Jonne Saloranta

Tekoälyn hyödyntäminen esineiden lajittelussa

Opinnäytetyö

Tekniikan ammattikorkeakoulututkinto

Sähkö- ja automaatiotekniikan koulutus

2024

Kaakkois-Suomen ammattikorkeakoulu

Kaakkois-Suomen ammattikorkeakoulu

Tutkintonimike	Insinööri (AMK)
Tekijä/Tekijät	Jonne Saloranta
Työn nimi	Tekoälyn hyödyntäminen esineiden lajittelussa
Vuosi	2024
Sivut	24 sivua, liitteitä 35 sivua
Työn ohjaaja(t)	Teemu Manninen

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli kehittää tekoälyä ja konenäköä hyödyntävä esineiden lajittelijan prototyyppi. Opinnäytetyössä käydään läpi ohjelmoitavan moottoriohjaimen suunnittelu, ohjelmiston kehittäminen valvomolle, moottoriohjaimelle sekä palvelimelle, jonka kautta valvomo ja moottorinohjain kommunikoivat.

Työ aloitettiin ohjelmoitavan moottoriohjaimen vaatimuksista ja komponenttien valinnasta. Käyttöjännitteeksi valittiin 12V, sillä se on hyvin yleinen jännite monissa virtalähteissä. Moottorinohjaimen piirilevyyn suunniteltiin myös jännitteenalenninpiiri, jolla saadaan mikroprosessorille sen vaatima 5V käyttöjännite. Valvomo ja kommunikointipalvelin toteutettiin käyttämällä Python-ohjelmointikieltä sekä erilaisia kirjastoja. Mikroprosessorin ohjelmointiin käytettiin Arduinoa, joka perustuu C/C++-ohjelmointikieleen. 3D-mallit suunniteltiin Autodesk Fusion360 -ohjelmalla, ja mallit tulostettiin käyttämällä Creality Ender3 -3D-tulostinta. Lajittelija suunniteltiin niin, että sillä saadaan lajiteltua kappaleet kolmeen eri kohtaan.

Opinnäytetyön tuloksena syntyi toimiva toteutus, jolla saadaan lajiteltua kolmea erilaista kappaletta korkealla tunnistusvarmuudella. Toteutus sisältää 3D-tulostetun lajittelijan, ohjelmoitavan logiikkalaitteen sekä ohjelmistot laitteen ohjaamiseen ja langattomaan kommunikointiin.

Asiasanat: Tekoäly, Konenäkö, Arduino, Piirilevy, Prototyyppi

South-Eastern Finland University of Applied Sciences

Degree title	Bachelor of Engineering
Author (authors)	Jonne Saloranta
Thesis title	Utilization of artificial intelligence in object sorting
Time	2024
Pages	24 pages, 35 pages of appendices
Supervisor	Teemu Manninen

ABSTRACT

The purpose of the thesis was to develop a prototype of an object sorter utilizing artificial intelligence and computer vision. The thesis outlines the design of a programmable motor controller, the development of software for the control room, motor controller and a server, through which the control room and motor controller communicate.

The work began with the requirements for the programmable motor controller and the selection of components. A supply voltage of 12V was chosen, as it is a very common voltage in many power supplies. The motor controller circuit board also included a step-down converter circuit to provide the 5V operating voltage required by the microprocessor. The control room and communication server were implemented using the Python programming language and various libraries. Arduino, based on the C/C++ programming language, was used for programming the microprocessor. The 3D models were designed using the Autodesk Fusion360 software and printed with the Creality Ender3 3D printer. The sorter was designed to sort objects into three different locations.

The result of the thesis was a functional implementation capable of sorting three different types of objects with high recognition accuracy. The implementation includes a 3D-printed sorter, a programmable logic device, and software for controlling the device and wireless communication.

Keywords: AI, Machine Vision, Arduino, PCB, Prototype

SISÄLLYS

LYHENTEET

GPT	Generative pre-trained transformers
PCB	Printed Circuit Board
PLC	Programmable Logic Controller
PLA	Polylactic Acid
NC	Normally Closed
NO	Normally Open
SMT	Surface Mount Technology
THT	Through Hole Technology
GPIO	General Purpose Input/Output
GND	Ground
BOM	Bill of Materials
RGB	Red/Green/Blue
FOSS	Free and Open-Source Software
TLS	Transport Layer Security
SSL	Secure Socket Layer

1 JOHDANTO

Tekoälyn kehitys on ottanut viime vuosina suuria harppauksia eteenpäin. Tekoälyä käytetään jopa muotisanana ilmaisemaan laitteen tai ohjelman edistyksellisyyttä. Tekoälyn hyödyt tulevat vastaan esimerkiksi monien tuntemasta ChatGPT-ohjelmistosta, joka pystyy tuottamaan lähes täydellistä tekstiä monilla eri kielillä.

Toisena esimerkkinä voidaan käyttää parkkihalleissa olevia kameroja, jotka tunnistavat parkkihalliin ajavien autojen rekisteritunnuksen tekoälyllä ja konenäöllä. Tällainen ohjelmisto olisi lähes mahdotonta toteuttaa perinteisillä menetelmillä. Tekoälyn kehittämisessä onkin tärkeää, että sen opettamiseen käytetään paljon hyvälaatuista dataa. Hyvälaatuisella datalla ei tarkoiteta mahdollisimman hyvää kuvaa, vaan sitä, että opetusdata ei sisällä ollenkaan tai mahdollisimman vähän virheitä.

Tässä opinnäytetyössä käydään läpi tekoälyä ja konenäköä hyödyntävän lajittelijan prototyypin kehittämisen eri vaiheita, kuten piirilevyjen suunnittelua, komponenttien valintaa, ohjelmistojen kehittämistä valvomolle, kommunikointipalvelimelle sekä itse ohjelmoitavalle moottorinohjaimelle. Laitteen muut fyysiset osat suunnitellaan Autodesk Fusion360 -ohjelmistolla, ja ne tulostetaan käyttäen Creality Ender3 -3D-tulostinta.

2 MENETELMÄT

Ohjelmistojen kehittämiseen valitsin Python-ohjelmointikielen sen helppouden ja laajan kirjastovalikoiman vuoksi. Kirjastoja, jotka ovat isossa osassa valvomon kehittämisessä ovat tkinter ja opencv2. Tkinter on käyttöliittymien tekemiseen tarkoitettu kirjasto, jolla saadaan tehtyä alustariippumattoamia käyttöliittymiä Windows, Linux ja Mac tietokoneille. Valvomoon tulee kamerasyöte USB-kamerasta opencv2 kirjaston avulla. Kommunikointipalvelin toteutetaan myös käyttämällä Pythonia ja sen sisältämää socket-moduulia. Palvelimen tarkoituksena on välittää komennot ja viestit lajittelijalle.

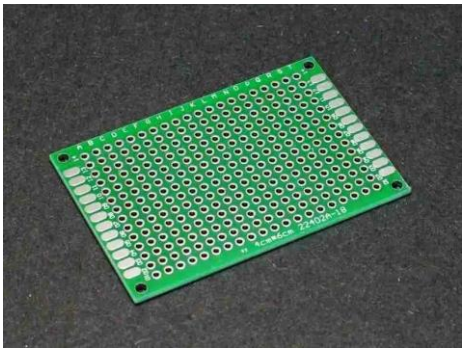
Ohjelmoitava moottoriohjain tehdään käyttämällä Arduino ja C/C++-ohjelmointikieltä. Ohjelmoinnin olisi myös voinut toteuttaa Pythonilla ja asentamalla mikroprosessorille MicroPython-ohjelman, mutta se ei ole tuttu, joten päätin käyttää Arduinoa. Moottorinohjaimen piirilevy suunnitellaan käyttämällä EasyEDA-ohjelmaa, jossa on kattava valikoima komponentteja sekä yhteisön luomia piirejä. Komponenttien valinta ei vaatinut ylimääräistä tarkkuutta, sillä lajittelijaa tullaan käyttämään vain kuivassa ja sisätiloissa, joten kosteuden tai kylmyyden kestävyydellä ei ole vaikutusta.

Lajittelijan muut fyysiset osat suunnitellaan käyttämällä Autodesk Fusio360 3D-mallinnus ohjelmalla. Käytän Fusion360-ohjelmaa sen helpon työnkulun takia. 3D-mallit saa tuotua ohjelmasta .obj tiedostona, joita Ultimate Cura tukee. Ultimate Cura on niin kutsuttu ”slicer”, joka muuttaa 3D-mallit gkoodiksi, jota 3D-tulostimet ymmärtävät. Kaikki osat tulostetaan käyttämällä Creality Ender3 tulostinta. Ender3 on aloittelija ystävällinen 3D-tulostin ja tulostus alue on 220x220x250mm, mutta käytettävä alue on oikeasti jonkun verran pienempi.

3 OHJELMOINTAVAN MOOTTORIOHJAIMEN SUUNNITTELU

Moottoriohjaimen piirien suunnittelulle on monta erilaista toteutustapaa. Helpoin tapa toteuttaa piirien kytkennät on käyttää koekytkentälevyä (kuva 1), mutta silloin piiri on hankala tehdä kestäväksi ja pysyväksi. Toisena vaihtoehtona voisi käyttää juotettavaa koekytkentälevyä (kuva 2), jolla saataisiin jo liikuttamista kestävä piirilevy. Päädyin kuitenkin valmistuttamaan piirilevyn siihen erikoistuneessa yrityksessä, jonka ansiosta piirilevystä saa kestävän sekä esteettisesti miellyttävämmän.

Kuva 1. Koekytkentälevy



Kuva 2. Juotettava koekytkentälevy

Piirilevyjen suunnitteluun päädyin käyttämään EasyEDA-ohjelmistoa. Se on ilmainen, helppokäyttöinen ja se tarjoaa piirilevyjen teetättämisen muutamalla klikkauksella. Piirilevyn komponentit saadaan myös tilattua samasta paikasta, kunhan käyttää ohjelmiston sisäistä komponenttikirjastoa. Tämä helpottaa piirilevyn suunnittelua, kun komponentteja ei tarvitse metsästää eri myyjiltä.

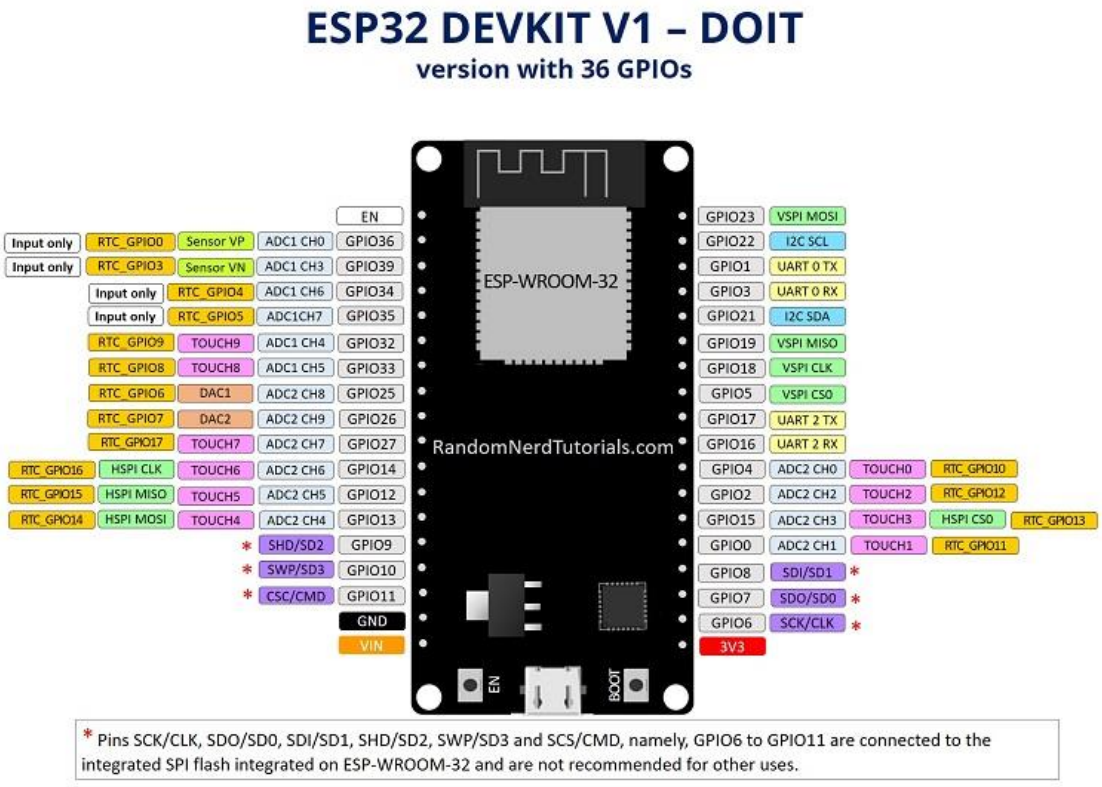
3.1 Komponenttien valinta

Piirilevyjen suunnittelussa yksi tärkeimmistä vaiheista on komponenttien valinta. Ominaisuuksia, joita kannattaa ottaa huomioon jo suunnitteluvaiheen alussa, ovat esimerkiksi komponenttien koko, lämpötilakestävyys ja käyttävätkö ne SMT- vai THT-teknologiaa. Päätin käyttää THT-teknologiaa kaikissa liittimissä ja SMT-teknologiaa muissa komponenteissa. [1.]

