio 35. Tulostusjonon käynnistys käyttäjän syötteestä

Kuviossa 36 näkyvä *start_print_queue* metodi käynnistää tulostusjonon suorittamisen kutsumalla deque_next_print metodia. Metodissa tarkistetaan järjestelmän ja tulostimen tila ennen kuin tulostusjono käynnistetään.

```
Class DobotPrinter():
    def start_print_queue(self):
        if len(self._printer_job_queue) == 0 or self._dobot.dobot_connected is False or self.printer_job_queue_active is True or self._printer.is_printing():
            self._logger.info("printti jono on tyhjä tai jonon tulostus on jo käynnissä")
            return

        self.printer_job_queue_active = True
        self.deque_next_print()
```

Kuvio 36. Tulostusjonon aloitus.

Lisäosan koodi ei saa häiritä OctoPrintin normaalia toimintaa. Jos lisäosassa on pitkää prosessointiaikaa vaativaa koodia kuten while-luuppeja, http-kyselyitä ei OctoPrintin muut komponentit saa tarpeeksi prosessointiaikaa koodinsa suorittamiseen. Esimerkiksi web-käyttöliittymä lakkaa toimimasta, kun OctoPrintin webserveri ei saa prosessorilta aikaa lisäosan käyttäessä kaikki resurssit. Osa Magi-

cianille annetavista komennoista on muun koodin suorituksen estävää, joten koodissa tulee ottaa edellä mainittu huomioon eli ajamme Magician käskyt omalla Thredillä.

Python Thredin kanssa käytetään PriorityQueue datatyyppiä. Siihen lisätään tehtävät, jotka halutaan ajaa erillisellä Thredillä. Kuviossa 37 metodi *deque_next_print* lisää *_dobot_task_queueen* tarvittavat tehtävät tulostusjonon edistämiseksi. Metodissa oleva ehtolause tarkistaa onko kyseessä ensimmäinen tulostus tulostusjonosta, jolloin Magicianin ei oletettavasti tarvitse vaihtaa tulostusalustaa.

```
class DobotPrinter():
  self._dobot_task_queue = queue.PriorityQueue()

  def deque_next_print(self):
      if self.printer_job_queue_active is False:
          return
      if self._worker_running is False:
          if self._current_job is not None:
              self._add_task_to_dobot_queue(
                  PrintTask(task="clear_print_surface", payload={}, priority=3, entry_counter=self._entry_counter))
              self._add_task_to_dobot_queue(
                  PrintTask(task="deque_print_job", payload={}, priority=3, entry_counter=self._entry_counter))
          else:
              self._add_task_to_dobot_queue(
                  PrintTask(task="deque_print_job", payload={}, priority=3, entry_counter=self._entry_counter))
          self._start_worker()

   def _add_task_to_dobot_queue(self, task):
        self._entry_counter += 1
        self._dobot_task_queue.put(task)

   def _start_worker(self):
       if self._worker_running is False:
           self._worker = threading.Thread(
               target=self._execute_dobot_task_queue, daemon=True, name="dobo")

           self._worker.start()
           self._worker_running = True
```

Kuvio 37. Seuraavan tulostuksen ottaminen tulostusjonosta

Thredille määritetty funktio *_execute_dobot_task_queue* (Kuvio 38) suorittaa *_dobot_task_queue* elementit yksi kerrallaan, tässä tapauksessa tulostusalustan vaihdon sekä seuraavan tulostuksen aloittamisen tulostusjonosta. Työjonoon lisätään myös muut tulostusoperaation tarvitsemat tehtävät kuten *deque_print_job*, vaikka tehtävän koodi ei olisi muun koodin suoritusta estävää. Näin koodin suorituksen järjestys pysyy yksinkertaisena.

```
class DobotPrinter():

    def _execute_dobot_task_queue(self):
        while self._dobot_task_queue.empty() is False:

            self.task_executing = True
            task = self._dobot_task_queue.get()

            if task.task == "connect":
                self._logger.info("connect dobot")
                self._connect_dobot(task.payload['serialport'], task.payload['homeDobotOnStartUp'])

            elif task.task == "disconnect":
                self._logger.info("disconnect dobot")
                self._disconnect_dobot()
                self._dobot_printer_fire_event("dobotDisconnected")

            elif task.task == "clear_print_surface":
                self._logger.info("clear table dobo queue")
                self._clear_print_surface()

            elif task.task == "deque_print_job":
                self._logger.info("deque print job")
                self._deque_print_job()
                self._dobot_printer_fire_event("dobotDequePrint")

            #jne...

            self.task_executing = False
            self._dobot_task_queue.task_done()
        self._worker_running = False
        self._logger.info("task que suoritettu loppuun")
```

