jamk

Laiteprototyypin kehitys avoimen lähde-koodin työkaluilla

Asko Ropponen

Opinnäytetyö, AMK
Toukokuu 2023
Tieto- ja viestintätekniikka

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Ropponen, Asko

Laiteprototyypin kehitys avoimen lähdekoodin työkaluilla

Jyväskylä: Jyväskylän ammattikorkeakoulu. Toukokuu 2023, 86 sivua.

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Erityistarkoituksiin tarkoitettuja sovelluksia, kuten esimerkiksi ääni- tai videoeditointiohjelmia ohjataan usein erillisellä ohjaimella kuin pelkästään tietokoneen hiirellä ja näppäimistöllä. Usein nämä ohjaimet ovat tarkoitettu ammattikäyttöön, ja ovat täten hintavia. Opinnäytetyön tavoitteena oli kehittää tällaisen ohjainlaitteen prototyyppi avoimen lähdekoodin työkaluja käyttäen, selvittääkseen onko mahdollista toteuttaa käyttökelpoinen laite harrastelijoiden saatavilla olevilla työkaluilla ja menetelmillä.

Suunnittelun lähtökohtana vertailtiin muutamaa olemassa olevaa laitetta ominaisuuksiltaan, ja tarkasteltiin mitä ominaisuuksia suunniteltavaan laitteeseen tulisi sisällyttää. Laitteessa käytettiin mikrokontrollerina Arduino Micro mikrokontrolleria ja laitteeseen suunniteltiin myös piirilevy ja fyysinen kotelo. Kotelo suunniteltiin Blender ohjelmassa, ja piirilevy KiCad-ohjelmistossa. Kotelo valmistettiin 3D-tulostamalla, ja piirilevy hankittiin erältä piirilevyjen valmistajalta. Laitteen ohjelmisto ohjelmoitiin Arduino-kehitysympäristössä, hyödyntäen valmiita ohjelmakirjastoja jotka helpottavat ohjelmiston kehittämistä. Työssä käytettiin esimerkiksi kirjastoja jotka emuloivat HID-laitteita, kuten hiirtä ja näppäimistöä. Ohjainlaite toimii emuloiden näppäimistön ja hiiren liikkeitä, jotta tietokoneelle ei täytyisi asentaa erillistä ajuriohjelmistoa, jolloin laite on mahdollisimman monikäyttöinen.

Ohjainlaitteen suunnittelu ja toteutus onnistui hyvin avoimen lähdekoodin työkaluja käyttäen. Huomattiin kuitenkin että avoimen lähdekoodin CAD-ohjelmistot eivät ole kovin helppokäyttöisiä, tästä syystä laitekotelo suunniteltiin Blender 3D-mallinnussovelluksessa, joka on tarkoitettu enemmänkin animaation ja grafiikan tekoon kuin fyysisten esineiden suunnitteluun. Blender soveltuukin 3D-tulostettavien kappaleiden suunnitteluun muutaman lisäosan avulla.

Lopullista ohjainlaitetta testattiin musiikintuotantosovelluksen kanssa. Ohjainlaite todettiin toimivaksi ja hyödylliseksi, joskin potentiaalisia muutoksia ja tulevaisuuden kehityskohteita ilmeni. Vaikka laite toimii laajalti eri sovelluksissa lähettämällä yksinkertaisia näppäinkomentoja ja hiiren liikkeitä, on useimmissa sovelluksissa eri pikanäppäimet kuin toisissa, joten yksi laitteen konfiguraatio ei käy joka sovellukseen. Lisäksi jokin toinen mahdollinen tapa lähettää komentoja, kuten esimerkiksi MIDI-standardi, olisi hyödyllinen joidenkin sovellusten kanssa. Kaiken kaikkiaan, ohjainlaite kuitenkin oli sopiva tarkoitukseensa.

Avainsanat (asiasanat)

sulautetut järjestelmät, HID, Arduino, avoin lähdekoodi, C++, piirilevyt, PCB, CAD, 3D-tulostus, KiCad, Blender

Muut tiedot

Liitteet: Laitteen piirikaavio (1 sivu), Laitteen ohjelmakoodi (18 sivua).

jamk

Kuvailulehti

Ropponen, Asko

Developing a device prototype with open source tools

Jyväskylä: JAMK University of Applied Sciences, May 2023, 86 pages.

Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

Special purpose software, such as audio or video editing software are often controlled with a separate, purpose built controllers, than just with a keyboard and mouse. However, these devices are often meant for professional use, and as such are expensive. The goal of this thesis was to develop a prototype of similar control device, using open source tools, to find out if it is viable to develop an prototype using only software and tools available to hobbyists.

Design process of the device was started by comparing a few existing devices and their features, from which the properties to be included in the designed device were decided. Arduino Micro was used as the micro controller for the device, and also a case and a PCB were designed for it. Blender was used for 3D modeling the casing, and the PCB was designed with KiCad. The case was 3D-printed, and the PCB acquired from a PCB fabrication house. Using Arduino ecosystem made developing easier, for example multiple libraries made for Arduino devices were utilized in this project. Most importantly various HID libraries, which emulated for example a computer keyboard and a mouse. Using this keyboard and mouse emulation approach, it is not necessary to install any special drivers to a computer, so the device can be used easily in wide variety of software and devices.

Creating the control device using only open source tools was successful. However it was deduced that open source CAD-software is not very user friendly, therefore the casing of the device was designed in Blender. Blender is designed more for animation and visual 3D-modelling, instead of CAD work, but using some addons made it possible to use Blender for modeling for 3D-printing.

Using the finalized control device was tested with an digital audio workstation software. The created control device was found to be usefull, but also some potential changes and improvements were discovered. Not all software uses same hotkeys, so a single hotkey configuration doesn’t work with all software, even if the key press based approach allows the device to control just about any software. In addition, some other way to send commands could be useful, like for example MIDI-standard. Overall, the control device was found to be fit for it’s purpose.

Keywords/tags (subjects)

embedded systems, HID, Arduino, open source, C++, circuit boards, PCB, CAD, 3D printing, KiCad, Blender

Miscellaneous

Appendixes: Circuit Diagram of the control device (1 page), Source code of the control device (18 pages).

Sisältö

Kuviot

Taulukot

1 Johdanto

Erityistarkoituksiin tarkoitettuja sovelluksia, kuten esimerkiksi ääni- tai videoeditointiohjelmia ohjataan usein erillisellä ohjaimella kuin pelkästään tietokoneen hiirellä ja näppäimistöllä, jotta ohjelmaa voidaan käyttää tehokkaammin. Tällaisia erikoiskäyttöön suunniteltuja laitteita on ollut markkinoilla pitkään, mutta usein niiden hinnat ovat todella korkeat, varsinkin jos ei tarvitse näitä työkaluja työssään. Myös erilaisia itse rakennettavia ratkaisuita löytyy internetistä jo ennestään useita, mutta ne eivät aina sovi kaikkiin käyttötarkoituksiin, saati jokaisen käyttäjän henkilökohtaisiin vaatimuksiin. Täten olisi hyödyllistä, että omiin käyttötarkoituksiin sopivan ohjainlaitteen toteuttaminen onnistuisi mahdollisimman matalalla kynnyksellä.

Opinnäytetyön tavoitteena oli kehittää ohjainlaitteen prototyyppi, jota voi käyttää tietokoneen ohjaamiseen. Samalla tarkasteltaisiin, kuinka tällaisen laitteen suunnittelu onnistuu ilmaisilla avoimen lähdekoodin työkaluilla. Tällöin laitteen toteutus ei estyisi mikäli ei ole mahdollista hankkia kalliita ohjelmistolisenssejä maksullisiin ohjelmistoihin (joita on yleisesti käytössä laitesuunnittelualalla). Työllä ei ole erillistä toimeksiantajaa, sillä tämän opinnäytetyön aihe syntyi omasta tarpeestani tehostaa työskentelyä eri sovelluksissa. Kuitenkin työn lopputuloksena olevan laitteen suunnitelmat ja ohjelmakoodi sekä muut tiedot jaettiin lopuksi sopivalla avoimella lisenssillä, jotta työn tulokset ovat hyödynnettävissä laajasti.

Laitteen suunnittelun alkupisteeksi valittiin suppeahko selvitys siitä, minkälaisia valmiita ohjainlaitteita on saatavilla, jotta voitiin määritellä minkälaisia ominaisuuksia suunniteltavaan laitteeseen tulisi tehdä. Lisäksi selvitettiin, minkälaisia ohjainlaitteita harrastelijat ovat toteuttaneet jo ennestään ja voiko näistä toteutuksista ottaa oppia tämän työn laitteen suunnittelussa. Näiden perusteella laadittiin alustava suunnitelma laitteen fyysisestä olomuodosta ja ominaisuuksista. Laitteen alustavassa suunnittelussa otettiin huomioon myös se, että laite soveltuisi helposti harrastelijatason työvälineillä valmistettavaksi. Esimerkiksi prototyypin laitekotelon suunnittelun lähtökohta oli, että se soveltuisi 3D-tulostettavaksi vaikka harrastelijatason 3D-tulostimilla ja piirilevyn suunnittelussa puolestaan otettiin huomioon, että sen voi hankkia edullisesti haluamaltaan toimittajalta tai tarvittaessa valmistaa itse. Laitteen ohjaimeksi valikoitiin Arduino -pohjainen mikrokontrolleri koska ne ovat laajalti käytössä. Niiden kanssa toimimiseen on siis saatavilla todella paljon apua ja resursseja jolloin kehitystyö on yksinkertaisempaa ja nopeampaa.

2 Taustatietoa ohjainlaitteista

2.1 HID-laitteet

HID-laite (Human Interface Device) termi tarkoittaa jotain laitetta, jota ihmiset käyttävät tietokoneen ohjaukseen. Tällaisia ovat esimerkiksi näppäimistöt sekä osoitinlaitteet. Myös kaikenlaiset laitteissa esiintyvät kontrollit ovat HID-laitteita, kuten esimerkiksi kaukosäätimet ja peliohjaimet. Lisäksi HID USB-laiteluokkaan kuuluu laitteita jotka eivät välttämättä vaadi ihmistä operoimaan niitä, mutta tuottavat dataa samoin kuin em. laitteet. Tällaisia ovat esimerkiksi viivakoodinlukijat ja anturit, kuten vaikka lämpömittarit. Jotkin HID-laitteet myös antavat palautetta käyttäjälleen, joten USB HID-luokka tukee myös kommunikaatiota laitteen suuntaan. (Device Class Definition for HID 1.11 2001, 11-12.)

Näitä HID-laitteita on tarpeen tutkia jotta saadaan kokonaiskuva erilaisista ohjainlaitteista, sekä missä käyttötarkoituksessa niitä käytetään. Markkinoilla on esimerkiksi kuvankäsittelyyn sekä ääni- tai videoeditointiin tarkoitettuja laitteita. Peliohjaimetkin voisi lukea tällaiseksi laitteeksi ja niitäkin on erilaisia eri tarkoituksiin. Jotkin laitteet myös käyttävät MIDI-standardia ohjatakseen yhteensopivaa sovellusta, kun taas tiettyjä sovelluksia voi ohjata vain yhteensopivalla ohjaimella.

2.2 Olemassa olevat toteutukset ja laitteet

2.2.1 Kaupalliset tuotteet

Edullisimmasta päästä alkaen löytyvät erilaiset peliohjaimet. Yleensä peliohjaimet kuitenkin soveltuvat huonosti muuhun kuin pelien ohjaukseen, koska niitä joko täytyy pidellä käsissään (jolloin kaikkia nappeja ei voi käyttää jos ohjain on vaikka päydällä) tai ovat suuria ja kiinteämmin asennettuja (kuten ratit ja lentokoneohjamet). Tavallisten peliohjainten lisäksi markkinoilta löytyy erilaisia näppäimistön ja erillisen peliohjaimen sekoituksia, esimerkiksi Razer Tartarus Pro, missä on 20 näppäimistön nappia sekä ristiohjain kuten peliohjaimessa (Epstein 2020). Hinnaltaan Tartarus Pro on hiukan kalliimmalla puolella, valmistajan omassa verkkokaupassa se maksaa 159.99€ (Razer Tartarus Pro. N.d). Razerin tuotteet ovat joka tapauksessa tuettuja vain Windows-käyttöjärjestelmällä, joten esimerkiksi Linux-käyttöjärjestelmällä on käytettävä kolmannen osapuolen ratkaisua nimeltä OpenRazer (Larabel 2022). Yleisesti peliohjaimissa on kuitenkin se huono puoli, että koh-

desovelluksen (kuten videoeditori) täytyy tukea peliohjainta ohjainlaitteena. Toinen vaihtoehto on että peliohjain emuloi näppäimistökomentoja, jolloin laite voi tehdä mitä vain näppäimistölläkin voi, mutta siinäkin on omat ongelmansa, kuten vaikka komentojen päällekkäisyys. Peliohjaimista myös usein puuttuu sellaisia ohjaintapoja jotka olisivat hyödyllisiä esimerkiksi videoeditoinnissa (kuten erilaiset rullat).

Videon, kuvien ja äänen käsittelyohjelmia varten suunniteltuja laitteita on markkinoilla eri hintaisia ja erilaisilla ominaisuuksilla. Esimerkiksi suomalainen yritys Loupedeck myy erilaisia monikäyttöisiä ohjainlaitteita, kuten esimerkiksi Loupedeck CT, missä on nappeja ja rullia ja lisäksi myös kosketusnäyttö. Laite tukee useita Adoben ohjelmistoja, kuin myös muita video ja valokuvan muokkausohjelmia (Fisher 2020a). Laitteen konfiguroiminen voi kuitenkin olla vaikeaa, esimerkiksi Fisher (2020a) kirjoittaa arvostelussaan, että asetusohjelma on monimutkainen ja hankala käyttää, jolloin halutun toiminnallisuuden saavuttaminen laitteella on haastavaa.

Eräs toinen vaihtoehto on modulaarinen Monogram Creative Console. Tämä laite koostuu eri ohjainmoduuleista joissa kussakin tietynlainen ohjain, kuten nappi tai rulla. Myös Monogrammin laite tukee Adoben ohjelmistoja ja joitain samoja video- ja valokuvanmuokkausohjelmia kuin Loupedeck CT, mutta esimerkiksi myös Unreal Engine -pelimoottorin editoria. Lisäksi se voi ohjata sovelluksia käyttäen MIDI-protokollaa tai emuloimalla näppäimistöä, hiirtä tai peliohjainta. (Fisher 2020b.) Myöskään Monogrammin asetusohjelmisto ei ole ongelmaton. Esimerkiksi Van Hemert (2021) kritisoi asetusohjelman käytettävyyttä, kuinka se on yleisesti sekava ja esimerkiksi eri toimintojen löytäminen on työlästä ja kaikkia toimintoja ei voinut edes säätää graafisesta käyttöliittymästä, vaan asetustiedostoa täytyi muokata manuaalisesti.

Muuta yhteistä näille tuotteille (kuin ohjelmistojen ongelmallisuus) on hinta, sillä se on korkea. Työn kirjoittamisen aikaan Loupedeck CT maksaa 499,00 € ja Monogram Creative Consolen hinta riippuen moduulien määrästä on noin 250-900 € (Loupedeck CT. N.d; Monogram Shop N.d.) Sama pätee myös useimmille muille tälläisille ammattilaisille tarkoitetuille tuotteille, jotkin ovat vielä kalliimpia.

2.2.2 Harrastelijoiden ratkaisut

Erilaisia harrastelijoiden tekemiä ohjainlaitetoteutuksia on myös monenlaisia. Esimerkiksi Open Transport (Kuvio 1) on Victor Khazen vuonna 2014 kehittämä musiikintuotantosovellusten ohjauslaite, joka käyttää Arduino mikrokontrolleria näppäimistön emulointiin. Tässä laitteessa on kytkimiä sekä rullia (jotka totetutettu pulssiantureilla) ja lisäksi näyttö. (Khaze 2014.) Orbion (Kuvio 2) puolestaan on CAD-hiiri mikä on tarkoitettu 3D-suunnitteluohjelmistojen kanssa käytettäväksi (Rowntree 2022). Tämä GitHub käyttäjän FaqTotum kehittämä laite on suunniteltu 3D-tulostettavaksi ja käyttää Arduino Pro Micro mikrokontrolleria. Orbion 3D-hiiressä on joystick ja pyöritettävä rulla sen päällä. (FaqTotum 2022.)

Kuvio 1: Victor Khazen Open Transport ohjainlaite. (Victor Khaze 2014)

Kuvio 2: Orbion 3D-hiiri. (FaqTotum 2022.)

2.2.3 Yhteenveto ja johtopäätökset

Tutkittujen laitteiden perusteella, kaupalliset ohjainlaitteet ovat usein kalliita, etenkin laitteet jotka on suunniteltu työkaluiksi ammattilaisille. Lisäksi ne toimivat (vaivatta) usein vain tiettyjen sovellusten kanssa ja niiden ohjelmistoissa on muitakin rajoituksia, varsinkin tehokäyttäjälle. Kuitenkin tietyissä käyttötapauksissa ne voivat olla kelvollisia. Harrastelijoiden ratkaisut ovat usein edullisempia, mutta tarvitsee tietotaitoa tällaisen laitteen valmistamiseen. Hyvänä puolena usein näiden ohjelmakoodi ja suunnitelmat ovat lisensoitu vapailla lisensseillä, joten kuka tahansa voi valmistaa sellaisen itselleen, taikka jatkokehittää tuotetta. Silloin esimerkiksi puutteellinen asetusohjelma ei olisi ongelma, jos mahdolliset puutteet on mahdollista korjata itse.

Vertailemalla laitteiden ominaisuuksia (Taulukko 1), huomataan että kaikissa video/äänieditointiin tarkoitetuissa laitteissa on nappeja, sekä pyöritettäviä nuppeja tai rullia. Viimeksi mainitut ovat hyviä juuri video- ja äänieditoinnissa, joten suunniteltavassa ohjainlaitteessa tulisi olla ainakin yksi tällainen. Puolestaan kaikenlaiset näytöt ovat toissijaisia, vaikkakin kosketusnäytöllä saa lisää toimintoja samalle fyysiselle tilalle, tuo se lisää monimutkaisuutta laitteeseen. Tavallisia nappeja taas on hyvä olla riittävästi, muttei kuitenkaan liikaa, jotta ohjainlaite pysyy pienikokoisena.

Taulukko 1: Tutkittujen laitteiden vertailu (tiedot: Epstein 2020; Razer Tartarus Pro. N.d); Fisher 2020a; Loupedeck CT. N.d; Fisher 2020b ; Monogram Shop. N.d; Khaze 2014; FaqTotum 2022).

Laite	Kuvaus	Ominaisuudet ja Huomiot	Hinta
Razer Tartarus Pro	peliohjain / -näppäimistö	• 20 nappia • 4 suuntainen ristiohjain • 1 hiiren rulla	159,99 €
Loupedeck CT	videon / valokuvanmuokkaus ohjain	• Kosketusnäyttö missä 12 toimintoa yhtä aikaa näkyvillä, useampia valikoiden takana • 20 nappia • 6 pyöritettävää nuppia • pyöritettävä rulla, missä myös kosketusnäyttö lisätoiminnoille	499 €
Monogram Creative Console	Modulaarinen videon / valokuvanmuokkaus ohjain	Moduulit: • Core-moduuli; sisältää kaksi nappia ja tavallisen näytön (ei kosketusnäyttö) • Keys-moduuli, 3 nappia. • Slider-moduuli, 3 liukuvaa säädintä • Dial-moduuli, 3 kierrettävää nuppia • Orbiter-moduuli, kierrettävä rulla ja joystick	250-900 €
Open Transport	Harrastelijan ratkaisu musiikintuotantosovellusten ohjaamiseen.	• Kaksi pyöritettävää nuppia • LCD näyttö • 8 nappia	Arduinon hinta + komponentit + kotelo
Orbion 3D Mouse	Tee-Se-Itse 3D-hiiri	• Joystick • kierrettävä rulla, missä kytkin • pieni OLED-näyttö ohjaimen asetusten säätöä varten	Arduinon hinta + komponentit + 3D tulostus.

3 Ohjainlaitteen toteutuksen tekniikat

3.1 Arduino ekosysteemi

3.1.1 Yritys ja tuotteet

Arduino on yritys, joka suunnittelee ja valmistaa laitteita ja ohjelmistoja joiden avulla voi tehdä laitteita jotka vuorovaikuttavat fyysisen maailman kanssa. Arduino kertoo omilla verkkosivuillaan että Arduinon tehtävä on antaa kenen tahansa parantaa elämäänsä saavutettavan elektroniikan ja digitaalisten teknologioiden kautta. (About Arduino 2021.) Tämä näkyy muun muassa sillä, että Arduinon laitteita käyttäviä on paljon, jolloin yhteisön tuki on suuri. Arduino-ekosysteemiin on esimerkiksi tehty lukuisia erilaisia ohjelmakirjastoja moneen eri tarkoitukseen ja ohjeita ja opastusta löytyy internetistä lähes kaikkeen aiheeseen liittyvään.

Arduinon tuotteet ovat lähtökohtaisesti lisensoitu avoimilla lisensseillä ja he esimerkiksi jakavat eri mikrokontrolliensa suunnittelutiedostot nettisivuillaan. Tämän takia laitteista on saatavilla erilaisia klooneja, jotka usein ovat halvempia. Arduinon lisensointiohjeessa (2023) kerrotaan, että heidän laitesuunnittelemiin perustuvat uudet laitteet tulisi olla yhtä lailla avoimella lisenssillä saatavilla. Lisäksi Arduinon nimeä ja tavaramerkkejä ei saa käyttää tällaisista laitteista. Kuitenkaan laitteissa joissa vain käytetään jotain Arduino laitetta (esimerkiksi mikrokontrolleria) osana itse laitetta, ei tarvitse mainita Arduinoa lainkaan. (Licensing for products based on Arduino 2023.)

3.1.2 Kehitysympäristö

Arduinolla on oma kehitysympäristö, Arduino IDE (integrated development enviroment), jolla voi kirjoittaa ohjelmakoodia, hallita käytettäviä laitteita sekä eri ohjelmakirjastoja (Overview of the Arduino IDE 1 N.d). Tämän ohjelman ensimmäinen versio on toteutettu Java-ohjelmointikielellä. Arduino julkaisi 1.2.2021 IDE:stä version 2.0, joka oli toteutettu Theia ja Electron teknologioilla. (Announcing the Arduino IDE 2.0 (beta) 2021). Electron käyttää usein paljon muistia verrattuna muihin käyttöliittymäratkaisuihin, esimerkiksi Koretić (2020) kirjoittaa julkaisussaan kuinka pieni testiohjelma käyttää 348.1 megatavua muistia, kun taas Qt:n QML kirjastolla toteutettu samankaltainen testiohjelma käyttää vain 92.6 megatavua muistia. Muistin käyttö pystyttiin todentamaan

testaamalla, Kuvio 3 näyttää kuinka IDE:n 2.0 versio käyttää enemmän muistia jopa verrattuna 1.8 Java pohjaiseen versioon.

Kehitysympäristön 2.0 versio lisäsi joitain uusia ominaisuuksia kuten automaattisen täydennyksen ja debugger toiminnon (vain tiettyjen laitteiden kanssa) (Dalmaris 2022). Kuitenkaan nämä uudet ominaisuudet eivät ole niin oleellisia joka käyttötarkoitukseen, että liiallinen muistin käyttö olisi hyväksyttävää. Tämä uusi käyttöliittymä myös reagoi hitaammin ja on muutenkin raskaampi käyttää kuin edellinen. Vanha 1.8 versio on vielä toimiva, ainakin toistaiseksi. Lisäksi on vielä olemassa komentoriviltä ajettava ohjelma nimeltä "arduino-cli", jolla voi tehdä samat asiat kuin Arduinon IDE:llä, kuten ohjelmien kääntämisen ja lataamisen laitteelle (Arduino CLI 2022). Tällöin ohjelmakoodin voisi kirjoittaa haluamallaan tekstieditorilla, ja vain käyttää em. ohjelmaa koodin puskemiseen mikrokontrollerille.

Task	PID	RSS ▲	CPU
java -DAPP_DIR=/usr/share/arduino -sp...	1496405	697,8 MiB	0%
/opt/arduino-ide/arduino-ide --type=re...	1799226	339,6 MiB	0%
/opt/arduino-ide/arduino-ide --type=gp...	1799050	186,9 MiB	0%
/opt/arduino-ide/arduino-ide /opt/ardui...	1799029	168,8 MiB	0%
/opt/arduino-ide/arduino-ide	1798925	159,2 MiB	0%
/opt/arduino-ide/resources/app/node_...	1799216	152,6 MiB	0%
/opt/arduino-ide/arduino-ide /opt/ardui...	1799417	86,1 MiB	0%
/opt/arduino-ide/arduino-ide /opt/ardui...	1799408	78,2 MiB	0%
/opt/arduino-ide/arduino-ide /opt/ardui...	1799221	68,9 MiB	0%
/opt/arduino-ide/arduino-ide /opt/ardui...	1799716	68,1 MiB	0%
/opt/arduino-ide/arduino-ide --type=util...	1799082	63,7 MiB	0%
/opt/arduino-ide/arduino-ide --type=zy...	1799001	45,2 MiB	0%
/opt/arduino-ide/arduino-ide --type=zy...	1798999	45,1 MiB	0%
/home/asko/.arduino15/packages/builti...	1799261	10,9 MiB	0%
/opt/arduino-ide/arduino-ide --type=zy...	1799011	10,0 MiB	0%
/home/asko/.arduino15/packages/builti...	1799262	3,9 MiB	0%
bash /usr/share/arduino/arduino	1496399	3,7 MiB	0%

Kuvio 3: Arduino IDE versio 1.8 ja 2.0 muistin käyttö. (Valittu ja alin prosessi kuuluvat ohjelman versiolle 1.8, muut ovat version 2.0 prosesseja)

3.1.3 Laitteet

Arduinolla on eri mikroprosessoreihin perustuvia laitteita. Omilla verkkosivuillaan Arduino jakaa laitteet kolmeen ryhmään, Nano, MKR ja Classic. Nano tuoteperheen laitteet ovat nimensä mukaisesti pienikokoisia ja niissä on usein sisäänrakennuttuna verkkokortteja tai antureita (Arduino Hardware 2022). Niissä käytetään vaihtelevasti mikrokontrolleita ja prosessoreja, esimerkiksi Arduino Nano käyttää ATMega328 mikrokontrolleria, kun taas Nano 33 tuotteet käyttävät ARM-prosessoreja (Arduino Nano – Arduino Official Store N.d; Arduino Nano 33 BLE – Arduino Official Store N.d). MKR tuotteisiin sisältyy niin itse mikrokontrollerikortteja kuin niiden päälle asennettavia laajennuskortteja. Kaikki MKR laitteet käyttävät Cortex-M0 prosessoria. Classic-kortteihin sisältyy

kaikki loput Arduinon mikrokontrollerikortit, kuten Arduino UNO, Mega, Leonardo, Micro ja muut. Näiden laitteiden prosessorit vaihtelevat enemmän. (Arduino Hardware 2022.)

USB:n yli toimivan ohjainlaitteen kehitys vaatii oikeanlaisen Arduino laitteen. Esimerkiksi Arduino Leonardon kauppasivulla kerrotaan laitteen kykenevän tähän, koska siinä käytettävässä ATMega32u4 mikrokontrollerissa on USB-väylä sisäänrakennuttuna, jolloin ei tarvita toista ohjainpiiriä hallitsemaan USB-kommunikaatiota. Tällöin Leonardo voi esimerkiksi emuloida tietokoneen näppäimistöä. (Arduino Leonardo – Arduino Official Store N.d.) Samaa mikrokontrolleria käyttää esimerkiksi Arduino Micro, joka on pienikokoisempi kuin Leonardo. (Arduino Micro – Arduino Official Store N.d). Arduino Micro siten soveltuu paremmin laitteen sisään asennettavaksi.

3.2 Arduino-kirjastot

3.2.1 HID-kirjastot

Arduinolle on saatavilla monia ohjelmakirjastoja, jotka esimerkiksi helpottavat tiettyjen antureiden kanssa toimimista tai helpottavat muuten ohjelmakoodin kirjoittamista. Arduino IDE:n mukana tulee useita kirjastoja, mutta niitä voi ladata lisää internetistä tai tehdä itse omia kirjastoja (Libraries 2022). Tietokonetta ohjaavan laitteen kannalta olennaisia kirjastoja ovat sellaiset kirjastot jotka voivat emuloida yleisiä tietokoneen ohjainlaitteita, kuten näppäimistöjä. Arduinon vakiokirjastot tukevat vakiona HID-laitteena toimimista ja tätä tukea käyttävät mm. kirjasto nimeltä "Keyboard" jonka avulla tietyt yhteensopivat mikrokontrollerit voivat lähettää näppäinpainalluksia tietokoneelle aivan kuin mikrokontrolleri olisi oikea näppäimistö, sekä kirjasto "Mouse" joka tekee vastaavan tietokonehiiren liikkeille (Keyboard 2022; Mouse 2022). Myös kolmannet osapuolet ovat luoneet tällaisia kirjastoja. Esimerkiksi "Arduino Joystick Library" kirjaston avulla mikrokontrolleri voi toimia peliohjaimen tavoin (Heironimus 2022).