Piirilevyn käyttöjännitteeksi valitsin 12V, sillä se on hyvin yleinen jännitetaso akuissa ja virtalähteissä. Mikroprosessorit käyttävät yleisesti 3,3V tai 5V jännitettä, joten niille pitää suunnitella jännitettä alentava piiri. [2.]
Moottoriohjainpiirilevy tarvitsee myös mikroprosessorin, jonka avulla voidaan ohjelmallisesti ohjata sisään- ja ulostulopinnejä eli GPIO-pinnejä. Sisääntulopinnien avulla voidaan lukea sisään tuleva jännite 0-5V väliltä. Sisääntulopinnejä on kahta erilaista, digitaalisia ja analogisia. Digitaaliseen sisääntulopinniin voidaan syöttää 5V jännite, jolloin mikroprosessori tulkitsee tämän olevan 1 tai päällä. Pinnin ollessa samassa potentiaalitasossa kuin GND-pinni, luetaan tämän olevan 0 tai sammuksissa. Tällä tavoin pystytään tunnistamaan, jos kytkintä on painettu. Analoginen sisääntulo taas pystyy antamaan arvon 0 ja 1 väliltä tai muulta ohjelmoidulta väliltä, kuten 0 ja 255. Analogisten pinnien käyttö on yleistä esimerkiksi lämpötila-antureissa, sillä ne perustuvat resistiivisyyden muutokseen kahden eri metallin välillä, joka voidaan havaita

jännitteen muutoksena. Moottorinohjaimeen ei tässä tapauksessa tule kuin digitaalisia pinnejä vaativia komponentteja,kuten kytkimiä.

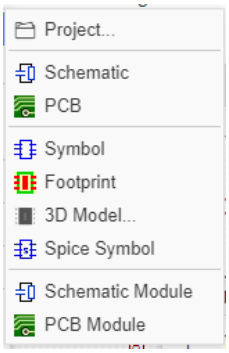
Mikroprosessoriksi valitsin DOIT ESP32 DEVKIT V1 -kehitysalustan, sillä se oli jo entuudestaan tuttu ja siinä oli paljon tarvittavia ominaisuuksia. Yksi isoimmista ominaisuuksista oli 2,4GHz WiFi-tuki, jonka avulla saadaan tehtyä lajittelijasta langattomasti toimiva. ESP32 tukee myös Arduinolla ohjelmointia, mikä helpottaa ja nopeuttaa prototyypin kehittämistä.



Kuva 3. ESP32-pinnien järjestys

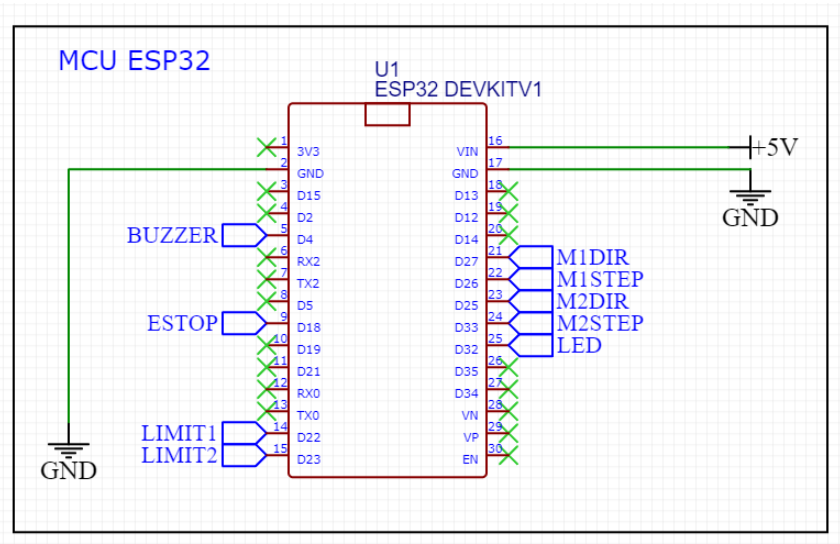
3.2 Piirilevyn suunnittelu

Piirilevyn suunnittelu aloitetaan luomalla projekti ja lisätään projektiin ”Schematic”-tiedosto eli kytkentäkaavio (kuva 4). Kytkentäkaavioon sijoitetaan tarvittavat komponentit ja tehdään kytkennät johdottamalla tai käyttämällä ”net port”-merkkejä. ”Net port” merkkien avulla saadaan kytkennöistä selkeämpiä ja eri piirit saadaan eroteltua visuaalisesti. Päätin myös käyttää ”JLCPCB-assebled”-palvelua, joka kasaa piirilevyt valmiiksi. Palvelua voidaan käyttää, kunhan kaikki komponentit on valittu kyseisestä kategoriasta, muuten niitä ei voida käyttää automatisoidussa kasausprosessissa.



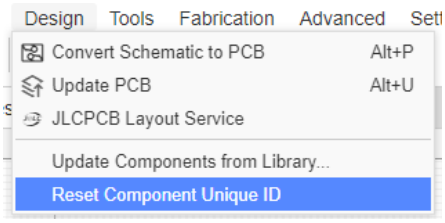
Kuva 4. Projektin kaaviotyypit

Käytin kuvaa kolme pinnien käytettävyyden tarkastamiseksi, sillä osaa pinneistä ei voida käyttää sisään/ulostulotarkoituksiin. Esimerkiksi VP, VN, D34 ja D35 voidaan käyttää vain sisääntulopinneinä. Vin kytketään +5V syöttöön, joka tulee 12V syötön ja jännitteenalennin piirikautta pinnille. Kaikki GND-pinnit kytketään piirilevyyn tehtävään maadoituskuparitasoon (kuva 5).



Kuva 5. Pinnien kytkentä kytkentäkaaviossa

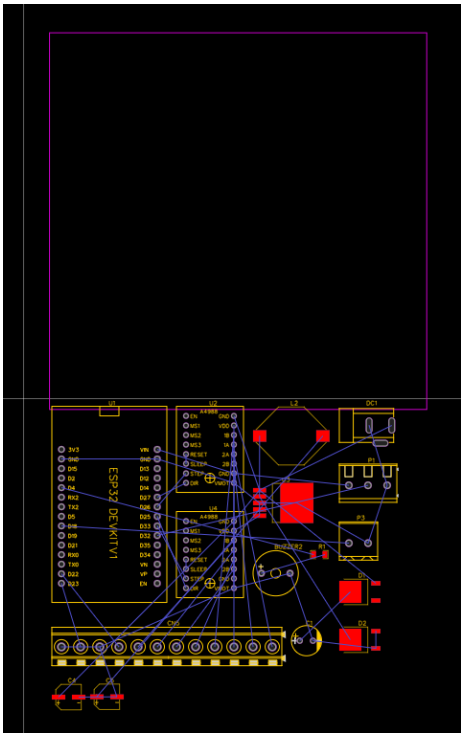
Kaiken kaikkiaan tehtiin yhdeksän eri piiriä, jotta saadaan kaikki tarvittavat komponentit toimimaan. Askelmoottoriohjaimille tehtiin molemmille omat piirinsä, 12V syöttö ja siitä 5V jännitteenalennuspiiri. Muita kytkentöjä oli mikroprosessorin, hätäseiskytkimen, ledivalon ja pienen summerin piirit. Kytkennät ovat valmiita, joten se voidaan muuttaa piirilevyksi painamalla Alt+P (kuva 6).



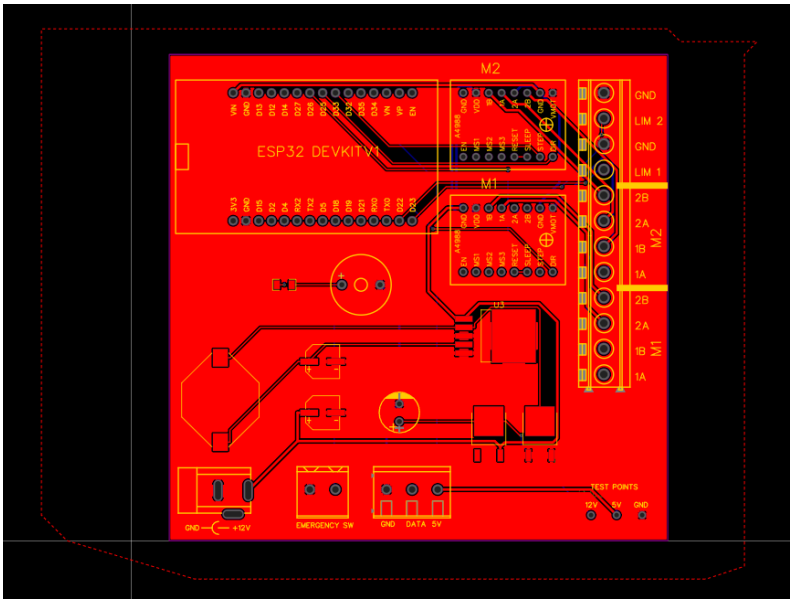
Kuva 6. Luodaan piirilevy kytkentäkaavioista painamalla Alt + P

3.3 Komponenttien sijoittaminen

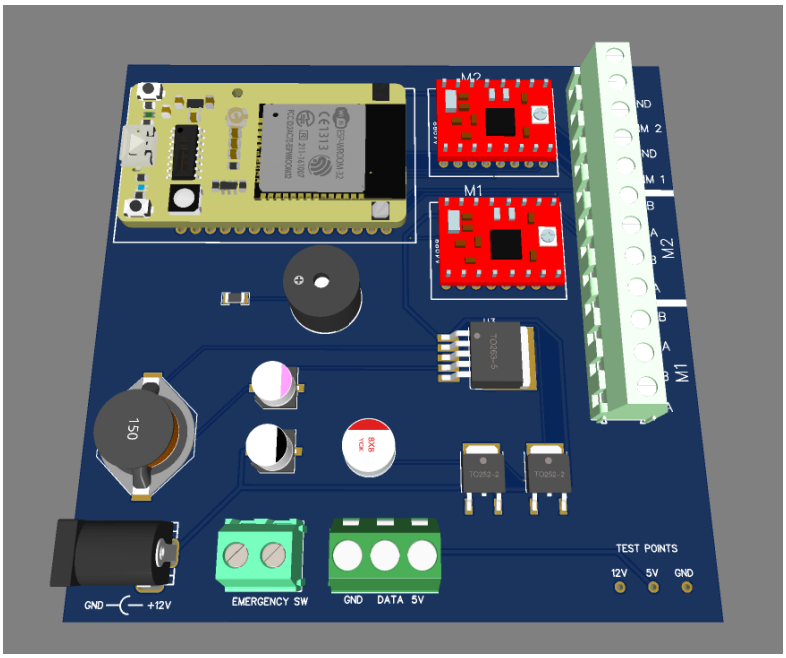
Piirilevyn koko on 100x100 mm, ja siihen tulee sijoittaa kaikki tarvittavat komponentit (kuva 7). Etenkin liittimiä sijoittaessa kannattaa tarkastaa, että ne asennetaan oikeinpäin ja että niihin kytkettävillä johdoilla on tarpeeksi tilaa. ESP32 sijoitettiin lähelle piirilevyn reunaa, jotta se on helppo ohjelmoida. Piirilevylle tehdään myös maadoitustaso, joka on kuparitaso piirilevyn sisällä, ja näin saadaan tehtyä helposti samassa potentiaalitasossa oleva maadoitus. Kuparitaso myös vähentää tarvittavien kuparireittien tekemisen piirilevylle (kuva 8). 3D-näkymästä voidaan tarkastella, millainen piirilevystä tulee ja vaativatko komponentit fyysisesti enemmän tilaa (kuva 9).



Kuva 7. Komponentit piirilevyn vieressä



Kuva 8. Komponentit sijoitettuna ja reitit reititettynä



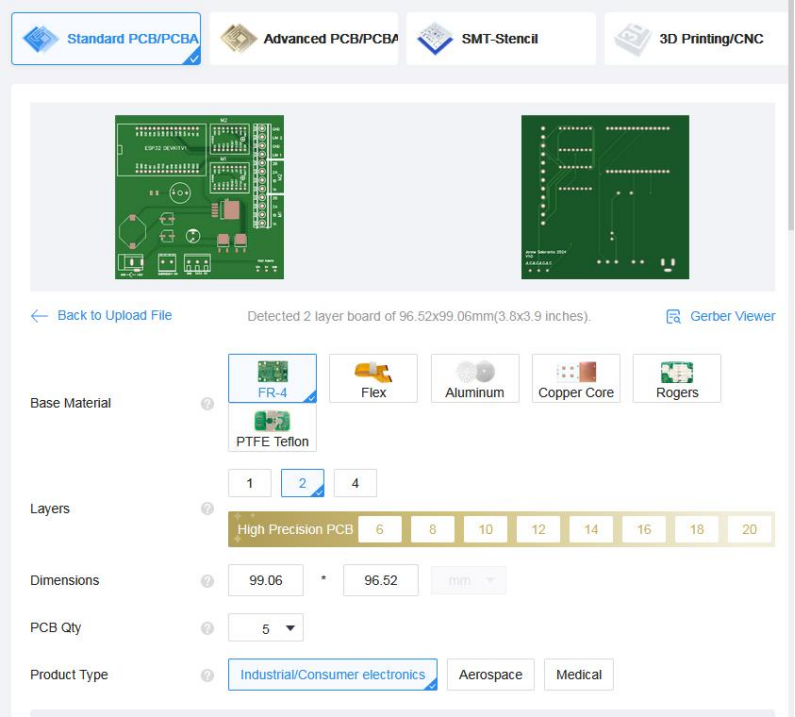
Kuva 9. 3D-näkymä piirilevystä

3.4 Kytkentöjen tarkistaminen

Kytkennät on hyvä tarkistaa silmämääräisesti ennen piirilevyn tilausta, ettei mitään kytketä väärin tai jätetä kokonaan kytkemättä. Näin vältetään ajan ja rahan tuhlaus, sillä piirilevyn sisäisiä kytkentöjä on hankala muokata jälkeenpäin. EasyEDA-ohjelmassa on sisäinen kytkentöjen tarkistus. Tämä tarkistaa, että kaikki kuparireitit ovat kytkettynä, ellei niitä ole erikseen merkitty ei-kytkettäväksi. Sisäisen tarkistuksen lisäksi on hyvä tarkastaa itse kaikki kytkennät. Kaikki kytkennät olivat ok, joten nyt voidaan tilata piirilevyt.