Kuvio 38. *PioritыQueuen* elementtien läpikäynti

Kuviossa 39 näkyvä metodi lähettää Magicianille tarvittavat liikekomennot tulostusalusta vaihtamista varten. Liikekomennot saavat parametrinsa *_bed_remove_path* Python dictionarysta, joka initialisoidaan kuviossa 25 näkyvässä tekstikentän sisällöstä. Kentän teksti annetaan JSON-formaatissa. *UseServoHook* ehtolauseen sisältö ohjaa kuviossa 15 näkyvää servomoottoria, joka on kytketty kuviossa 7 näkyvään IO-paneelin. Liikekomennoille annettava Z-parametri selitetään omassa luvussa 4.2.3.

```
class DobotPrinter():

    def _clear_print_surface(self):

        if self._bed_remove_path is None:
            self._dobot_printer_fire_event("dobotError",
                                           dict(type="dobotJsonError",
                                                msg="bedRemovePath JSON puuttuu tai oli virheellistä."))
            return
        #pwm pulssit
        neutral = 6.6
        max = 13.2
        min = 2
        #pitäsisi olla  pedin vastakappaleen keskikohta lol (ja onkin!)
        z = self._probe_point_z - self._probe_and_tool_offset - self._printer_bed_lift_tool_center

        if self.useServoHook == True:
            result = self._dobot.setIOMultiplexing(address=11, type=2)
            result = self._dobot.setIOMultiplexing(address=10, type=1)
            result = self._dobot.setIODO(address=10, level=1)
            #avataan koukku varalta jos jäänyt kiinni asentoon
            result = self._dobot.setPWM(address=11, frequency=50, dutyCycle=neutral)

        result = self._ptp_komento(self._bed_remove_path["point"]["movementType"],
                                   self._bed_remove_path["point"]["x"],
                                   self._bed_remove_path["point"]["y"],
                                   z,
                                   self._bed_remove_path["point"]["r"])

        result = self._ptp_komento(self._bed_remove_path["point2"]["movementType"], ...
        if self.useServoHook == True:
            result = self._dobot.setPWM(address=11, frequency=50, dutyCycle=min) #suljetaan koukku
            time.sleep(1)
        result = self._ptp_komento(self._bed_remove_path["point3"]["movementType"],...
        pose = self._dobot.getDobotPose() #dobo pyörii y akselilla
        result = self._ptp_komento(self._bed_remove_path["point4"]["movementType"], ...

        #Levyn asetus takaisin pedille.
        time.sleep(3)
        result = self._ptp_komento(self._bed_remove_path["point4"]["movementType"],...
        result = self._ptp_komento(self._bed_remove_path["point2"]["movementType"],...
        if self.useServoHook == True:
            result = self._dobot.setPWM(address=11, frequency=50, dutyCycle=neutral)
            time.sleep(1)
            result = self._dobot.setIODO(address=10, level=0)
        result = self._ptp_komento(self._bed_remove_path["point"]["movementType"],...
        result = self._ptp_komento("linear", 220, 0, 15, 0)

    def _ptp_komento(self, ptpType, x, y, z, r):
        if ptpType == "jump":
            result = self._dobot.ptpJumpCommand(True, x, y, z, r)
        elif ptpType == "linear":
            result = self._dobot.ptpLinearCommand(True, x, y, z, r)
        elif ptpType == "joint":
            result = self._dobot.ptpJointCommand(True, x, y, z, r)

        if result:
            if result[1] == 0:
                self._logger.info("ptpCMD onnistui")
                self._dobot_printer_fire_event("ptpCMD")
                return True
            else:
                self._logger.info("ptpCMD epäonnistui")
                self._dobot_printer_fire_event("dobotError", dict(type="dobotCMDError",result=result[1],
                                               msg="ptpCMD epäonnistui"))
                return False
        else:
            self._logger.info("ptpCMD epäonnistui, Dobot ei yhdistetty?.")
            self._dobot_printer_fire_event("dobotError", dict(type="dobotCMDError",result=result[1],
                                           msg="ptpCMD epäonnistui, Dobot ei yhdistetty?"))
            return False
```