Kuten luvussa 3.1.3 on mainittu, Arduinon omia USB kirjastoja käyttäessä **l**aitte**e**ssa täytyy olla oikeanlainen mikroprosessori/USB-moduuli. Kuitenkin myös joidenkin muiden laitteiden (mitkä eivät ole soveltuvia käyttäessä Arduinon omia kirjastoja) käyttö onnistuu NicoHoodin tekemällä HID-kirjastolla. Hänen projektinsa mahdollistaa lähes kaikkien Arduino mikrokontrollerien käytön USB-HID laitteena. Lisäksi hänen HID-kirjastossansa on enemmän toiminnallisuutta kuin Arduinon vakio

HID-kirjastoissa, sillä se tukee mm. näppäimistöistä löytyvien medianäppäinten emulointia. (Nico-Hood 2021.)

3.2.2 MIDI-kirjastot

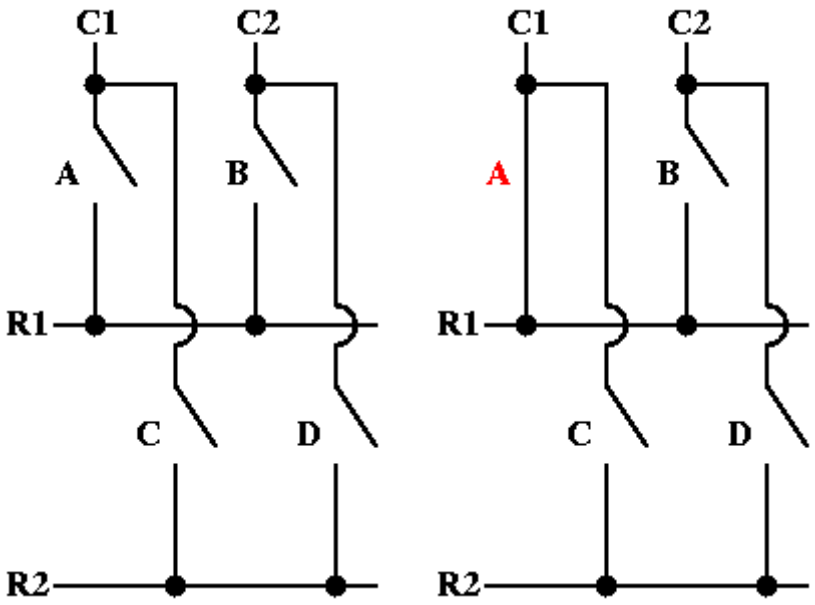
MIDI (Musical Instrument Digital Interface) on vuonna 1983 julkaistu standardi jolla eri musiikki-laitteet voivat kommunikoida keskenään MIDI komentojen avulla. Esimerkiksi kosketinsoitin voi lähettää soitetut nuotit MIDI-standardin avulla toiselle soitin laitteelle, joka soittaa niiden perus-teella omia ääniään. Myös tietokoneet joissa on sopivat lisälaitteet voivat käyttää MIDI-standardia, jolloin MIDI-laitteita on helppo automatisoida. Nykyään yleisesti käytetään MIDI-ohjaimia, jotka kytketään USB:llä tietokoneeseen. (Gibson N.d.) Tällaisella ohjaimella voi ohjata yhteensopivia sovelluksia, joissa on MIDI tuki. Yleensä tällainen sovellus on musiikintuotantosovellus, joten mikä-li tahdotaan laajempaa käyttöalaa, on ohjainlaite viisaampaa toteuttaa muulla tapaa.

Arduinon virallisista kirjastoista löytyy kirjasto MIDIUSB, jonka avulla Arduino mikrokontrolleri voi toimia MIDI-ohjaimena USB-portin yli (MIDIUSB 2022). Toinen MIDI-ohjainkirjasto on Control Sur-face. Se on vielä laajempi kirjasto ja tehty erityisesti MIDI-ohjainten tekoon. Se tukee eri midi-so-vittimia, erilaisia kontrolleja (potentiometrit, liukusäätimet/faderit, pulssianturit, napit yms.), merkkivaloja (mm. erilaiset LED-valot, mittarit, näytöt) sekä eri asetuspankkeja (eli samat kontrollit voivat tehdä eri asioita eri pankeissa). (Control Surface 2023.)

3.3 Elektroniikkakytkennät ja komponentit

3.3.1 Kytkimet ja näppäimistömatriisi

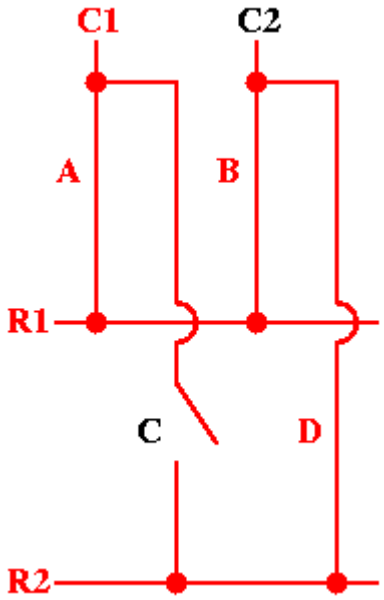
Laitteeseen tulee useita kytkimiä sekä pyöritettäviä rullia. Kytkimet ovat tavallisia palautuvia painonappeja, ja rullat toteutetaan pulssiantureilla. Ohjainten suuren määrän ja ATMega32u4 mikrokontrollerin määrältään rajoittuneiden sisääntulojen vuoksi on tarpeellista käyttää näppäimistömatriisia jotta saadaan useampi nappi vähemmillä sisääntuloilla. Dribin (2000) kertoo oppaassaan, että näppäimistömatriisi koostuu johtimista, jotka on järjestetty riveihin ja sarakkeisiin. Kukin näppäimistön näppäin (kytkin) yhdistää painettaessa jonkin rivin johonkin toiseen sarakkeeseen, jolloin piiri sulkeutuu (Dribin 2000, luku 2). Esimerkiksi kun oheisessa kuviossa (Kuvio 4) kuvatussa näppäimistöpiirissä painetaan näppäintä A, yhdistyy sarake 1 ja rivi 1.



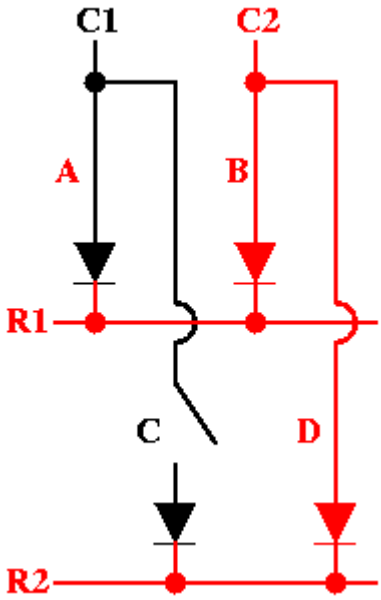
Kuvio 4: Näppäimistömatriisin toimintaperiaate. (Dribin 2000)

Matriiseihin liittyy myös haamupainallusilmiö (eng. Ghosting), joka on hyvä välttää. Esimerkiksi mikäli em. matriisista painetaan yhtaikaa näppäimiä A, B ja D, yhdistyy sarake yksi sekä riviin yksi että kaksi. Ensimmäinen on odotettua koska se vastaa näppäintä A, mutta toinen vastaisi näppäintä C, mitä ei ole painettu. (Dribin 2000, luku 6.) Tämä johtuu siitä että virtapiiri yhdistyy muitten painettujen näppäinten kautta. (Kuvio 5) Dribin (2000, luku 7) kertoo myös ns. maskausongelmasta (eng. Masking): esim. mikäli em. tilanteesta (missä siis painettuna näppäimet A, B ja C) painetaan C näppäintä, ei näppäimistön ohjain tajua että nappia on painettu, koska se luuli jo ennestään että C

nappi oli painettuna. Dribin (2000, luku 8) ehdottaa näiden ongelmien poistamiseksi diodien sijoittamista virtapiiriin, jokaisen näppäimen kanssa sarjaan, jolloin em. tilanne ratkeaa koska virta ei voi kulkea ns. väärään suuntaan virtapiirissä. (Kuvio 6).



Kuvio 5: Haamupainallukset (Dribin 2000)



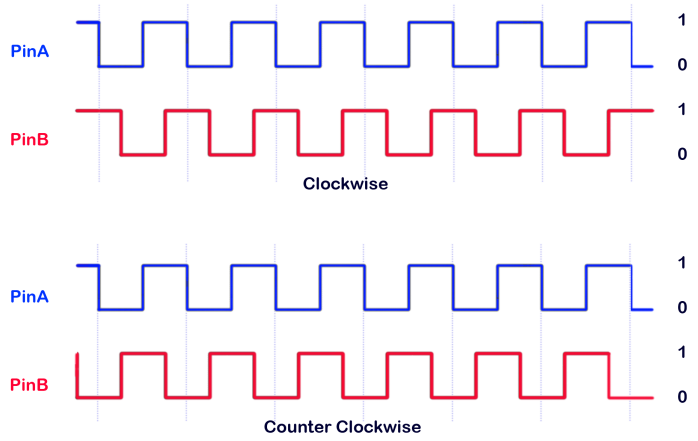
Kuvio 6: Haamupainallusten esto diodein (Dribin 2000)

3.3.2 Kulma-anturit

Kulma-anturit (käytetään myös nimityksiä pulssianturi, pulssienkooderi, eng. rotary encoder) ovat antureita jotka muuntavat akselin kulman ja/tai liikkeen analogiseksi tai digitaaliseksi signaaliksi. Käytännössä kulma-anturin avulla saadaan selville akselista sen nykyinen asento sekä kääntymisen suunta ja nopeus. (Murray 2019.) Kulma-antureita on kahta eri tyyppiä, inkrementti- tai pulssiantureita (Incremental rotary encoder) sekä absoluuttiantureita. Absoluuttianturit tietävät oman absoluuttisen asentonsa aina, kun taas pulssianturit pystyvät ainoastaan lukemaan mihin suuntaan ja miten paljon kulma muuttuu, mutta anturin aloitusasento ei ole tiedossa. Pulssianturit ovat yleisempiä ja edullisempia. Lisäksi on kulma-antureita joissa ei ole pykäliä, jolloin ne soveltuvat paremmin esimerkiksi nopeasti kierrettäväksi rullaksi. (Bell & Hansell 2018.) Tässä työssä ei käsitellä absoluuttiantureita, joten lopussa tekstissä viitataan vain inkrementtiantureihin.

Pulssianturin rakenne on seuraava; pyöritettävässä varressa on kiinni levy, jossa on metallikontakteja tietyn välimatkoin. Levyyn ottaa kiinni kaksi vierekkäin olevaa kontaktia, ”data” ja ”clock”, jot-

ka anturin ollessa paikallaan sijaitsevat levyssä olevien kontaktien välissä. Kun anturin akselia käännetään, osuu jompikumpi näistä kontakteista levyssä olevaan kontaktiin ensin. Jos datakontakti yhdistyy ensin levyn kontakteihin, tämä yleensä tarkoittaa että akseli on liikkunut myötäpäivään (ja päin vastoin). (Murray 2019.) Ulkoisesti useimmissa pulssiantureissa on 3 pinniä (maa, sekä yksi pinni kullekin kääntösuunnalle. Nämä kaksi pinniä vaihtelevat ”HIGH” ja ”LOW” tasojen väillä, riippuen anturin pyörityssuunnasta. Tällöin pulssianturilla on 4 eri tilaa, bitteinä ilmaistuina siis 00, 01, 10 ja 11. (Meyer 2012.) Pyörimissuunnan voi siis määrittää näiden perusteella, esim. jos tila siirtyy 00->01->11->10->00 (eli kierrettäessä ensimmäiseksi yksi pinni muuttuu ”HIGH”-tilaan, sitten toinenkin pinni ja sen jälkeen ensimmäinen pinni menee ”LOW”-tilaan), on anturi kääntynyt tiettyyn suuntaan. Tämän pystyy esittämään kanttiaaltona (Kuvio 7).



Kuvio 7: Pulssianturin signaalin kanttiaalto (Quadrature Rotary Encoder N.d.)

3.4 Muut käytettävät työkalut ja ohjelmistot

3.4.1 Suunnittelu- ja valmistusmenetelmät

Arduinon kehitysympäristön lisäksi tarvitaan ohjelmistoja laitteen muun fyysisen rakenteen suunnitteluun, kuten laitteen piirilevyjen ja kotelon piirtoon. Laitteen piirilevyn suunnitteluun on monia vaihtoehtoja, useimmat ohjelmistot ovat kuitenkin kaupallisia mutta joistain on myös saatavilla ilmainen/edullisempi versio harrastelijoille. Myös avoimen lähdekoodin (ja lisenssin) sovelluksia on, kuten KiCad joka onkin todella pätevä ohjelmisto piirilevyjen suunnitteluun. Itse fyysisen piirilevyn voisi valmistaa itse (esimerkiksi siirtokalvolla tai UV-valotusmenetelmällä), mutta nykyään on palveluita joista voi tilata pienenkin erän piirilevyjä kohtuuhintaan vaikka juuri prototyyppitarkoituksiin, mikä on usein järkevämpää harrastelijallekin kuin syövyttää levynsä itse.

Laitteen kotelon voisi suunnitella millä tahansa CAD-ohjelmistolla (tietokoneavusteinen suunnittelu, eng. computer aided design), myös avoimen lähdekoodin CAD-ohjelmia on olemassa kuten FreeCAD ja OpenSCAD. Näistä jälkimmäinen on ohjelmointiin perustuva, eli 3D-malli luodaan kirjoittamalla ohjelmakoodi joka kuvaa kappaleen luonnin ja OpenSCAD renderoi mallin koodin perusteella (OpenSCAD - About N.d). FreeCAD -ohjelmaa puolestaan on kritisoitu vaikeakäyttöiseksi, esimerkiksi Junk ja Kuen (2016) kertovat tutkimuksessaan, että FreeCAD kärsii vertailussa juuri käytettävyyden takia, vaikka siinä onkin paljon ominaisuuksia. Myös Blender 3D-mallinnusohjelmaa voi käyttää 3D-tulostettavien kappaleiden suunnitteluun. Se voi olla järkevää etenkin sellaiselle harrastelijalle joka on jo oppinut Blenderin käytön muussa kontekstissa, kuten vaikka pelien 3D-mallien teossa.

3.4.2 Piirilevyn suunnittelu

Ennen itse piirilevyn suunnittelua suunnitellaan kytkentäkaavio. Kytkentäkaavio (eng. schematic, circuit diagram) on ”moniosaisen sähkölaitteen rakennetta standardoidulla tavalla kuvaava kaavio” (Kotimaisten kielten keskus ja Kielikone Oy 2022b). Käytännössä se tarkoittaa, että kytkentäkaaviossa kuvataan, mitä komponentteja laitteessa käytetään, sekä miten ne on kytketty toisiinsa. Kytkentäkaavio näyttää kytkennät mahdollisimman selkeästi, kaikki kytkennät on piirretty mahdollisimman suorilla viivoilla. Komponenttien todellinen sijoittelu ei yleensä vastaakaan kytkentäkaavion sijoittelua (Hewes N.d). Kytkentäkaavio nimensä mukaisesti kertookin vain piirin kytkennät. Kytkentäkaavioissa käytetään symboleita, jotka on määritelty IEC 60617 -standardissa, missä jokai-

selle symbolille on määritelty referenssinumero, nimi, graafinen esitys sekä jotain lisätietoa mikäli tarpeellista (IEC 60617 - Graphical Symbols for Diagrams N.d). Kaikissa kytkentäkaaviossa käytetään siis samoja standardisymboleita kuvaamaan eri komponentteja. Kytkentäkaavion voi piirtää käsin tai käyttää jotain ohjelmistoa, useimmissa ohjelmissa millä suunnitellaan piirilevyjä, on myös työkalut kytkentäkaavion piirtoon.

Piirilevy (eng. Printed Circuit Board, PCB) on ”levy johon on kiinnitetty sähköisiä komponentteja ja jonka pintaan (ja sisäpuolelle) on muodostettu näiden komponenttien välisiä kytkentöjä” (Kotimaisten kielten keskus ja Kielikone Oy 2022a). Yleensä piirilevyt koostuvat sähköä johtamattomasta aineesta, jonka pinnalle on tehty kerros sähköä johtavasta aineesta, missä on johtimia ja komponenttien kiinnityspaikkoja. Piirilevyissä on vähintään yksi sähköä johtava kerros, mutta useamman kerroksen käyttö tekee piirilevystä pienikokoisemman ja helpottaa sen suunnittelua. Esimerkiksi nelikerroksissa piirilevyssä on mahdollista jättää yksi johdinkerros piirin maaksi ja toinen käyttöjännitteelle, jolloin loput (uloimmat) kerrokset jäävät vapaaksi komponenttien asettelulle. (Keim 2020.) Monikerroksisen piirilevyn valmistus itse (esimerkiksi luvussa 3.4.1 mainituilla tekniikoilla) on kuitenkin lähes mahdotonta.

3.4.3 KiCad

KiCad on avoimen lähdekoodin elektroniikkasuunnitteluohjelmisto, millä voi tehdä kytkentäkaavioita sekä suunnitella piirilevyjä. Ohjelmisto toimii Windows, Linux sekä macOS -käyttöjärjestelmillä ja itse ohjelma on lisensoitu GNU GPLv3 -lisenssillä. (About KiCad – KiCad EDA N.d.) KiCadin symbolikirjastot (jotka sisältävät eri komponenttien kytkentäkaaviosymboleita ja jalanjälkikuvia, eli minkälaisen alan komponentti vie piirilevyltä) puolestaan ovat Creative Commons CC-BY-SA 4.0 lisenssin alaisuudessa, jotta kirjastojen sisältöä voi käyttää rajoitteitta niin kaupallisissa, suljetuissa kuin ei-kaupallisissa projekteissa. Käytännössä siis jos käyttää KiCadin kirjastojen sisältöä projektissaan ei vaadi saman lisenssin käyttöä tai mitään mainintaa kirjastojen lisenssistä, mutta kirjastoja itsessään jakaessa (muutettuna tai sellaisenaan), täytyy tämä tehdä saman lisenssin alla. (Libraries License – KiCad EDA.) KiCadin mukana tulevien symbolikirjastojen lisäksi symbolikirjastoja on mahdollista asentaa muista lähteistä ja jopa tehdä itse, sillä vakiokirjastot eivät voi millään kattaa kaikkea.

Tyypillinen työtapa KiCadissa koostuu kahdesta osasta, kytkentäkaavion piirrosta ja piirilevyn suunnittelusta. KiCadissa kummallekin toimenpiteelle on omat näkymänsä, *Schematic Editor* sekä *PCB Editor*. Lisäksi omat näkymänsä on komponenttien symboleiden ja fyysisen koon muokkaamiseen, *Symbol Editor* ja *Footprint Editor*. Näillä voi joko muokata esimerkiksi em. mukana tulevien kirjastojen komponentteja tai luoda omia määrittelyjä. KiCadissa käytetään projektimuotoista työtapaa, eli kaikki yhden projektin tiedostot (kuten vaikka kytkentäkaavio ja piirilevyn suunnitelma) ovat saman projektin alla. Projektiin kuuluvia tiedostoja kannattaa ensisijaisesti avata vain projektin alaisuudessa, sillä muutoin jotkin projektin tiedot saattavat puuttua. (Getting Started in KiCad – KiCad Documentation, Basic Concepts and Workflow.) Lisäksi itse piirilevyeditori ja kytkentäkaavioeditori toimivat yhteen, esimerkiksi kytkentäkaavio tuodaan piirilevyeditoriin, ja mahdolliset muutokset virtapiiriin tehdään aina kytkentäkaavioon, mikä tuodaan uudelleen piirilevyeditoriin.

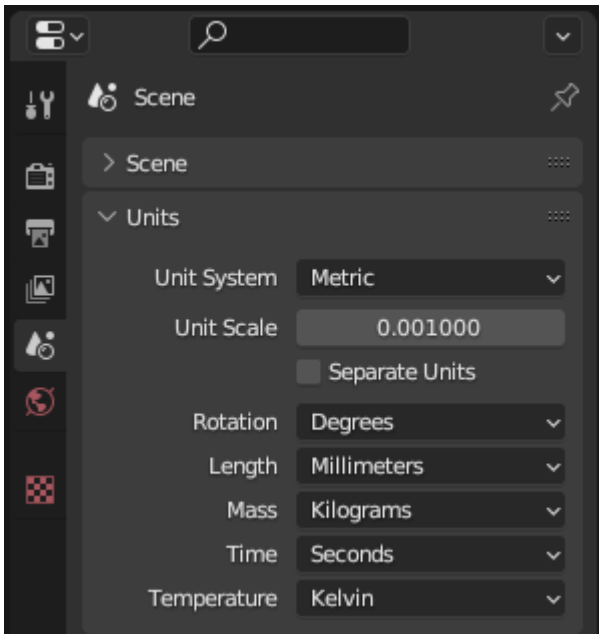
3.4.4 3D-Tulostus

3D-tulostuksessa on kyse kappaleiden valmistamisesta materiaalia lisäävillä menetelmillä (toisin kuin vaikka jyrsin, joka poistaa materiaalia saavuttaakseen kappaleen halutun muodon). 3D-tulostus on jo niin kehittynyt ala, että kuluttajaluokassakin on saatavilla todella tarkkaan tulostusjälkeen soveltuvia 3D-tulostimia. Jo vuonna 2018 Tukesin julkaisemassa *Kysymyksiä ja vastauksia 3D-tulostamisesta* (2018) julkaisussa kerrotaankin, että 3D-tulostimien hinnat ovat laskeneet merkittävästi, mikä on madaltanut 3D-tulostamisen ammatti- ja kotikäytön välistä rajaa. Samassa Tukesin julkaisussa myös todetaan, että useimmat harrastelijoiden tulostimet perustuvat useimmiten muovin ekstruusioon lämmitetyn suuttimen läpi (FDM, fused deposition modeling). Nykyään myös eri hartsitulostustekniikat (Resin printing) kuten SLA (stereolitografia), LCD ja DLP ovat hintansa puolesta kuluttajien ja harrastelijoiden saatavilla. Korkeatarkkuuksisen hartsitulostimen saa jopa alle kahdensadan yhdysvaltain dollarin (Frey, Gherke 2023). Oli tulostimen teknologia mikä tahansa, ovat ne hyödyllisiä prototyyppien nopeatahtisessa kehittämisessä.

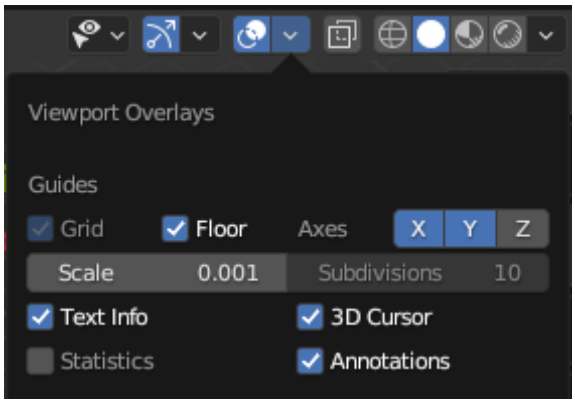
3.4.5 Blender 3D-tulostuksessa

Blender on Blender Foundation -säätiön kehittämä avoimen lähdekoodin 3D mallinnusohjelmisto, jossa on työkaluja niin itse mallinnukseen kuin animaatioon, simulointiin, renderöintiin, liikkeenkaappaukseen ja jopa videoeditointiin sekä pelien tekemiseen (About – blender.org N.d). Blender onkin suunniteltu enemmänkin visuaalisen puolen 3D-mallinnukseen, toisin kuin CAD-ohjelmistot, jotka on tarkoitettu fyysisten asioiden suunnittelemiseen. Kuitenkin Blenderillä on mahdollista tehdä 3D-malleja myös 3D-tulostusta (tai muuta fyysistä valmistamista) varten, kunhan vain säätää asetukset sopiviksi.

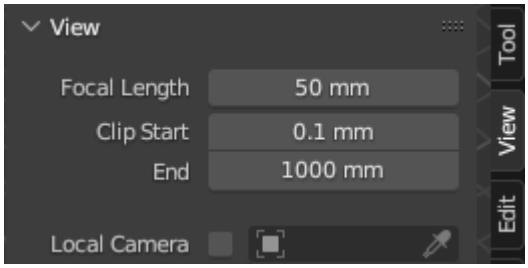
Tärkeintä on käyttää yksikköinä oikeita yksikköjä, 3D-tulostuksessa yleisesti millimetrejä. Tämän voi asettaa *Scene* -asetusvälilehden alta kohdasta *Units*. *Unit System* -valikon arvoksi asetetaan *Metric* ja *Unit Scale* -arvoksi 0.001, joka tarkoittaa millimetrejä. (Kuvio 8) Skaalan muuttamisen takia myös Blenderin ruudukon kokoa on säädettävä. Tämä löytyy *Viewport Overlays* -valikosta näkymän (viewport) yläreunasta. (Kuvio 9) Skaala 0.001 tarkoittaa, että yksi ruutu on 1mm. Myös kameran takarajaa kannattaa säätää, sillä muuten objektit eivät välttämättä näy näkymässä kokonaan. Tämä asetus löytyy *Properties* -valikosta näkymän oikeasta reunasta (pikanäppäin **N**) kohdasta *View → (Clip) End*. (Kuvio 10) Lisäksi on hyödyllistä asettaa eri kappaleiden väri satunnaiseksi, varsinkin jos samassa tiedostossa on useita eri osia (Kuvio 11).