3.5 Piirilevyjen tilaus

Piirilevyn tilaaminen onnistuu yhdellä klikkauksella. EasyEDA luo automaattisesti GRB-tiedostot, joilla piirilevyt tehdään, ja BOM-tiedoston, joka sisältää kaikki tarvittavat komponentit. Painetaan valikosta ”Fabrication” ja alavalikosta ” One-click order PCB/SMT”, joka ohjaa selaimeen valitsemaan piirilevyn värin, määrän jne. (Kuva 10.)



Kuva 10. Piirilevyn tilausasetukset

4 OHJELMISTON KEHITTÄMINEN

Ohjelmistojen kehittäminen on nykypäivän yksi keskeisimmistä aiheista minkä tahansa laitteen valmistamisessa. Ohjelmistot pitää kehittää varmatoimisiksi sekä tietoturvallisiksi. Etenkin teollisuudessa laitteet voivat olla painavia ja nopeita, joten ne voivat aiheuttaa niin taloudellisia kuin henkilövahinkojakin.

Tämän prototyypin ohjelmistossa huomioidaan muutamia turvallisuusseikkoja, kuten datan lähetyksen ja vastaanottamisen varmistaminen sekä hätäseis-painikkeen toiminta. Muita toimintaan vaikuttavia tekijöitä ovat esimerkiksi lajittelijan asennon tunnistaminen rajakytkimien avulla. Näin lajittelija tietää, milloin se on ns. 0-asennossa, eikä asento pääse harhailemaan vääriin asentoihin.

Tietoturva on huomioitu tässä opinnäytetyössä vain siten, että lajittelija on yhteydessä WPA2- suojattuun langattomaan verkkoon. Kuka vain verkkoon pääsevä voisi siis keskustella ja ohjata laitetta.

4.1 Valvomo

Valvomon käyttöliittymän luomiseen käytettiin Python 3.11 -versiota ja sen valmiiksi sisältämää tkinter-pakettia. Se on kirjasto, jonka tarkoituksena on helpottaa käyttöliittymien luontia eri alustoille, kuten Windows, Linux ja Mac.

Käyttöliittymään on lisätty kamerasyöte, jonka avulla voidaan valvoa lajiteltavia kappaleita tai mahdollisia ongelmia. Kamerasyötteessä näkyy myös rajaava suorakulmio, joka on tekoälyllä tunnistettujen kappaleiden ympärillä. Valvomon sisältää tarvittavat painonapit, joilla ohjataan prototyypin eri toimintoja. Valvomon painonapit on toteutettu käyttäen tkinterin sisäänrakennettua Button-funktiota, jolla saadaan tehtyä helposti erilaisia toimintoja. Käyttöliittymässä on myös tekstikentät, joihin voi syöttää kommunikointipalvelimen IP-osoitteen ja portin, jos ne eivät ole samat kuin valmiiksi annetut 127.0.0.1 ja 12345. Tämä helpottaa yhdistämisen eri palvelimille.

4.2 Lajittelija

Lajittelijalle tulevat kaksi askel moottoria mahdollistavat sen ohjaamisen kahdella eri akselilla. Näin saadaan esineet lajiteltua niiden omille paikoilleen. Lajittelijan pohjan muoto on neliö, ja näin saadaan yksi reuna johdotuksille ja kolme muuta reunaa lajiteltavien esineiden lajittelu sijainneiksi.

4.2.1 Tilamerkkivalo

Tilamerkkivalon on tärkeä turvalaite kaikissa laitteissa. Tällä tavoin laitteen lähellä olevat ihmiset näkevät, missä tilassa laite on ja merkkivalon värit auttavat vikojen diagnosoinnissa. He myös näkevät jo kaukaa, jos joku on painanut hätäseis-nappia.

Lajittelijan toiminta tilan merkkivalona toimii yksittäin ohjattava RGB-ledinauha, jossa on 7kpl WS2812B-mallin lediä. Yksittäin ohjattavat ledit ovat hyviä moniin eri käyttötarkoituksiin, sillä jokaista lediä voidaan ohjata erikseen erilaisella kirkkaudella ja värillä. Käytin For-silmukkaa luomaan häivytyksen ledien vilkkumiselle. Tätä käytetään langattomaan verkkoon ja palvelimeen yhdistettäessä ilmaisemaan onko yhdistäminen onnistunut. (Kuva 11.)

```
void fade_flash(uint32_t color, int flashes, int speed, bool used_in_core)
{

    strip.fill(color, 0, number_of_leds);
    for (int i = 0; i < flashes; i++)
    {
        for (int j = 0; j < 255; j++)
        {
            strip.setBrightness(j);
            strip.show();
            if (used_in_core)
            {
                vTaskDelay(speed / portTICK_PERIOD_MS);
            }
            else
            {
                delay(speed);
            }
        }
        for (int j = 255; j > 0; j--)
        {
            strip.setBrightness(j);
            strip.show();
            if (used_in_core)
            {
                vTaskDelay(speed / portTICK_PERIOD_MS);
            }
            else
            {
                delay(speed);
            }
        }
    }
}
```

Kuva 11. Ledien vilkkumisen funktio

4.2.2 Moottoriohjaus

Moottorien ohjaukseen päätin käyttää hyvin yleistä A4988-mallin askelmoottoriohjainsovitinkorttia. Ohjaimen suurin sallittu käyttö virta on 2A, joka on riittävä Nema 17 Pancake -mallin askelmoottorille. Askelmoottorin suurin käyttövirta on 1A. Pienemmän käyttövirran ansiosta askelmoottoriohjain pysyy viileämpänä ja moottoreita voidaan päivittää tarvittaessa tehokkaampiin.

Askelmoottoriohjaimen virtaa rajoitetaan käyttämällä A4988-datalehdestä löytyvää kaavaa. Kyseissä A4988-sovitinkortissa oleva virranmittaus vastus on 0,1Ω ja ITripMAX tulisi olla sama kuin askelmoottorin suurin käyttövirta eli 1A. Kaavasta 1 pitää ratkaista Vref, jonka avulla saadaan säädettyä virranrajoitus oikeaksi. [3]

$$I_{TripMAX} = \frac{V_{Ref}}{8 * R_s} \quad (1)$$

$$V_{ref} = I_{TripMAX} * R_s * 8$$

$$V_{ref} = 1A * ,1\Omega * 8 = 0,8V$$

Moottoreita ohjataan käyttämällä BasicStepperDriver-kirjastoa. Kirjaston avulla moottoreiden ohjaaminen on helppoa. Tein myös moottoreiden ohjaamiseen muutamia lisäfunktioita, kuten right_90 ja down_45 jne.

```
#include "BasicStepperDriver.h"
…
const int M1DIR = 27;
const int M1STEP = 26;
BasicStepperDriver stepper1(MOTOR_STEPS, M1DIR, M1STEP);
…
void left_90()
{
    stepper1.rotate(-90);
}

…
if (command.startsWith("potentiometer:"))
{
    if (!busy)
    {
        busy = true;
        vTaskDelay(1000 / portTICK_PERIOD_MS);
        right_90();
        vTaskDelay(100 / portTICK_PERIOD_MS);
        down_45();
        vTaskDelay(100 / portTICK_PERIOD_MS);
        up_45();
        vTaskDelay(100 / portTICK_PERIOD_MS);
        left_90();
        vTaskDelay(100 / portTICK_PERIOD_MS);
        clear_command();
        busy = false;
    }
}
```

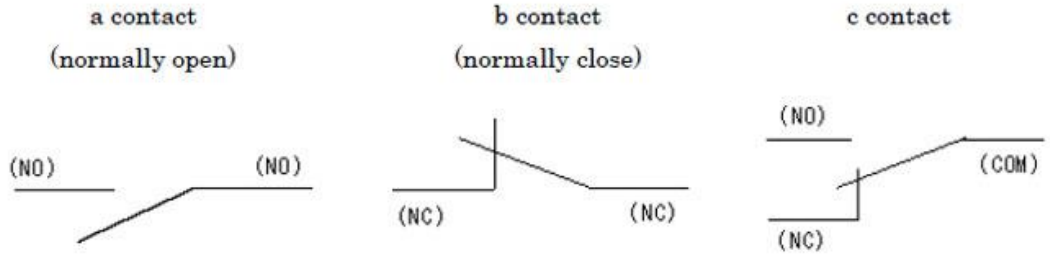
Kuva 12. Moottorin ohjaukseen käytettyjä koodinpätkiä

4.2.3 Hätäseis

Fyysinen hätäseis on toteutettu käyttämällä hätäseis-kytkintä. Tällä tavoin lajittelija voidaan pysäyttää, jos se tekee jotain odottamatonta. Hätäkytkimet ovat yleisesti NC-kytkimiä eli normaalisti suljettuja, jolloin kytkintä painettaessa kytkin avautuu. Ohjelmallisesti hätäseis on toteutettu käyttämällä Arduinon pin-

Mode-funktiota, jonka avulla voidaan asettaa mikroprosessorin pinnit sisääntuloksi tai ulostuloksi. Hätäkytkimen tapauksessa laitetaan pinMode-sisääntuloksi, koska kyseistä pinniä halutaan ohjata. PinMode laitetaan myös käyttämään ESP32:n sisäistä ylösvetovastusta, jolloin saadaan pinni loogiseen päällä-tilaan. Kytkimen ollessa kytkettynä pinni on loogisessa 0-tilassa, ja kytkintä painettaessa kytkin avautuu ja se kytkeytyy loogiseen 1-tilaan.

.



Kuva 13. NC- ja NO-kytkimien toiminta

4.2.4 Rajakytkimet

Rajakytkimiä käytetään tunnistamaan erilaisten laitteiden ääriasentoja. Tässä käytetään kahta rajakytkintä tunnistamaan pystyakselin ääriasennot. Näin moottoreiden johdot eivät pääse sotkeutumaan tai rikkoutumaan. Rajakytkimet ovat myös PinMode-funktion avulla kytkettynä loogiseen 1-tilaan, kun ne ovat auki-tilassa. Lajittelijan käynnistyessä se suorittaa kalibroinnin, jolla se löytää oman sijaintinsa kokeilemalla molemmat rajakytkimet kääntymällä niitä vasten.

4.2.5 Palvelinyhteys

Lajittelijan käynnistyessä se yrittää yhdistää ennalta määritettyyn langattomaan WiFi-verkkoon. Yhdistämisen onnistuessa se yrittää yhdistää samassa verkossa toimivaan kommunikointipalvelimeen portin 12345 kautta. Tässä vaiheessa lajittelija on valmis ottamaan vastaan komentoja joko tekoälyn tunnistuksista tai ihmisen antamista komennoista.

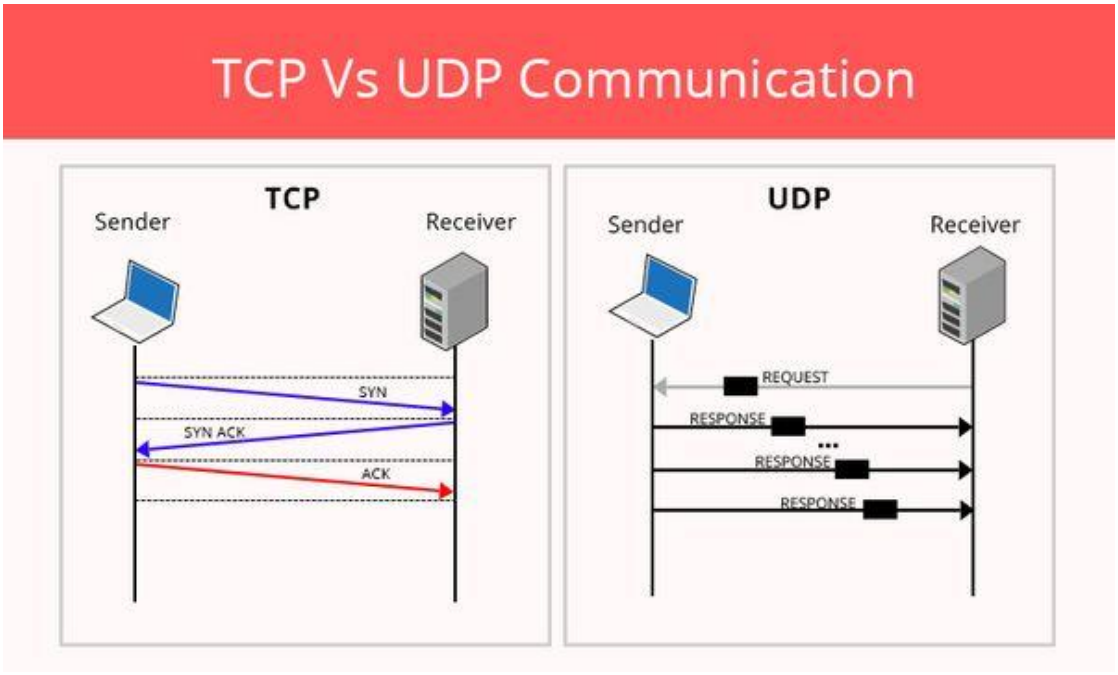
4.2.6 Ohjelmalliset turvamekanismit

Ohjelmallisten turvamekanismien tarkoitus on estää vahingot, jotka voivat johtua viallisesta anturista tai muusta laitteesta. Hätäkytkimelle on tehty turvamekanismi, jonka avulla hätäseis-tilannetta ei voi kytkeä pois päältä, vaikka kytkin irrotettaisiin sen liittimistä.