Kuvio 39. Magicianille syötettävät liikekomennot tulostusalustan vaihtamiseksi

Kuvion 40 *_deque_print_job metodi* ottaa tulostusjonosta seuraavan tulostettavan tiedoston ja aloittaa uuden tulostuksen käyttämällä *_printer propertya*, joka on OctoPrintin sisäinen moduuli yhdistetyn 3D-tulostimen hallintaan. Jos tulostusjono on tyhjä, lopetetaan tulostusjonon suorittaminen.

```
class DobotPrinter(self):

  def _deque_print_job(self):
        if len(self._printer_job_queue) > 0:
            self._current_job = self._printer_job_queue.pop()
            #tulostin aloittaa valitun tiedoston tulostamisen
            self._printer.select_file(self._current_job, False, printAfterSelect=True)
        else:
            self._current_job = None
            self.printer_job_queue_active = False
```

Kuvio 40. Seuraavan tulostuksen aloittaminen tulosjonosta

Käyttäjän käynnistettyä tulostusjonon ja ensimmäisen tulostuksen valmistuttua siirtyy tulostusjonon edistyminen OctoPrint EventHandlerPlugin pohjaluokan *on_event* metodille tulostusjonon tyhjentymiseen asti (Kuvio 41). PrintDone eventti kutsuu samaa metodia mitä kutsuttiin käyttäjän aloittaessa tulostusjono.

```
class DobotprintPlugin(
    octoprint.plugin.StartupPlugin,
    octoprint.plugin.SettingsPlugin,
    octoprint.plugin.AssetPlugin,
    octoprint.plugin.TemplatePlugin,
    octoprint.plugin.EventHandlerPlugin,
    octoprint.plugin.SimpleApiPlugin
):

    def on_event(self, event, payload):
        if "PrintDone" == event:
            self._logger.info("Tulostus valmis")
            self._dobot_printer.deque_next_print()
            self.update_frontend()
        if "PrintCancelled" == event:
            self.update_frontend()
        if "PrintPaused" == event:
            self._dobot_printer.pause_print_queue()
            self.update_frontend()
        if "PrintResumed" == event:

        #jne...
```

Kuvio 41. OctoPrint PrintDone event

4.2.3 Probe

Jos 3D-tulostimen ja Magicianin paikka toisiinsa nähden muuttuu, ei aiemmin annetut liikekomennot enää osu kohdalleen. Tulostimen ja Magicianin tulisi aina olla

kohtisuorassa toisiinsa nähden, jotta tulostusalustan vastakappaleen ja nostotyökalun kohdistaminen olisi helppoa. Kuviossa 16 näkyvässä testialustassa tulostin ja Magician ovat kiinteästi kiinni alustassa. Tulostimen pedin tasauksen jälkeen liikekomennot eivät enää toimineet, sillä tasauksessa tulostusalusta voi siirtyä useita millimetrejä ylös- tai alaspäin. Tällöin liikekomennon Z-komponentti ei enää ole oikealla kohdalla. Parhaassa tapauksessa tulostimen peti olisi kiinteästi kiinni eikä tätä tarvitsisi säätää, mutta halvoissa 3D-tulostimissa kuviossa 12 näkyvät säätörullat ovat yleinen keino kompensoida halpoja epätasaisia tulostin petejä.