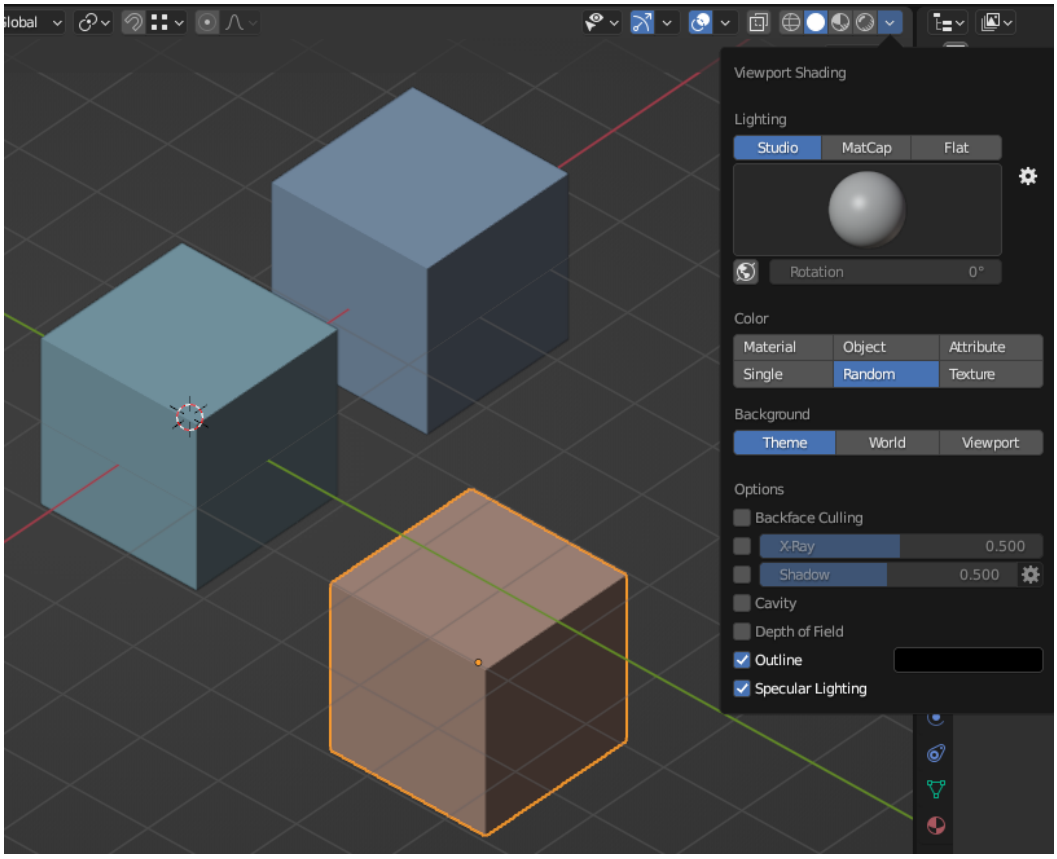
Kuvio 8: Blenderin mittayksikköasetukset



Kuvio 9: Blenderin ruudukkoasetukset Viewport Overlays -valikossa

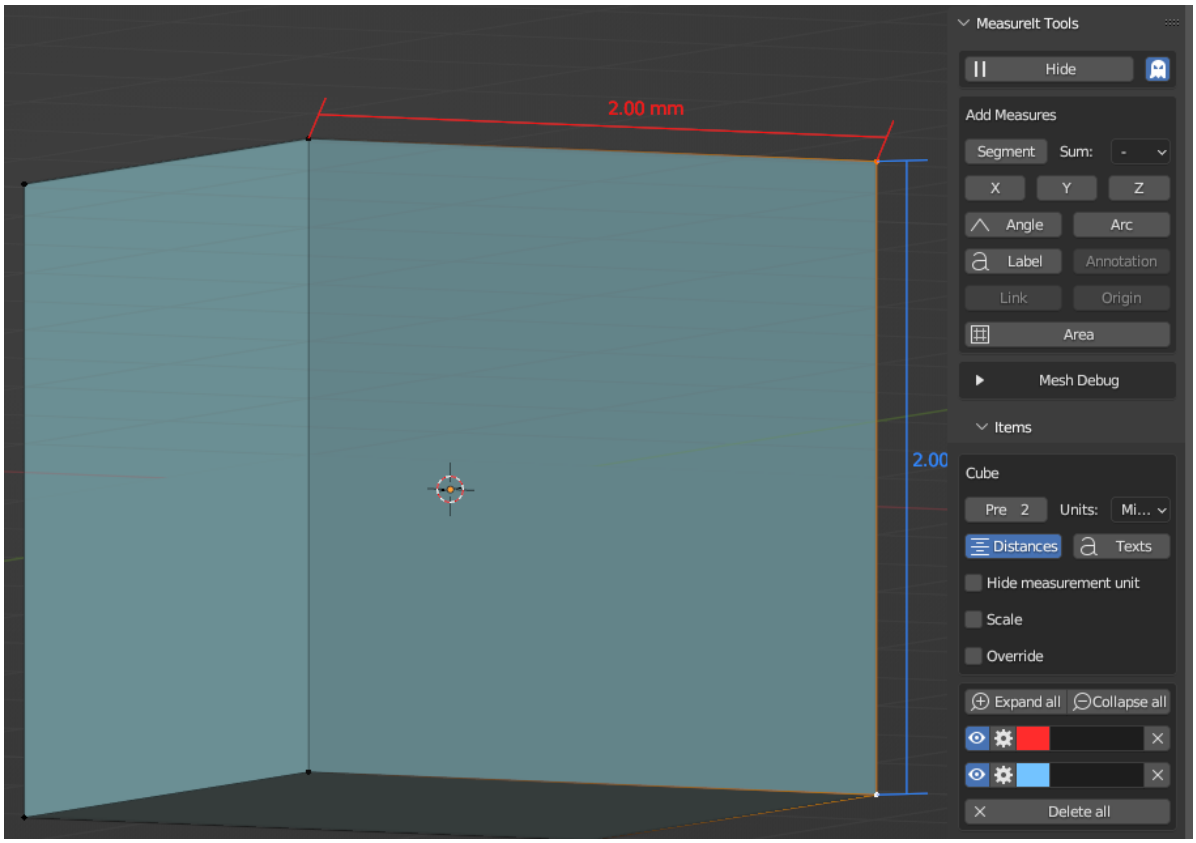


Kuvio 10: 3D-näkymän kameran rajat Blenderissä



Kuvio 11: Kappaleiden väriasetukset Blenderissä.

Blenderiin on saatavilla myös lisäosia jotka auttavat nimenomaan 3D-tulostuksessa ja suunnittelussa. Esimerkiksi *3D Print Toolbox* -lisäosa näyttää kappaleista tietoa josta on hyötyä 3D-tulostusta varten. Sillä näkee muun muassa, onko 3D-mallissa reikiä, yksipuolisia sivuja tai liian ohuita geometrioita. (3D Print Toolbox – Blender Manual 2023.) Toinen tärkeä lisäosa on *Measure It*. Se on lisäosa (kappaleiden) mittojen näyttämiseen 3D-näkymässä, mikä on tärkeää mm. 3D-tulostuksessa (Measure It – Blender Manual 2023). Esimerkiksisi voidaan mitata kahden verteksin (pisteen) välinen mitta, valitsemalla kaksi verteksiä ja klikkaamalla nappia *Segment* lisäosan paneelissa. . (Kuvio 12) Molemmat lisäosat lisäävät valikkonsa *Properties* -valikkoon (pikanäppäin **N**).



Kuvio 12: Measure It -lisäosa Blenderissä

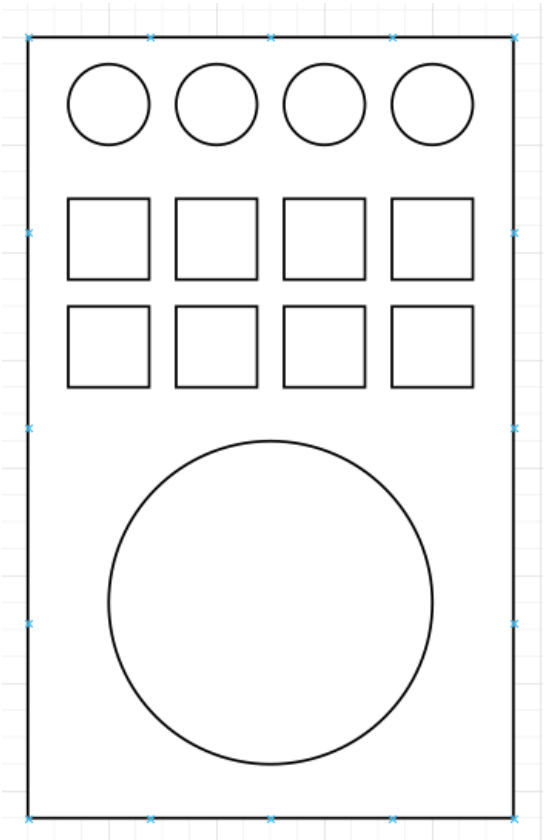
Suurimman osan ajasta kannattaa käyttää isometristä kameraa perspektiivikameran sijasta (pikanäppäin numeronäppäimistön 5). Joissain tapauksissa perspektiivikameralla pääsee kuitenkin joihinkin kulmiin käsiksi paremmin. Myös lukuisat muokkaajat (modifiers) ovat hyödyllisiä 3D-tulostus mallien mallinuksessa, kuten *Array ja Subdivision Surface. Array* -muokkaajalla voi tehdä kopioita olemasta olevasta geometriasta, kun taas *Subdivision Surface* jakaa automaattisesti geometrian sivut useampiin pienempiin osiin, silottaen sitä. Tällöin itse geometriasta ei tarvitse tehdä niin monimutkaista, ja sitä on helpompi muokata myöhemmin. Muokkaajat toimivat niin että ne eivät tee pysyviä muutoksia 3D-mallin geometriaan (ellei tätä erikseen haluta), vaan niiden muutokset tavallaan asetetaan pohjalla olevan mallin päälle. Nämä muutokset kuitenkin tulevat mukaan jos 3D-malli viedään ulkoiseen tiedostoon (kuten *stl* jota käytetään 3D-tulostuksessa).

4 Toteutus

4.1 Ohjainlaitteen suunnittelu

4.1.1 Alustava suunnitelma

Alustavasti ohjainlaitteeseen suunniteltiin yksi iso pyöritettävä rulla keskelle, jota mahdollisesti pystyisi lisäksi liikuttelemaan vaakatasossa nelisuuntaisen ristiohjaimen tavoin. Laitteeseen myös tulisi myös jonkin kokoinen näppäimistön kaltainen painikerykelmä. Painikkeiden yläpuolelle suunniteltiin 4 pyöritettävän nupin rivi, jossa nupit toteuttaisin pulssiantureilla. Mikäli Arduinossa olisi vielä sen jälkeen ylimääräisiä pinnejä, voitaisiin laitteeseen lisätä vielä muutama pulssianturi. Nämä eri ohjaintavat aseteltaisiin laitteeseen alustavasti kuvion 13 mukaan, missä pyöreät ovat rullia ja neliskanttiset näppäimiä.



Kuvio 13: Ohjainlaitteen alustava asetelmasuunnitelma

4.1.2 Arduinon valinta ja sen rajoitteet

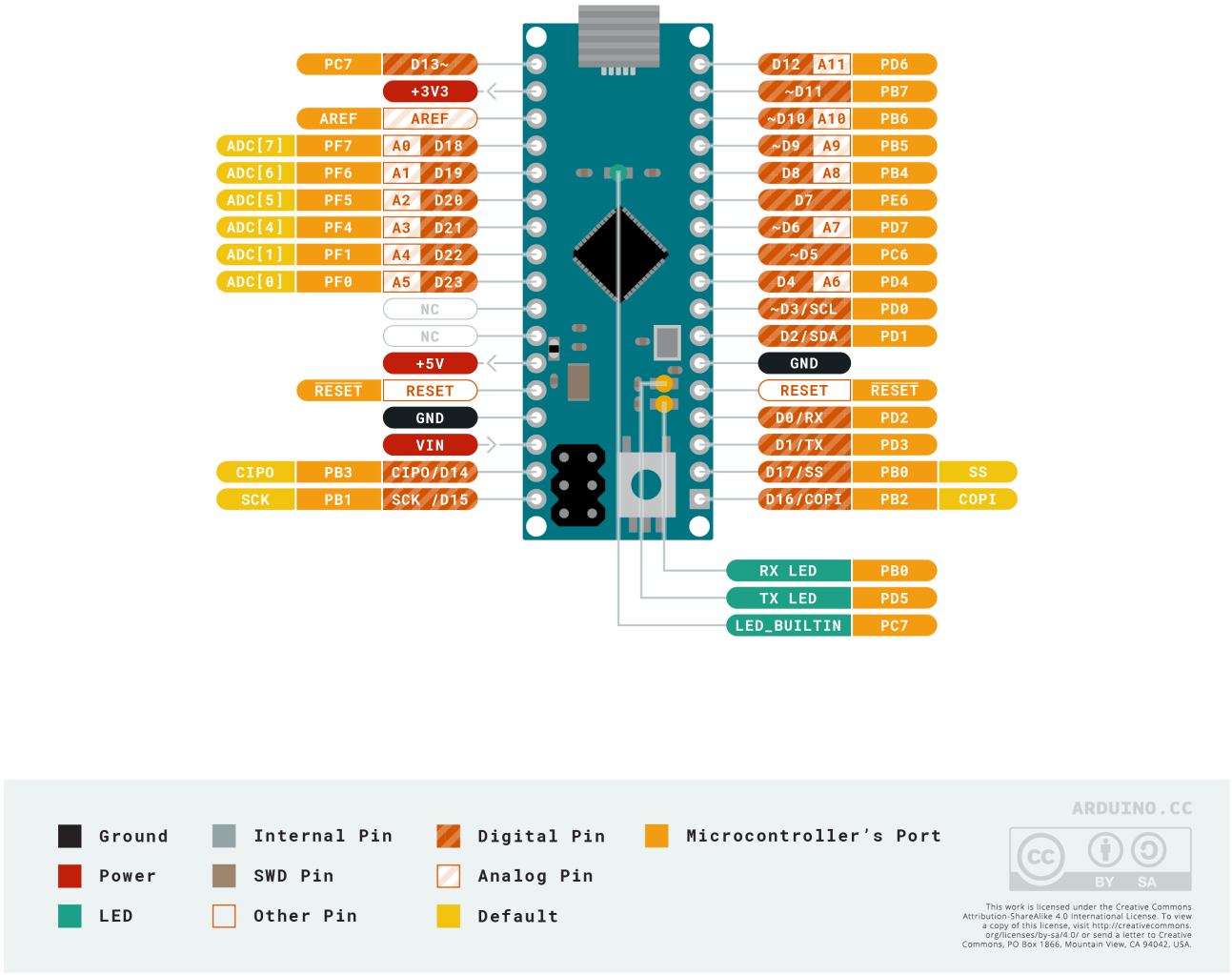
Käytettäväksi Arduino laitteeksi valikoitui ATMega32u4 mikrokontrolleriin perustuva Arduino Micro yhteensopiva klooni, merkkiä RobotDyn. Kuten luvussa 3.1 käsiteltiin, Arduinon lisensointi mahdollistaa laitteiden valmistuksen hyvin vapaasti, joten saatavilla on huomattavasti edullisempia laitteita. Kyseinen laite on täysin yhteensopiva alkuperäisen Arduino Micron kanssa, niin fyysiseltä kooltaan, kuin mikrokontrollerin ja pinnien osalta (pinnien lukumäärä, sijainti ja toiminnallisuus vastaavat alkuperäisen Arduino Micron pinnejä). Arduino Microssa on useita digitaalisia pinnejä, joista kutakin voi käyttää joko ulos- tai sisääntulona. (Kuvio 14) Lisäksi joissain pinneissä on erikoistoimintoja, mainittavia ovat esimerkiksi keskeytyspinnit (interrupts), PWM toimintoa tukevat pinnit sekä analogiset sisääntulot pinneissä A0-A5 ja A6-A11, jotka ovat päällekkäin joidenkin digitaalisten pinnien kanssa (Arduino Micro – Arduino Official Store. N.d). Analogisia pinnejä ei kuitenkaan tarvita tässä projektissa, joten käytössä on kaikki digitaaliset pinnit. Toisin kuin Arduino Micron kauppasivulla kerrotaan, Microssa digitalisia pinnejä on käytettävissä yhteensä 24 kappaletta (eikä vain 20). Tosin kahteen näistä pinneistä on kytketty Arduinoon sisäänrakennettu vastus ja LED, mikä voi hankaloittaa pinnin käyttöä sisääntulona (ks. luku 4.1.4).

Yksinkertaisimmillaan kukin pulssianturi vaatii Arduinosta kaksi pinniä sekä yhteyden virtapiirin maahan. Monimutkaisemmat kytkennät ovat myös mahdollisia, mutta hyödynnettäessä valmiita kirjastoja on helpointa käyttää em. kytkentätapaa. Pulssiantureiden lukemiseen on mahdollisesti hyödyllistä käyttää keskeytyspinnejä, jotka nimensä mukaisesti keskeyttävät koodin ajamisen. Näitä pinnejä Arduino Microssa on kuitenkin vain 5 kappaletta, 0(RX), 1(TX), 2, 3 ja 7 (Arduino Micro – Arduino Official Store. N.d). Pulssiantureita voi koettaa lukea myös ilman pinnikeskeytyksiä, myös suoraan pääohjelman ”`loop`” funktiossa, mutta se voi olla hankalaa ajoituksen kannalta. Eräs parempi vaihtoehto on käyttää jotain ohjelmistokirjastoa joka hoitaa antureiden lukemisen (ks. luku 4.2.1).

Laitteeseen tulevia kytkimiä, kuten pulssiantureissa olevia kytkimiä tahtoisi lukea yksinkertaisimmalla tavalla, se vaatisi yhden pinnin kullekin kytkimelle. Kytkimet kuitenkin tahdottiin yhdistää näppäimistömatriisin avulla (ks. luku 3.3.1), jotta pinnejä tarvitsee yhteensä vähemmän. Laitteeseen suunniteltiin tulevan yhteensä ainakin 16 kytkintä matriisiin kytkettynä, jolloin näppäimistömatriisi säästää tämän näppäimistön tapauksessa 8 pinniä sen sijaan, että käytettäisiin jokaiselle napille omaa pinniä. Myös näppäimistömatriisien lukemiseen on valmiita kirjastoja.

ARDUINO

ARDUINO
MICRO



Kuvio 14: Arduino Micro pinnikaavio (Arduino Micro Pinout Diagram N.d)

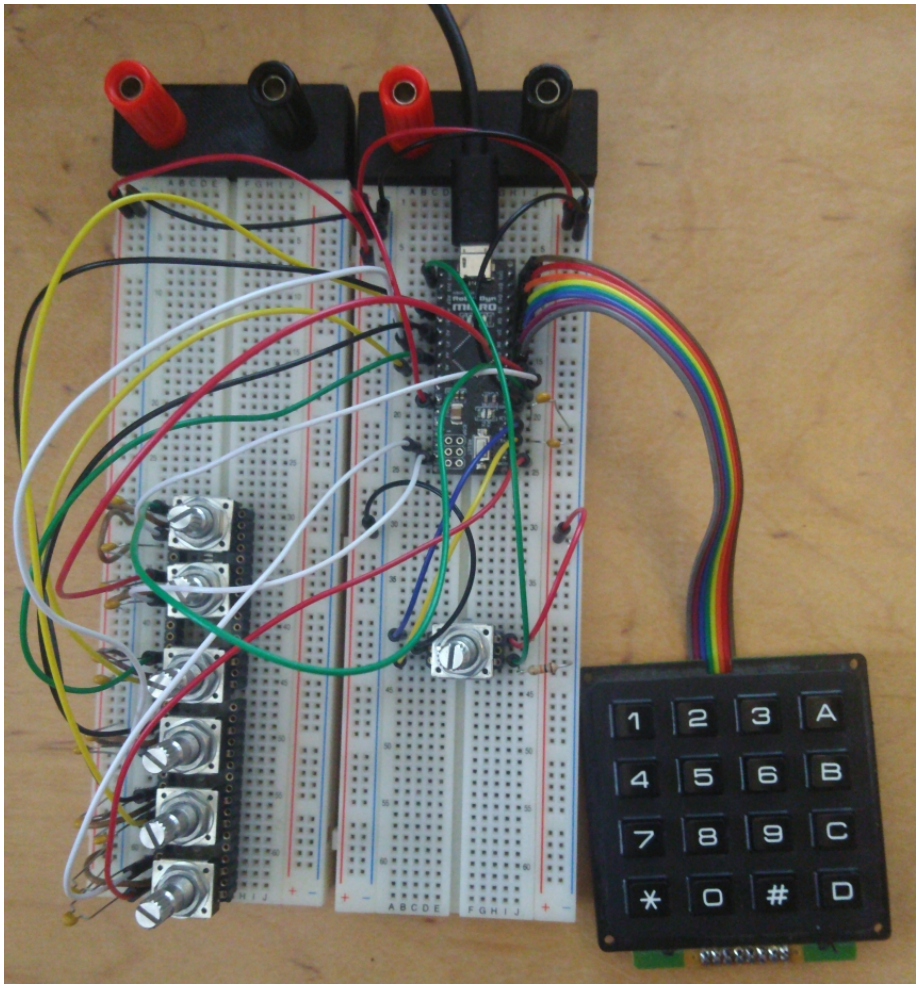
4.1.3 Muut komponenttivalinnat

Pulssianturiksi kävisi melkein mikä tahansa yleinen pulssianturi, mutta laitteeseen valittiin anturi jonka voi kiinnittää varrestaan mutterilla paneeliin, pelkän piirilevykiinnityksen lisäksi. Työssä käytetyt anturit ovat Bourns yhtiön PEC11r sarjasta, mutta muutkin samantyyppiset pulssianturit soveltuisivat käytettäväksi niiden tilalla, tosin suunnitellussa piirilevyssä on vain PEC11 sarjan (tai saman kokoisille) pulssiantureille mitoitetut paikat. Laitteeseen suunniteltiin kytkettäväksi useita antureita hiukan eri ominaisuuksilla. Joissain on pykälät, joissain ei kun taas joissain on painokytkimet ja toisissa ei. Alustavassa suunnitelmassa määriteltiin, että isoin rulla olisi painokytkimellä mutta ilman pykäliä, kun taas ylärivin nupit olisivat pykälillä sekä painikkeella. Mikäli on tarpeen, on kuitenkin mahdollista vaihtaa jokin pulssianturi erityyppiseen, jos ilmenisi että laite toimisi paremmin tämän muutoksen myötä.

Muiksi näppäimiksi valikoitui yleisesti näppäimistössä käytetyt Cherry valmistajan MX-sarjan MX Blue kytkimet, tai mallinumeron mukaan Cherry MX1A-E1NW. Tässä kytkimessä on selvä kytkeytymisraja sekä ääni. Toki mainittakoon, jos jotakuta miellyttää eri tyyppinen kytkin, kaikki sarjan näppäintyypit käyvät sen paikalle, mikäli mallin fyysiset kriteerit ovat samat. Mallinumeron toisen osan (E1NW) ensimmäinen merkki määrittää kytkimen mallin (E) ja toiseksi viimeinen mahdollisen valaistuksen. Jotta ohjainlaite olisi yksinkertaisempi toteuttaa, valittiin kytkimen mallin missä ei ole LED-valaistusta (N). Viimeinen merkki määrittää kytkimen ulkoisen rakenteen. Tässä W tarkoittaa sitä, että kytkimessä on muoviset pinnit joilla sen voi kiinnittää suoraan piirilevylle. (Developer Page – Cherry MX N.d.) Tämä mahdollisesti helpottaa kotelon suunnittelua, kun välttämättä ei tarvitse suunnitella kytkimille erillistä kiinnityslevyä. Lisäksi Cherry MX kytkimien päälle tarvitsee näppäimistöhatut. Tätä varten hankittiin eräästä verkkokauppapaikasta MX tyyppisille kytkimille tarkoitettu kuvioton (eli näppäimissä ei ole kirjainmerkintöjä) näppäimistöhattusarja.

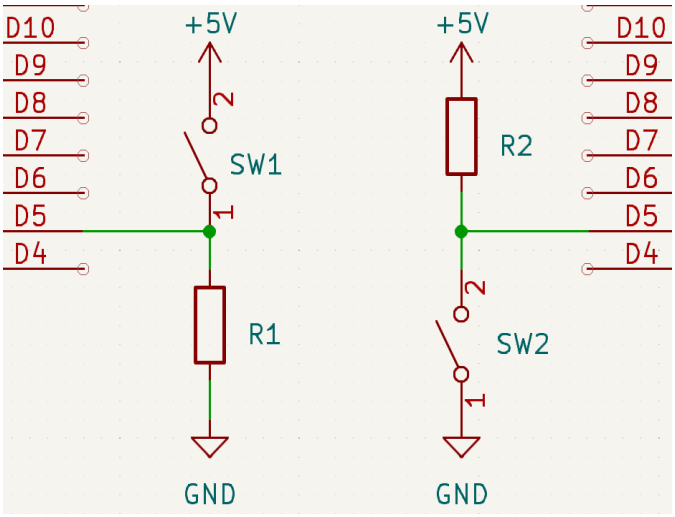
4.1.4 Piirin suunnitteluprosessi

Virtapiiriin suunnittelu tapahtui iteratiivisesti, kytkemällä komponentteja ensin yksitellen koekytkentälevylle. Aluksi kytkettiin pelkästään yhden pulssianturit kytkimineen Arduinoon, ja kokeiltiin sen lukemista Arduinolla. Kun tämä oli toimivassa tilassa, alkoi lopullisen laitteen virtapiirin suunnittelu KiCad-sovelluksella. Työ aloitettiin piirikaavion suunnittelulla, mutta pian kävi ilmi että on pakko piirtää myös itse fyysistä piirilevyä samaan aikaan, jotta komponenttien kytkennät voidaan asetella piirilevylle mahdollisimman helposti. Kun komponentteja aseteltiin KiCadissa piirilevylle, huomattiin usein, että jokin komponentti on mahdollista kytkeä helpommin jos sen siirtää eri pinniin Arduinossa, jolloin myös piirikaavio muuttui. Tätä tapahtui useaan otteeseen, kunnes työ oli siinä pisteessä, että lopullisen virtapiirin pystyi jo kytkemään koekytkentälevylle. Sen sijaan että näppäimistömatriisi olisi kytketty irtonaisista kytkimistä, käytettiin testausvaiheessa valmista numeronäppäimistömoduulia, jossa myös 16 näppäintä. Sen näppäimet on kytketty matriisiin, mutta siinä ei ole diodeja (ks. luku 3.3.1), joten se ei ole täysin vastaava lopullisen kytkennän kanssa, mutta tarpeeksi lähellä tässä vaiheessa. Piirikaavioon tuli vielä jälkeenpäinkin pieniä muutoksia tämänkin jälkeen.



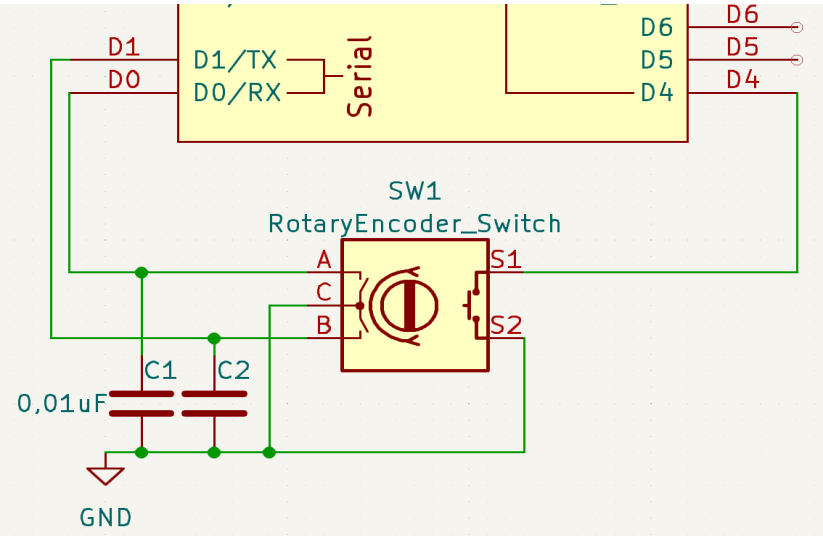
Kuvio 15: Laitteen piiri koekytkentälevyllä.

Kytkimien kytkemisessä täytyi ottaa huomioon, että on käytettävä ylös- tai alasvetovastusta, jotta pinnin tila on aina tiedetyssä tilassa, eikä ”leiju”. (eng. floating) Ylösvetovastus kytketään maan ja mikrokontrollerin pinnin välissä olevalta kytkimeltä positiiviseen jännitteeseen, kun taas alasvetovastus kytketään positiivisen jännitteen ja mikrokontrollerin pinnin välissä olevalta kytkimeltä piirin maahan. (Kuvio 16) Ylösvetovastusta käytetään yleisemmin, ellei ole erikseen tarpeen käyttää alasvetovastusta kuten esimerkiksi edellisessä kappaleessa mainutussa erikoistapauksessa. Arduinon mikrokontrollereilla on mahdollista käyttää joko ulkoista ylösvetovastusta, tai Arduinoon sisäänrakennettuja ylösvetovastuksia. Nämä ovat esimerkiksi Arduino Micron tapauksessa suuruudeltaan 20-50k ohmia (Arduino Micro – Arduino Official Store. N.d). Ulkoista ylösvetovastusta on tarpeellista käyttää esimerkiksi silloin kun piiri vaatii tietyn suuruisen ylösvetovastuksen. Arduinokorteissa ei myöskään ole sisäisiä alasvetovastuksia, joten jos kytkennässä tarvitaan sellaista, on myös käytettävä ulkoista vastusta.



Kuvio 16: Ylös- ja alasvetovastuksen kytkentä. Alasveto vastus vasemmalla, ylösvetovastus oikealla

Pulssiantureiden kytkentä oli suoraviivaista. Pulssianturin yhteinen pinni (yleensä keskimmäinen) kytkettiin maahan, ja kaksi muuta pinniä kukin omaan pinniinsä Arduino kortissa. Pulssiantureiden kytkennässä täytyy ottaa huomioon myös kosketinvärähtely (kytkimen kytkeytyessä sen kontaktit eivät yhdisty kerralla vaan värähtelevät jonkin verran, aiheutten haamupainalluksia) tai enemmänkin sen minimointi (eng. debounce). Kosketinvärähtelyn suodatus onnistuisi ohjelmallisesti, mutta myös virtapiiriin rakennettu ratkaisu toimii hyvin. Tällöin myös ohjelmakoodiin ei tule ylimääräistä. Kytkin pulssianturit kuvion 17 mukaan, jolloin kondensaattorit C1 ja C2 suodattavat piikit pois pulssianturin signaalista.