4.3 TCP/IP-palvelin

Palvelin hyödyntää Pythonin socket-moduulia, joka auttaa lähettämään ja vastaanottamaan viestejä päätepisteiden välillä. Tässä palvelimen tapauksessa kaikki laitteet keskustelevat ainoastaan palvelimen kanssa ja palvelin jakaa tarvittavan tiedon eteenpäin muille laitteille.

Oikean tiedon varma lähettäminen ja vastaanottamien on kriittistä, joten suunnittelussa piti valita TCP/IP- ja UDP/IP-protokollien väliltä. Alla olevasta kuvasta voidaan nähdä TCP- ja UDP-protokollien erot. TCP vaatii aina, että vastaanottaja vahvistaa, että on saanut tiedot ja että se on sama tieto tarkistussumman avulla. UDP on nopeampi kuin TCP, mutta se ei varmista, onko tieto saatu tai onko se tullut ehjänä perille. Tästä syystä kommunikointi protokollaksi valitsin TCP-protokollan.



Kuva 14. TCP- ja UDP-protokollien erot

Palvelimesta tehtiin hyvin yksinkertainen, sillä sen ei tarvitse kuin välittää vastaanottamansa viestit. Palvelimen ainoat asetukset tai muuttujat ovat palvelimen IPv4-osoite, josta se kuuntelee yhteyksiä. IP-osoitteeksi asetetaan 0.0.0.0, jolloin palvelin kuuntelee yhteyksiä kaikista paikallisverkon sisällä olevista osoitteista. Portiksi numeroksi valitsin satunnaisen numeron 12345, joka ei ole käytössä muulla ohjelmalla. [5.]

```
HOST = '0.0.0.0'
PORT = 12345

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    server.bind((HOST, PORT))
except socket.error as e:
    print(f"Could not bind to port {PORT}! Error: {e}")
    exit()
except Exception as e:
    print(f"Error: {e}")
    exit()
```

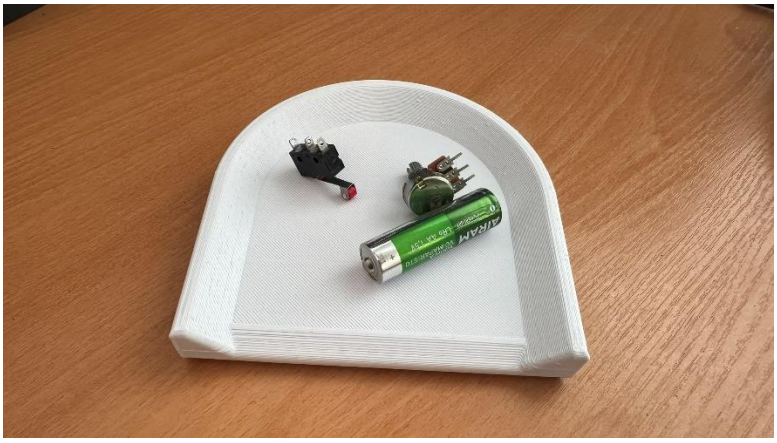
Kuva 15. TCP-palvelimen luonti ja portin sitominen

4.4 Tekoäly konenäöllä

Esineiden tunnistukseen käytin Ultralytics Python -kirjastoa. Ultralytics pohjautuu YOLO-tunnistusmallin, ja tekoälymallin kehittäminen on tehty todella helpoksi. Mallin luominen ei vaadi kuin yhden komennon tai koodin pätkän ja kirjasto hoitaa lopun [6]. Konenäköä varten käytetään opencv2 python -kirjastoa ja HD 1080p USB-kameraa.

4.4.1 Kuvat

Tekoälyn opettamista varten otin 74 kuvaa eri etäisyyksiltä, asennoissa ja valaistuksessa. Tällä tavoin saadaan malli tunnistamaan esineet isommalla todennäköisyydellä. Osa kuvista on hyvä laittaa tekoälyn validointia varten erilleen opetusdatasta. Laitan n.10 % kuvista ”valid”-nimiseen kansioon ja loput kuvista ”train”-nimiseen kansioon niiden merkitsemisen jälkeen.

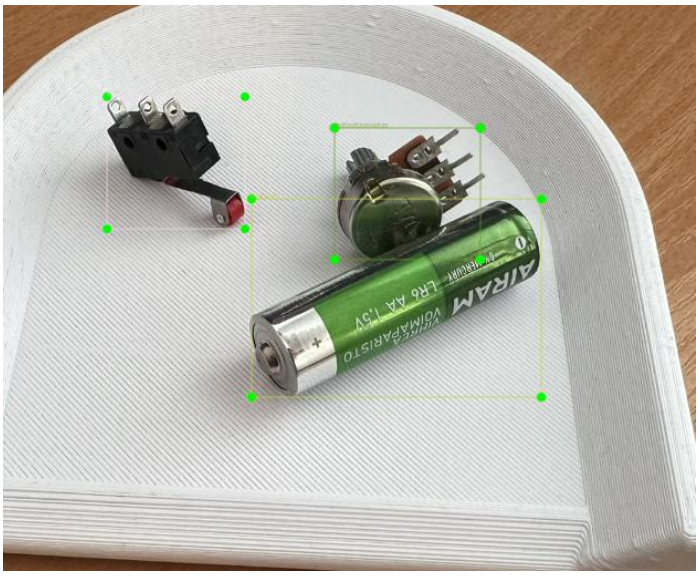


Kuva 16. Tekoälyn opetuskuva

4.4.2 Opettaminen

Tekoälyn opettamiseen käytin Ultralytics- ja yolov8n-mallia. n-malli eli nano on yolov8:n pienikokoisin ja nopein malli, mutta se oli yhtä tarkka kuin isommat mallit. Valitsin nano-mallin, koska sitä käytetään kannettavalla tietokoneella, jossa ei ole tehokasta prosessoria, eikä tekoälyä tukevaa näytönohjainta. Tekoäly opetetaan käyttämällä Google Colab -sivustoa, joka sopii akateemiseen, tekoäly- tai muuhun paljon tehoa vaativaan ohjelmointiin. [7.]

Esineiden merkitsemiseen kuvissa käytin ”labelImg” FOSS -ohjelmistoa. Ohjelman avulla kuviin voidaan merkitä niissä esiintyvät esineet oikeilla ”etiketillä”, jolloin tekoäly osaa etsiä yhtäläisyyksiä merkityiltä alueilta. Tällä tavoin tekoäly oppii tunnistamaan erilaisia esineitä. [8.]



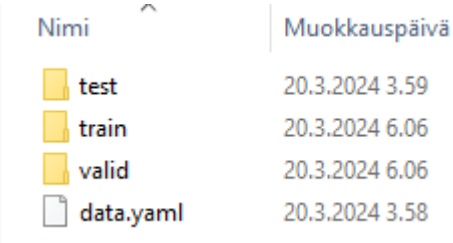
Kuva 17. Esineiden merkintä etiketeillä

Kun kaikki kuvissa esiintyvät esineet on merkitty, jaetaan ne kansioihin. Kansioiden sijainnilla tai nimillä ei ole väliä, sillä niiden sijainnit laitetaan data.yaml-tiedostoon (kuva 18), jolloin Ultralytics löytää kuvat. Hyvä tapa on kuitenkin laittaa kansiot ja data.yaml-tiedosto yhteen kansioon, joka helpottaa kuvien käyttämistä (kuva 19).
Pääkansio ladattiin Google Driveen, ja se pystytään yhdistämään Google Colabiin.

```
path: ../drive/MyDrive/thesis-ai-images/Dataset/Sorter
train: ../train/
val: ../valid/
test: ../test/

nc: 3
names: ['limitswitch', 'battery', 'potentiometer']
```

Kuva 18. data.yaml sisältämät asetukset



Kuva 19. Tarvittavat kansiot

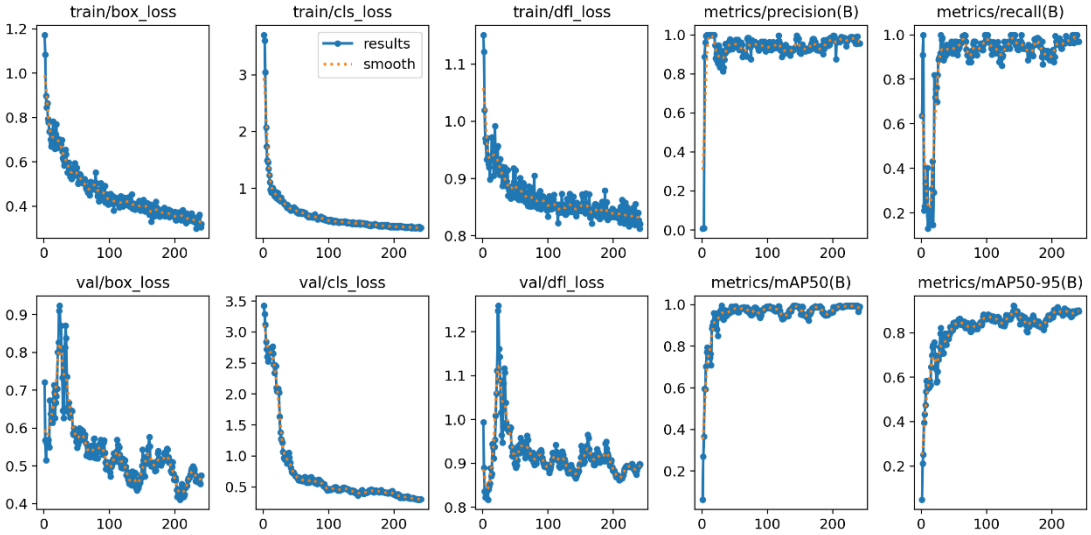
Ultralytics-kirjaston avulla saadaan tunnistusmalli tehtyä helposti yhdellä komennolla. Tärkeimmät asetukset ovat ”model”, ”data” ja ”epochs”. ”model”-asetukseksi laitetaan valmiiksi opetetun mallin nimi, joka tässä tapauksessa on ”yolov9n.pt”. Nano-malli on nopein ja pienin muista valittavista malleista, jonka ansiosta se toimii nopeammin heikompitehoisilla tietokoneilla. ”data”-asetukseen tulee oman opetusdatan tietoja pitävän tiedoston sijainti. ”epochs”-asetus tarkoittaa sitä, kuinka monta kertaa data käydään läpi opetuksen aikana. Tämä vaikuttaa opetusajan pituuteen sekä mallin tehokkuuteen. [9]

Tilastoista nähdään kuinka ajan mittaa tekoäly kehittyy ja sen tarkkuus paranee. Tärkeimpiä tilastoja ovat mAP50 ja mAP50-95. mAP50-tilasto kuvaa ns.

helppoja tunnistuksia, ja mAP50-95 kuvaa (kuva 21), kuinka hyvin malli tunnistaa eri vaikeusasteilla. Kuvasta nähdään, että tunnistukset tehdään yli 90 % varmuudella. [10]

```
!yolo task=detect mode=train model=yolov8n.pt data=../content/drive/MyDrive/thesis-ai-images/Dataset/Sorter/data.yaml epochs=400 imgsz=640
```

Kuva 20. Tekoälyn opetuksen komento Colabissa



Kuva 21. Opetetun tekoälyn tilastot

4.4.3 Testaus

Tekoälymallia kokeiltaessa huomasin, että se tunnistaa todella hyvin opetetut esineet eri kuvakulmissa ja eri etäisyyksiltä. Esineiden tunnistusvarmuus oli korkea, eikä se tehnyt vääriä tunnistuksia. Nyt tekoälymalli on valmis käytettäväksi valvomon käyttöliittymässä. (Kuva 22.)