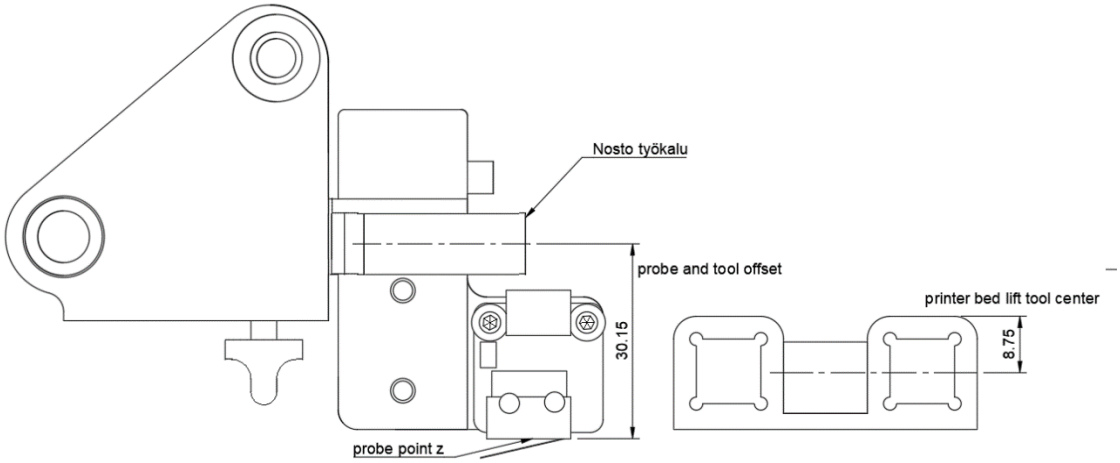
Kokeiluna toteutin Magicianille probe end-effectorin, joka koostuu Ender3-tulostimen rajakytkimestä sekä adapterista, probe yhdistetään kuvion 7 IO-takapaneeliin (Kuvio 17). Proben toiminnallisuus toteutetaan kuvion 42 metodissa, jonka kulku menee seuraavasti. Tulostimelle lähetetään G-koodikäsky, jossa tulostimen peti ajetaan 230 milliä eteenpäin. Tämän jälkeen Magician ajaa itsensä tulostusalustan vastakappaleen yläpuolelle ja alkaa ajamaan probea kohti vastakappaletta juoksutusmoodissa y-akselilla. Tämä tarkoittaa sitä, että Magician jatkaa liikettä, kunnes se saa käskyn pysähtyä. Proben ajaessa kohti vastakappaletta pollataan proben IO-paneelin porttia ja pysäytetään Magicianin liike, kun kytkin laukeaa eli vetää virran maihin. Tämän jälkeen Magicianilta kysytään nykyistä asentoa, jota käytetään liikekomennoille annettavan Z-komponentin laskemisessa.

```
class DobotPrinter():

  def _probe_bed(self):

        if self._probe_path is None:
            self._dobot_printer_fire_event("dobotError", dict(type="dobotJsonError",
                                                              msg="probePath JSON puuttuu tai oli virheellisä."))
            return

        self._printer.commands(["G28 Y", "G1 Y230 F1200"])
        self._logger.info("g-koodi komento test")
        time.sleep(20) # _prinnter.commands funktio suoriutuu asyncisti niin pitää odottaa...
        # [0] napin tila, [1] kommunikaation onnistuminen probe irti == taso 1

        result = self._ptp_komento(self._probe_path["point"]["movementType"],
                                   self._probe_path["point"]["x"],
                                     self._probe_path["point"]["y"],
                                       self._probe_path["point"]["z"],
                                         self._probe_path["point"]["r"])
        if result is not True:
            return

        set_probe_multiplex = self._dobot.setIOMultiplexing(address=14, type=3)  # input 3.3V
        if set_probe_multiplex:
            if set_probe_multiplex[1] == 0:
                probe_level = self._dobot.getIODI(14)
                if probe_level: # proben tason luku palautti jonkin tuloksen, [] = ei tehty dobot komentoa.
                    if probe_level[1] == 0: # probe tason luku onnistui
                        if probe_level[0]: # proben taso, 0 tai 1
                            self._logger.info("dobot probe ei kytketty?")
                            self._dobot_printer_fire_event("dobotError",
                                                           dict(type="dobotProbeError",result=probe_level[0],
                                                                msg="Dobot probe ei kytketty?"))
                            return
                        else:
                            self._logger.info("dobot probe OK!")
                            # probeta tässä
                            self._dobot.setJogCmd(6) # -Y tila
                            while True:
                                probe_level = self._dobot.getIODI(14)[0]
                                if not probe_level:
                                    continue
                                else:
                                    self._dobot.setJogCmd(0) # idle tila
                                    self._probe_point_z = self._dobot.getDobotPose()[2] # z komponentti

                                    # https://community.octoprint.org/t/sync-octopi-settings-when-changing-them-in-the-backend/52107
                                    self._settings.set(["probeZ"], self._probe_point_z)
                                    self._settings.save(trigger_event=True)
                                    time.sleep(2)
                                    break
                            #result = self._ptp_komento("jump", self._dobot.homePose.x, self._dobot.homePose.y, self._dobot.homePose.z, 0)
                            self._dobot.ptpJumpCommand(True, self._dobot.homePose.x, self._dobot.homePose.y, self._dobot.homePose.z, 0)
                            # if result is not True:
                            #     return
                            self._logger.info("dobot probetus valmis")
                    else:
                        self._logger.info("dobot probe epäonnistui, ei saatu luettua proben tasoa?")
                        self._dobot_printer_fire_event("dobotError", dict(type="dobotProbeError",
                                                                          result=probe_level[1],
                                                                            msg="Dobot probe, ei saatu luettua proben tasoa?"))
                else:
                    self._logger.info("dobot probe ei kytketty?")
                    self._dobot_printer_fire_event("dobotError", dict(type="dobotProbeError",
                                                                      result=set_probe_multiplex[1],
                                                                        msg="Dobot probe päälle kytkeminen epäonnistui?"))
        else:
            self._logger.info("dobot probe ei kytketty?.")
            self._dobot_printer_fire_event("dobotError", dict(type="dobotProbeError",
                                                              result=set_probe_multiplex[1],
                                                                msg="Probe päälle kytkeminen epäonnistui?, Dobot ei yhdistetty?"))
```