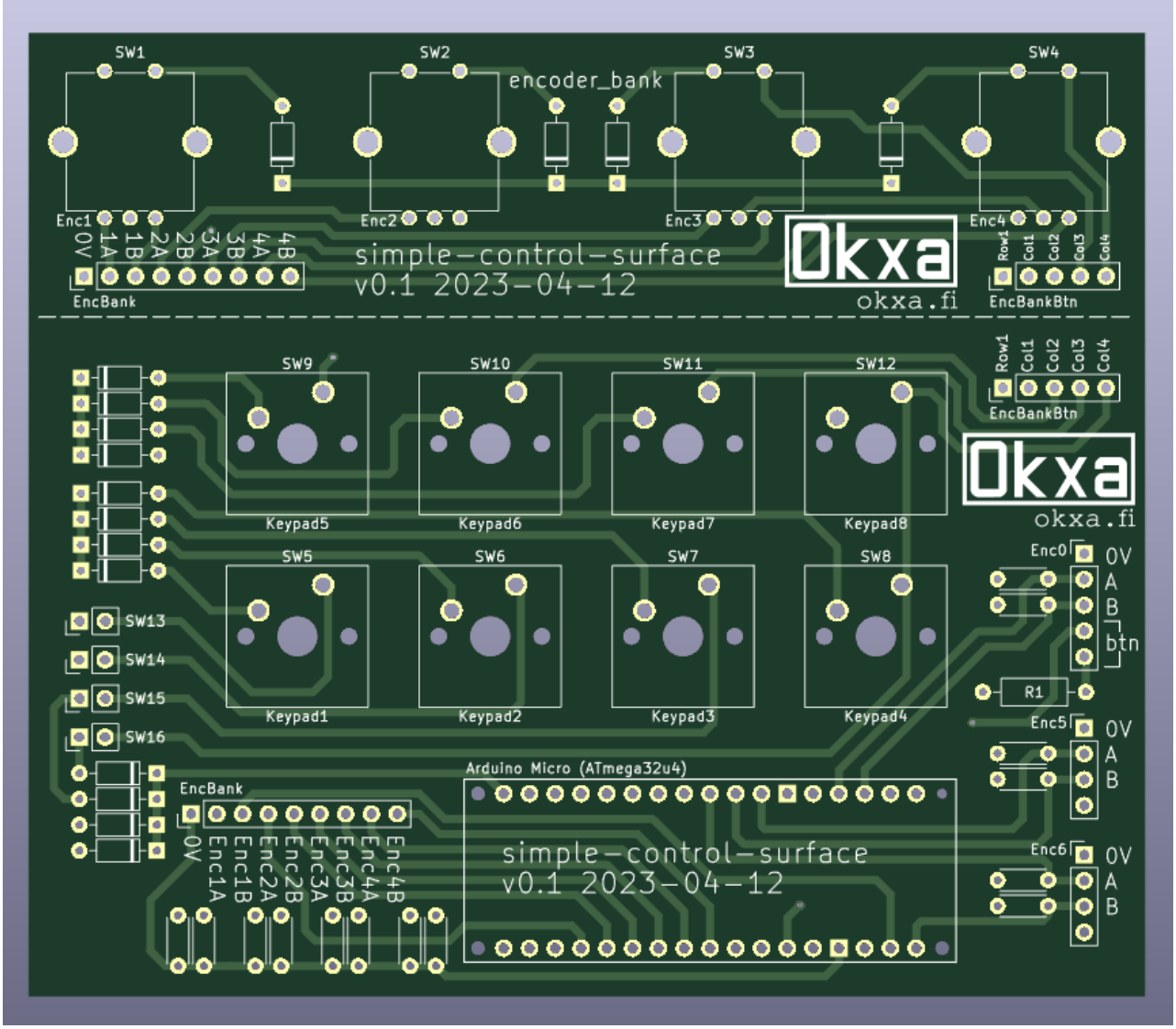
Kuvio 17: Pulssianturin kosketinvärähtelyn suodatus

4.1.5 Lopullinen virtapiiri ja piirilevy

Lopulliseen virtapiiriin (Liite 1. Ohjainlaitteen piirikaavio) kuului 7 pulssianturia ja yhteensä 17 kytkintä, joista 5 on pulssiantureiden painikkeita. Lähes kaikki kytkimet kytkettiin näppäimistömatriisiin, jonka kanssa käytettiin sisäisiä ylösvetovastuksia. Koska kytkimiä oli niin paljon, täytyi yksi kytkin (päärullan painokytkin) kytkeä erikseen vielä yhteen pinniin ja käyttää ulkoista alasvetovastusta (koska pinni oli nro. 13). Teoriassa laitteessa olisi vielä vapaana yksi pinni, mutta myös siihen on myös kytketty vastus ja LED (samoin kuin pinnissä 13) (Schematic - Micro ATmega32u4-MU N.d). Tämän pinnin LED-valo näyttää USB kommunikaation tilanteen, joten senkään takia ei ole välttämättä suositeltavaa käyttää tätä pinniä tavallisena digitaalisena ulos- tai sisääntulona (Arduino Micro – Arduino Official Store N.d).

Piirilevystä (Kuvio 18) tuli kaksipuoleinen, 110mm leveä ja 95mm korkea. Levy suunniteltiin kaksiosaiseksi jotta ylärivin nupit voi tarvittaessa nostaa eri tasolle näppäimistöstä. Levy on jaettu päälevyyn jossa on näppäimistön napit, sekä erilliseen pienempään osioon johon kiinnitettiin ylärivin 4 pulssianturia. Lisäksi piirilevyyn kytketään liittimillä ne pulssianturit ja kytkimet mitkä eivät sijaitse itse piirilevyllä. Piirilevy suunniteltiin niin että siinä ei käytetä lainkaan pintaliitoskomponentteja, sillä niitä on hankala asentaa käsin. Ainoat komponentit itse Arduino kortin, pulssiantureiden ja kytkimien lisäksi ovat 16 kpl 1n4148 (tai samantyyppisiä) diodeja (näppäimistömatriisia varten), sekä 14kpl 0.01µF keraamisia kondensaattoreita (pulssiantureiden suodatus) ja yksi 10kΩ vastus (erikseen olevan kytkimen alasvetovastus).

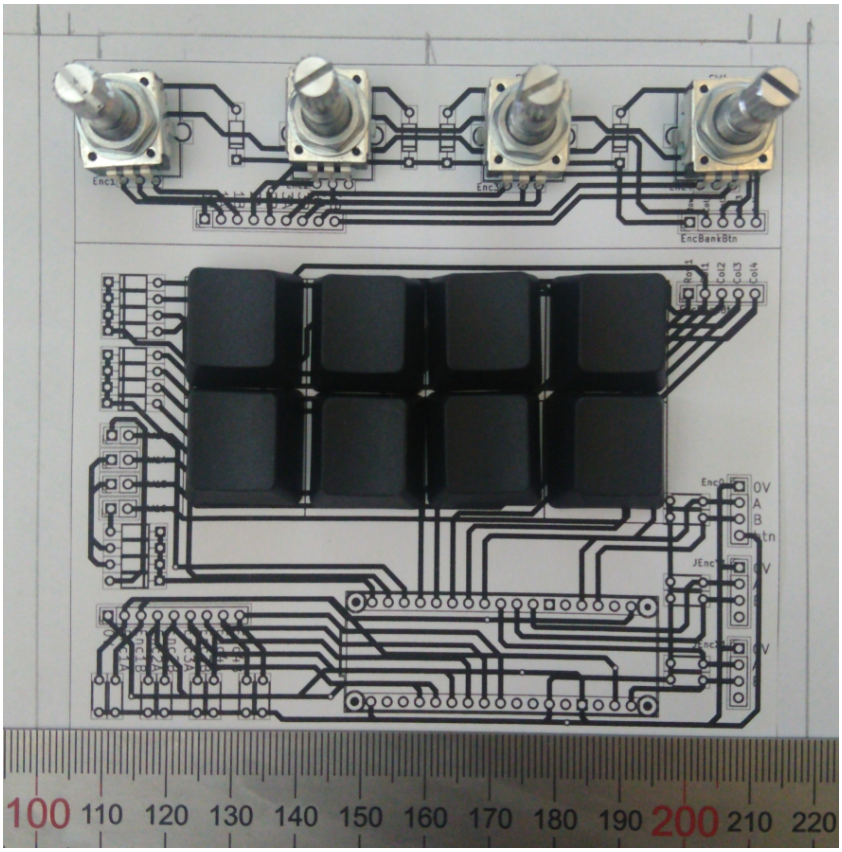
Piirilevyn suunnittelu KiCad ohjelmistolla oli suhteellisen helppoa. KiCadin mukana tulevassa komponenttikirjastoista (ks. luku 3.2) ei löytynyt kaikkia työssä tarvittavia komponentteja, joten täytyi asentaa joitain kolmannen osapuolen lisäkirjastoja. Ensimmäinen kirjasto oli ”arduino-kicad-library” joka sisältää useamman Arduino kortin, kuin myös Arduino kortin päälle asennettavan lisäkortin symbolit ja jalanjäljet, myös työssä käytetyn Arduino Micron tiedot (Parks Yong 2023.) Toinen kirjasto joka asennettiin oli ”keyswitch-kicad-library”, mikä sisältää useiden näppäimistökytkintyyppien jalanjäljet, kuten esimerkiksi Cherry MX sarjan ja yhteensopivien kytkimien jalanjäljet (Silva 2022).



Kuvio 18: Laitteen lopullisen piirilevyn 3D-esikatselu KiCad-ohjelmassa.

4.1.6 Laitekotelon suunnittelu

Laitteen fyysisen olomuodon suunnittelu aloitettiin yksinkertaisen suunnitelman mukaan (ks. luku 4.1.1), mutta tarkat mitat tarkentuivat vasta laitteen piirilevyä suunnitellessa. Piirilevylle asennettavat pulssiantureiden ja kytkimien asemoiminen kohdilleen alkoi arvioimalla kuinka lähellä pulssianturit voivat olla toisiaan. Lopullinen etäisyys oli 30mm. Näppäimistökytkimille tarkoitettu etäisyys on 19mm, ja se olikin määritelty käyttämässäni komponenttikirjastossa, joten niiden kytkinten asettelu ei aiheuttanut hankaluuksia. Lopullinen testi tapahtui tulostamalla piirilevyn suunnitelma paperille, ja asettamalla komponentit sen päälle niille varatuille paikoille (Kuvio 19).

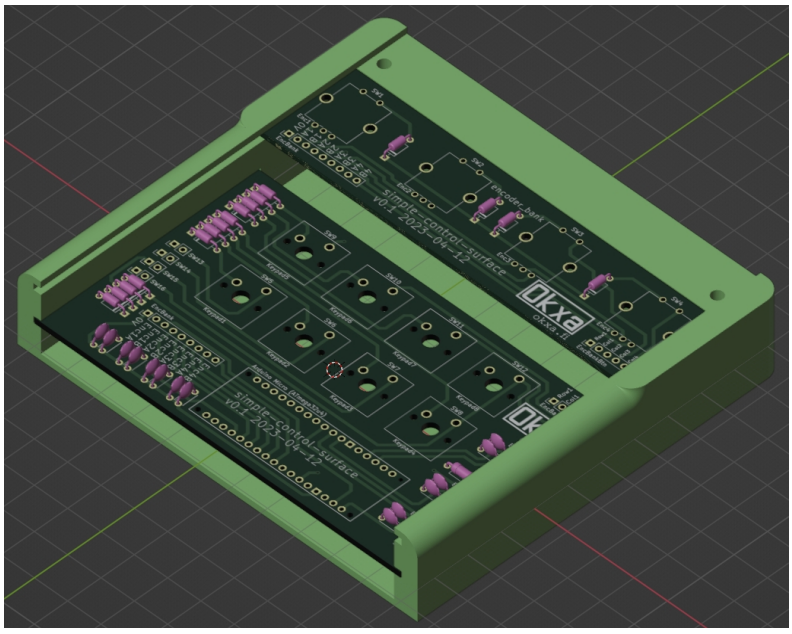


Kuvio 19: Ohjainlaitteen mittasuhteiden hahmottelu tulostetun piirilevysuunnitelman avulla

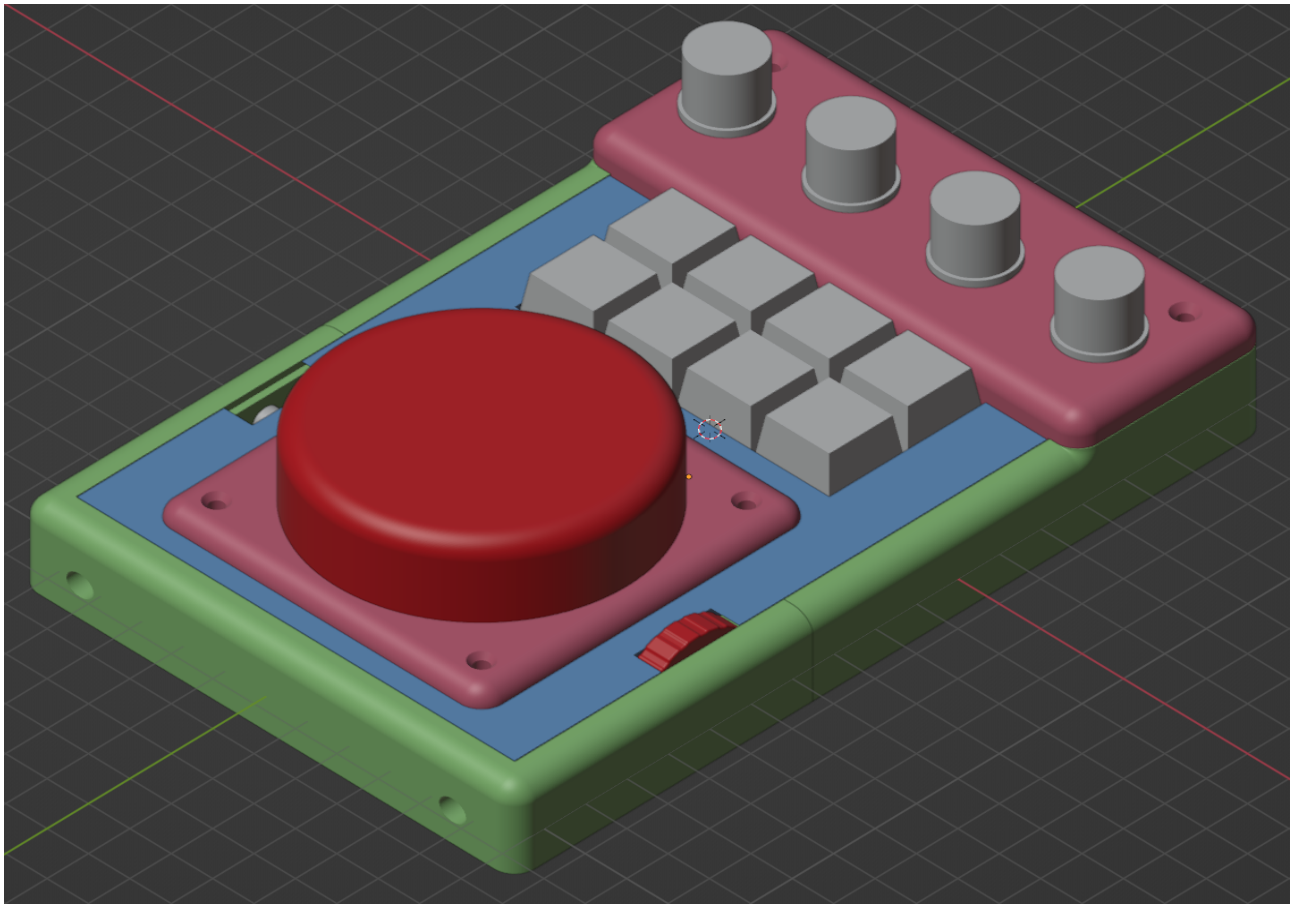
Kun piirilevy oli suunniteltu kokonaan (ja siten sen ulkoiset mitat olivat tiedossa), aloitettiin laitekotelon suunnittelu Blender ohjelmalla. Työstämistä helpotti piirilevyn 3D-mallin tuominen Blenderiin. Se täytyi tehdä hiukan mutkan kautta, sillä KiCad sovelluksesta pystyy viemään 3D-mallin esimerkiksi VRML-tiedostoon, mutta sen tuottama 3D-malli oli aivan liian tarkka, ja sen käyttö Blenderissä olisi hankalaa. Joten sopivan 3D-mallin tuottaminen täytyi tehdä mutkan kautta, ensin viemällä piirilevy STEP-tiedostoon, jonka voi sitten avata esimerkiksi FreeCAD-sovelluksessa, jolla mallin voi viedä esimerkiksi Gltf tai Obj muotoiseen tiedostoon, mitä Blender osaa lukea. Lisäksi

tehtiin vielä tekstuurikuva KiCadin 3D-näkymän piirilevystä, jotta piirilevyn kaikki ominaisuudet näkyisivät Blenderissä ja komponenttien sijoittelu kotelon valmistusta varten olisi helpompaa. Ensin KiCadin 3D-näkymän tausta asetettiin yksiväriseksi (Preferences -> 3D Viever -> Colors -> Background Gradient Start/End), sekä kamera asetettiin ortografiseen tilaan. Kuvankaappauksen voi ottaa valikosta ”Edit -> Copy 3D Image”, jolloin kuva kopioituu leikepöydälle. Tässä kuvassa on vielä reunat, joten kuvaa muokattiin GIMP kuvankäsittelyohjelmalla, rajaten piirilevyn ulkopuolinen osa pois kuvasta. Tämän kuvan sitten pystyy asettamaan piirilevyn 3D-mallin tekstuuriksi Blenderissä. Vielä ennen itse kotelon mallinnuksen aloitusta kannattaa tarkistaa onko piirilevyn 3D-malli oikeissa mitoissa. Käyttäen luvussa 3.4.5 mainittua MeasureIt-lisäosaa, piirilevyn mallista mitattiin levyn korkeus, leveys ja etenkin paksuus. Korkeus ja leveys olivatkin tasan määritellyn kokoiset, mutta mitattu paksuus Blenderissä oli vain 1.58mm kun piirilevyn oikea paksuus on 1.6mm, joten piirilevyn paksuutta täytyi kasvattaa 0.02mm.

Koska piirilevyyn ei piirretty kiinnitysreikiä, levy ajateltiin kiinnitettäväksi kiskoihin jotka olisivat sisäänrakennettuna laitekoteloon. Tämä toteutettiin tekemällä kotelon alapuoleen kiskot kullekin piirilevylle (eli jaotellun piirilevyn eri osille ks. luku 4.1.5) joihin piirilevyt liu'utetaan. (Kuvio 20) Samoja kiskoja käytetään kotelon kannen kiinnitykseen. (Kuvio 21, sininen kappale) Samassa kuviossa on myös näkyvillä laitteen lopullinen kotelo kokonaisuudessaan. Harmaan väriset osat eivät ole osa itse 3D-tulostettavaa koteloa, vaan ne esittävät muita osia kuten näppäimiä ja pulssiantureita (ja niiden nuppeja). Laitteeseen tulostetaan päärullan nuppi, sekä kaksi pienempää pyörää päärullan molemmille puolille. (Väritetty punaisella kuviossa 21)



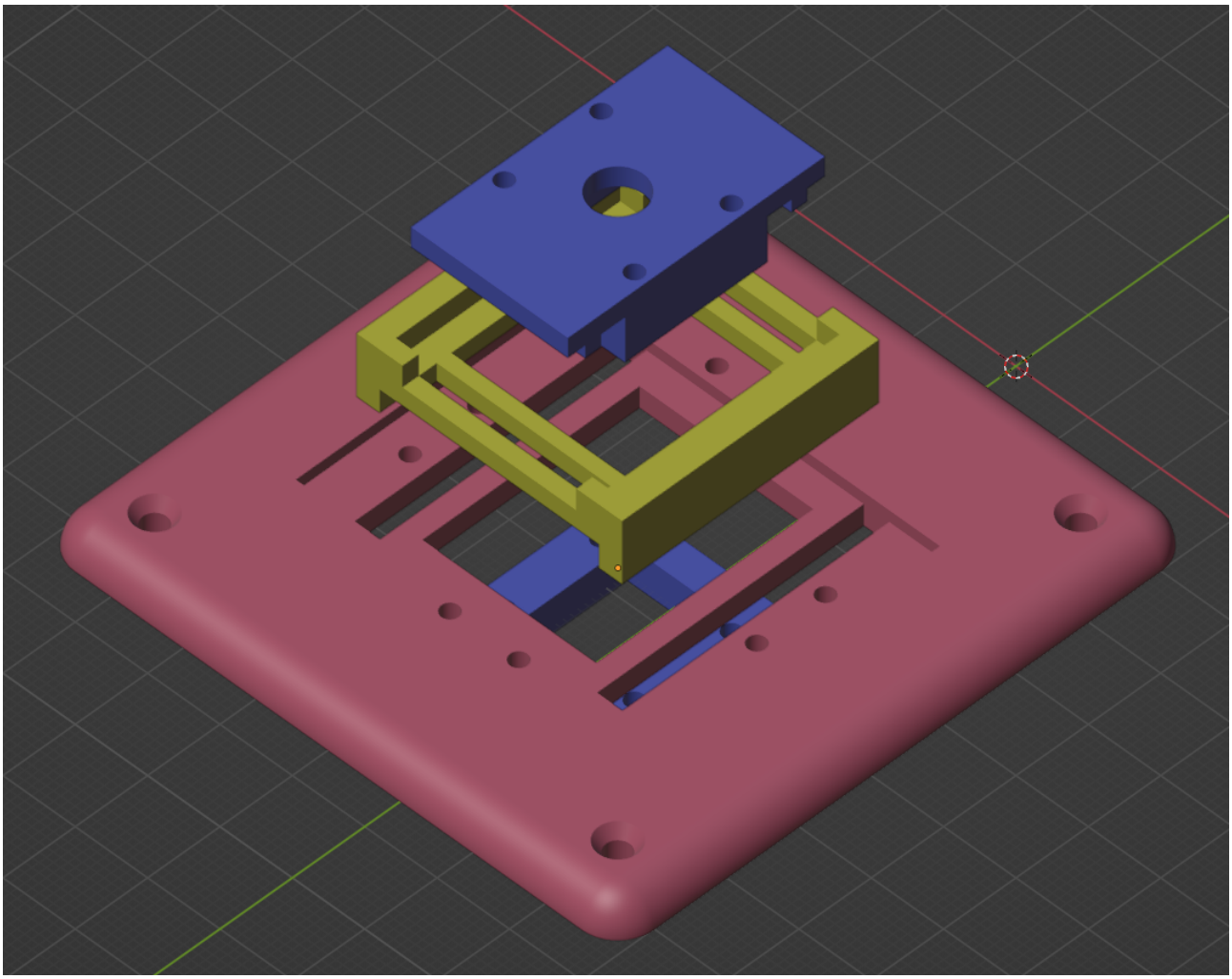
Kuvio 20: Laitekotelon alapuoli Blenderissä ja piirilevyjen kiinnitystapa.



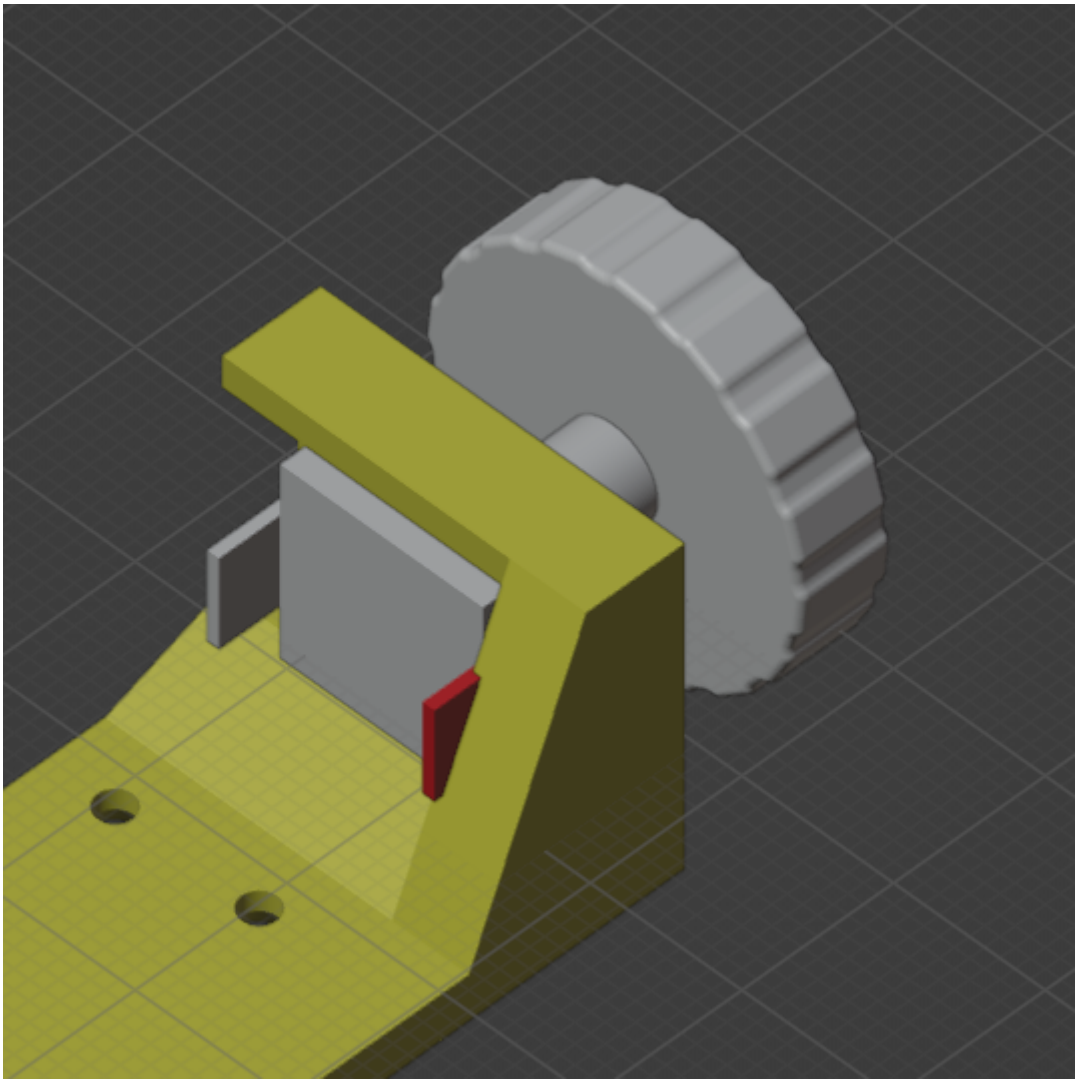
Kuvio 21: Laitekotelo kokonaisuudessaan

Kaikki kotelon osat on suunniteltu niin että ne ovat mahdollisimman yksinkertaisia tulostaa myös kohtuuhintaisilla harrastelijatason FDM-tyyppisillä 3D-tulostimilla. Käytännössä tämä tarkoittaa sitä, että osien suunnittelussa on otettu huomioon millaisia rakenteita tällainen 3D-tulostin voi tulostaa. Esimerkiksi tukirakenteiden tarve on minimoitu, sillä vaikka tukirakenteiden tulostaminen on mahdollista, ei niistä jää yhtä siisti jälki kuin jos kappaleen suunnittelee tulostettavaksi ilman tukirakenteita. Usein tukirakenteiden käytön voi välttää, kun vain suunnittelee kappaleen niin että mikään osa ei tulostaessa jää roikkumaan tyhjän päälle. Esimerkiksi em. kotelon tapauksessa, pohjakappale on suunniteltu tulostettavaksi pystyasennossa, niin että siihen tulevat kiskot ovat pystysuunnassa, jolloin kiskoihin ei tarvitse tukimateriaalia sisälle, joka olisi hankala poistaa pienestä raosta.

Mekaanisesti monimutkaisin osuus laitteessa on kotelon päällislevyyn kiinnitettävä laitteen päärullan mekanismi. (Kuvio 22) Mekanismi koostuu levystä (kuvassa punainen kappale), jossa on urat seuraavaa kappaletta (keltainen) varten, joka puolestaan liukuu näissä urissa pystysuuntaan (vaakatasossa). Ylimpänä sijaitseva osa (sininen) liikkuu vaakasuunnassa edellisen komponentin (keltainen) päällä. Sinisen osan pohjaan kiinnitetään ruuveilla lisäosa, joka aktivoi, alimmaiseen punaisella merkityn levyn alapuolelle kiinnitetyt mikrokytkimet. Eli kokonaisuudessaan mekanismi liikkuu kahdella akselilla, ja aktivoitujen mikrokytkimien perusteella mikrokontrollerilla voidaan lukea nupin vaakasuuntaiset liikkeet. Nupin kiertoliike puolestaan luetaan pulssianturilta, joka kiinnitetään päällimmäisessä sinisessä osassa olevaan reikään. Lisäksi päärullan sivuilla olevat pienempien rullien pulssianturit kiinnitetään päärullan levyn alle tulevaan kappaleeseen. (Kuvio 23)



Kuvio 22: Päärullan mekanismi.