Kuva 22. Esineiden tunnistuksen kokeilua

5 LAJITTELULAITTEEN SUUNNITTELU JA 3D-TULOSTUS

Lajittelijan 3D-tulostettavat osat suunniteltiin käyttämällä Autodesk Fusion360 -ohjelmaa. Mallilla ei ole muita vaatimuksia kuin taso, johon esineet laitetaan tunnistettavaksi, kiinnityspaikat moottoreille sekä teline kameralle. Askelmoottorin pitomomentti on 0,16Nm ja varren pituus on 4cm, joten nämä pitää ottaa huomioon tasoa suunniteltaessa [11]. Tason suurin paino, jonka se jaksaa pitää saadaan laskemalla newtonit (kaava 2) ja muuttamalla newtonit grammoiksi. Lopputulokseksi saadaan 408g, mutta tämä on askelmoottorin maksimi pitomomentti ja moottorin liikkuessa se laskee huomattavasti. Moottorin valmistajalta ei kuitenkaan löydy tästä minkäänlaista tietoa, joten taso joudutaan suunnittelemaan kokeilemalla.

$$M = F * r \tag{2}$$

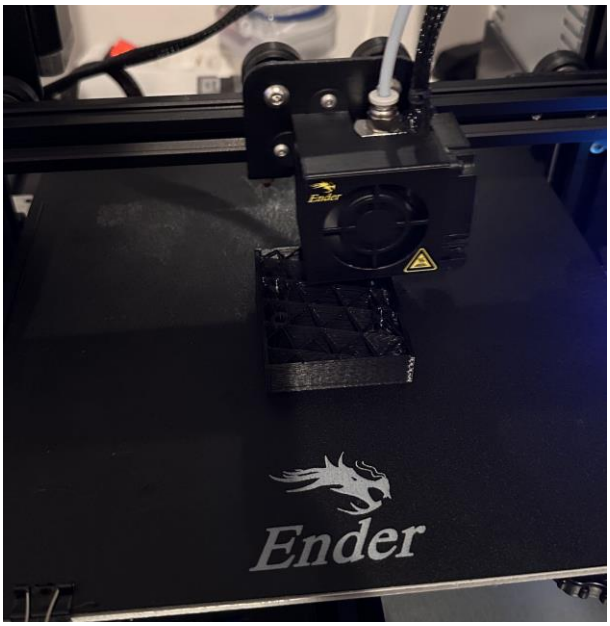
$$F = \frac{M}{r}$$

$$F = \frac{0{,}16Nm}{0{,}04m} = 4N$$

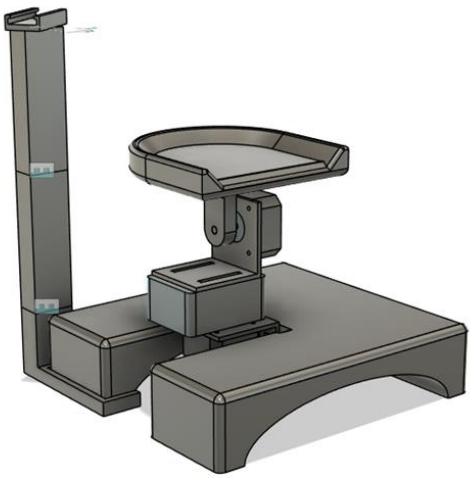
$$F = ma \tag{3}$$

$$m = \frac{F}{a} = \frac{4N}{9{,}8\frac{m}{s^2}} = 0{,}408kg = 408g$$

Lajittelijan tason painoksi tuli lopulta alle 100g, joka on ~25% maksimi painosta. Lajittelijan osien tulostamiseen käytin Creality Ender3 -3D-tulostinta ja 1,75 mm PLA-filamenttia, ja sitä kului noin 1 kg verran (kuva 23). Kameran teline ja lajittelijan jalka liimattiin yhteen käyttämällä pikaliimaa ja askelmoottoreiden kiinnikkeet laitettiin kiinni koneruuveilla. Valmiista lajittelijasta tuli siisti ja helposti kasattava. Kameran pidike on lajittelijan jalanympärillä kiinni kitkan avulla. (Kuva 24.)



Kuva 23. Osien tulostamista

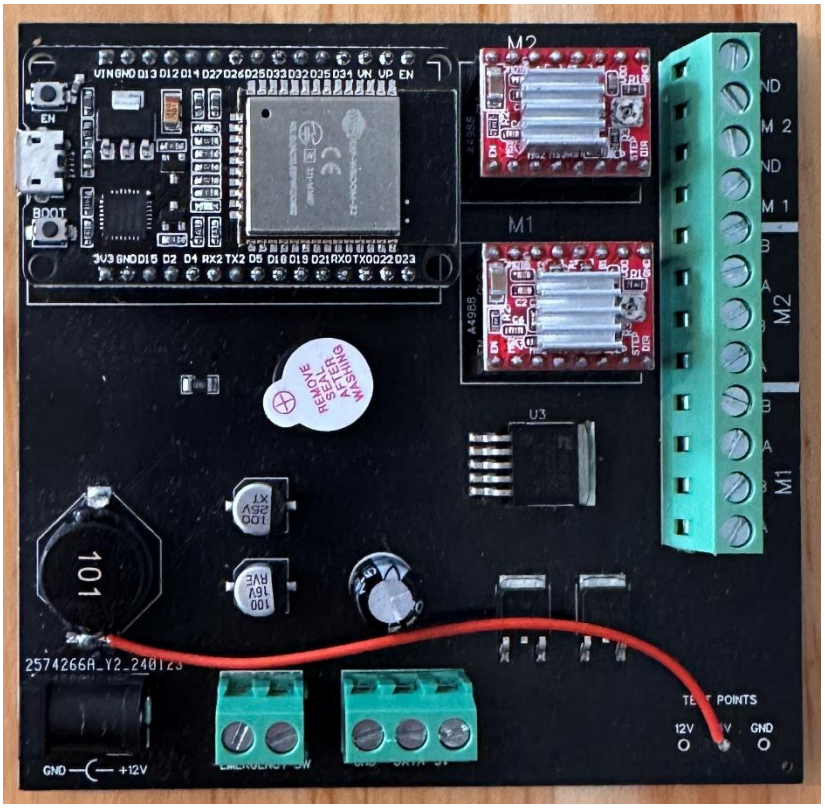


Kuva 24. Valmis lajittelijan 3D-malli

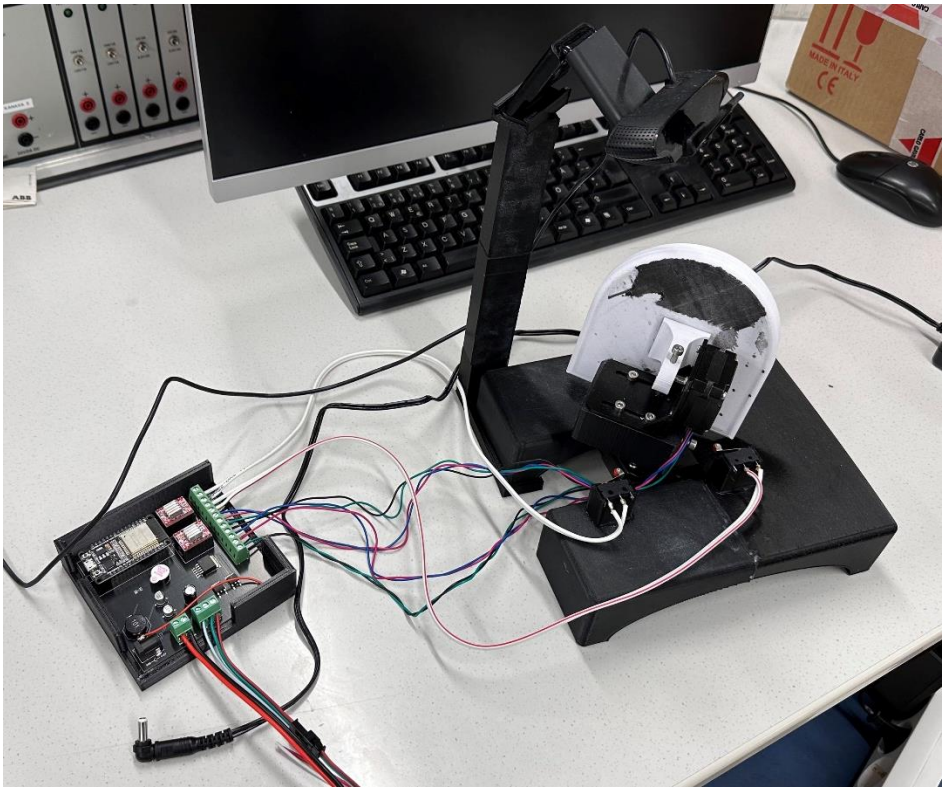
6 POHDINTA

Lajittelijan prototyypistä tuli toimiva kokonaisuus. Opin todella paljon uutta piirilevyjen valmistuksesta, tekoälystä ja ohjelmoinnista. Lajittelijaa voisi käyttää pienien kappaleiden lajittelussa esimerkiksi yritystoiminnassa. Piirilevyjen suunnittelun aikana oli ongelmia löytää, mitkä ESP32:n pinnit saavat olla kytkettyinä maahan ja mitä ei voida käyttää ollenkaan. Tämä johti siihen, että muutama ESP32 kärähti ja piirilevyt jouduttiin tilaamaan uudelleen. Uusilla piirilevyillä lähes kaikki toimi. Suunnittelun aikana +5V reitti oli jäänyt pois jännitteenalenninpiirin ja ESP32 Vin-pinnin väliltä, mutta onneksi tämän pystyi jälkikäteen korjaamaan pienellä pätkällä johtoa (kuva 25).

Lajittelijan kehityskohteita voisi olla USB-kameran vaihtaminen IP-kameraan, jolloin siitä saisi kokonaan langattoman. Kommunikointipalvelimen voisi päivittää käyttämään TLS/SSL-protokollaa, jolla saataisiin laitteiden välinen viestintä salattua. Työ käytiin esittelemässä Teemu Manniselle Mikkelin kampuksella (kuva 26). Kaiken kaikkiaan prototyypin kehittäminen onnistui hyvin ja opittuja asioita tullaan käyttämään tulevaisuudessa.



Kuva 25 Piirilevyn puuttuvan reitin korjaus



Kuva 26 Valmiin prototyypin esittely

LÄHTEET

[1] Harvin. The difference between tht and smt. WWW-dokumentti. Saatavissa: https://www.harwin.com/blog/the-difference-between-tht-and-smt/ [viitattu 8.5.2024].

[2] New York University. Understanding DC power supplies. WWW-dokumentti. Saatavissa: https://itp.nyu.edu/physcomp/lessons/electronics/understanding-dc-power-supplies/ [viitattu 8.5.2024].

[3] Pololu. A4988 Datasheet. WWW-dokumentti. Saatavissa: https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf [viitattu 14.5.2024].

[4] Omron. FAQ03206 of Basic Switches FAQ. WWW-dokumentti. Saatavissa: https://www.ia.omron.com/support/faq/answer/29/faq03206/index.html [viitattu 14.5.2024].

[5] Realpython. Socket API Overview. WWW-dokumentti. Saatavissa: https://realpython.com/python-sockets/#socket-api-overview [viitattu 10.5.2024].

[6] Github. Ultralytics. WWW-dokumentti. Saatavissa: https://github.com/ultralytics/ultralytics [viitattu 12.5.2024].

[7] Google. Colaboratory. WWW-dokumentti. Saatavissa: https://research.google.com/colaboratory/faq.html [viitattu 14.5.2024].

[8] Github. labelImg. WWW-dokumentti. Saatavissa: https://github.com/HumanSignal/labelImg [viitattu 14.5.2024].

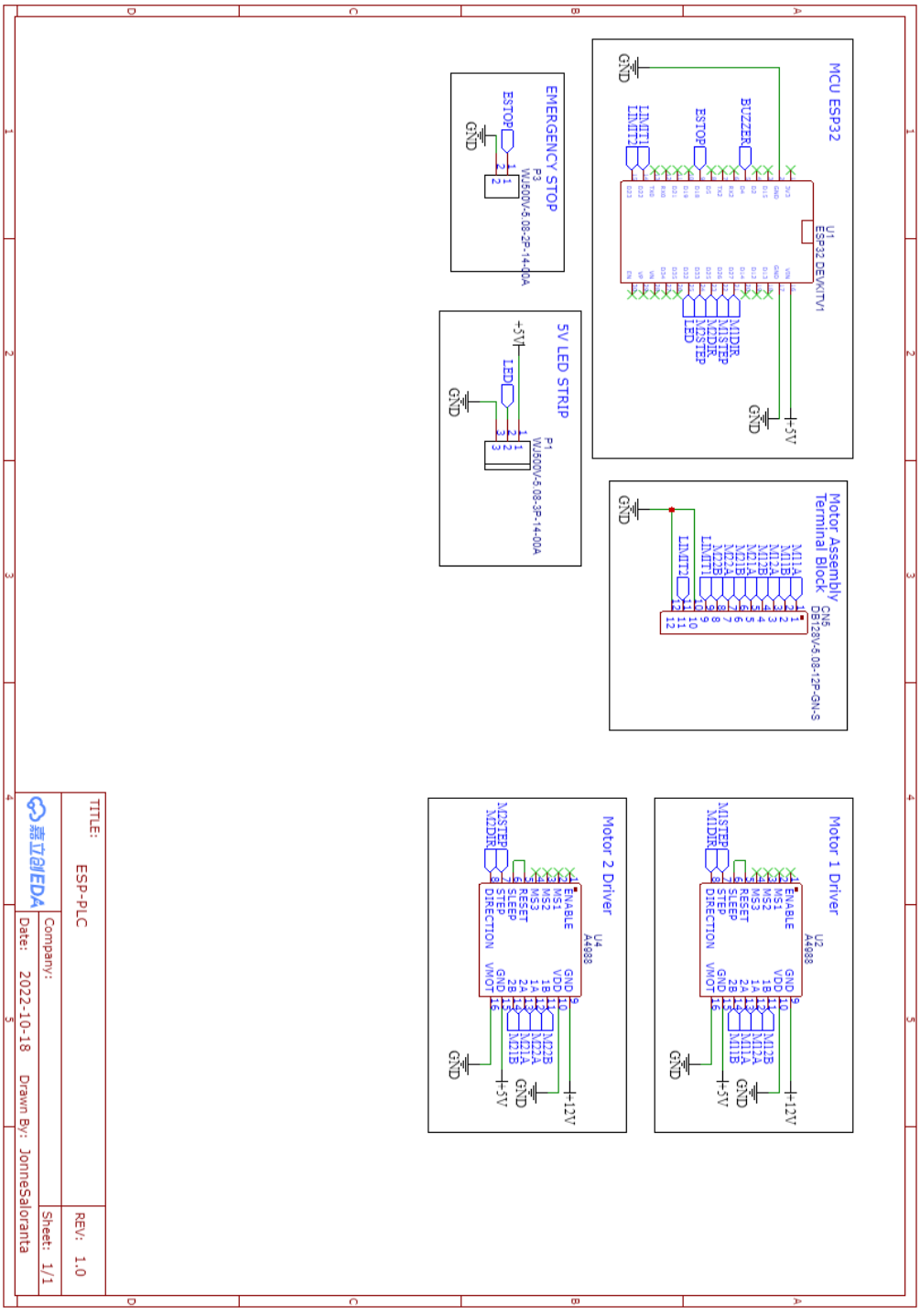
[9] Ultralytics. Model Training with Ultralytics YOLO. WWW-dokumentti. Saatavissa: https://docs.ultralytics.com/modes/train/#train-settings [viitattu 16.5.2024].