Kuvio 42. Nostotyökalun vastakappaleen pinnan probetus

Z-komponentti saadaan yksinkertaisella laskutoimituksella, joka näkyy kuviossa 39 metodin alussa. Laskutoimituksen *_probe_point_z* saadaan probetuksen tuloksena ja kaksi muuta komponenttia on peräisin CAD-mallennusohjelmasta (Kuvio 43). Omaksi yllätykseni tämä toimi ensi yrittämällä ja nostotyökalu osui vastakappaleeseen ilman että minun piti manuaalisesti selvittää liikekomennon Z-

komponentti. Tätä olisi mahdollisesti voinut hyödyntää myös muiden liikekomennon komponenttien selvittämiseen, mutta aika ei riittänyt tähän.



Kuvio 43. Laskutoimituksen lukujen havainnollistaminen

5 POHDINTA

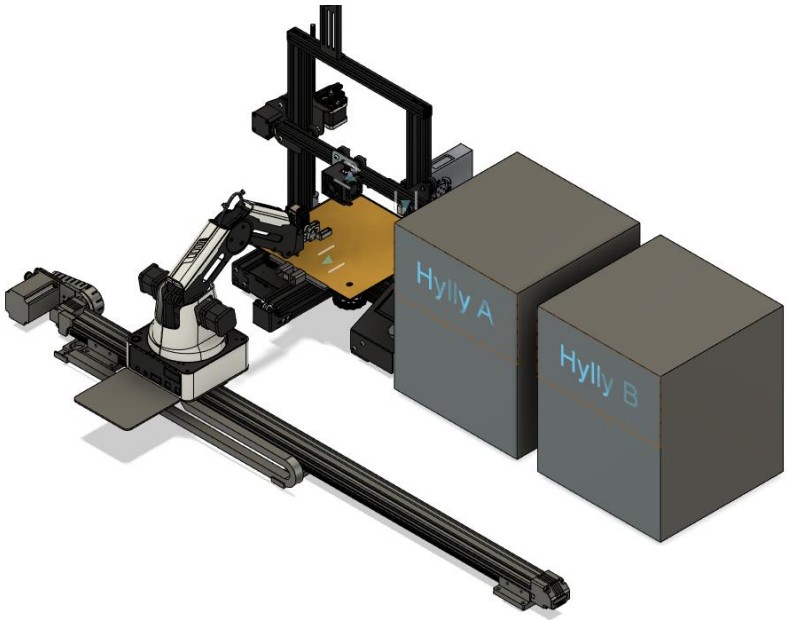
OctoPrint oli projektiin hyvä valinta ja lisäosan kehittäminen oli sujuvaa, vaikka minulla ei aiempaa kokemusta tästä ollut. Tässä auttoi se, että OctoPrintin dokumentaatio on hyvä, kehitysyhteisö on aktiivinen ja koodi esimerkkejä löytyy paljon. Lisäosa tekee sen mitä alkujaan suunnittelin eli käyttäjä lisää G-koodi tiedostoja tulostusjonoon, lisäosa aloittaa tulosteen jonosta ja Magician tekee toimenpiteeseensä tulostuksen valmistuttua ja tämä sykli jatkuu, kunnes jono on tyhjä.