Kuvio 23: Päärullan sivuilla olevien rullien kiinnitys.

4.2 Ohjelmakoodi

4.2.1 Sisääntulojen lukeminen manuaalisesti

Laitteen ohjelmakoodia työstettiin yhtäaikaisesti raudan suunnittelun kanssa. Aluksi kokeiltiin yhden pulssianturin ja kytkimen lukemista täysin ilman ohjelmistokirjastoja (kuten luvun 4.1.4 alussa mainittiin), jotta saataisiin kuva kuinka ohjelmakoodi toimii. Ensin määriteltiin käytettävät pinnit, jotka sitten asetetaan ”setup” funktiossa haluttuun tilaan:

```
1  // määritellään pinnit
2  const uint8_t enc0PinL = 0;
3  const uint8_t enc0PinR = 1;
4  const uint8_t enc0PinSw = 13;
5
6  void setup() {
7      Serial.begin(9600);
8      // asetetaan pinnien tilat
9      pinMode(enc0PinL, INPUT_PULLUP);
10     pinMode(enc0PinR, INPUT_PULLUP);
11     pinMode(enc0PinSw, INPUT_PULLUP);
12     // keskeytetään ohjelma pulssianturin käännöllä
13     attachInterrupt(digitalPinToInterrupt(enc0PinL), readEncoder, CHANGE);
14     attachInterrupt(digitalPinToInterrupt(enc0PinR), readEncoder, CHANGE);
15 }
```

Tässä esimerkissä pinnit on määritelty vakioina (”`const`”-sanalla) riveillä 2-4. Vakiot ovat lauseita, joilla on muuttumaton arvo (Constants N.d). Tällainen arvo ei voi muuttua ohjelman ajon aikana. Arduino ympäristössä funktio ”`setup`” joka alkaa rivillä 6, ajetaan vain kerran mikroprosessorin käynnistyksen aikana. Tässä esimerkissä tuossa funktiossa ensin käynnistetään sarjaporttiyhteys tietokoneen kanssa rivillä 7 ja sitten asetetaan kukin pinni tilaan ”`INPUT_PULLUP`”, jolloin käytetään mikroprosessorin sisäistä ylösvetovastusta. Muut tilat digitaalisille pinneille ovat ”`INPUT`” ja ”`OUTPUT`”, jotka tarkoittava vastaavasti että pinni on sisääntulotilassa, mutta käytetään ulkoista ylös- tai alasvetovastusta ja että pinni toimii ulosmenona, ohjaten esimerkiksi jotain LED-valoa. Lisäksi riveillä 12 ja 13 kiinnitetään pulssiantureiden pinneihin keskeytykset, jotta anturin arvo voidaan lukea mahdollisimman nopeasti (ks. luku 4.1.2). Keskeytykset kutsuvat ”`readEncoder`”-funktiota. Itse ajon aikana tapahtuva koodi on ”`loop`” funktiossa:

```
1  void loop() {
2      // luetaan pulssianturin painokytkin
3      if (!digitalRead(enc0PinSw)) {
4          Serial.println("pushbutton");
5      }
7  }
```

Funktio on nimensä mukaan loputtomasti itseään toistava. Tässä esimerkissä luetaan Arduinon ympäristössä määritetyllä ”digitalRead”-funktiolla pulssianturin pinnin tila ja mikäli se **ei ole** ”HIGH”-tilassa (eli se on ”LOW”-tilassa), tulostetaan sarjaportin kautta tekstiä. Tämä näin päin siksi että käytettäessä ylösvetovastusta on kytkimen vakiotila ”HIGH”, jolloin painettaessa kytkintä yhdistyy piiri maahan ja pinni vetäytyy maahan ja ”LOW”-tilaan. Tässä esimerkissä tekstiä virtaa jatkuvasti sarjaportin kautta niin kauan kuin kytkin on kytkettynä. Jotta painikkeet toimivat hyödyllisemmin, täytyy tarkistaa kytkimen tila. Tämä onnistuu esimerkiksi seuraavalla tavalla:

```
1  static uint8_t enc0PinSwLast{0}; // static, arvo sama joka suorituskerralla
2  uint8_t enc0PinSwValue = digitalRead(enc0PinSw);
3  if (enc0PinSwValue != enc0PinSwLast) {
4      if (enc0PinSwValue == HIGH) {
5          Serial.println("pushbutton click");
6      }
7      else {
8          Serial.println("pushbutton release");
9      }
10 }
11 enc0PinSwLast = enc0PinSwValue;
```

Käytännössä pidetään kirjaa ja tarkistetaan, mikä arvo kytkimellä on ollut viimeksi. Ensimmäinen ehtolauseke (rivi 3) läpäisee vain jos arvo on muuttunut suorituskertojen välillä. Lisäksi sisempi ehtolauseke (rivit 4-9), tarkistaa onko nykyinen arvon muutos painokytkimen vapautus vai painallus. Lopuksi vielä asetetaan nykyinen arvo edellisen arvon muuttujaan.

Kokonaisen näppäimistömatriisin lukeminen tapahtuu yksinkertaisimmiltaan skannaamalla matriisin sarake- ja rivipinnit vuoron perään, kytkemällä kukin sarakepinni vuorotellen ”OUTPUT”-tilaan, ja sitten lukemalla mitkä rivipinnit ovat aktiivisia sillä skannausiteraatiolla. Samoin kuin vain yhden näppäimen tapauksessa, pidetään myös kirjaa siitä että mikä on matriisin edellinen tila. Tämä tapahtuu esimerkiksi tallettamalla kunkin matriisin näppäimen tilan kaksiulotteiseen taulukkoon:

```
int matrixRows = 4; // kokorajat
int matrixCols = 4;
bool matrixState[matrixRows][matrixCols] {}; // matriisin tila
bool matrixStateLast[matrixRows][matrixCols] {};
// määritellään näppäimistömatriisin pinnit
// (asetettaisiin start() funktiossa oikeaan tilaan)
uint8_t rows[matrixRows] = { 6,7,8,9}; // rivit
uint8_t cols[matrixCols] = { 1,3,4,5}; // sarakkeet
// skannataan matriisi
for (int c = 0; c < matrixCols; c++) {
    byte col = cols[c]; // haetaan sarakkeen pinni taulukosta
    pinMode(col, OUTPUT); // asetetaan sarakkeen pinni OUTPUT -tilaan
    digitalWrite(col, LOW); // kirjoitetaan sarakepinniin LOW
    for (int r = 0; r < matrixRows; r++) {
        byte row = rows[r]; // haetaan rivin pinni taulukosta
        pinMode(row, INPUT_PULLUP); // asetetaan rivipinni sisääntuloksi
        matrixState[r][c] == digitalRead(row); // skannattaan rivipinni
        // tarkistetaan painikkeen tila
        if (matrixState[r][c] != matrixStateLast[r][c])
            if (matrixState[r][c]) {
                Serial.println(" matrix click");
            }
            else {
                Serial.println(" matrix release");
            }
            Serial.print(" key: ");
            Serial.print(r);
            Serial.println(c);
        }
        pinMode(row, INPUT); // palauta pinni INPUT pinniksi
        // asetetaan matriisin viimeinen tila
        matrixStateLast[r][c] = matrixState[r][c];
    }
    pinMode(col, INPUT);
}

```

Pulssianturin tila luetaan omalla funktiollaan (jota kutsutaan siis em. keskeytyksillä). Jos pulssiantureiden arvot tahdotaan nopeasti ja tehokkaasti, kannattaa toimia tavujen tasolla. Monenlaisia esimerkkejä löytyy internetistä, myös vielä paremmin optimoituja kuin tämä paljolti Adam Meyerin (2012) toteutukseen perustuva esimerkki:

```
void readEncoder() {
    static uint8_t enc0Last = 0; // viimeisin arvo
    int MSB = digitalRead(enc0PinL); // eniten merkitsevä tavu
    int LSB = digitalRead(enc0PinR); // vähiten merkitsevä tavu

    int enc = (MSB << 1) | LSB; // yhdistetään molemmat arvot
    int sum  = (enc0Last << 2) | enc; // yhdistetään edelliseen arvoon
    if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011) {
        Serial.println("CW");
    }
    if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum == 0b1000) {
        Serial.println("CCW");
    }
    enc0Last = enc;
}
```

Tässä antureiden tilat tallennetaan yhteen bittiin. (Bitissä kahdeksan tavua, eli binäärimuotoisena `0b00000000`) Rivillä kuusi oleva lauseke siirtää ensin ”`MSB`”-muuttujan viimeisen tavun yhden askeleen vasemmalle (eli jos MSB oli ”`1 == 0b00000001`” tulee lausekeen ”`(MSB << 1)`” jäkeen arvoksi ”`0b00000010`”). Lausekkeen toisessa osassa ”`| LSB`” pystyviiva on TAI bittioperaattori. Se asettaa ne bitin arvot ykköseksi mitkä ovat arvossa yksi, joten jos ”`LSB`”-muuttuja on arvossa 1 (”`1 == 0b00000001`”), seuraa että ”`enc`” muuttujan arvo tulee olemaan ”`0b00000011`”. Seuraava rivi tekee samankaltaisen operaation yhdistäen nykyisen lukeman edelliseen arvoon, jolloin saadaan arvo muotoa ”`0b00001011`”. Tätä ”`sum`” arvoa sitten vertaillaan rivien 8-13 ehtolausekkeissa. Luvussa 3.3.2 esitellyn logiikan mukaan, anturin suunta voidaan selvittää missä järjestyksessä pulssit tapahtuvat.

4.2.2 Sisääntulojen lukeminen kirjastojen avulla

Kuten edellisessä luvussa esitettiin, ei optimoidun ohjelmakoodin tuottaminen ole yksinkertaista (ja edellisen luvunkaan esimerkit eivät ole kaikkein optimaalisimpia). Täten oli järkevää hyödyntää valmiita ohjelmakirjastoja, joissa on jo valmiiksi hyvin optimaaliset algoritmit. Lisäksi tällaiset kirjastot usein helpottavat itse ohjelman kirjoitusta muutenkin. Näissä kirjastoissa on usein määritelty luokka joka kuvaa kyseessä olevaa ohjainlaitetta, kuten pulssianturia tai näppäimistömatriisia. Tällöin laitteen voi määritellä yhdellä lausekkeella ja arvojen lukeminenkin helpottuu.

Pulssiantureiden lukemiseen valittiin ClickEncoder-kirjasto, joka lukee pulssianturit ajastinpohjaisilla keskeytyksillä yhden millisekunnin välein. Lisäksi kirjastolla voi myös lukea pulssianturien painokytkimien tilan. (0xPIT 2014) Tosin 0xPIT-käyttäjä vaikuttaa lopettaneen ClickEncoder-kirjaston ylläpidon, mutta koska kirjasto on lisensoitu avoimella lisenssillä, siitä on saatavilla päivitettyjä versioita. Esimerkiksi Schallbertin päivittämä versio sisältää joitain uusia ominaisuuksia sekä paremman algoritmin antureiden lukemiseen, joten valitsin tämän version kirjastosta (Schallbert 2023). Schallbertin (kuin myös alkuperäinenkin) versio ClickEncoder-kirjastosta on lisensoitu MIT-lisenssillä (Schallbert 2021). Täten kirjastoa voisi tarvittaessa käyttää kaupallissakin projekteissa, sillä lisenssin ehdot eivät tätä rajoita.

Myös näppäimistömatriisiin lukemiseen on saatavilla eri kirjastoja, mutta useimmat näistä on tarkoitettu enemmän vain yleisten 4x4 tai 3x4 kokoisten numeronäppäimistöjen (samanlaisten kuin tässäkin työssä käytettiin kytkettäessä piiriä koekytkentälevyllä, ks. Kuvio 15) lukemiseen ja ne

usein tekevät oletuksia raudasta, esimerkiksi juuri näppäimistön koon suhteen (rivien/sarakkeiden määrä). Työssä käyttöön valikoitui Nick Gammonin Keypad Matrix -kirjasto, jolla voi lukea minkä kokoisen näppäimistö matriisin tahansa (Gammon 2018a). Myös Keypad Matrix kirjasto on lisensoitu MIT-lisenssillä, kuten ohjelman lähdekoodissa on kerrottu (Gammon 2018b).

Kumpaakin kirjastoa käytetään luomalla ilmentymä kirjaston edustamasta luokasta. ClickEncoder-kirjaston tapauksessa voi esimerkiksi määritellä fyysistä pulssianturia vastaavan objektin luokan ”`ClickEncoder`” avulla. Tästä luokasta ilmentymää luodessa määritellään mihin Arduinon pinneihin pulssianturi sekä sen pinnit on kytketty, sekä pulssianturin askeleiden määrä yhden pykälän kohdalla (eli kuinka monta eri tilaa pulssianturin pulsseissa on käännettäessä anturia yhden pykälän verran yleisimmin 4). Viimeinen parametri luokan muodostajafunktiossa (”konstruktori”, eng. constructor) määrittää onko anturi ja sen kytkin aktiivinen ”`HIGH`” vai ”`LOW`”-tilassa. Luokalla on muutamia funktioita, osa liittyy anturin arvon lukemiseen ja osa eri ominaisuuksien, kuten kiihtyvyyden, päälle asettamiseen. Käytännössä anturin lukeminen tapahtuu niin että sen ”`service`”-funktiota kutsutaan ajastinkeskeytyksellä, esimerkiksi käyttäen Arduinon ajastinta 1 ”`TimerOne`”-kirjastolla:

```
 1  const uint16_t timerInterval = 1000; // ajastimen intervalli, 1ms (1000 us)
 2  static TimerOne timer;
 3
 4  void setup() {
 5      timer.initialize(timerInterval);
 6      timer.attachInterrupt(timerInterrupt);
 7  }
 8
 9  void timerInterrupt()
10  {
11      // pulssiantureista tallennettu osoittimen taulukkoon,
12      // kustakin pulssianturista kutsutaan service funktiota
13      for (int x = 0; x < encodersSize; x++) {
14          encoders[x]->service();
15      }
16  }
```

Tämä funktio lukee itse fyysisen pulssianturin tilan ja tallettaa sen arvon. Sitten muualla koodissa, esimerkiksi ”`loop`”-funktiossa, voidaan lukea anturin arvo käyttäen joko ”`getIncrement`” tai ”`getAccumulate`”-funktioita. Näistä ensimmäinen palauttaa joko 0, 1 tai -1, riippuen anturin kiertosuunnasta ja jälkimmäinen kokonaisluvun joka on joko negatiivinen tai positiivinen tai 0 anturin asennosta riippuen, pitäen kirjaa laitteen käynnistyksestä asti. ”`getButton`”-funktio taas nimensä mukaisesti palauttaa painokytkimen tilan.

”ClickEncoder”-luokka oikeastaan koostuu kahdesta eri luokasta, ”Encoder” ja ”Button”. Näitä luokkia on mahdollista käyttää myös itsestään, mahdollistaen esimerkiksi vain pulssianturin määrittelyn kirjaston avulla. Tätä tarvittiinkin, sillä kaikki (paitsi yksi ja sekin täytyi kytkeä alasveto-vastuksella (ks. luku 4.1.5), jolloin sen aktiivinen tila on eri kuin itse pulssianturilla, joten senkin kytkin täytyi määritellä erikseen itse pulssianturista) ohjainlaitteeseen tulevat kytkimet on kytketty erikseen näppäimistömatriisiin. Näppäimistömatriisiluokan ilmentymä määritellään seuraavasti:

```
// matriisin pinnit taulukoissa
const uint8_t matrix1Rows[] = {9,10,11,12};
const uint8_t matrix1Cols[] = {5,6,7,8};
// lasketaan koot pinnitaulukoille
const int mRC = sizeof(matrix1Rows)/sizeof(matrix1Rows[0]);
const int mCC = sizeof(matrix1Cols)/sizeof(matrix1Cols[0]);

// fyysisten nappien määrittely
const char physButtons[matrix1RowsSize][matrix1ColsSize] = {
    {00, 01, 02, 03},
    {10, 11, 12, 13},
    {20, 21, 22, 23},
    {30, 31, 32, 33},
};
// määritellään matriisi, Nick Gammonin Keypad_Matrix -kirjastolla
Keypad_Matrix matrix1 = Keypad_Matrix(makeKeymap(physButtons), mRows, mCols, mRC, mCC);
```

Luokan ”Keypad_Matrix” muodostojafunktio ottaa vastaan kaavion näppäimistön asettelusta, sekä matriisin käyttämät pinnit taulukkomuodossa (tai oikeastaan annetaan vain osoitin tähän taulukkoon, käyttäen kirjastossa määriteltyä ”makeKeymap”-funktiota), sekä kuinka monta riviä ja saraketta näppäimistössä on (yllä olevassa esimerkissä nämä on laskettu pinnitaulukkojen koon mukaan). Näppäimet määritellään merkkitaulukkona, mutta koska merkki-tietotyypin voi esittää myös kokonaislukuina, voidaan taulukkoon tallentaa muutakin tietoa. Tässä tapauksessa se millä matriisin rivillä ja sarakkeella kukin painike sijaitsee. Itse matriisin lukeminen toimii seuraavasti; ”setup”-funktiossa asetetaan matriisi toimintaan, sekä asetetaan matriisille näppäimien painallusta ja nostoa käsittelevät funktiot:

```
void setup() {
    matrix1.begin();
    matrix1.setKeyDownHandler(onKeyDown);
    matrix1.setKeyUpHandler(onKeyUp);
}
```

Sitten näissä funktioissa voidaan käsitellä näppäinten painallukset, tässä tapauksessa vain tulostetaan sarjaportin kautta se näppäin, joka on painettuna. Funktiossa arvo ”key” on edellä määritel-

lystä taulukosta. Huomauttamisen arvoista on myös, että matriisilta voi tarkistaa onko jokin muu näppäin samanaikaisesti painettuna, joten näppäinyhdistelmätkin ovat mahdollisia:

```
 1  void onKeyDown(const char key) {
 2      Serial.print("pressed ");
 3      Serial.println(key);
 4      if (matrix1.isKeyDown('A')) {
 5          Serial.println("A is also down");
 6      }
 7  }
 8  void onKeyUp(const char key) {
 9      Serial.print("released ");
10      Serial.println(key);
11  }
```

4.2.3 Komentojen lähettäminen tietokoneelle kirjastojen avulla

Sen lisäksi että kirjastoja käytettiin antureiden ja näppäinten lukemiseen, tarvittiin myös kirjastoja joilla voi lähettää haluttuja ohjauskomentoja tietokoneelle. Luvussa 3.2 käsitellyistä vaihtoehdoista ensimmäiseksi käyttöön otettiin yksinkertaisimmat ”Keyboard” ja ”Mouse”-kirjastot, sillä niiden avulla on helpointa testailla laitteen toiminnallisuutta, kuin erikoisempien kirjastojen (kuten ControlSurface tai eri MIDI-kirjastoja) kanssa. Toisekseen pelkkien näppäinkomentojen lähetys tekee laitteesta mahdollisimman monikäyttöisen. Mikäli nämä vakiokirjastot jossain vaiheessa todettaisiin liian rajoittuneeksi, voitaisiin tarvittaessa siirtyä käyttämään vaikka samassa luvussa käsiteltyä NicoHoodin HID-kirjastoa, mutta ainakin aluksi nämä vakiokirjastot todettiin riittäväksi.

Keyboard ja Mouse-kirjastojen peruskäyttö on yksinkertaista. Ensin sisällytetään kirjastot ohjelmaan ja käynnistetään ne kutsumalla ”`setup`”-funktiossa kummankin kirjaston ”`begin`” funktiota. Sitten voidaan kutsua kirjastojen metodeja, jotka lähettävät näppäimistön tai hiiren komentoja tietokoneelle. Esimerkiksi luvussa 4.2.1 esitellyn esimerkin tapauksessa voitaisiin lähettää näppäimistön painallus vaikka seuraavasti:

```
 1  void loop() {
 2      static uint8_t enc0PinSwLast{0};
 3      uint8_t enc0PinSwValue = digitalRead(enc0PinSw);
 4      if (enc0PinSwValue != enc0PinSwLast) {
 5          if (enc0PinSwValue == HIGH) {
 6              Keyboard.press('n');
 7          }
 8          else {
 9              Keyboard.release('n');
10          }
11      }
12      enc0PinSwLast = enc0PinSwValue;
13  }
```

Keyboard-kirjastoon on myös määritelty makroilla useimmat erikoisnäppäimet, joten niiden käyttäminen on helppoa. Myös eri näppäimistöasettelujen erikoisnäppäimet (kuten esimerkiksi Å,Ä ja Ö) on huomioitu kirjastossa. Silloin täytyy käyttää kunkin asettelun lisäkirjastoa joka lisää makrot näille näppäimille. (Keyboard Modifiers and Special Keys 2022.) Keyboard kirjaston mukana tulee tanskan, saksan, espanjan, ranskan, italian ja ruotsin kielien näppäimistöasettelun mukaiset asettelukirjastot. Ruotsin asettelu on käytännössä sama kuin suomalainen, joten sitä voisi käyttää jos tarvitsee juuri niitä näppäimiä suomalaisessa näppäimistössä.

Ohjainlaitteen lopulliseen koodiin tahdottiin jonkinlainen keskitetty tapa asettaa tietokoneelle lähetettävät näppäinkomennot, jotta kunkin fyysisen kytkimen tai nupin toiminnon asettaminen on mahdollista helposti, muuttamatta itse ohjelmakoodia. Tätä varten luotiin tietue, jossa voidaan määritellä tietokoneelle ajettavat komennot, sekä ajettavien komentojen tyyppi:

```
1  struct HIDAction {
2      enum { DEVICE_KEYBOARD, DEVICE_MOUSE } actionDevice;
3      const char *values;
4  };
```

Tietueessa ”`actionDevice`” on lueteltu tyyppi (eng. enumeration, tai ”enum”), joka määrittää minkä tyyppisen laitteen komennosta on kyse, sillä tätä samaa tietuetta käytetään myös hiiren liikkeisiin. Tämä onnistuu sillä Mouse kirjaston ”`move`”-funktion parametrit ovat tyyppiä etumerkillinen merkki (eng. ”signed char”) (Mouse – Mouse.move() 2022). Itse lähetettävät komennot ovat merkkitaulukossa ”`values`”, jonka sisältö riippuu toiminnon ”`actionDevice`”-arvosta. Näppäimistölle ne ovat yksinkertaisesti Keyboard-kirjaston hyväksymiä näppäin arvoja, hiirelle puolestaan numeerisia arvo hiiren liikkeistä (Hiiren liikkeen X ja Y sekä rullan liike ja painikkeet).

Tietuesta tehtiin sitten ilmentymiä, joista sitten tallennettiin osoittimet taulukoihin, jotta toimintojen etsiminen painetun kytkimen perusteella olisi helppoa. Tämä tapa hallinnoida toimintoja mahdollistaa esim. samojen toimintojen asettamisen usealle näppäimelle. Lisäksi tämä toteutus mahdollistaa samanlaisten toimintomäärittelyiden käytön kaikkien fyysisten sisääntulojen, niin näppäinten kuin pulssiantureiden, toimintojen määrittelyyn. Toimintotaulukot ovat moniulotteisia taulukoita, jotka kertovat millä toimintokerroksella (ks. alla) ja fyysisellä sisääntulolla (näppäimellä tai pulssianturilla), toiminto sijaitsee. Alla esimerkiksi luodaan ja asetetaan toimintoja sekä irrallisille näppäimille sekä matriisissa sijaitseville näppäimille.

```
1  // makro jonka avulla komentojen määrittely on helpompaa
2  #define HIDActionValues (char[maxActionValues])
3  // luodaan toimintoja
4  const HIDAction act_key_a = {HIDAction::DEVICE_KEYBOARD, HIDActionValues{'a'} };
5  const HIDAction act_key_b = {HIDAction::DEVICE_KEYBOARD, HIDActionValues{'b'} };
6  const HIDAction act_key_ctrl_a = {HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_LEFT_CTRL, 'a'} };
7  // asetetaan toiminnot taulukkoon
8  const HIDAction *singleButtonActions[maxLayers][singleButtonsSize][maxActionsOnLayer] {
9      { // ensimmäisen komentokerroksen toiminnot
10         { &act_key_a } // ensimmäisen painikkeen komennot
11         { &act_key_b } // toisen painikkeen komennot
12     },
13     { // toisen komentokerroksen toiminnot
14         { &act_key_ctrl_a }
15         { }
16     }
17 };
18 // matriisin toiminnot
19 const HIDAction *matrixActions[maxLayers][mRowsSize][mColsSize][maxActionsOnLayer] {
20     { // ensimmäisen komentokerroksen toiminnot
21         { // matriisin 1. rivi
22             { act_key_a }, // matriisin 1. rivin 1. sarakkeen toiminnot
23             {  },
24             {  },
25             {  }
26         },
27         { // rivi 2
28             {  },
29             {  },
30             {  },
31             {  }
32         },
33         { // rivi 3
34             {  },
35             {  },
36             {  },
37             {  }
38         },
39         { // rivi 4
40             {  },
41             {  },
42             {  },
43             {  }
44         },
45     }, // sekä seuraavat komentokerrokset...
```

Koska tietokoneelle lähetettävät komennot ovat merkkijonotaulukoita, oli ne helpointa määrittää osoittimena ja luoda merkkijonotaulukot tietueen ilmentymien luonnin yhteydessä, tavalla joka tunnetaan englannin kielellä nimellä Compound Literal. Se on tuettu C99 standardissa ja GCC-kääntäjän laajennoksena se on tuettu myös C++ ja C89 moodeissa (Using the GNU Compiler Collection (GCC) 2005, luku 5.20). On myös otettava huomioon että C++ kielessä ”compound literal” -arvon elinikä kestää vain sen määrittelyn loppuun, mutta mikäli arvo on globaali muuttuja tai ”`const`” eli vakio tyyppinen, sillä on staattinen elinikä (Using the GNU Compiler Collection (GCC) 2005, luku 5.20). Koska ohjainlaitteen koodissa toiminnot on määritelty globaaleiksi vakioiksi, ei mitään ongelmia ole. Compound literal -taulukko määritellään esimerkiksi merkkijonotaulukon tapauksessa asettamalla alustuslistan eteen ”`(char[])`”. Koodin luettavuuden helpottamiseksi luotiin makro joka laajenee tallaiseksi. Esimerkiksi yllä olevassa koodissa rivillä 2 määritelty makro ”`HIDActionValues{'a'}`” laajenee muotoon ”`(char[maxActionValues]){’a’}`”.

Kuten edellä mainittiin, toimintoja on myös asetettu useampaan toimintokerrokseen. Toimintokerros yksinkertaisesti on toinen kerros toimintoja kullekin fyysiselle näppäimelle. Toimintokerros aktivoidaan painamalla määriteltyä näppäintä. Käytännössä voidaan määritellä näppäinyhdistelmiä, joita aktivoimalla ajetaan eri komento kuin vain pelkkää näppäintä yksinään painamalla. Tätä varten luotiin tietue joka määrää mitä fyysisiä painikkeita tulee olla painettuna jotta kukin toimintokerros aktivoidaan. Lisäksi taas kerran näistä ilmentymistä asetetaan osoittimet taulukkoon jotta toimintokerroksien käsittely olisi helppoa:

```
1  // millainen fyysinen sisääntulo on
2  typedef enum { INPUT_BUTTON, INPUT_MATRIX, INPUT_ENCODER } INPUT_TYPE;
3
4  #define Inputs (char[maxComboInputs])
5  #define InputTypes (INPUT_TYPE[maxComboInputs])
6
7  struct ComboInput {
8      const char *inputs; // lähetettävät komennot
9      const INPUT_TYPE *inputTypes; // missä painike sijatsee
10 };
11
12 // määritellään comboInput tietueita
13 const ComboInput noComboKey = { Inputs{  }, InputTypes{  } };
14 const ComboInput oneComboKey = { Inputs{ 33 }, InputTypes{ INPUT_MATRIX } };
15 const ComboInput twoComboKey = { Inputs{ 33, 32 }, InputTypes{ INPUT_MATRIX, INPUT_MATRIX } };
16
17 const ComboInput *comboLayers[maxLayers] {
18     &noComboKey,
19     &oneComboKey,
20     &twoComboKey
21 };
```

”ComboInput”-tietueessa on kaksi taulukkoa, ensimmäisessä määritellään näppäimet jotka aktivoivat kunkin toimintokerroksen, toisessa missä kukin näppäin sijaitsee (matriisissa vai irrallaan). Myös tässä käytetään samankaltaisia makroja avustamaan painikkeiden määrittelyä, kuin itse toimintojen määrittelyssä. Itse painikkeet ovat merkkejä, samoin kuin esimerkiksi näppäimistömatriisin määrittelyssä käytetty taulukko. Tällöin näiden arvojen vertaaminen on mahdollista suoraan.