[10] Ultralytics. YOLO Performance Metrics. WWW-dokumentti. Saatavissa: https://docs.ultralytics.com/guides/yolo-performance-metrics/#class-wise-metrics [viitattu 14.5.2024].

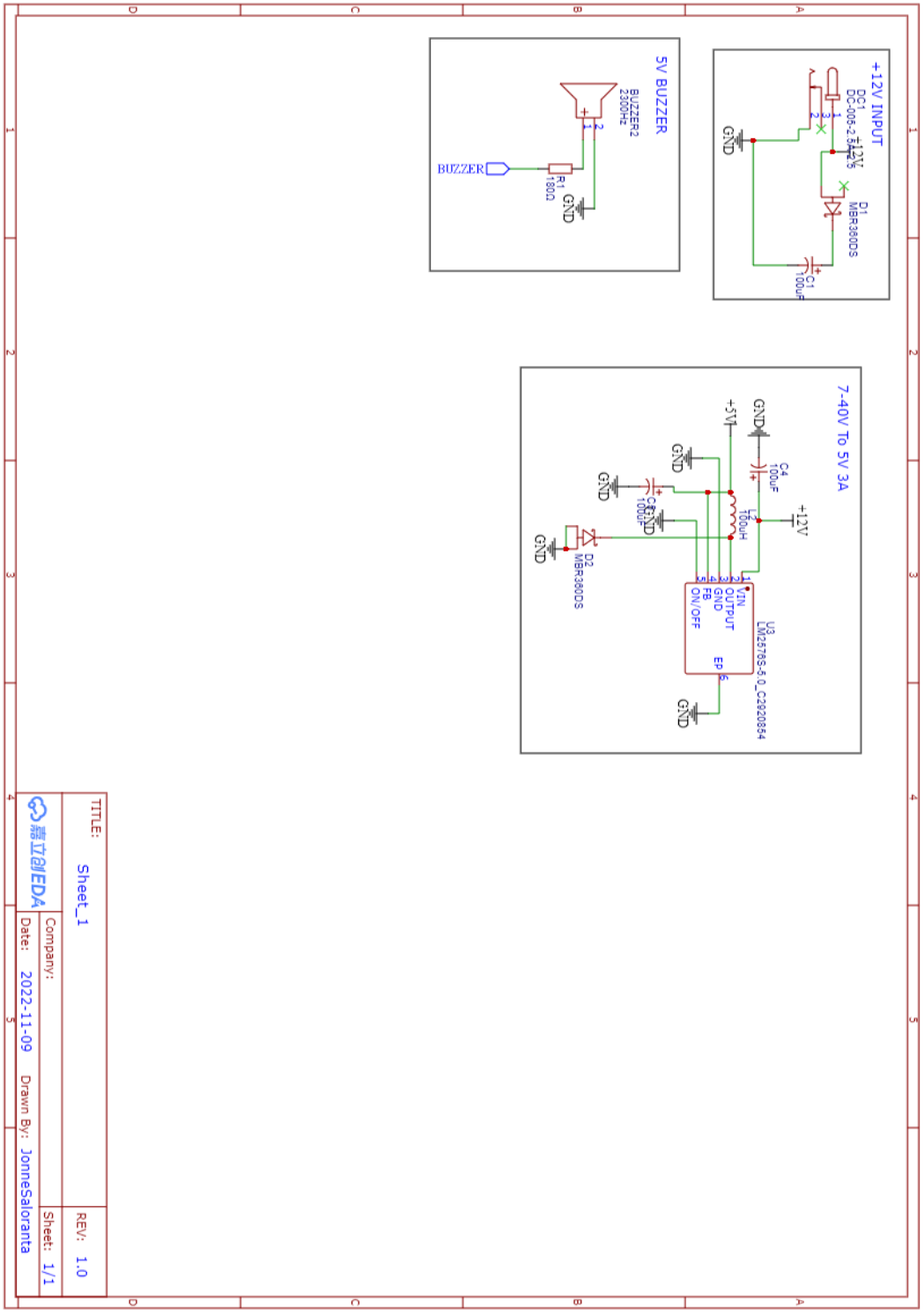
[11] Stepperonline. Askelmoottorin datalehti. WWW-dokumentti. Saatavissa: https://www.omc-stepperonline.com/index.php?route=product/product/get_file&file=3268/17HS08-1004S_Full_Datasheet.pdf [viitattu 16.5.2024].

KUVALUETTELO

Liite 1. Kytkentäkaavio 1



Liite 2. Kytkentäkaavio 2



Liite 3. ESP32-ohjelma

```
#include <WiFi.h>
#include <freertos/FreeRTOS.h>
#include <freertos/event_groups.h>
#include <FastLED_NeoPixel.h>
#include "BasicStepperDriver.h"

EventGroupHandle_t wifiEventGroup;
const int WIFI_CONNECTED_BIT = BIT0;

WiFiClient client;

const char *ssid = "ArduinoLabra";
const char *password = "12345678";

// const char *server_ip = "192.168.1.101";
const char *server_ip = "192.168.1.100";
const int server_port = 12345;

const int emergencyPin = 18;

const int buzzer = 4;
const int beep_delay = 500;

const int limit1 = 22;
const int limit2 = 23;

#define MOTOR_STEPS 200
#define M1RPM 30
#define M2RPM 30
#define MICROSTEPS 1

const int M1DIR = 27;
const int M1STEP = 26;
BasicStepperDriver stepper1(MOTOR_STEPS, M1DIR, M1STEP);

const int M2DIR = 25;
const int M2STEP = 33;
BasicStepperDriver stepper2(MOTOR_STEPS, M2DIR, M2STEP);

#define BUFFER_SIZE 1024
char buffer[BUFFER_SIZE];
int bufferIndex = 0;

const int ledPin = 32;
const int number_of_leds = 7;
const int brightness = 50;

unsigned long lastMillis = 0;

int max_brightness = 32;
```

```
CRGB leds[number_of_leds];
FastLED_NeoPixel_Variant strip(leds, number_of_leds);

uint32_t black = strip.Color(0, 0, 0);
uint32_t red = strip.Color(255, 0, 0);
uint32_t green = strip.Color(0, 255, 0);
uint32_t blue = strip.Color(0, 0, 255);
uint32_t yellow = strip.Color(255, 255, 0);
uint32_t purple = strip.Color(255, 0, 255);
uint32_t aqua = strip.Color(0, 255, 255);
uint32_t white = strip.Color(255, 255, 255);

enum State
{
    STOPPED,
    RUNNING,
    EMERGENCY,
    CONNECTING_TO_WIFI,
    CONNECTED_TO_WIFI,
    CONNECTING_TO_SERVER,
    CONNECTED_TO_SERVER
};

bool busy = false;

enum State process_state;

String command = "";

// Function declarations
void maintainWiFi(void *parameter);
void handleClient(void *parameter);

void clear_command()
{
    command = "";
}

void number_of_flashes(uint32_t color, int ledPin, int flashes, int
speed, bool used_in_core)
{
    strip.fill(color, 0, number_of_leds);
    for (int i = 0; i < flashes; i++)
    {
        strip.show();
        if (used_in_core)
        {
            vTaskDelay(speed / portTICK_PERIOD_MS);
        }
        else
        {
```

```
            delay(speed);
        }
        strip.fill(black, 0, number_of_leds);
        strip.show();
        if (used_in_core)
        {
            vTaskDelay(speed / portTICK_PERIOD_MS);
        }
        else
        {
            delay(speed);
        }
    }
}

void fade_flash(uint32_t color, int flashes, int speed, bool
used_in_core)
{

    strip.fill(color, 0, number_of_leds);
    for (int i = 0; i < flashes; i++)
    {
        for (int j = 0; j < 255; j++)
        {
            strip.setBrightness(j);
            strip.show();
            if (used_in_core)
            {
                vTaskDelay(speed / portTICK_PERIOD_MS);
            }
            else
            {
                delay(speed);
            }
        }
        for (int j = 255; j > 0; j--)
        {
            strip.setBrightness(j);
            strip.show();
            if (used_in_core)
            {
                vTaskDelay(speed / portTICK_PERIOD_MS);
            }
            else
            {
                delay(speed);
            }
        }
    }
}

void set_solid_color(uint32_t color)
```

```
{
    strip.fill(color, 0, number_of_leds);
    strip.show();
}

void set_led_color(uint32_t color)
{
    strip.fill(color, 0, number_of_leds);
    strip.show();
}

void set_led_color(uint32_t color, int led)
{
    strip.setPixelColor(led, color);
    strip.show();
}

void set_led_brightness(int brightness)
{
    strip.setBrightness(brightness);
    strip.show();
}

void setup_ready()
{
    tone(buzzer, 2500, 50);
    delay(200);
    noTone(buzzer);
    tone(buzzer, 500, 25);
    delay(200);
    noTone(buzzer);
    process_state = CONNECTING_TO_WIFI;
}

void ready_sound()
{
    tone(buzzer, 2500, 50);
    delay(200);
    noTone(buzzer);
    tone(buzzer, 500, 25);
    delay(200);
    noTone(buzzer);
}

void emergency_state()
{
    // Serial.println("Emergency state");
    number_of_flashes(red, ledPin, 1, 1000, true);
}

bool check_limit(int pin)
{
```

```
    return digitalRead(pin) != false;
}

void tip_holder()
{
    stepper2.rotate(-45);
    delay(500);
    stepper2.rotate(45);
    delay(500);
}

void left_90()
{
    stepper1.rotate(-90);
}

void right_90()
{
    stepper1.rotate(90);
}

void up_45()
{
    stepper2.rotate(45);
}

void down_45()
{
    stepper2.rotate(-45);
}

void calibrate()
{
    bool limit1Reached = false;
    bool limit2Reached = false;

    // Initialize stepper for calibration
    stepper2.rotate(180);
    delay(500);
    stepper2.rotate(-86);
    delay(500);

    // Continuously move stepper1 towards limit1
    while (!limit1Reached)
    {
        if (check_limit(limit1))
        {
            stepper1.move(1); // Move a single step towards limit1
            delay(10);        // Adjust delay for desired speed
        }
        else
        {
```

```
            stepper1.stop(); // Stop if limit switch is hit
            limit1Reached = true;
        }
    }

    delay(250);

    // Continuously move stepper1 towards limit2
    while (!limit2Reached)
    {
        if (check_limit(limit2))
        {
            stepper1.move(-1); // Move a single step towards limit2
            delay(10);         // Adjust delay for desired speed
        }
        else
        {
            stepper1.stop(); // Stop if limit switch is hit
            limit2Reached = true;
        }
    }

    delay(500);

    stepper1.rotate(95);
}

void setup()
{
    Serial.begin(115200);

    strip.begin(FastLED.addLeds<WS2812B, ledPin, GRB>(leds, num-
ber_of_leds));
    strip.setBrightness(max_brightness);

    // fade_flash(red, 1, 2, false);

    // pinMode(M1STEP, OUTPUT);
    // pinMode(M1DIR, OUTPUT);
    // pinMode(M2STEP, OUTPUT);
    // pinMode(M2DIR, OUTPUT);

    pinMode(limit1, INPUT_PULLUP);
    pinMode(limit2, INPUT_PULLUP);

    pinMode(emergencyPin, INPUT_PULLUP);

    pinMode(ledPin, OUTPUT);

    // set_direction(M1DIR, HIGH);
    // move_motor(M1STEP, 100, 2000);
```

```
    stepper1.begin(M1RPM, MICROSTEPS);
    stepper2.begin(M2RPM, MICROSTEPS);

    calibrate();

    tip_holder();

    // Create an event group
    wifiEventGroup = xEventGroupCreate();

    // Create tasks
    xTaskCreatePinnedToCore(
        maintainWiFi,
        "MaintainWiFi",
        10000,
        NULL,
        1,
        NULL,
        0);

    xTaskCreatePinnedToCore(
        handleClient,
        "HandleClient",
        10000,
        NULL,
        1,
        NULL,
        1);

    set_led_color(black);
    set_led_brightness(max_brightness);

    setup_ready();
}

void loop()
{
    // Empty loop, as tasks are handled in FreeRTOS tasks
}

void maintainWiFi(void *parameter)
{
    // Connect to WiFi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.println("Connecting to WiFi..");
        number_of_flashes(blue, ledPin, 1, 1000, true);
    }
    process_state = CONNECTED_TO_WIFI;
    Serial.println("Connected to the WiFi");
```

```
    number_of_flashes(blue, ledPin, 2, 500, true);

    // Signal that WiFi is connected
    xEventGroupSetBits(wifiEventGroup, WIFI_CONNECTED_BIT);

    for (;;)
    {

        if (digitalRead(emergencyPin) == HIGH && process_state != EMER-
GENCY)
        {
            process_state = EMERGENCY;
            continue;
        }

        if (command.startsWith("Reset"))
        {
            process_state = STOPPED;
            clear_command();
            continue;
        }

        if (command.startsWith("Emergency"))
        {
            process_state = EMERGENCY;
            clear_command();
            continue;
        }

        if (command.startsWith("Stop"))
        {
            process_state = STOPPED;
            clear_command();
            continue;
        }

        if (command.startsWith("Start"))
        {
            process_state = RUNNING;
            set_solid_color(green);
            clear_command();
            continue;
        }

        if (process_state == EMERGENCY)
        {
            emergency_state();
            continue;
        }

        if (process_state == STOPPED)
        {
```

```
        number_of_flashes(yellow, ledPin, 1, 1000, true);
        continue;
    }

    if (process_state == RUNNING)
    {
        if (command.startsWith("limitswitch:"))
        {
            if (!busy)
            {
                busy = true;
                vTaskDelay(1000 / portTICK_PERIOD_MS);
                left_90();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                down_45();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                up_45();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                right_90();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                clear_command();
                busy = false;
            }

            clear_command();
        }

        if (command.startsWith("battery:"))
        {
            if (!busy)
            {
                busy = true;
                vTaskDelay(1000 / portTICK_PERIOD_MS);
                down_45();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                up_45();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                clear_command();
                busy = false;
            }
        }

        if (command.startsWith("potentiometer:"))
        {
            if (!busy)
            {
                busy = true;
                vTaskDelay(1000 / portTICK_PERIOD_MS);
                right_90();
                vTaskDelay(100 / portTICK_PERIOD_MS);
                down_45();
                vTaskDelay(100 / portTICK_PERIOD_MS);
```

```
                    up_45();
                    vTaskDelay(100 / portTICK_PERIOD_MS);
                    left_90();
                    vTaskDelay(100 / portTICK_PERIOD_MS);
                    clear_command();
                    busy = false;
                }
            }
        }
        vTaskDelay(10 / portTICK_PERIOD_MS);
    }
}

void handleClient(void *parameter)
{
    // Wait for the WiFi connection
    xEventGroupWaitBits(wifiEventGroup, WIFI_CONNECTED_BIT, pdFALSE,
pdTRUE, portMAX_DELAY);

    // Now attempt to connect to the TCP server
    while (!client.connect(server_ip, server_port))
    {
        Serial.println("Connection to server failed. Is the server
online? retrying...");
        fade_flash(red, 1, 1, true);
        vTaskDelay(500 / portTICK_PERIOD_MS);
    }
    process_state = CONNECTED_TO_SERVER;
    Serial.println("Connected to a server");
    fade_flash(green, 1, 1, true);
    process_state = STOPPED;

    // Handle client communication
    while (true)
    {
        if (client.connected())
        {
            // Serial.println("Client connected");
            if (client.available())
            {
                // Serial.println("Client available");

                char c = client.read(); // Read a character
                // Serial.println(c);
                if (c == '\n' || c == '\r' || bufferIndex == BUFFER_SIZE
- 1)
                {
                    buffer[bufferIndex] = '\0'; // Null-terminate the
string
                    if (bufferIndex > 0)        // Check if there's
something in the buffer
                    {
```

```
                        command = String(buffer); // Assign to command
                        Serial.println(command);
                        // Reset buffer for the next command
                        bufferIndex = 0;
                        memset(buffer, 0, BUFFER_SIZE);
                    }
                }
                else
                {
                    buffer[bufferIndex++] = c; // Store character in
buffer
                }
            }
        }
        else
        {
            // Attempt to reconnect if the connection is lost
            while (!client.connect(server_ip, server_port))
            {
                Serial.println("Reconnecting to server...");
                process_state = CONNECTING_TO_SERVER;
                fade_flash(blue, 1, 1, true);
                vTaskDelay(10 / portTICK_PERIOD_MS);
            }
        }
        vTaskDelay(1 / portTICK_PERIOD_MS); // Small delay to prevent
task from hogging CPU
    }
}
```