Magician soveltuu tehtävään aika hyvin, isona kysymys merkkinä oli Magicianin kyky tuottaa tarpeeksi voimaa sekä liikkeiden toistettavuus. Projektin aikana selvisi, että Magicianin nostovoima on riittävä valmiiden tulosteiden pois siirtämiseksi tulostimen pediltä. Magicianin toistettavuus näyttäisi myös olevan riittävä testien perusteella. Testeissä lisään tulostusjonoon kymmenen G-koodi tiedostoa ja annan lisäosan suorittaa jonon tyhjäksi. Testissä seurasin, muuttuiko tulostusalustan paikka tulostimen pedillä Magicianin suorittaessa tulostusalustan vaihdon tulosteen valmistuttua. Magician suoriutui testeistä hyvin ja sai nostettua ja palautettua tulostusalustan onnistuneesti lähtöpisteeseen eli Magicianin toistettavuus on kohtuu hyvä. Suoritin testin pariin kertaan samoilla tuloksilla.

Projektissa käytettävä Ender3-tulostin voi olla heikkolenkki järjestelmän luotettavuuden kannalta, mutta hyvin säädettynä tulostuksen onnistumisprosentti on kohtuu hyvä. Arvio tulostimen luotettavuuteen perustuu henkilökohtaiseen kokemukseen.

Lisäosassa ensimmäinen asia, jota lähtisin viemään eteenpäin, on Magicianin hallintaan lisäosassa käytettävän C++ jaetun kirjaston korvaamisen kokonaan Pythonilla. Tämä vähentäisi tarvittavien riippuvuuksien määrää sekä mahdollisesti parantaisi lisäosan luotettavuutta tarjoamalla paremmin kontrollin koodista, jolla Magiciania ajetaan. Jaetun kirjaston korvaaminen Pythonilla on toteutettavissa, koska Magicianin valmistaja tarjoaa sivustollaan kattavan dokumentaation protokollasta, jota Magician ymmärtää. Inspiraatiota voi hakea myös itse jaetusta kirjastosta koska tämän lähdekoodi on avoimesti saatavilla.

Toisena ja tärkeämpänä kehityskohteena olisi linjaston loppuunsaattaminen eli Dobot lineaariraiteen implementointi sekä tulostusalustan vaihto-operaation loppuun suunnittelu. Alkuperäisenä ideana on, että Magician latoo valmiit tulosteen hyllyyn ja nostaa toisesta hyllystä uuden tulostusalustan (Kuvio 44). Pitää myös tilata kiinalaisia halpoja joustavia PEI-tulostusalustaja, jotta järjestelmää voitaisiin testata kokonaisuudessaan.



Kuvio 44. Illustraatio mihin suuntaan linjastoa alettaisiin viemään

Yksi tärkeä kehityskohde olisi myös PrusaSlicer-profiilin luonti. Tavoitteena olisi luoda profiili, joka nopeuttaisi uuden tulostuksen aloittamista tulostusjonosta. Profiilissa muokattaisiin lähinnä lopetus ja aloitus G-koodia, jonka PrusaSlicer lisää tulostettavan G-koodin loppuun. Muutoksia voisi esimerkiksi olla, että tulostin ei tee kotiutuskomentoa tulostuksien alkuun tai ettei lämmityselementtejä kytketä pois päältä tulostuksen lopuksi.