Itse toimintojen lähettäminen toimii niin, että ensiksi skannataan näppäinyhdistelmät, ja niiden perusteella valitaan käytettävä toimintokerros. Sen jälkeen skannataan kaikki fyysiset sisääntulot (irralliset näppäimet, matriisi, pulssianturit) ja kultakin kytketyltä sisääntulolta aktivoidaan kaikki niille asetetut toiminnot. Toiminnon aktivoiminen toimii hieman eri lailla riippuen siitä että onko kyseessä pulssianturi vai painike. Pulssianturit vain lähettävät komennon ja sitten vapauttavat sen heti (koska pulssiantureita ei voi pitää pohjaan painettuna), toisin kuin näppäimet jotka vapauttavat toiminnon vasta kun näppäin itsessään vapautetaan. Lisäksi ohjelma sisältää toiminnallisuuden

joka näppäinyhdistelmässä olevan näppäimen vapautuessa vapauttaa kyseisen yhdistelmän aktivoiman kerroksen, sekä aktivoi mahdollisen seuraavan toimintokerroksen.

```
 1  static bool singleButtonsStateLast[singleButtonsSize];
 2  bool singleButtonsState[singleButtonsSize];
 3  for (int b = 0; b < singleButtonsSize; b++) {
 4      singleButtonsState[b] = digitalRead(singleButtons[b]->pin);
 5      // mikäli toimintokerros on vaihtunut, asetetaan toiminnot
 6      // kunkin kytkimen viime tilasta välittämättä
 7      // muulloin vain jos kytkimen tila on muuttunut
 8      if (layerChanged || singleButtonsState[b] != singleButtonsStateLast[b]) {
 9          if (singleButtonsState[b] == HIGH) { // kytkin on painettu
10              // tarkistetaan aktivoiko painettu kytkin tämän toimintokerroksen
11              if (isComboInputInLayer(singleButtons[b], layer)) {
12                  // jos niin, vapautetaan kaikki kerroksen alapuoliset kerrokset
13                  releaseOtherActionLayers(layer, false);
14                  hasChangedLayer = true;
15              }
16              // suoritetaan painetun painikkeen toiminnot
17              parseLayerActions(singleButtonActions[layer][b], layer);
18          }
19          else { // kytkin on vapautettu
20              // onko vapautettu kytkin määritelty aktivoimaan mitään toimintokerrosta
21              if (isComboInput(singleButtons[b])) {
22                  // jos niin, vapautetaan kaikki kerroksen yläpuoliset kerrokset
23                  releaseOtherActionLayers(layer, true);
24                  hasChangedLayer = true;
25              }
26              // vapautetaan painetun painikkeen toiminnot
27              parseLayerActions(singleButtonActions[layer][b], layer, true);
28          }
29      }
30      if (hasChangedLayer) {
31          layerChanged = true; // kerros on vaihtunut
32      }
33      else {
34          layerChanged = false;
35      }
36      singleButtonsStateLast[b] = singleButtonsState[b];
37  }
```

Funktiot jotka vapauttavat tai aktivoivat toimintokerroksia vain käyvät läpi kaikki kerrokset, ja riippuen viimeisen argumentin arvosta, tekevät sen vain annettua kerrosnumeroa ylemmille tai alemmille toimintokerroksille. Näiden funktioiden toteutus ja koko laitteen lopullinen ohjelmakoodi löytyy liitteestä 2.

4.2.4 Ohjelman lataaminen Arduino-laitteelle

Ohjelmakoodi käännettiin ja ladattiin laitteelle käyttäen arduino-cli ohjelmaa (luvussa 3.1.2 mainituista syistä). Ohjelman kääntäminen onnistuu komennolla ”`arduino-cli compile`”, kuitenkin komennolle täytyy antaa argumentti ”`--fqbn arduino:avr:micro`” millä määritellään kohdealusta, tässä tapauksessa Arduino Micro. Kohdealusta täytyy määritellä myös samoin ohjel-

maa ladattaessa laitteelle komennolla ”`arduino-cli upload`”. Lisäksi ”`upload`” komennolle täytyy antaa sen sarjaportin nimi, minkä tietokone on antanut Arduinolle. Tämä tapahtuu argumentilla ”`-p /dev/ttyACM0`”. Itse sarjaportin näkee esimerkiksi seuraavalla komennolla:

```
$ arduino-cli board list
Port         Protocol Type              Board Name    FQBN              Core
/dev/ttyACM0 serial   Serial Port (USB) Arduino Micro arduino:avr:micro arduino:avr
```

Ohjelma myös tulostaa tietoa sarjaporttiin, mikä auttaa kehityksessä. Sarjaporttia voi kuunnella arduino-cli:lla komennolla ”`arduino-cli monitor`”, jolle myös täytyy antaa sarjaportin nimi argumentilla ”`-p /dev/ttyACM0`”. Sarjaporttiin tulostaminen ei ole kuitenkaan ohjelman valmistuttua tarpeellista, joten se täytyy pystyä kytkemään pois päältä. Ohjelmakoodissa käytettiin omaa määritystä ”SERIAL_ENABLED”, joka täytyi asettaa ohjelman käännösvaiheessa, argumentilla ”`--build-property build.extra_flags="-DSERIAL_ENABLED=0"`”, jossa arvo 0 tarkoittaa että sarjaporttiin tulostus on pois päältä ja 1 että se on päällä. Itse koodissa tämä määritys toimii niin että ”`#if`”-koodiblokin sisällä oleva koodi käännetään ohjelmaan vain jos tämä ”`SERIAL_ENABLED`” määritys on arvossa 1:

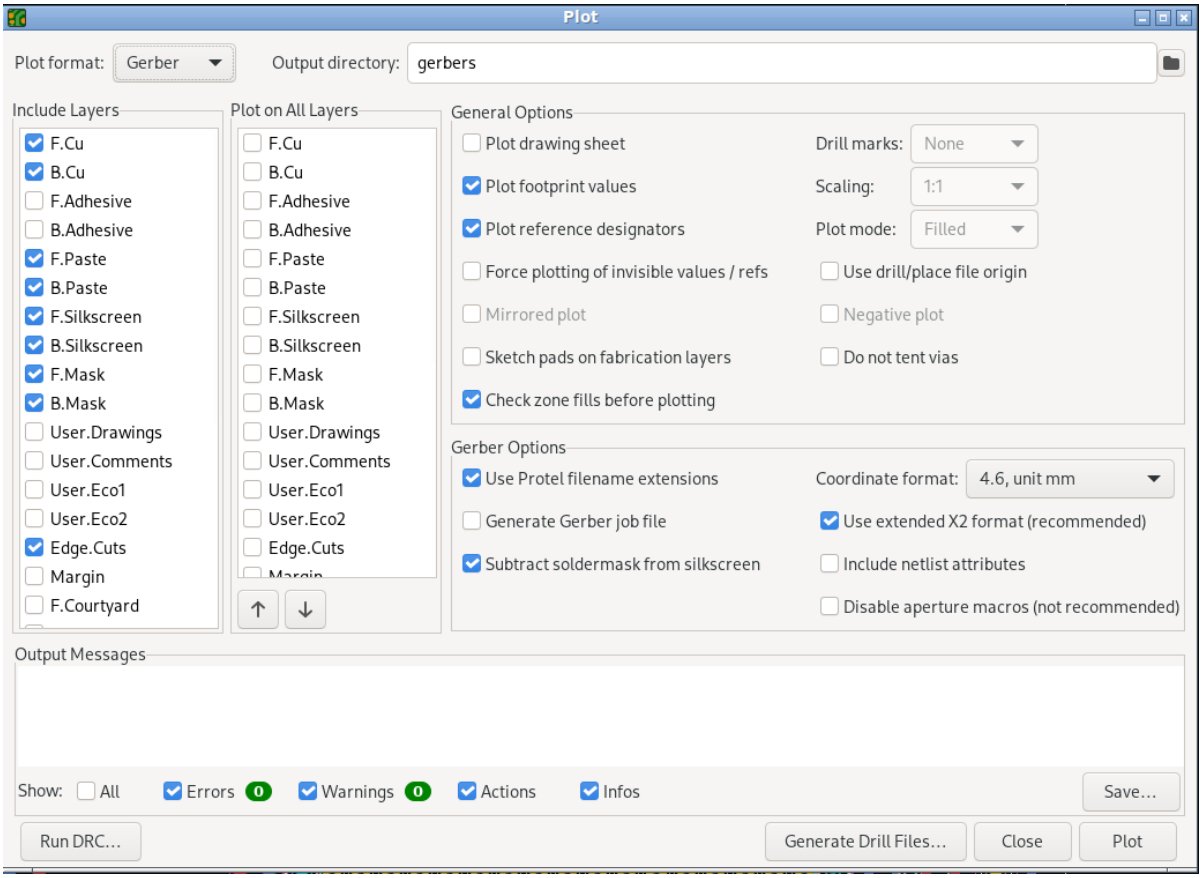
```
1  #if SERIAL_ENABLED
2  Serial.print("Tulostetaan sarjaporttiin!");
3  #endif
4  // tehdään loput ohjelmasta
```

4.3 Ohjainlaitteen fyysinen toteutus

4.3.1 Piirilevyn hankkiminen

Piirilevy hankittiin eräältä valmistajalta, jonka palvelusta voi tilata pienen erän piirilevyjä edullisesti prototyyppitarkoituksiin. Piirilevyjä tilattiin 5kpl ja levyt maksoivat kokonaisuudessaan (levyt, rahti ja ALV) noin 14€. Tämä hinta saavutettiin valitsemalla halvimmat materiaalit sekä postitustapa. Lisäksi pienentämällä piirilevy sopivan kokoiseksi (110x95mm), saatiin postikuluja vielä laskettua n. 10 eurosta n. 3,5 euroon. Toki halvin postitustapa tarkoitti, että levyjen saapumisessa kesti useita viikkoja.

Itse levyn suunnitelmien lähettäminen valmistajalle tapahtui viemällä ensin piirilevyn suunnitelmat KiCad-ohjelmasta Gerber ja Drill -tiedostoihin. Tämä tapahtui KiCad ohjelman piirilevyeditorin valikosta ”File -> Fabrication Outputs -> Gerbers”. Tästä avautuu kuvion 24 mukainen ikkuna, jossa valitaan että mitä suunnitelman kerroksia viedään Gerber tiedostoon ja millä asetuksilla. Nämä asetukset asetetaan piirilevyn valmistajan vaatimusten mukaan, jotka usein löytyvät valmistajan verkkosivuilta. Lisäksi tarvitaan Drill-tiedostot piirilevyn reikien porausta varten. Ne voi generoida Gerber-vienti ikkunan painikkeesta avautuvassa ikkunassa. Myös niiden vientiin liittyvät asetukset täytyy asettaa valmistajalle sopivaksi.

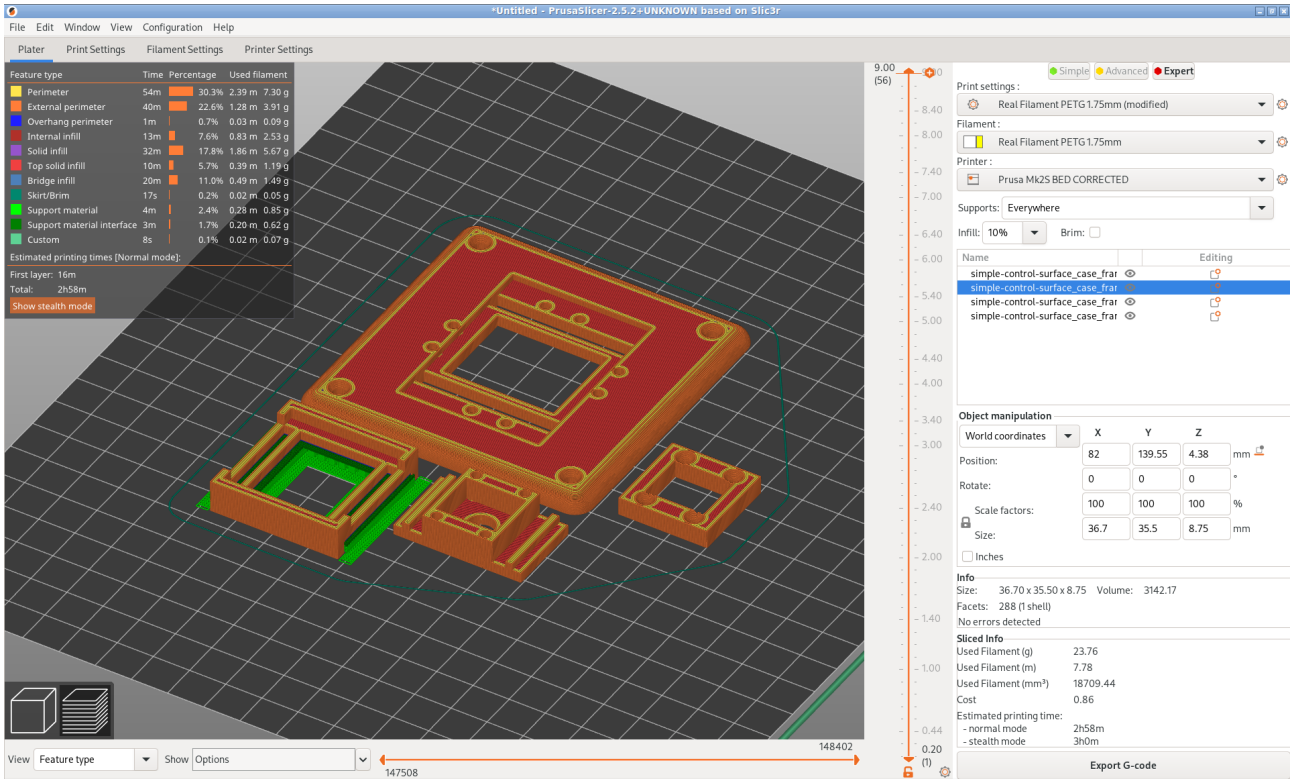


Kuvio 24: Gerber-tiedostojen vientiasetukset KiCad-ohjelmassa.

4.3.2 Laitekotelon 3D-tulostus

Jotta laitteen Blenderissä suunnitellun 3D-mallin voi tulostaa, se on vietävä ensin stl-tiedostoiksi, jotka voidaan sitten muuntaa eri sovelluksella tulostimen ymmärtämäksi G-koodiksi. Tämä tehtiin PrusaSlicer sovelluksella, joka on räätälöity erityisesti Prusan 3D-tulostimien kanssa käytettäväksi. Sovellusta voi toki käyttää muidenkin 3D-tulostimien kanssa, ja on olemassa myös muita vastaavia sovelluksia. PrusaSlicer sovellus on lisensoitu GPL lisenssillä ja pohjautuukin Slic3r-nimiseen sovellukseen (PrusaSlicer 2021).

Sovellukseen tuodaan halutut stl-muotoiset 3D-mallit ja sovellus siivuttaa osat kerroksiin halutuilla asetuksilla. Siivutus tehdään koska 3D-tulostin liikkuu XYZ koordinaatistossa G-koodin perusteella jonka ohjelma generoi siivutusvaiheessa. Siivutusohjelmalla voi myös esikatsella tulostimen tulostuspään reittiä sekä kerroksia joita se siten rakentaa, joten esikatselusta voi huomata mahdollisia virheitä tulostettavissa osissa. Siivutusohjelmalla voi myös luoda tukirakenteet sellaisten osien alle joita ei ole tuettu alapuolelta alkuperäisessä 3D-mallissa. Kuviossa 25 nämä näkyvät vihreällä. Ohjelma myös näyttää arvion tulostuksen kestosta sekä käytetystä filamentin määrästä.



Kuvio 25: Laitteen osia siivutettuna PrusaSlicer-ohjelmassa.

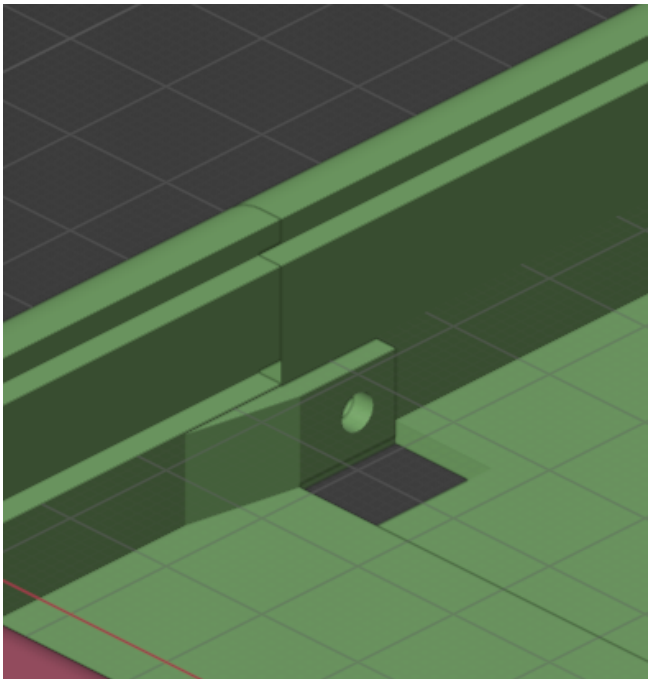
Itse tulostus onnistui helposti, tämä toki riippuu käytetystä 3D-tulostimesta. Tässä tulostamiseen käytettiin Prusa i3 MK2 tulostinta, joka on FDM-tyyppinen 3D-tulostin. Kotelon materiaalina käytettiin PETG muovia, sillä se on kestävämpää kuin PLA (esimerkiksi PLA-muovin sulamislämpötila on n. 60°C, kun taas PETG alkaa pehmenemään vasta n. 85°C), mutta helpommin tulostettavaa kuin esimerkiksi ABS (Ryder N.d). Tulostus aloitettiin monimutkaisimmista osista, jotta olisi aikaa muuttaa niiden 3D-malleja, mikäli tulostamisen jälkeen huomattaisiin mitään ongelmia. Joitain osia täytyikin hieman muokata ja tulostaa uudelleen (ks. luku 4.4.1). Tulostamisessa oli myös tärkeää asetella kappaleet oikeaan asentoon, jotta kappaleiden tulostus onnistuu mahdollisimman vähillä tukirakenteilla. Tämä onnistuikin hyvin, ainoastaan päärullan mekanismin keskimmäisen levyn tulostamiseen vaadittiin tukirakenteet.

4.4 Laitteen kokoaminen & testaus

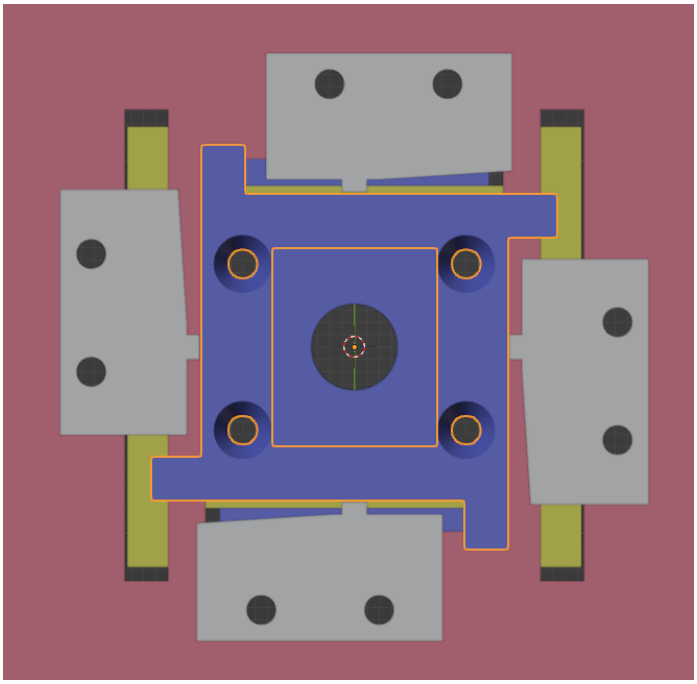
4.4.1 Kokoamisprosessi

Mekanismin osia täytyi hieman käsitellä tulostuksen jälkeen käsin. Esimerkiksi liukuvien kappaleiden reunoista täytyi veistellä pois ylimääräisiä reunuksia. Lopuista kappaleista ei löytynyt suurempia ongelmia tulostuksen jälkeen. Kuitenkin esimerkiksi ylärivin pulssiantureiden päälle tulevan kappaleen reiät olivat liian pieniä, joten niitä porattiin suuremmiksi jälkikäteen, koska se oli nopeampaa kuin kappaleen uudelleen tulostaminen. Itse 3D-malliin tämä puute korjattiin. Lisäksi itse koteloa täytyi pidentää keskikohdastaan 5mm, koska muulloin laitekotelossa oleva ruuvi osui Arduino korttiin. Kotelon alaosiin myös lisättiin ulokkeet ja ruuvinreiät kotelon kiinnittämiseksi toisiinsa (Kuvio 27). Tämän prototyyppilaitteen tapauksessa ei kuitenkaan alapuolen piirilevyn puoleista osaa tulostettu uudestaan. Sen sijaan jo tulostettuun osaan porattiin reiät ja kotelon puoliskot kiinnitettiin toisiinsa pienellä levyllä ja ruuveilla.

Myös päärullan mekanismi muuttui tulostuksen jälkeen. Esimerkiksi keskimmäisen kappaleen liukupinnat olivat liian paksut ja eivät sopineet alemman kappaleen kiskoihin, joten kappaletta täytyi muokata ja sitten tulostaa uudelleen. Myös mekanismin pohjakappaleen muotoa täytyi muuttaa, jotta mekanismi toimii luotettavammin. Siihen lisättiin siivekkeet kuhunkin kulmaan, jotta kappale ei pääse putoamaan keskellä olevaan reikään. (Kuvio 26).

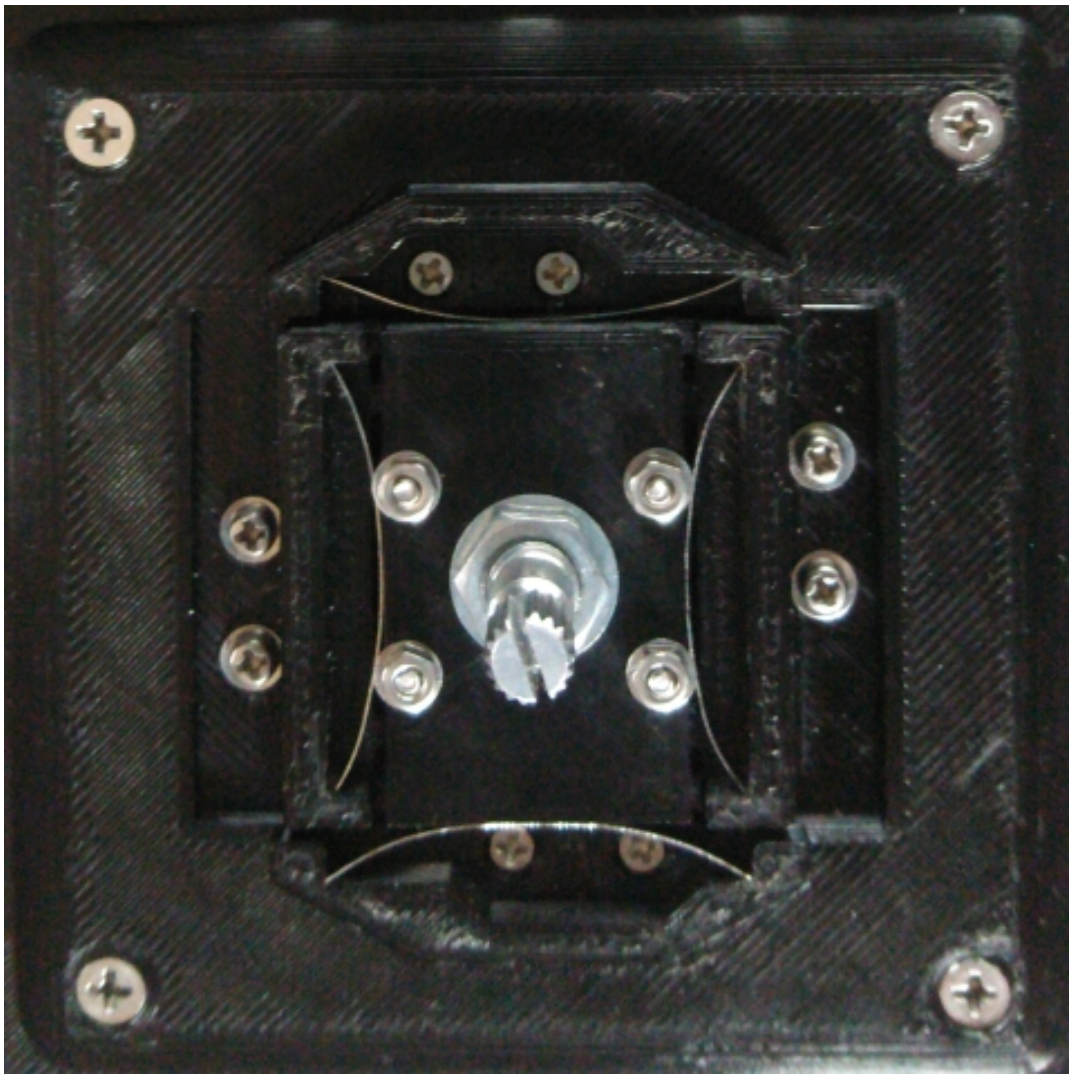


Kuvio 27: Laitekotelon puoliskojen kiinnittäminen toisiinsa.



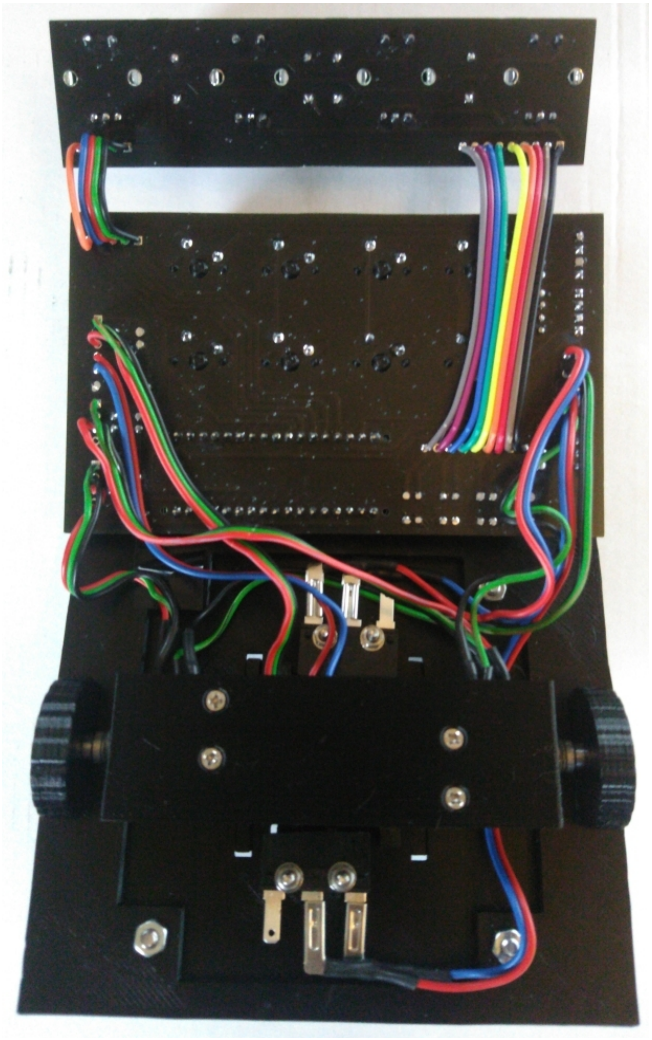
Kuvio 26: Päärullan mekanismin pohjimmaisen kappaleen uusi muoto.

Lisäksi mekanismi vaati ylimääräiset palautusjouset, koska pelkät mikrokytkimien sisäiset jouset eivät olleet tarpeeksi vahvat palauttamaan mekanismia keskiasentoon. Tätä varten lisättiin mekanismin pohjalevyyn paikat jousille. Itse jouset tehtiin ohuesta joustavasta metallista. (Kuvio 28) Toimivan kokoisten jousien tekeminen vaati kokeilua, joten sopivan kokoiset valmiit jouset olisivat parempi vaihtoehto, mutta mekanismi näyttäisi toimivan kohtuullisesti tällaisenaankin.