Liite 4. Palvelimen ohjelma

```
import socket
import sys
import threading
from datetime import datetime

HOST = '0.0.0.0'
PORT = 12345

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
try:
    server.bind((HOST, PORT))
except socket.error as e:
    print(f"Could not bind to port {PORT}! Error: {e}")
    exit()
except Exception as e:
    print(f"Error: {e}")
```

```
    exit()


try:
    server.listen()
except Exception as e:
    print("Could not start listening!")
    exit()

print(f"Server listening on {HOST}:{PORT}")

clients = []

# def get_current_time():
#     now = datetime.now()
#     current_time = now.strftime("%H:%M:%S")
#     return current_time


def broadcast(message):
    for client in clients[:]:  # Iterate over a copy of the list
        try:
            client.send(message)
        except Exception as e:
            print(f"Error broadcasting to client: {e}")
            clients.remove(client)
            client.close()  # Ensure client socket is properly closed

def broadcast_except(message, sender):
    for client in clients:
        if client != sender:
            try:
                client.send(message)
            except Exception as e:
                print(f"Error broadcasting to client: {e}")
                clients.remove(client)

def handle_client(client):
    while True:
        try:
            message = client.recv(1024)
            if not message:
                raise ConnectionError("Client disconnected")
            # add newline to the end of the message
            decoded_message = message.decode('utf-8')
            decoded_message = decoded_message.strip() + '\n'
            broadcast(decoded_message.encode('utf-8'))
            print(f"Message from {client.getpeername()}: {decoded_mes-
sage}")
        except ConnectionError:
            client_address = client.getpeername()
            clients.remove(client)
```

```
            broadcast(f"Client {client_address} disconnected".en-
code('utf-8'))
            print(f"Client {client_address} disconnected")
            client.close()
            break
        except Exception as e:
            print(f"Error handling client: {e}")
            break

def receive():
    while True:
        client, address = server.accept()
        print(f"Connected with {str(address)}")
        clients.append(client)

        broadcast_except(f"> Client {str(address)} connected".en-
code('utf-8'), client)
        client.send(f"> Connected...".encode('utf-8'))

        thread = threading.Thread(target=handle_client, args=(client,))
        thread.start()

try:
    receive()
except KeyboardInterrupt:
    print("Server stopped!")
    exit()
```

Liite 5. Valvomon ohjelma

```
import logging
import tkinter as tk
from tkinter import messagebox
import cv2
from PIL import Image, ImageTk
import threading
from ultralytics import YOLO
import math
from enum import Enum
from src.settings import Settings
from src.labels import Labels
from src.connection_manager import SocketClient
import time

class State(Enum):
    IDLE = 0
    RUNNING = 1
    STOPPED = 2
    EMERGENCY_STOPPED = 3
```

```
    WARNING = 4

class MonitorClient:
    def __init__(self, window):
        self.labels = Labels()
        self.settings = Settings()
        self.socketclient = SocketClient()
        self.state = State.IDLE
        self.det_confidence = 0.4
        self.window = window
        self.window.title(f"{self.labels.get('window_title_label')}")
        self.window.geometry(self.settings.get("window_geometry"))
        self.now = int(round(time.time() * 1000))
        self.type = self.settings.get("type")
        self.model = None

        self.running = False
        self.cap = None
        self.connected_webcams = []

        # Create the GUI elements
        self.create_gui()


    def create_gui(self):
        # Create a frame to contain the GUI elements
        self.gui_frame = tk.Frame(self.window)
        self.gui_frame.pack()

        # Status label
        self.status_label = tk.Label(self.gui_frame, text=f"{self.la-
bels.get('status_label')}: {self.state.name}")
        self.status_label.grid(row=3, column=1, padx=10, pady=10,
sticky="w")

        # Dropdown for selecting webcams
        self.webcam_dropdown = tk.StringVar(self.gui_frame)
        self.webcam_dropdown.set(f"{self.labels.get('select_webcam_la-
bel')}")
        self.update_webcam_list()

        self.webcam_menu = tk.OptionMenu(self.gui_frame,
self.webcam_dropdown, *self.connected_webcams)
        self.webcam_menu.grid(row=3, column=2, columnspan=2, padx=10,
pady=10, sticky="w")

        # Webcam screen
        self.webcam_screen = tk.Label(self.gui_frame, bg="black")
        self.set_screen_blank()
        self.webcam_screen.grid(row=0, column=0, columnspan=4, padx=10,
pady=10)
```

```
        # Server connection settings
        self.server_ip_label = tk.Label(self.gui_frame, text=f"{self.la-
bels.get('server_ip_label')}")
        self.server_ip_label.grid(row=1, column=0, padx=10, pady=10,
sticky="w")

        self.server_ip_text = tk.StringVar(self.gui_frame,
value=f"{self.settings.get('conn_ip')}")
        self.server_ip_entry = tk.Entry(self.gui_frame, textvaria-
ble=self.server_ip_text)
        self.server_ip_entry.grid(row=1, column=1, padx=10, pady=10,
sticky="w")

        self.server_port_label = tk.Label(self.gui_frame,
text=f"{self.labels.get('server_port_label')}")
        self.server_port_label.grid(row=2, column=0, padx=10, pady=10,
sticky="w")

        self.server_port_text = tk.StringVar(self.gui_frame,
value=f"{self.settings.get('conn_port')}")
        self.server_port_entry = tk.Entry(self.gui_frame, textvaria-
ble=self.server_port_text)
        self.server_port_entry.grid(row=2, column=1, padx=10, pady=10,
sticky="w")

        self.server_connect_button = tk.Button(self.gui_frame,
text=f"{self.labels.get('connect_button_label')}", command=self.con-
nect_to_server)
        self.server_connect_button.grid(row=3, column=0, columnspan=2,
padx=10, pady=10, sticky="w")

        self.console_box = tk.Text(self.gui_frame, width=50, height=20)
        self.console_box.grid(row=0, column=5, columnspan=4, padx=10,
pady=10, sticky="w")
        self.console_box.config(state="disabled")

        #self.draw_boxes_checkbox = tk.Checkbutton(self.gui_frame,
text=f"{self.labels.get('draw_boxes_label')}")
        #TODO: Add functionality to toggle drawing detection boxes

        # Buttons
        self.connect_camera_button = tk.Button(self.gui_frame,
text=f"{self.labels.get('camera_open_label')}", command=self.start_cam-
era)
        self.start_button = tk.Button(self.gui_frame, text=f"{self.la-
bels.get('start_button_label')}", command=self.start)
        self.stop_button = tk.Button(self.gui_frame, text=f"{self.la-
bels.get('stop_button_label')}",command=self.stop)
        self.emergency_button = tk.Button(self.gui_frame,
text=f"{self.labels.get('emergency_stop_label')}", command=self.emer-
gency_stop)
```

```
        self.load_detection_model_button = tk.Button(self.gui_frame,
text=f"{self.labels.get('load_model_label')}", command=self.load_detec-
tion_model)
        # self.load_cameras_button = tk.Button(self.gui_frame,
text=f"{self.labels.get('load_cameras_label')}", command=self.up-
date_webcam_list)
        self.reset_button = tk.Button(self.gui_frame, text=f"{self.la-
bels.get('reset_button_label')}", command=self.reset)

        self.connect_camera_button.grid(row=0, column=4, padx=10,
pady=10, sticky="w")
        self.connect_camera_button.config(state="disabled")

        self.start_button.grid(row=2, column=2, padx=10, pady=10,
sticky="w")
        self.stop_button.grid(row=2, column=3, padx=10, pady=10,
sticky="w")
        self.emergency_button.grid(row=3, column=3, columnspan=2,
padx=10, pady=10, sticky="w")
        self.load_detection_model_button.grid(row=4, column=1, padx=10,
pady=10, sticky="w")
        self.reset_button.grid(row=4, column=2, padx=10, pady=10,
sticky="w")



        self.start_button.config(state="disabled")
        self.stop_button.config(state="disabled")
        self.server_connect_button.config(state="disabled")



        # self.load_cameras_button.grid(row=4, column=1, padx=10,
pady=10, sticky="w")

    def update_webcam_list(self):
        self.connected_webcams = []
        self.connected_webcams.append(f"{self.labels.get('se-
lect_webcam_label')}")

        self.connected_webcams = [f"{self.labels.get('camera_number_la-
bel')} {i}" for i in range(self.settings.get('cameras_to_load')) if
self.check_camera_available(i)]
        if len(self.connected_webcams) == 0:
            self.connected_webcams = [f"{self.labels.get('no_webcam_la-
bel')}"]

        else:
            self.webcam_dropdown.set(self.connected_webcams[0])

    def check_camera_available(self, index):
        cap = cv2.VideoCapture(index)
```

```
        if not cap.isOpened():
            return False
        else:
            ret, frame = cap.read()
            cap.release()
            return ret and frame is not None

    def update_frame(self):
        while self.running:
            ret, frame = self.cap.read()
            if ret:
                # Convert to RGB for PIL compatibility
                rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                pil_image = Image.fromarray(rgb_frame)

                # Perform object detection
                results = self.model(pil_image, conf=self.set-
tings.get("ai_confidence"))

                # annotated_frame = results[0].plot(boxes=False)
                annotated_frame = results[0].plot(boxes=True)  # plot
the results
                detected_objects = []
                confidence = 0
                cls = 0

                for r in results:
                    boxes = r.boxes

                    for box in boxes:
                        confidence = math.ceil((box.conf[0]*100)) / 100
                        cls = int(box.cls[0])
                        # print(f"{self.class_names[cls]}: {confi-
dence}")
                        detected_objects.ap-
pend(f"{self.class_names[cls]}: {confidence}")
                self.console_box.config(state="normal")
                self.console_box.delete(1.0, tk.END)
                self.console_box.insert(tk.END, f"{detected_objects}")
                self.console_box.see(tk.END)
                self.console_box.config(state="disabled")

                if self.now + 250 < int(round(time.time() * 1000)):
                    self.now = int(round(time.time() * 1000))
                    if self.socketclient.is_connected() and self.state
== State.RUNNING:
                        if len(detected_objects) > 0:
                            # self.socketclient.send_mes-
sage(f"{self.class_names[cls]}: {confidence}")
                            self.socketclient.send_message(f"{de-
tected_objects[0]}\n")
```

```
                # Convert the annotated frame back to ImageTk format
                annotated_image = Image.fromarray(cv2.cvtColor(anno-
tated_frame, cv2.COLOR_BGR2RGB))
                frame_tk = ImageTk.PhotoImage(image=annotated_image)

                # Update the webcam screen with the modified frame
                self.webcam_screen.configure(image=frame_tk)
                self.webcam_screen.image = frame_tk

            self.window.update()
        self.set_screen_blank()

    def load_detection_model(self):
        self.model = YOLO(self.settings.get("ai_model", "yolov8n.pt"))
        self.class_names = self.model.names
        self.connect_camera_button.config(state="normal")
        self.load_detection_model_button.config(state="disabled")

    def start_camera(self):
        if self.running:
            messagebox.showinfo("Info", f"{self.labels.get('camera_al-
ready_running')}")
            return

        selected_webcam = int(self.webcam_dropdown.get().split(" ")[-1])

        self.connect_camera_button.config(text=f"{self.labels.get('open-
ing_camera')}", state="disabled")

        # Initialize and start the camera in a separate thread
        threading.Thread(target=self.initialize_camera, args=(se-
lected_webcam,), daemon=True).start()

    def initialize_camera(self, camera_index):
        self.cap = cv2.VideoCapture(camera_index)

        if not self.cap.isOpened():
            messagebox.showerror("Error", f"{self.labels.get('cam-
era_failed_to_open')}")
            return

        # Set the size of the webcam screen based on the webcam's reso-
lution
        self.webcam_screen.config(width=640, height=480)

        self.running = True
        self.window.after(0, self.set_camera_button_to_stop)

        self.server_connect_button.config(state="normal")
        self.start_button.config(state="normal")
        self.stop_button.config(state="normal")
```

```
        # Start updating frames in a separate thread
        threading.Thread(target=self.update_frame, daemon=True).start()

    def set_camera_button_to_stop(self):
        self.connect_camera_button.config(text=f"{self.labels.get('cam-
era_stop')}", command=self.stop_camera, state="normal")
        self.set_screen_blank()

    def stop_camera(self):
        if self.running:
            self.running = False
            self.connect_camera_button.config(text=f"{self.la-
bels.get('camera_open_label')}", command=self.start_camera, state="nor-
mal")
            self.socketclient.disconnect()
            self.server_connect_button.config(text=f"{self.la-
bels.get('connect_to_server_button')}", command=self.connect_to_server)
            self.console_box.config(state="normal")
            self.console_box.delete(1.0, tk.END)
            self.console_box.insert(tk.END, f"Disconnected from server
due to camera stop")
            self.console_box.see(tk.END)
            self.console_box.config(state="disabled")
            self.cap.release()

    def emergency_stop(self):
        self.status_label.config(text=f"{self.labels.get('status_la-
bel')}: {State.EMERGENCY_STOPPED.name}")
        self.socketclient.send_message("Emergency")
        self.state = State.EMERGENCY_STOPPED

        self.start_button.config(state="disabled")
        self.stop_button.config(state="disabled")
        self.emergency_button.config(state="disabled")
        self.reset_button.config(state="normal")

    def start(self):
        self.status_label.config(text=f"{self.labels.get('status_la-
bel')}: {State.RUNNING.name}")
        self.socketclient.send_message("Start")
        self.state = State.RUNNING
        self.start_button.config(state="disabled")
        self.stop_button.config(state="normal")
        self.emergency_button.config(state="normal")
        self.reset_button.config(state="normal")

    def stop(self):
        self.status_label.config(text=f"{self.labels.get('status_la-
bel')}: {State.STOPPED.name}")
        self.socketclient.send_message("Stop")
        self.state = State.STOPPED
```

```
        self.start_button.config(state="normal")
        self.stop_button.config(state="disabled")
        self.emergency_button.config(state="normal")
        self.emergency_button.config(state="normal")

    def reset(self):
        self.status_label.config(text=f"{self.labels.get('status_la-
bel')}: {State.IDLE.name}")
        self.socketclient.send_message("Reset")
        self.state = State.IDLE

        self.start_button.config(state="normal")
        self.stop_button.config(state="normal")
        self.emergency_button.config(state="normal")
        self.reset_button.config(state="normal")

    def set_screen_blank(self):
        blank_image = Image.new('RGB', (640, 480), (0, 0, 0))
        blank_photo = ImageTk.PhotoImage(blank_image)
        self.webcam_screen.configure(image=blank_photo)
        self.webcam_screen.image = blank_photo

    def connect_to_server(self):
        ip = self.server_ip_text.get()
        port = int(self.server_port_text.get())

        logging.info(f"Connecting to
{self.server_ip_text.get()}:{self.server_port_text.get()}")
        self.server_connect_button.config(text=f"{self.labels.get('dis-
connect_from_server_button')}", command=self.disconnect_from_server)
        threading.Thread(target=lambda: self.socketclient.connect(ip,
port), daemon=True).start()

    def disconnect_from_server(self):
        logging.info(f"Disconnecting from
{self.server_ip_text.get()}:{self.server_port_text.get()}")
        self.socketclient.disconnect()
        self.server_connect_button.config(text=f"{self.labels.get('con-
nect_to_server_button')}", command=self.connect_to_server)

    def on_close(self):
        if messagebox.askokcancel("Quit", f"{self.labels.get('con-
firm_exit')}"):
            self.stop_camera()
            self.socketclient.disconnect()
            self.window.destroy()

if __name__ == "__main__":
    import cProfile
    import pstats
```

```
    debug = False

    if debug:
        pr = cProfile.Profile()
        pr.enable()

    window = tk.Tk()
    app = MonitorClient(window)
    logging.basicConfig(level=logging.INFO)
    window.protocol("WM_DELETE_WINDOW", app.on_close)
    window.mainloop()

    if debug:
        pr.disable()
        stats = pstats.Stats(pr)
        stats.sort_stats('cumulative').print_stats(10)  # Print the top
10 time-consuming functions
```