Projektin aikana syntyneessä järjestelmässä on mielestäni potentiaalia ja tästä saisi toimivan, jos se vietäisiin loppuun. Keskittyisin enemmän linjaston sekä nostotyökalun ja tämän vastakappaleen jatkosuunnitteluun sillä näissä on paljon parantamisen varaa. Lisäosa suoriutuu tehtävästään jo ihan hyvin eli järjestelmä hyötyisi enemmän fyysisten komponenttien suunnitellusta.

Projektin aikana aiempi osaamiseni kehittyi CAD-mallinnuksen sekä Python ja JavaScript ohjelmointikielien osalta. Uusia asioita oli Dobot Magician robottikäsivarsi sekä OctoPrint-lisäosien kehitys, joista minulle jäi kummastakin hyvä käsitys ja osaaminen. Osaan antaa Magicianille komentoja sekä ymmärrän kuinka Magician komennot käsittelee sisäisesti, osaan myös käyttää Magicianin takaa löytyvää IOT-paneelina hyödyksi uuden toiminnallisuuden toteuttamiseksi. Olen aiemmin käyttänyt OctoPrintiä normaaliin 3D-tulostamiseen, mutta nyt tiedän OctoPrintin toiminasta syvällisemmin ja kykenisin lisäämään tähän omaa toiminnalisuutta lisäosajärjestelmän avulla. En usko, että hyödyn Dobot Magician osaamisestani ellen jotenkin päädy viemään tätä projektia loppuun, mutta OctoPrint lisäosakehitys osaamisestani voin mahdollisesti käyttää, vaikka omissa projekteissa.

LÄHTEET

Duet3D 2023a GCode Everywhere. Viitattu 26.7.2023 https://docs.duet3d.com/en/User_manual/Reference/Gcodes.

Duet3D 2023b Innovation. Viitattu 26.7.2023 https://docs.duet3d.com/User_manual/RepRapFirmware/RepRapFirmware_overview.

Duet3D 2023c Introduction. Viitattu 30.7.2023 https://docs.duet3d.com/User_manual/Machine_configuration/SBC_setup.

Duet3D 2023d Duet Web Control Manual. Viitattu 19.8.2023 https://docs.duet3d.com/User_manual/Reference/Duet_Web_Control_Manual.

Duet3D 2023e DWC .Viitattu 19.8.2023 https://docs.duet3d.com/manual/dwc/dwc23_01_intro_dark.png.

Duet3D 2023f SBC setup. Viitattu 30.7.2023 https://docs.duet3d.com/manual/configuration/sbc_setup_03.jpg.

Haley, N. 2022. What is Klipper Firmware and why would you want it. Viitattu 15.7.2023 https://www.obico.io/blog/what-is-klipper-3d-printer-firmware/.

Florian, D. 2023a. Firmware. Viitattu 8.7.2023 https://www.drdflo.com/pages/Guides/How-to-Build-a-3D-Printer/Firmware.html.

Florian, D. 2023b. Stepper Drivers. Viitattu 8.7.2023 https://www.drdflo.com/pages/Guides/How-to-Build-a-3D-Printer/Stepper-Driver.html.

Klipper 2023 Features. Viitattu 16.7.2023 https://www.klipper3d.org/Features.html.

Lapland Robotics 2020. Projekti. Viitattu 8.7.2023 https://laplandrobotics.com/fi/projekti/.

Marlin 2023 What is Marlin. Viitattu 9.7.2023 https://marlinfw.org/docs/basics/introduction.html.

Moonraker 2023 Welcome to Moonraker Documentantion. Viitattu 13.8.2023 https://moonraker.readthedocs.io/en/latest/.

OctoPrint 2023 The snappy interface for your 3D pinter. Viitattu 26.8.2023 https://octoprint.org/#compatible-and-extendable.

RepRap 2020 RepRap Firmware. Viitattu 25.7.2023 https://reprap.org/wiki/RepRap_Firmware.

Shenzhen Yuejianf Technology Co., Ltd. 2020. Dobot Magician User Guide. Viitattu 29.8.2023 https://www.dobot-robots.com/service/download-center?keyword=&products%5B%5D=316.