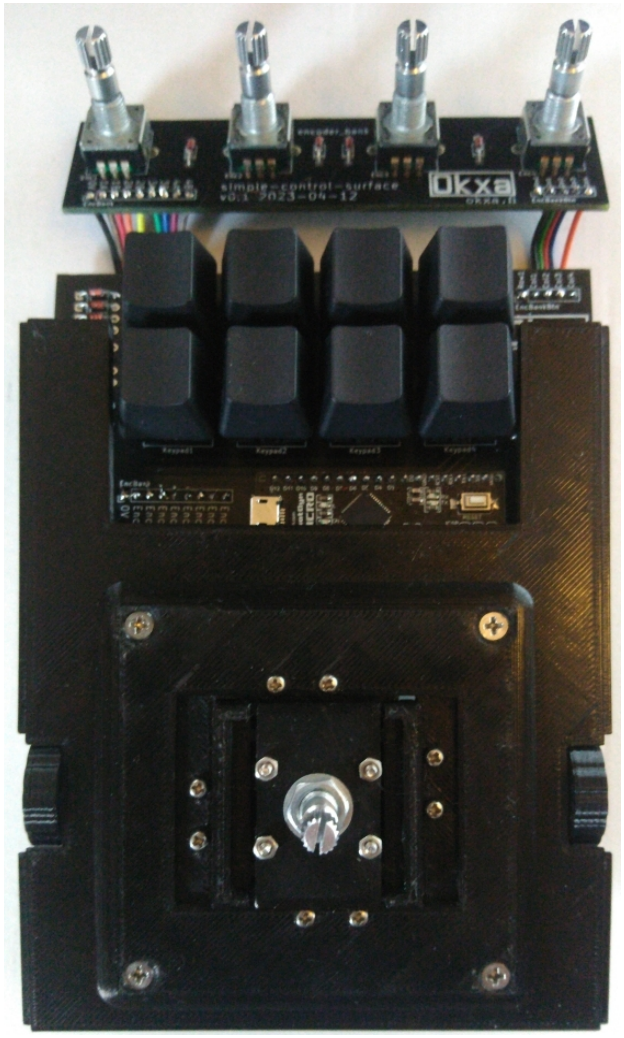


Kuvio 28: Päärullan jousimekanismi.

Laitteen elektroniikan kokoaminen onnistui sujuvasti. Piirilevyn sahaaminen kahteen osaan onnistui tavallisen rautasahan terällä. Molempia piirilevyjä sekä irrallisia pulssiantureita ja kytkimiä yhdistävät johdot kytkettiin piirilevyn alapuolelta (Kuvio 29), koska kotelossa oli sillä puolella enemmän tilaa. Muut komponentit kuin itse kytkimet ja pulssianturit kiinnitettiin levyn yläpuolelle (Kuvio 30). Irralliset komponentit ovat kiinni itse kotelossa, tai sen päällislevyssä. Tämä levy, sekä piirilevyt, liu'utetaan kotelon ala- ja yläkappaleissa oleviin kiskoihin.



Kuvio 29: Ohjainlaitteen elektroniikat koottuna (Alapuoli)



Kuvio 30: Ohjainlaitteen elektroniikat koottuna (Yläpuoli)

Kootessa huomattiin joitain ongelmia myös piirilevyn suunnitelmassa. Isoimpana se että pulssiantureiden kiinnitysjalkojen reiät piirilevyssä huomattiin olevan liian pienet. Tämä johtui pienestä sekaannuksesta koska työssä käytettiin PEC11R sarjan pulssiantureita ja KiCadin komponenttikirjastossa oli vain PEC12R pulssianturin jalanjälki. Nämä komponentit ovat melkein samanlaisia, ja nopealla vilkaisulla nämä olisivat käyneet ristiin, sillä molempien pystymallisten antureiden kiinnitysreiät olivat tarpeeksi lähellä kooltaan. Kuitenkaan anturit eivät sopineet paikalleen piirilevyjen saavuttua. Kävi ilmi että KiCadin kirjastossa määriteltiin pystymallisen jalkojen kooksi 2,0x2,1mm. Tämä johtui siitä että PEC12R sarjassa on kaksi erilaista pystymallin pulssianturia. PEC12R määritelmädokumentissa kerrotaan että mallille ”PEC12R-3xxxF-xxxxx” reikien koot ovat 2,0x2,6mm, kun taas mallille ”PEC12R-4xxxF-xxxxx” reikien koot ovat 2,0x2,1mm (PEC12R Series – 12mm Incremental Encoder 2021). PEC11R sarjan pystymalliselle anturille reiät ovat kooltaan 1,8x2,6mm (PEC11R Series – 12mm Incremental Encoder 2023.)

Mikäli siis KiCadin kirjastossa anturin jalanjälki olisi mallinnettu mallin ”PEC12R-3xxxx-xxxxx” mukaan, olisi käytetty PEC11R sarjan pulssianturi sopineet paikalleen suoraan. Tässä prototyypissä antureiden asennus onnistui kun porattiin kiinnitysjalkojen reikiä isommaksi, mutta itse piirilevyn suunnitelmaan reiät korjattiin. Tällaisten ongelmien välttämiseksi kannattaa tarkistaa ainakin tärkeimpien komponenttien mitat KiCadissa. Piirilevyssä ei ollut muuta suoranaista vikaa, mutta kokoamisen helpottamiseksi esimerkiksi joitain johtojen paikkoja kannattaisi vaihtaa piirilevyn suunnitelmassa, jotta johtoja ei tarvitsisi kytkeä ristiin. Esimerkiksi irrallisille pulssiantureille menevien johtojen paikat kannattaisi järjestää uudestaan vastaamaan pulssiantureiden pinniasettelua, jossa maa on keskellä. Lopulta laite saatiin koottua lopulliseen muotoonsa (Kuvio 31), ja voitiin aloittaa sen testaus.

Kuvio 31: Ohjainlaite koottuna.

4.4.2 Lopputuloksen testaus

Laitteen käyttöä testattiin REAPER-musiikintuotantosovelluksessa, jossa lähes kaikille ohjelman toiminnoille voi asettaa sen aktivoivan näppäinyhdistelmän. Sovellus siis sopii hyvin tämän laitteen kanssa käytettäväksi. Ensimmäiseksi kokeiltiin asettaa laitteen päärullan sivuilla oleviin pulssiantureihin komennot jotka suurentaisivat ja pienentäisivät näkymää. REAPER-ohjelmistossa tämä on asetettu yhdistelmään ctrl-näppäin + hiiren rulla, joten ohjainlaitteen ohjelmistoon luotiin toiminto joka painaa ctrl-näppäintä ja siirtää hiiren rullaa ylös tai alas, riippuen pulssianturin pyörityssuunnasta. Lisäksi ohjainlaitteen näppäimiin asetettiin komentoja, jotka ohjaavat esimerkiksi toiston tilaa, tai ajavat komentoja REAPERissa jotka käsittelevät ääntä jollain tapaa.

Testaillessa laitetta, ohjelmakoodissa huomattiin vielä pari ongelmaa nimenomaan pulssianturei-den kanssa, koska niiden lähettämät komennot vapautetaan heti kun ne on lähetettykin. Ensimmäinen oli se että hiiren liikutuskomennot (eli X, Y akselit ja rullan liike) lähettiin myös kun toiminto vapautettiin. Tämä ratkaistiin yksinkertaisella ehtolausekkeella joka lähettää hiiren liikkumiskomennot vain jos kyseessä ei ole toiminnon vapautus. Toinen ongelma oli että REAPER-sovellus ei näyttänyt huomaavan että ctrl-näppäin oli samanaikaisesti painettuna kun hiiren rullaa liikutettiin. Esimerkiksi verkkoselaimessa tämä komento toimi oikein, ja verkkosivu skaalautui kuten normaalistikin ctrl+rullaus yhdistelmällä. Tämä ratkaistiin asettamalla ”`delay`”-funktiolla 50ms viive ennen kuin pulssianturin toiminto vapautettiin sen lähettämisen jälkeen. Näiden muokkausten jälkeen laitteen lähettämät näppäinkomennot toimivat oikein myös REAPERissa.

Laite toimi hyvin yhden sovelluksen kanssa testattuna. Toiminnot kuitenkin täytyy asettaa sovelluskohtaisesti, varsinkin jos sovelluksessa ei voi vaihtaa käytettyjä pikanäppäimiä. Asettamalla toiminnot yleisesti käytettyihin näppäimiin (kuten välilyönti, home, end, page up/down) laitteella pystyy ohjaamaan useampia sovelluksia jonkin verran (käyttäen samaa konfiguraatiota), mutta kaikki sovellukset eivät hyödynnä näitäkään näppäimiä. Loppujen lopuksi laite kuitenkin näyttää soveltuvan tarkoitukseensa. Komentoja voisi säätää loputtomasti, mutta tällä lyhyellä testillä laitteen käyttö oli toimivaa, ja lisää tarvittavia komentoja voidaan asettaa helposti tarpeen vaatiessa.

5 Pohdinta

Tavoitteena oli kehittää tietokoneen ohjainlaitteen prototyyppi käyttäen avoimen lähdekoodin työkaluja, ja niin tehtiinkin. Itse laite ja sen suunnitelmat ovat käyttökelpoisia, mutta laitetta voisi myös parannella niin fyysisesti kuin ohjelmistonkin osalta. Esimerkiksi päärullan alla olevan neli-suuntaisen ohjaimen huomattiin olevan ongelmallinen valmistaa 3D-tulostamalla. Harrastelijataso-son 3D-tulostimen tarkkuus ei riitä kovin hyvin tällaisen tarkkuutta vaativan mekanismin valmista-miseen, joten mekanismin toteuttaminen jollain eri tavalla kuin pelkästään 3D-tulostamalla voisi olla parempi vaihtoehto. Lisäksi mekanismia ei välttämättä pysty käyttämään kovin hyvin hyödyksi työskennellessä, koska sen päällä on pyörivä rulla. Toki laitteen modulaarisuuden takia laitetta on helppo muuttaa. Jos mekanismin poistaisi laitteesta, sen osana olevat mikrokytkimet olisi mahdol-lista korvata jollain muilla kytkimillä, kuten esimerkiksi näppäimillä jotka aktivoisivat toimintoker-roksia. Muuten laitteeseen valitut ohjaintavat toimivat hyvin. Painokytkimet sopivat hyvin tavallis-ten näppäinkomentojen lähettämiseen, ja ylärivin useat pulssianturit olivat hyödyllisiä usean ko-mennon yhtäaikaiseen käyttöön. Päärullan pulssianturi voisi olla tarkempi, kuin siinä käytetty puls-sianturi, joka on samanlainen kuin kaikki muutkin laitteen pulssianturit. Nyt päärullassa on välystä, joten rullaa voi pyörittää pari astetta mutta mitään ei tapahdu. Kuitenkin rulla toimii suhteellisen hyvin.

Laitteen ohjelmakoodi toimii hyvin, niissä puitteissa mihin se on suunniteltu. Näppäin- ja hiiriko-mentojen määrittäminen on helppoa. Laite on monikäyttöinen koska komennot ovat yksinkertai-sesti näppäimistön ja hiiren painalluksia ja liikkeitä, joten esimerkiksi hiiren rullaksi asetettu ko-mento toimii vaikkapa internet selaimessa. Kuitenkin laitteen käyttö on jonkin verran rajautunutta yhteen sovellukseen kerrallaan, koska näppäinkomennot ovat usein sovelluskohtaisia. Yksi vaihto-ehto olisi lisätä laitteeseen ominaisuus, joka antaisi valita usean eri komentokonfiguraation välillä, mutta toki täytyy pitää mielessä Arduino mikrokontrollerin rajallinen muistin määrä. Lisäksi joitain sovelluksia ei voi ohjata hyvin pelkillä näppäimistön komennoilla, joten esimerkiksi MIDI-komento-jen lähetys saattaisi olla hyvä ominaisuus tulevaisuudessa. Tämä onnistuisi luvussa 3.2.2 käsitellyil-lä kirjastoilla. Myös muita kirjastoja olisi hyvä harkita, esimerkiksi HID-kirjastoja, joissa on enem-män ominaisuuksia ja jotka tukevat enemmän laitteita kuin vain näppäimistöä ja hiirtä.

Työn suunnittelu käytetyillä ohjelmistoilla oli sujuvaa. Etenkin KiCad on todella pätevä ohjelmisto piirilevyjen suunnitteluun. Kuitenkin KiCadia käyttäessä on syytä tarkastaa pitävätkö mitat ja muut tiedot paikkansa, ainakin tärkeimpien komponenttien osalta, sillä komponenttikirjastot eivät välttämättä ole täysin oikein määriteltyjä (ks. luku 4.4.1). Avoimen lähdekoodin CAD- ja 3D-suunnitteluohjelmistojen osalta tilanne on hieman huonompi. Kuten luvussa 3.4.1 mainittiin, vaikka FreeCAD onkin suhteellisen ominaisuusrikas ohjelmisto, se ei ole kovin helppokäyttöinen. Muita avoimia vaihtoehtoja CAD-ohjelmille ei juuri ole, ainakaan yhtä hyvin varusteltuja kuin jotkin kaupalliset sovellukset. Työssä käytetty Blender-ohjelma toimii hyvin 3D-tulostukseen tarkoittavien mallien luomiseen, mutta sopii muuhun niin sanottuun oikeaan CAD-suunnitteluun huonommin, koska Blender on ensisijaisesti 3D-mallinnus- ja animaatiosovellus. Asennetut lisäosat (ks. luku 3.4.5) helpottivat mallinnustyötä paljon, koska niillä sai esimerkiksi mitattua kappaleiden tarkat mitat.

Arduinon kehitysympäristö ja laaja tuki helpotti laitteen kehittämistä huomattavasti. Ohjelmistokirjastot helpottivat laitteen kehitystä. Esimerkiksi ei tarvinnut tutkia ja toteuttaa miten HID-laitteet lähettävät komentoja tietokoneelle, vaan sen sijaan voitiin vain käyttää valmiita kirjastoja. Ohjelmakirjastoja on kuitenkin todella paljon, ja jotkin ovat toteutettu paremmin kuin toiset, joten kannattaa tutustua eri vaihtoehtoihin. On hyvä myös pitää mielessä että tarvitseeko jotain ohjelmakirjastoa lainkaan. Esimerkiksi tässäkin laitteessa matriisin lukemiseen käytetty kirjasto voitaisiin todennäköisesti korvata helposti itse kirjoitetulla koodilla. Arduinon dokumentaatiot ovat pääsääntöisesti hyvin toteutettuja ja laitteille löytyi myös useita esimerkkejä erilaisista ohjelmista, joten apua toteutukselle oli saatavilla.

Kaiken kaikkiaan, tällaisen ohjainlaitteen prototyypin toteutus vain avoimen lähdekoodin ohjelmia ja työkaluja käyttäen on mahdollista ja suhteellisen helppoa. Arduino, KiCad ja Blender toimivat hyvin tämänkaltaisen laitteen suunnitteluun. Tietysti kaivattaisiin helpommin käytettäviä avoimen lähdekoodin CAD-ohjelmistoja. Itse ohjainlaite on suhteellisen käyttökelpoinen sellaisenaan, mutta myös jatkokehityskohteita löydettiin, niin ohjelmistosta kuin ulkomuodosta. Laitteen suunnitelmat ja ohjelmakoodi myös jaettiin vapaalla lisenssillä, jotta opinnäytetyön tuloksista olisi hyötyä mahdollisimman monelle.

Lähteet

0xPIT. 2014. ClickEncoder. Viitattu 15.3.2023. https://github.com/0xPIT/encoder/blob/master/README.md

3D Print Toolbox – Blender Manual. 2023. Blender-ohjelmiston käyttöohje. Blender Foundation. Viitattu 1.3.2023. https://docs.blender.org/manual/en/latest/addons/mesh/3d_print_toolbox.html

About – blender.org. N.d. Blender projektin kotisivut. Viitattu 1.3.2023. https://www.blender.org/about/

About Arduino. 2021. Verkkosivu Arduinon sivuilla. Viitattu 10.2.2023. https://www.arduino.cc/en/about

About KiCad – KiCad EDA. N.d. KiCad-projektin kotisivu. Viitattu 4.3.2023. https://www.kicad.org/about/kicad/

Announcing the Arduino IDE 2.0 (beta). 2021. Viitattu 10.2.2023. https://blog.arduino.cc/2021/03/01/announcing-the-arduino-ide-2-0-beta/

Arduino CLI. 2022. Arduinon dokumentaatio. Viitattu 10.2.2023. https://arduino.github.io/arduino-cli/0.30/

Arduino Hardware. 2022. Kategoriasivu Arduinon sivuilla. Viitattu 13.2.2023. https://www.arduino.cc/en/hardware

Arduino Leonardo – Arduino Official Store. N.d. Tuotesivu Arduinon verkkokaupassa. Viitattu 14.2.2023 https://store.arduino.cc/products/arduino-leonardo-with-headers

Arduino Micro – Arduino Official Store. N.d. Tuotesivu Arduinon verkkokaupassa. Viitattu 14.2.2023 https://store.arduino.cc/products/arduino-micro

Arduino Micro Pinout Diagram. N.d. Kuva Arduinon verkkokaupassa. Viitattu 15.3.2023. https://content.arduino.cc/assets/Pinout-Micro_latest.png

Arduino Nano – Arduino Official Store. N.d. Tuotesivu Arduinon verkkokaupassa. Viitattu 14.2.2023 https://store.arduino.cc/products/arduino-nano

Arduino Nano 33 BLE – Arduino Official Store. N.d. Tuotesivu Arduinon verkkokaupassa. Viitattu 14.2.2023. https://store.arduino.cc/products/arduino-nano-33-ble

Bell, R & Hansell, M. 2018. Rotary Encoders. DIYODE Magazine. Viitattu 23.2.2023. https://diyode-mag.com/education/what_the_tech_rotary_encoders

Constants. N.d. cplusplus.com. Viitattu 19.4.2023. https://cplusplus.com/doc/tutorial/constants/

Control Surface. 2023. Control Surface projektin dokumentaatio. Viitattu 10.2.2023. https://ttta-pa.github.io/Control-Surface-doc/Doxygen/index.html

Dalmaris, P. 2022. Arduino IDE 2.0 is here. Tech Explorations. Viitattu 10.2.2023. https://techexplo-rations.com/blog/arduino/arduino-ide-2-0/

Developer Page – Cherry MX. N.d. Cherry AG. Viitattu 15.3.2023. https://www.cherrymx.de/en/dev.html

Device Class Definition for HID 1.11. 2001. USB Implementers Forum, Inc. Viitattu 14.2.2023. https://www.usb.org/document-library/device-class-definition-hid-111

Dribin, D. 2000. Keyboard Matrix Help. dribing.org. Viitattu 1.3.2023. https://www.dribin.org/dave/keyboard/one_html/

Epstein, M. 2020. Razer Tartartus Pro Review. PCMag. Viitattu 20.2.2023. https://www.pcmag.com/reviews/razer-tartarus-pro

FaqTotum. 2022. Orbion The OpenSource 3D Space Mouse. Viitattu 14.2.2023. https://git-hub.com/FaqT0tum/Orbion_3D_Space_Mouse/blob/main/README.md

Fisher, J. 2020a. Monogram Creative Console Studio Review, PCMag. Viitattu 20.2.2023. https://www.pcmag.com/reviews/monogram-creative-console-studio

Fisher, J. 2020b. Loupedeck CT Review, PCMag. Viitattu 20.2.2023. https://www.pcmag.com/re-views/loupedeck-ct

Frey, S & Gherke, F. 2023. The Best Resin 3D Printers in 2023. All3DP. Viitattu 1.3.2023. https://all3dp.com/1/best-resin-dlp-sla-3d-printer-kit-stereolithography/

Gammon N. 2018a.Keypad Matrix library. Viitattu 19.4.2023. https://github.com/nickgammon/Keypad_Matrix

Gammon N. 2018b. Keypad_Matrix.h. Keypad Matrix library. Viitattu 19.4.2023. https://github.com/nickgammon/Keypad_Matrix/blob/master/Keypad_Matrix.h

Getting Started in KiCad – KiCad Documentation. N.d. KiCad-projektin dokumentaatio. Viitattu 4.3.2023. https://docs.kicad.org/6.0/en/getting_started_in_kicad/getting_started_in_kicad.html

Gibson, J. N.d. Introduction to the MIDI Standard. Indiana University Bloomington. Viitattu 13.2.2023. https://cecm.indiana.edu/361/midi.html

Heironimus, M. 2022. Arduino Joystick Library. Viitattu 10.2.2023. https://github.com/MHeironimus/ArduinoJoystickLibrary/blob/master/README.md

Hewes, J. N.d. Circuit Diagrams. Electronics Club. Viitattu 4.3.2023. https://electronicsclub.info/circuitdiagrams.htm

IEC 60617 - Graphical Symbols for Diagrams. N.d. IEC 60617 database, International Electrotechnical Commission. https://std.iec.ch/iec60617

Junk, S & Kuen, C. 2016. Review of Open Source and Freeware CAD Systems for Use with 3D-Printing. University of Applied Sciences Offenburg. Elsevier B.V. Viitattu 1.3.2023. https://doi.org/10.1016/j.procir.2016.04.174

Keim, R. 2020. What Is a Printed Circuit Board (PCB)?. All About Circuits. Viitattu 4.3.2023. https://www.allaboutcircuits.com/technical-articles/what-is-a-printed-circuit-board-pcb/

Keyboard. 2022. Arduinon dokumentaatio. Viitattu 10.2.2023. https://www.arduino.cc/reference/en/libraries/keyboard/

Keyboard Modifiers and Special Keys. 2022. Arduinon dokumentaatio. Viitattu 26.4.2023. https://www.arduino.cc/reference/en/language/functions/usb/keyboard/keyboardmodifiers/

Khaze, V. 2014. Open Transport. Viitattu 14.2.2023. https://victorkhaze.com/open-transport

Koretić, P. 2020. Quick look: Electron vs Qt/QML app memory usage. Medium.com. Viitattu 14.2.2023. https://pkoretic.medium.com/quick-look-electron-vs-qt-qml-app-memory-usage-e8769008534f

Kotimaisten kielten keskus ja Kielikone Oy. 2022a. Piirilevy. Kielitoimiston Sanakirja. Viitattu 4.3.2023.https://www.kielitoimistonsanakirja.fi/#/piirilevy

Kotimaisten kielten keskus ja Kielikone Oy. 2022b. Kytkentäkaavio. Kielitoimiston Sanakirja. Viitattu 4.3.2023. https://www.kielitoimistonsanakirja.fi/#/kytkent%C3%A4kaavio

Kysymyksiä ja vastauksia 3D-tulostamisesta. 2018. Turvallisuus- ja kemikaaliviraston (Tukes) julkaisu sivulla https://tukes.fi/3d-tulostus. Viitattu 1.3.2023. https://tukes.fi/documents/5470659/8579343/Kysymyksi%C3%A4+ja+vastauksia+3D-tulostamisesta

Larabel, M. 2022. OpenRazer 3.5 Brings Support For Newer Razer Devices On Linux. Phoronix. Viitattu 20.2.2023. https://www.phoronix.com/news/OpenRazer-3.5-Released

Libraries License – KiCad EDA. N.d. KiCad-projektin kotisivu. Viitattu 4.3.2023. https://www.kicad.org/libraries/license/

Libraries. 2022. Arduinon dokumentaatio. Viitattu 10.2.2023. https://www.arduino.cc/reference/en/libraries/

Licensing for products based on Arduino. 2023. Tukiartikkeli arduinon verkkosivuilla. Viitattu 10.2.2023 https://support.arduino.cc/hc/en-us/articles/4415094490770-Licensing-for-products-based-on-Arduino

Loupedeck CT. N.d. Tuotesivu Loupedeck-yhtiön verkkokaupassa. Viitattu 26.4.2023. https://loupedeck.com/products/loupedeck-ct/

Measure It – Blender Manual. 2023. Blender-ohjelmiston käyttöohje. Blender Foundation. Viitattu 1.3.2023. https://docs.blender.org/manual/en/latest/addons/3d_view/measureit.html

Meyer A. 2012. Rotary Encoder + Arduino. Adam Meyerin kotisivut. Julkaistu alunperin bildr.org verkkosivustolla (Arkistoitu 30.1.2022 osoitteessa http://web.archive.org/web/20220130040958/https://bildr.org/2012/08/rotary-encoder-arduino/). Viitattu 19.4.2023. http://adam-meyer.com/arduino/Rotary_Encoder

MIDIUSB. 2022. Arduinon dokumentaatio. Viitattu 10.2.2023. https://www.arduino.cc/reference/en/libraries/midiusb/

Monogram Shop. N.d. Monogram-yhtiön verkkokauppa. Viitattu 26.4.2023. https://monogramcc.com/shop/

Mouse. 2022. Arduinon dokumentaatio. Viitattu 10.2.2023. https://www.arduino.cc/reference/en/libraries/mouse/

Mouse – Mouse.move(). 2022. Arduinon Mouse kirjaston dokumentaatio. Viitattu 27.4.2023. https://www.arduino.cc/reference/en/libraries/mouse/mouse.move/

Murray, M. 2019. How Rotary Encoders Work – Electronics Basics. The Geek Pub. Viitattu 23.2.2023. https://www.thegeekpub.com/245407/how-rotary-encoders-work-electronics-basics/

NicoHood. 2021. Arduino HID Project. Viitattu 10.2.2023. https://github.com/NicoHood/HID/blob/master/Readme.md

OpenSCAD - About. N.d. Openscad projektin verkkosivuilla. Viitattu 25.4.2023. https://openscad.org/about.html

Overview of the Arduino IDE 1. 2023. Viitattu 10.2.2023. https://docs.arduino.cc/software/ide-v1/tutorials/Environment

Parks Yong N. 2023. KiCad Symbol & Footprint Library for Arduino Modules. Viitattu 19.4.2023. https://github.com/Alarm-Siren/arduino-kicad-library/blob/master/README.md

PEC11R Series – 12mm Incremental Encoder. 2023. Bourns. Viitattu 6.5.2023. https://www.bourns.com/docs/product-datasheets/pec11r.pdf

PEC12R Series – 12mm Incremental Encoder. 2021. Bourns. Viitattu 6.5.2023. https://www.bourns.com/docs/product-datasheets/pec12r.pdf

PrusaSlicer. 2021. Viitattu 24.4.2023. https://github.com/prusa3d/PrusaSlicer/blob/master/README.md

Quadrature Rotary Encoder. N.d. .NET nanoFramework projektin dokumentaatio. Viitattu 1.3.2023. https://docs.nanoframework.net/devicesdetails/RotaryEncoder/README.html

Razer Tartarus Pro. N.d. Tuotesivu Razer-yhtiön verkkokaupassa. Viitattu 26.4.2023. https://www.razer.com/eu-en/gaming-keypads/Razer-Tartarus-Pro/RZ07-03110100-R3M1

Rowntree, D. 2022. A DIY CAD Mouse You Can Actually Build. Hackaday. Viitattu 14.2.2023. https://hackaday.com/2022/01/18/a-diy-cad-mouse-you-can-actually-build/

Ryder B. N.d. The 3D Printing Holy Trinity: PLA, ABS, and PETG. Slice Engineering. Viitattu 24.4.2023. https://www.sliceengineering.com/blogs/news/the-3d-printing-holy-trinity

Schallbert. 2021. Clickencoder projektin lisenssidokumentti, LICENCE. Viitattu 19.4.2023. https://github.com/Schallbert/encoder/blob/master/LICENSE

Schallbert. 2023. ClickEncoder. Schallbert:in kotisivu, schallbert.de. Viitattu 15.3.2023. https://schallbert.de/projects-software/encoder/

Schematic - Micro ATmega32u4-MU. N.d. Tuotteen dokumentaatio RobotDyn verkkokaupassa robotdyn.com. Sivu arkistoitu 7.9.2021. Viitattu 13.3.2023. http://web.archive.org/web/20211107040343/http://robotdyn.com/pub/media/0G-00005086==Micro-ATmega32U4MU/DOCS/Schematic==0G-00005086==Micro-ATmega32U4MU.pdf