Liite 6. Palvelimen yhteydenhallintaohjelma

```
import socket
from src.labels import Labels
from src.settings import Settings
import threading

class SocketClient:
    def __init__(self):
        self.labels = Labels()
        self.settings = Settings()

        self.host = None
        self.port = None
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.connected = False
        self.running = False

        self.receive_thread = threading.Thread(target=self.receive, dae-
mon=True)
        self.send_thread = threading.Thread(target=self.send, dae-
mon=True)

    def is_connected(self):
        return self.connected

    def connect(self, ip, port):
        self.client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.host = ip
        self.port = port
        try:
            self.client.connect((self.host, self.port))
            self.connected = True
```

```
            self.running = True

            # Start threads only after successful connection
            self.receive_thread = threading.Thread(target=self.receive,
daemon=True)
            self.receive_thread.start()
            self.send_thread = threading.Thread(target=self.send, dae-
mon=True)
            self.send_thread.start()
        except socket.error as e:
            print(f"{self.labels.get('error_connecting')}{e}")

    def receive(self):
        while self.running:
            try:
                message = self.client.recv(1024).decode('utf-8')
                if not message:
                    # print(f"{self.labels.get('server_disconnected')}")
                    # self.disconnect()
                    break
                print(message)
            except Exception as e:
                print(f"{self.labels.get('error_occured')}{e}")
                break


    def send(self):
        while self.running:
            try:
                message = input('')
                if message:
                    self.client.send(message.encode('utf-8'))
                else:
                    break
            except Exception as e:
                print(f"{self.labels.get('error_occured')}{e}")
                break

    def send_message(self, message):
        if not self.connected:
            print(f"{self.labels.get('not_connected_to_server')}")
            return

        try:
            self.client.send(message.encode('utf-8'))
        except Exception as e:
            print(f"{self.labels.get('error_occured')}{e}")
            self.disconnect()


    def disconnect(self):
        if self.connected:
```

```
            self.running = False
            self.client.close()
            self.connected = False
            print(f"{self.labels.get('disconnected_from_server')}")


    def set_ip(self, ip):
        self.host = ip
        self.settings.set("conn_ip", ip)
        self.settings.save()

    def set_port(self, port):
        self.port = port
        self.settings.set("conn_port", port)
        self.settings.save()

    def is_connected(self):
        return self.connected

if __name__ == "__main__":
    socketclient = SocketClient()
    socketclient.connect('127.0.0.1', 12345)
    try:
        while socketclient.is_connected():
            message = input('')
            if message:
                socketclient.send_message(message)
            else:
                break
    except KeyboardInterrupt:
        socketclient.disconnect()
```

Liite 7. Valvomon tekstienhallintaohjelma

```
import logging
from src.setting_manager import SettingsManager

class Labels:

    def __init__(self):
        self.mgr = SettingsManager("messages.yaml")

        messages = {
            "window_title_label": "AI Sorter",
            "confirm_exit": "Do you want to quit?",
            "start_button_label": "Start",
            "stop_button_label": "Stop",
            "emergency_stop_label": "Emergency Stop",
            "server_ip_label": "Server IP:",
            "server_port_label": "Server Port:",
```

```
            "status_label": "Status:",
            "select_webcam_label": "Select Webcam",
            "no_webcam_label": "No webcams available",
            "camera_number_label": "Camera:",
            "camera_already_running": "Webcam is already running.",
            "camera_failed_to_open": "Failed to open the webcam.",
            "camera_stop": "Stop Camera",
            "camera_open_label": "Open Camera",
            "camera_close_label": "Close Camera",
            "error_occured": "Error occured!",
            "error_connecting": "Could not connect to server! Error: ",
            "connected_to_server": "Connected to ",
            "load_model_label": "Load Detection Model",
            "load_cameras_label": "Load Cameras",
            "connect_to_server_button": "Connect...",
            "disconnect_from_server_button": "Disconnect",
            "server_disconnected": "Server disconnected!",
            "connect_button_label": "Connect",
            "disconnect_button_label": "Disconnect",
            'not_connected_to_server': 'Not connected to server!',
            'disconnected_from_server': 'Disconnected from server!',
            'opening_camera': 'Opening camera...',
            'draw_boxes_label': 'Draw boxes',
            'reset_button_label': 'Reset',
        }
        self.settings = self.mgr.load_or_create_settings(messages)

    def get(self, key, default=None):
        # Splitting the key to support nested dictionaries
        keys = key.split('.')
        value = self.settings
        try:
            for k in keys:
                value = value[k]
            return value
        except KeyError:
            logging.warning(f"Key '{key}' not found in settings file.")
            return default


if __name__ == "__main__":
    logging.basicConfig(level=logging.WARNING)
    labels = Labels()
    print(labels.get("window_title_label"))
    print(labels.get("server_port_label"))
    print(labels.get("this_is_missing"))
```

Liite 8. Valvomon asetustenhallintaohjelma

```
import os
```

```
import yaml

class SettingsManager():
    def __init__(self, config_file):
        self.config_file = config_file

    def load_or_create_settings(self, settings):
        if not os.path.exists(os.path.join(os.getcwd(), self.con-
fig_file)):
            with open(self.config_file, 'w') as file:
                yaml.dump(settings, file, default_flow_style=False)
            return settings
        else:
            # check for missing keys and if they are missing, add them
to the settings file and then return the settings
            with open(os.path.join(os.getcwd(), self.config_file), 'r')
as file:
                loaded_settings = yaml.safe_load(file)
                for key in list(loaded_settings.keys()):
                    if key not in settings:
                        print(f"Removing key {key} from settings file.")
                        del loaded_settings[key]
                for key in settings:
                    if key not in loaded_settings:
                        print(f"Key {key} not found in settings file.
Adding it with default value '{settings[key]}'")
                        loaded_settings[key] = settings[key]
                with open(self.config_file, 'w') as file:
                    yaml.dump(loaded_settings, file, de-
fault_flow_style=False)
                return loaded_settings

    def get_setting(self, key, default=None):
        # Splitting the key to support nested dictionaries
        keys = key.split('.')
        value = self.settings
        try:
            for k in keys:
                value = value[k]
            return value
        except KeyError:
            print(f"Key {key} not found in settings file.")
            return default
```

Liite 9. Valvomon asetustiedoston luomisohjelma

```
from src.setting_manager import SettingsManager

class Settings:
```

```
    def __init__(self):
        self.mgr = SettingsManager("settings.yaml")

        settings = {
            "type": "1",
            "conn_ip": "127.0.0.1",
            "conn_port": 12345,
            "default_camera": 0,
            "cameras_to_load": 5,
            "window_width": 1280,
            "window_height": 720,
            "ai_model": "yolov8n.pt",
            "ai_confidence": 0.4,
            "draw_boxes": True,
        }
        self.settings = self.mgr.load_or_create_settings(settings)

    def get(self, key, default=None):
        # Splitting the key to support nested dictionaries
        keys = key.split('.')
        value = self.settings
        try:
            for k in keys:
                value = value[k]
            return value
        except KeyError:
            return default


if __name__ == "__main__":
    settings = Settings()
```

Liite 10. Tekoälyn opetuskomennot

```
# %%
!nvidia-smi

# %%
from google.colab import drive
#drive.mount('/content/drive')

# %%
!ls

# %%
!pip install ultralytics

# %%
from ultralytics import YOLO
```

```
# %%
!yolo task=detect mode=train model=yolov8n.pt data=../con-
tent/drive/MyDrive/thesis-ai-images/Dataset/Sorter/data.yaml epochs=400
imgsz=640

# %%
!ls

# %%
!zip -r detect.zip runs/detect/

# %%
from google.colab import files
files.download('detect.zip')
```

Liite 11. data.yaml-tiedoston sisältö

```
path: ../drive/MyDrive/thesis-ai-images/Dataset/Sorter
train: ../train/
val: ../valid/
test: ../test/

nc: 3
names: ['limitswitch', 'battery', 'potentiometer']
```