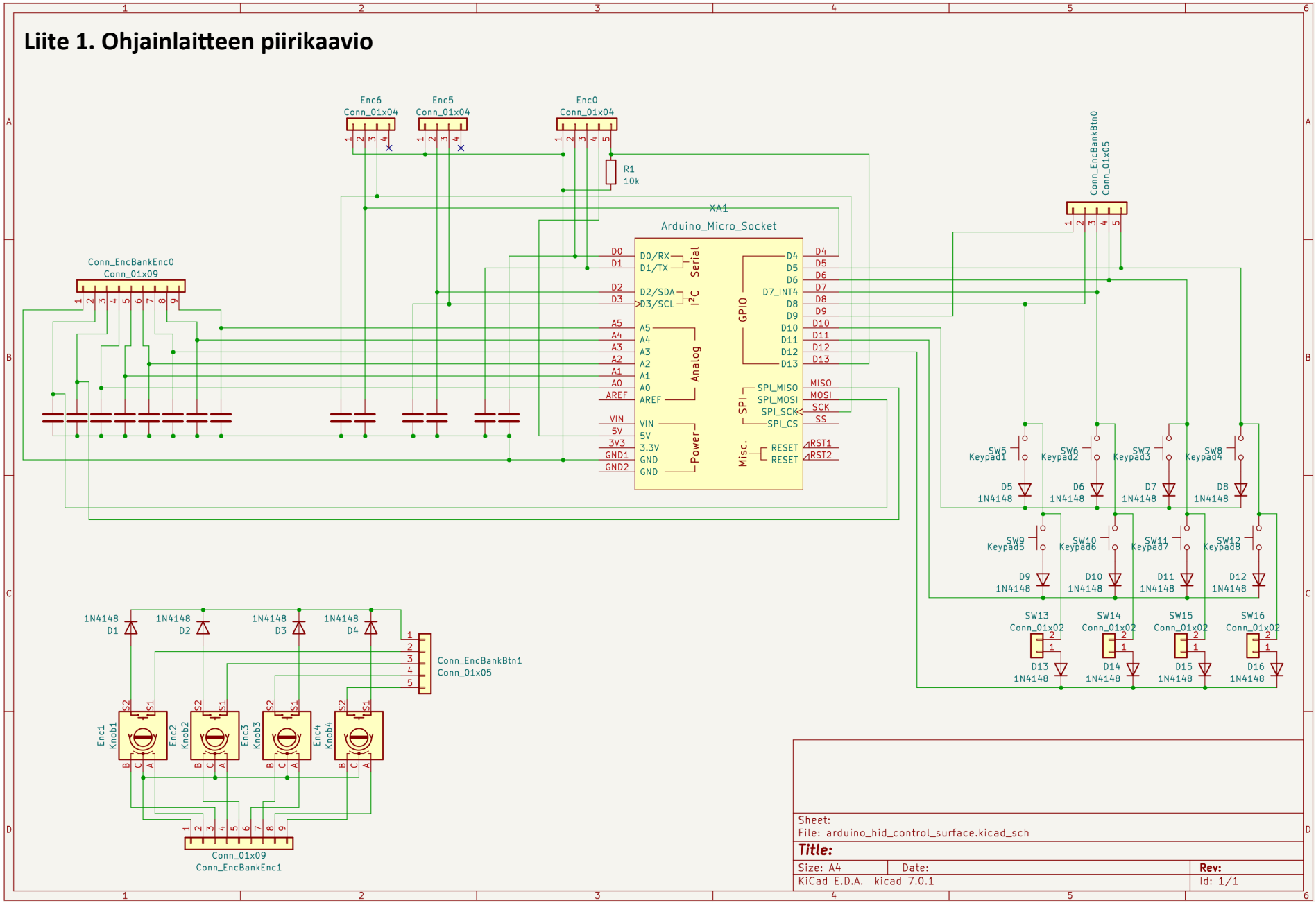
Silva R. 2022. Keyswitch Kicad Library. Viitattu 19.4.2023. https://github.com/perigoso/keyswitch-kicad-library/blob/main/README.md

Using the GNU Compiler Collection (GCC). 2005. Free Software Foundation. Viitattu 2.5.2023. https://gcc.gnu.org/onlinedocs/gcc-4.2.4/gcc/index.html

Van Hemert, T. 2021. ”I wanted to LOVE this!!! - Monogram Creative Console” -videoarvostelu ShortCircuit YouTube-kanavalla. Viitattu 20.2.2023. https://youtu.be/-X1-Sz-Hd1U

Liitteet

Liite 1. Ohjainlaitteen piirikaavio



Liite 2. Laitteen ohjelmakoodi

```
// used for input
#include <TimerOne.h>
#include <ClickEncoder.h> // Schallberts ClickEncoder library https://github.com/Schallbert/encoder/
#include <Keypad_Matrix.h> // using Nick Gammons' Keypad_Matrix library https://github.com/nickgammon/Keypad_Matrix
                           // technically useless for this, could be replaced with own matrix scan code, currently used mainly because it implements isKeyDown()
// output libraries
#include "Mouse.h" // modified Mouse library which supports horizontal scroll https://arduino.stackexchange.com/questions/46055/can-the-mouse-library-scroll-horizontally
#include <Keyboard.h>
// toggle debug serial output

// HARDWARE DEFINITIONS
// ENCODERS ###########################################################################################################################################

const uint8_t encNotches = 4; // encoder steps (pulse edges in signal) on single notch
const uint8_t encoderReleaseDelay = 50; // delay in ms between activating and releasing actions on encoder turn
                                        // as some software does not realize modifier key is pressed if it is released too fast
                                        // especially when sending 2 actions like CTRL + mouse scroll

// encoder pins, by digital pin numbers (by default as on arduino micro)
const uint8_t enc0PinA = 0;
const uint8_t enc0PinB = 1;
const uint8_t enc1PinA = 14;
const uint8_t enc1PinB = 16;
const uint8_t enc2PinA = 18;
const uint8_t enc2PinB = 19;
const uint8_t enc3PinA = 20;
const uint8_t enc3PinB = 21;
const uint8_t enc4PinA = 22;
const uint8_t enc4PinB = 23;
const uint8_t enc5PinA = 2;
const uint8_t enc5PinB = 3;
const uint8_t enc6PinA = 4;
```

```
33  const uint8_t enc6PinB = 15;
34
35  /* Defining encoders as Encoders (without buttons), as opposed to ClickEncoders,
36     because their buttons are wired separately in a matrix.
37     also use internal pullup resistors, by setting active to LOW (false) */
38  Encoder enc0{enc0PinA, enc0PinB, encNotches, LOW};
39  Encoder enc1{enc1PinB, enc1PinA, encNotches, LOW}; // invert encoder direction by changing pins around
40  Encoder enc2{enc2PinA, enc2PinB, encNotches, LOW};
41  Encoder enc3{enc3PinA, enc3PinB, encNotches, LOW};
42  Encoder enc4{enc4PinA, enc4PinB, encNotches, LOW};
43  Encoder enc5{enc5PinB, enc5PinA, encNotches, LOW};
44  Encoder enc6{enc6PinA, enc6PinB, encNotches, LOW};
45  // Store pointers to encoders in an array to make handling encoders easier
46  static Encoder *encoders[] = { &enc0, &enc1, &enc2, &enc3, &enc4, &enc5, &enc6 };
47  const int encodersSize = sizeof(encoders)/sizeof(encoders[0]); // calculate array size
48
49  const uint16_t timerInterval = 1000; // timer interval, 1ms (1000 us) interval recommended for the librarys stock settings
50  static TimerOne timer; // define timer, encoders are polled with this
51
52  // BUTTONS ##########################################################################################################################
53
54  // buttons which are not in matrix. but instead have their own pins
55  struct singleButton {
56      const uint8_t pin;
57      const bool activeState;
58  };
59
60  const singleButton enc0btn = {13, HIGH};
61
62  const singleButton *singleButtons[] = { &enc0btn }; // stores pins
63  const uint8_t singleButtonsSize = sizeof(singleButtons)/sizeof(singleButtons[0]);
64
65  // most buttons are in matrix, store row and col pins in arrays
66  const uint8_t matrixRows[] = {9,10,11,12};
67  const uint8_t matrixCols[] = {5,6,7,8};
```

```
68  const int matrixRowsSize = sizeof(matrixRows)/sizeof(matrixRows[0]); // calc keypad array sizes:
69  const int matrixColsSize = sizeof(matrixCols)/sizeof(matrixCols[0]);
70
71  /* define physical keyboard matrix button mapping
72     these are not actual char type ASCII character codes but they instead indicate row/column
73     this allows us to use row & column index instead of char codes
74     because most keypad libs use char codes, and not row/col index
75     numbers below are ROW, then COLUMN, starting from 1 (because default value for empty is 0, so we can check if key empty) */
76  const char matrixPhysButtons[matrixRowsSize][matrixColsSize] = {
77      {11, 12, 13, 14},
78      {21, 22, 23, 24},
79      {31, 32, 33, 34},
80      {41, 42, 43, 44}
81  };
82  // Defining keyboard matrix
83  Keypad_Matrix matrix = Keypad_Matrix(makeKeymap(matrixPhysButtons), matrixRows, matrixCols, matrixRowsSize, matrixColsSize);
84
85  bool matrixState[matrixRowsSize][matrixColsSize] = {}; // store matrix state
86
87  typedef enum { INPUT_BUTTON, INPUT_MATRIX, INPUT_ENCODER } INPUT_TYPE; // what kind of physical input
88
89  // define structs
90  struct HIDAction {
91      enum { DEVICE_KEYBOARD, DEVICE_MOUSE } actionDevice;
92      const char *values;
93  };
94
95  struct ComboInput {
96      const char *inputs;
97      const INPUT_TYPE *inputTypes;
98  };
99  // LIMITS ##########################################################################################################################
100 // Action array limits, adjust these if more keys are needed, but more actions takes more space in memory...
101 const uint8_t maxLayers = 3; // max action layers
102 const uint8_t maxComboInputs = 5; // max simultaneous physical input keys to active a action layer
```

```
103 const uint8_t maxActionsOnLayer = 3; // max HIDActions on one inputs single action layer
104 const uint8_t maxActionValues = 5; // max values on single HIDAction, should be >= 5 when using mouse actions
105
106 /*  Define some macros to make initializing actions easier
107     NOTE: These macros expand to (char[]){'a'}, this initialization is apparently supported in gcc/clang by extension
108           It should work in globalscope/static only according to a comment on this answer https://stackoverflow.com/a/23027131
109           Another option would've been to init these arrays first and then add them to the structs,
110           which is more typing, and also seemed to take more resources */
111 #define HIDActionValues (char[maxActionValues])
112 #define Inputs (char[maxComboInputs])
113 #define InputTypes (INPUT_TYPE[maxComboInputs])
114
115 // COMBO LAYERS ###############################################################################################################
116 // actions are defined into layers and comboLayer struct defines which physical inputs
117 // need to be activated in order to trigger any action on that action layer
118 const ComboInput noComboKey = { Inputs{  }, InputTypes{  } }; // bottom most input layer, blank by default (no assigned keys),
119                                                               // not sure on behaviour when a key is assigned to bottom layer
120
121 // Defined combo layers:
122 // const ComboInput oneComboKey = { Inputs{ 0 }, InputTypes{ INPUT_BUTTON } };
123
124 // store pointers to them in array for easier access.
125 // Place comboLayer with least combo keys topmost
126 const ComboInput *actionLayersComboInputs[maxLayers] {
127     &noComboKey
128     // &enc0buttonComboKey
129 };
130 const int comboInputsSize = sizeof(actionLayersComboInputs)/sizeof(actionLayersComboInputs[0]); // calculate array size
131
132 // INIT ACTIONS ###############################################################################################################
133
134 // modifiers by themselves (to use with mouse actions)
135 const HIDAction act_key_ctrl = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_LEFT_CTRL} };
136 const HIDAction act_key_shift = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_LEFT_SHIFT} };
137 const HIDAction act_key_alt = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_LEFT_ALT} };
```

```
138
139 const HIDAction act_mouse_scroll_up = { HIDAction::DEVICE_MOUSE, HIDActionValues{0,0,1,0,0} };
140 const HIDAction act_mouse_scroll_dn = { HIDAction::DEVICE_MOUSE, HIDActionValues{0,0,-1,0,0} };
141
142 const HIDAction act_mouse_scroll_l = { HIDAction::DEVICE_MOUSE, HIDActionValues{0,0,0,1,0} };
143 const HIDAction act_mouse_scroll_r = { HIDAction::DEVICE_MOUSE, HIDActionValues{0,0,0,-1,0} };
144
145 const HIDAction act_key_arrow_up = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_UP_ARROW} };
146 const HIDAction act_key_arrow_down = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_DOWN_ARROW} };
147 const HIDAction act_key_arrow_left = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_LEFT_ARROW} };
148 const HIDAction act_key_arrow_right = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_RIGHT_ARROW} };
149
150 const HIDAction act_key_home = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_HOME} };
151 const HIDAction act_key_end = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{KEY_END} };
152
153 const HIDAction act_key_space = { HIDAction::DEVICE_KEYBOARD, HIDActionValues{' '} };
154
155 // ACTION BINDS ##########################################################################################################
156
157 // single button actions
158 const HIDAction *singleButtonActions[maxLayers][singleButtonsSize][maxActionsOnLayer] {
159     { // layer 0
160         {  }
161     }
162 };
163
164 // keyboard matrix actions
165 const HIDAction *matrixActions[maxLayers][matrixRowsSize][matrixColsSize][maxActionsOnLayer] {
166     { // layer 0 ##########################
167       // no actionLayersComboInputs used for activating these
168         { // row 1
169             {  }, // actions for the first key (== column) in first row etc.
170             {  },
171             {  },
172             {  }
```

```
        },
        { // row 2
            { &act_key_end },
            { &act_key_space },
            { &act_key_shift, &act_key_space },
            { &act_key_home }
        },
        { // row 3
            {  },
            {  },
            {  },
            {  }
        },
        { // row 4
            { &act_key_arrow_left },
            { &act_key_arrow_down },
            { &act_key_arrow_right },
            { &act_key_arrow_up }
        },
    },
    // rest of the layers are activated by actionLayersComboInputs, see array above.
    { // layer 1 ##########################
        {
            {  },
            {  },
            {  },
            {  }
        },
        {
            {  },
            {  },
            {  },
            {  }
        },
        {
```

```
208             {  },
209             {  },
210             {  },
211             {  }
212         },
213         {
214             {  },
215             {  },
216             {  },
217             {  }
218         },
219     },
220     { // layer 3 ##########################
221         {
222             {  },
223             {  },
224             {  },
225             {  }
226         },
227         {
228             {  },
229             {  },
230             {  },
231             {  }
232         },
233         {
234             {  },
235             {  },
236             {  },
237             {  }
238         },
239         {
240             {  },
241             {  },
242             {  },
```

```
            {  }
        }
    }
};

// encoder actions clockwise
const HIDAction *encoderActionsCW[maxLayers][encodersSize][maxActionsOnLayer] {
    {
        { &act_mouse_scroll_dn }, // enc0
        { &act_mouse_scroll_l }, // enc1
        {  }, // enc2
        {  }, // enc3
        {  }, // enc4
        { &act_key_ctrl, &act_key_shift, &act_mouse_scroll_dn }, // enc5
        { &act_key_ctrl, &act_mouse_scroll_dn }  // enc6
    },
    {
        {  }, // enc0
        {  }, // enc1
        {  }, // enc2
        {  }, // enc3
        {  }, // enc4
        {  }, // enc5
        {  }  // enc6
    }
};

// encoder actions counter-clockwise
const HIDAction *encoderActionsCCW[maxLayers][encodersSize][maxActionsOnLayer] {
    {
        { &act_mouse_scroll_up }, // enc0
        { &act_mouse_scroll_r }, // enc1
        {  }, // enc2
        {  }, // enc3
        {  }, // enc4
```

```
278         { &act_key_ctrl, &act_key_shift, &act_mouse_scroll_up }, // enc5
279         { &act_key_ctrl, &act_mouse_scroll_up }  // enc6
280     },
281     {
282         {  }, // enc0
283         {  }, // enc1
284         {  }, // enc2
285         {  }, // enc3
286         {  }, // enc4
287         {  }, // enc5
288         {  }  // enc6
289     }
290 };
291
292 void setup() {
293     // set single buttons pin states
294     for (int b = 0; b < singleButtonsSize; b++) {
295         if (singleButtons[b]->activeState) {
296             pinMode(singleButtons[b]->pin, INPUT);
297         }
298         else {
299             pinMode(singleButtons[b]->pin, INPUT_PULLUP);
300         }
301     }
302
303     // begin keyboard matrix
304     matrix.begin();
305     matrix.setKeyDownHandler(onKeyDown);
306     matrix.setKeyUpHandler(onKeyUp);
307
308     // set encoder props
309     enc0.setAccelerationEnabled(true);
310
311     timer.initialize(timerInterval); // init timer for reading encoders
312     timer.attachInterrupt(timerInterrupt);
```

```
313
314     // start input libraries
315     Mouse.begin();
316     Keyboard.begin();
317
318     #if SERIAL_ENABLED
319     Serial.begin(9600);
320     #endif
321 }
322
323 // called by the timer
324 void timerInterrupt()
325 {
326     for (int x = 0; x < encodersSize; x++) {
327         encoders[x]->service(); // updates the actual encoder value for each encoder
328     }
329 }
330 // forward declarations
331 void parseLayerActions(HIDAction *actions[], int currentLayer, bool release = false, int valueMultiplier = 1);
332 void setAction(HIDAction *action, bool release = false);
333 void setActionLayer(int layer, bool release = false);
334 void releaseOtherActionLayers(int layer, bool above = false);
335
336 void loop() {
337     matrix.scan(); //scan matrix
338     static bool layerChanged; // if layer has changed while holding inputs down, see below
339     bool hasChangedLayer = false; // helper to set above
340
341     // get current action layer (== different actions if any modifier is pressed)
342     int layer = getActiveActionLayer();
343
344     // read each encoder
345     for (int e = 0; e < encodersSize; e++) {
346         int16_t val = encoders[e]->getIncrement();
347         if (val != 0) {
```

```
348             #if SERIAL_ENABLED
349             Serial.print("Enc");
350             Serial.print(e);
351             Serial.print(" turned, value: ");
352             Serial.println(val);
353             #endif
354             if (val < 0) {
355                 parseLayerActions(encoderActionsCW[layer][e], layer, false, abs(val)); // set actions
356                 delay(encoderReleaseDelay); // delay before release to mitigate some software that does not realize key is pressed
357                 parseLayerActions(encoderActionsCW[layer][e], layer, true); // encoders cannot be pressed so release actions right away
358             }
359             else if (val > 0) {
360                 parseLayerActions(encoderActionsCCW[layer][e], layer, false, abs(val));
361                 delay(encoderReleaseDelay);
362                 parseLayerActions(encoderActionsCCW[layer][e], layer, true);
363             }
364         }
365     }
366     // read single buttons
367     static bool singleButtonsStateLast[singleButtonsSize];
368     bool singleButtonsState[singleButtonsSize];
369     for (int b = 0; b < singleButtonsSize; b++) {
370         singleButtonsState[b] = digitalRead(singleButtons[b]->pin);
371         if (layerChanged || singleButtonsState[b] != singleButtonsStateLast[b]) {
372             #if SERIAL_ENABLED
373             Serial.print("singleButton ");
374             Serial.print(b);
375             Serial.print(" pin: ");
376             Serial.print(singleButtons[b]->pin);
377             #endif
378             if (singleButtonsState[b] == HIGH) {
379                 #if SERIAL_ENABLED
380                 Serial.println(" pressed");
381                 #endif
382                 if (isComboInputInLayer(singleButtons[b], layer)) { // if pressed key is comboInput in the just activated layer
```

```
383                releaseOtherActionLayers(layer, false); // release layers below (x < layer)
384                hasChangedLayer = true;
385            }
386            parseLayerActions(singleButtonActions[layer][b], layer);
387        }
388        else {
389            #if SERIAL_ENABLED
390            Serial.println(" released");
391            #endif
392            if (isComboInput(singleButtons[b])) { // if released key was comboInput
393                releaseOtherActionLayers(layer, true); // release layers above (layer < x)
394                hasChangedLayer = true;
395            }
396            parseLayerActions(singleButtonActions[layer][b], layer, true);
397        }
398    }
399    if (hasChangedLayer) {
400        layerChanged = true; // layer has changed previous layer, on next loop set next layer actions
401    }
402    else {
403        layerChanged = false;
404    }
405    singleButtonsStateLast[b] = singleButtonsState[b];
406 }
407 // read matrix, similiar to single button code
408 static bool matrixStateLast[matrixRowsSize][matrixColsSize]; // store last matrix state
409 for (int r = 0; r < matrixRowsSize; r++) {
410    for (int c = 0; c < matrixColsSize; c++) {
411        if (layerChanged || matrixState[r][c] != matrixStateLast[r][c]) {
412            if (matrixState[r][c]) { // pressed
413                if (isComboInputInLayer(matrixPhysButtons[r][c], layer)) {
414                    releaseOtherActionLayers(layer, false);
415                    hasChangedLayer = true;
416                }
417                parseLayerActions(matrixActions[layer][r][c], layer);
```

```
418                 }
419                 else { // released
420                     if (isComboInput(matrixPhysButtons[r][c])) {
421                         releaseOtherActionLayers(layer, true);
422                         hasChangedLayer = true;
423                     }
424                     parseLayerActions(matrixActions[layer][r][c], layer, true);
425                 }
426             }
427             if (hasChangedLayer) {
428                 layerChanged = true;
429             }
430             else {
431                 layerChanged = false;
432             }
433             matrixStateLast[r][c] = matrixState[r][c];
434         }
435     }
436 }
437
438 void setAction(HIDAction *action, bool release = false, int valueMultiplier = 1) {
439     #if SERIAL_ENABLED
440     if (release) {
441         Serial.print("Released, Action type: ");
442     }
443     else {
444         Serial.print("Pressed, Action type: ");
445     }
446     #endif
447     if (action->actionDevice == HIDAction::DEVICE_KEYBOARD) {
448         #if SERIAL_ENABLED
449         Serial.print("Keyboard, action keys: ");
450         Serial.print("  ");
451         #endif
452         for (int i = 0; i < maxActionValues; i++) {
```

```
453             if (action->values[i] == 0) { break; } // check for null member in actionKeys and stop loop if encountered
454             if (release) {
455                 Keyboard.release(action->values[i]);
456             }
457             else {
458                 Keyboard.press(action->values[i]);
459             }
460             #if SERIAL_ENABLED
461             Serial.print((int)action->values[i]);
462             Serial.print(" (");
463             Serial.print(action->values[i]);
464             Serial.print("), ");
465             #endif
466         }
467         #if SERIAL_ENABLED
468         Serial.println("");
469         #endif
470     }
471     else if (action->actionDevice == HIDAction::DEVICE_MOUSE) {
472         // assume mouse move actions are set as desired, any value of 0 means no movement on that axis, so no problem
473         if (!release) {
474             // move mouse, amount of action * valueMultiplier. valueMultiplier is supplied for example if using encoders with accelration
475             Mouse.move(action->values[0] * valueMultiplier, action->values[1] * valueMultiplier, action->values[2] * valueMultiplier, action->values[3] *
valueMultiplier);
476         }
477         // handle mouse click
478         if (action->values[4] != 0) {
479             if (release) {
480                 Mouse.release(action->values[4]);
481             }
482             else {
483                 Mouse.press(action->values[4]);
484             }
485         }
486         #if SERIAL_ENABLED
```

```
487         Serial.print("Mouse, action values (x,y,wheel,click): ");
488         Serial.print((int)action->values[0]);
489         Serial.print(", ");
490         Serial.print((int)action->values[1]);
491         Serial.print(", ");
492         Serial.print((int)action->values[2]);
493         Serial.print(", ");
494         Serial.println((int)action->values[3]);
495         Serial.print(", ");
496         Serial.println((int)action->values[4]);
497         Serial.print(", multiplier: ");
498         Serial.println(valueMultiplier);
499         #endif
500     }
501 }
502 
503 // used to parse and execute actions
504 void parseLayerActions(HIDAction *actions[], int currentLayer, bool release = false, int valueMultiplier = 1) {
505     for (int a = 0; a < maxActionsOnLayer - 1; a++) {
506         // if empty index
507         if (actions[a] == 0) {
508             break;
509         }
510         #if SERIAL_ENABLED
511         Serial.print(" -Layer: ");
512         Serial.print(currentLayer);
513         Serial.print(", Action: ");
514         Serial.print(a);
515         Serial.print(": ");
516         #endif
517         setAction(actions[a], release, valueMultiplier);
518     }
519 }
520 
521 int getActiveActionLayer() {
```

```
522     // iterate through layers backwards, because simplest layers are at front
523     for (int l = maxLayers - 1; l >= 0; l--) {
524         if (actionLayersComboInputs[l] == 0) { break; }
525         bool hasAllComboInputsPressed = true;
526         bool notAllComboInputsPressed = false;
527         for (int i = 0; i < maxComboInputs; i++) {
528             if (actionLayersComboInputs[l]->inputs[i] == 0) { break; }
529             bool comboInputIsPressed = false;
530             // check where a physical comboInput is at
531             if (actionLayersComboInputs[l]->inputTypes[i] == INPUT_BUTTON) {
532                 comboInputIsPressed == true; // TODO: implement separate button as combokey
533             }
534             else if (actionLayersComboInputs[l]->inputTypes[i] == INPUT_MATRIX) {
535                 comboInputIsPressed = matrix.isKeyDown(actionLayersComboInputs[l]->inputs[i]);
536             }
537             // set
538             if (comboInputIsPressed) {
539                 hasAllComboInputsPressed = true;
540             }
541             else {
542                 notAllComboInputsPressed = true;
543             }
544         }
545         if (hasAllComboInputsPressed && !notAllComboInputsPressed) {
546             return l; // return activated comboLayer index
547         }
548     }
549     return 0; // no combos matched, return base layer
550 }
551
552 // set all actions on a single actionLayer
553 void setActionLayer(int layer, bool release = false) {
554     for (int b = 0; b < singleButtonsSize; b++) {
555         parseLayerActions(singleButtonActions[layer][b], layer, release);
556     }
```

```
557     for (int r = 0; r < matrixRowsSize; r++) {
558         for (int c = 0; c < matrixColsSize; c++) {
559             parseLayerActions(matrixActions[layer][r][c], layer, release);
560         }
561     }
562 }
563 
564 // returns true at the first comboLayer that the key is part of
565 bool isComboInputInLayer(char key, int layer) {
566     for (int i = 0; i < maxComboInputs; i++) {
567         if (actionLayersComboInputs[layer]->inputs[i] == 0) {
568             return false;
569         }
570         if (key == actionLayersComboInputs[layer]->inputs[i]) {
571             return true;
572         }
573     }
574     return false;
575 }
576 // returns true if key is comboInput
577 bool isComboInput(char key) {
578     for (int l = 0; l < maxLayers; l++) {
579         if (actionLayersComboInputs[l] == 0) { break; }
580         for (int i = 0; i < maxComboInputs; i++) {
581             if (actionLayersComboInputs[l]->inputs[i] == 0) { break; }
582             if (key == actionLayersComboInputs[l]->inputs[i]) {
583                 return true;
584             }
585         }
586     }
587     return false;
588 }
589 
590 void releaseOtherActionLayers(int layer, bool above = false) {
591     #if SERIAL_ENABLED
```

```
592     Serial.print("###### Layer switched to ");
593     Serial.println(layer);
594     #endif
595     for (int l = maxLayers - 1; l >= 0; l--) {
596         if (actionLayersComboInputs[l] == 0) { break; } // not a valid layer
597         if (above && l > layer) {
598             #if SERIAL_ENABLED
599             Serial.print("#### Releasing layer: ");
600             Serial.print(l);
601             Serial.println(" above, ");
602             #endif
603             setActionLayer(l, true);
604         }
605         else if (!above && l < layer) {
606             #if SERIAL_ENABLED
607             Serial.print("#### Releasing layer: ");
608             Serial.print(l);
609             Serial.println(" below, ");
610             #endif
611             setActionLayer(l, true);
612         }
613     }
614 }
615
616 // matrix handling
617 void onKeyDown(const char key) {
618     #if SERIAL_ENABLED
619     Serial.print("Matrix: ");
620     Serial.print((int)key);
621     Serial.println(" pressed ");
622     #endif
623     int row = getRowFromKey(key);
624     int col = getColFromKey(key);
625     matrixState[row][col] = true;
626 }
```

```
627
628 void onKeyUp(const char key) {
629     #if SERIAL_ENABLED
630     Serial.print("Matrix: ");
631     Serial.print((int)key);
632     Serial.println(" released ");
633     #endif
634     int row = getRowFromKey(key);
635     int col = getColFromKey(key);
636     matrixState[row][col] = false;
637 }
638
639 // convert key number to row/col index, physical input numbers start from 1 but indexes from 0!
640 int getRowFromKey(char key) {
641     return ((key / 10) % 10) - 1; // return leftmost digit minus one (10)
642 }
643 int getColFromKey(char key) {
644     return (key  % 10) - 1; // return rightmost digit minus one (01)
645 }
```