Tomi Palomäki

Hihnakuljettimen automatisoitu energiakulutuksen optimointi tekoälyalgoritmin avulla

Opinnäytetyö
Kevät 2020
SeAMK Tekniikka
Insinööri (AMK), Automaatiotekniikka

SeAMK
SEINÄJOEN AMMATTIKORKEAKOULU
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Seinäjoen ammattikorkeakoulu

Tutkinto-ohjelma: Automaatiotekniikka

Suuntautumisvaihtoehto: Sähköautomaatio

Tekijä: Tomi Palomäki

Työn nimi: Hihnakuljettimen automatisoitu energiakulutuksen optimointi tekoälyalgoritmin avulla

Ohjaaja: Mikko Ylihärsilä

Vuosi: 2020 Sivumäärä: 41

Työ tehtiin Seinäjoen ammattikorkeakoulun Mixed reality & collaborative robotics -hankkeeseen demonstraationa.

Tavoitteena oli selvittää, miten pienellä datamäärällä on mahdollista optimoida kuljetinhihnan energiankulutus, sekä selvittää, mitä vaiheita täysin automatisoitu keskitetty hihnakuljettimen datankeruujärjestelmä vaatii, kun kuljetin ei ole tuotannossa kiinni ja käytetään apuna yhteistyörobottia testikappaleiden lastaamisessa.

Teoriaosuudessa käsitellään aiheita, joita automaattisessa keskitetyssä datankeruujärjestelmän ohjelmoinnissa python-ohjelmointikielellä vaaditaan, sekä neuroverkkojen teoriaa ja esitellään työssä käytetty python-moduuli, joka sisältää työssä käytetyn neuroverkkoalgoritmin.

Käytännön osuudessa käydään läpi työn eri vaiheet, jotka on jaoteltu jokaisessa aihealueessa siten, että ensin esitellään vaatimukset, joita seuraa niiden toteuttaminen. Työssä ohjelmoitiin python-sovellus, joka sisältää tietokannan, neuroverkon, automaattisen kuljettimen sekä robotin ohjaamisen Ethernet-verkon yli.

Työssä selvisi, minkä verran tarvitaan dataa, että voidaan hyödyntää onnistuneesti neuroverkkoa hihnakuljettimen energiankulutuksen optimoinnissa sekä mitä vaiheita keskitetyn datankeruujärjestelmän ohjelmoiminen vaatii. Työstä tuotettiin myös videodokumentti.

Avainsanat: hihnakuljetin, kuljetin, neuroverkko, energiankulutus, optimointi, data, tekoäly, tietokanta, python, pieni data, ohjelmointi

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Automation Engineering

Specialisation: Electric Automation

Author: Tomi Palomäki

Title of thesis: Automated Energy Consumption Optimization of a Belt Conveyor Using an Artificial Intelligence Algorithm

Supervisor: Mikko Ylihärsilä

Year: 2020 Number of pages: 41

The work was done as a demonstration for Seinäjoki University of Applied Sciences Mixed reality & Collaborative Robotics project. The aim was to see how little data is needed to optimize the energy consumption of the conveyor belt, and to find out which steps a fully automated centralized belt conveyor data acquisition system requires when the conveyor is not in production and a collaborative robot is used to load test pieces.

The theory part concentrated on the topics required when programming an automatic centralized data acquisition system in the Python programming language. Also the theory of neural networks was studied and the Python module used in the work was introduced. The python module included the neural network algorithm used in the work.

The practical part reviewed the different stages of the work, which were divided in each topic area so that first the requirements were introduced and then followed their implementation. A Python application was programmed, including a database, neuro network, automatic conveyor and controlling a robot over an Ethernet network.

The thesis revealed how much data is needed to successfully utilize the neural network when optimizing the energy consumption of the belt conveyor and which steps are required to program a centralized data collection system. A video documentary was also produced about the work.

Keywords: belt conveyor, conveyor, neural network, energy consumption, optimization, data, artificial intelligence, database, python, small data, programming

SISÄLTÖ

Kuvaluettelo

Käytetyt termit ja lyhenteet

Kirjasto	Kirjasto sisältää valmiita ohjelmamoduuleita eri tarkoituksiin.
Luokka	Luokka on ohjelmoinnissa nimitys olio-ohjelmoinnissa ohjelmakoodin osalle, jossa luodaan olioita määriteltyjen ohjeiden perusteella.
Metodi	Metodi on ohjelmakoodissa lohko, jossa määritellään, miten asiat metodin sisällä tehdään eli se sisältää aina jotain tekemistä.
Modbus	Modbus on master-slave-sarjaliikenneprotokolla, jota käytetään eri laitteiden välisessä kommunikoinnissa.
Muuttuja	Muuttuja on ohjelmoinnissa tiedon tallennusvarasto, jolle annetaan nimi, ja jonka arvo voi nimensä mukaan muuttaa arvoaan sen mukaan, miten muuttujan toiminta ohjelmakierrossa määritellään.
Olio	Olio on ohjelmoinnissa käytetty tietorakenne, joka sisältää tietoja sekä metodeja tietojen käsittelyyn
Palvelinsovellus	Palvelinsovellus prosessoi asiakassovelluksen lähettämän tiedon ja palauttaa tämän perusteella vastauksen asiakassovellukselle. Asiakassovelluksen lähettämä tieto voi olla esimerkiksi pyyntö ”mikä on seuraava liike, joka täytyy suorittaa”.
Perceptron	Varhaisen neuroverkon binäärinen neuroni eli algoritmi, jolla toteutetaan yksikerroksinen binäärinen lineaarinen luokittelu.
Portti	Portti on tietoliikenneyhteyden kommunikaation päätepiste.

Protokolla	Protokolla on sovittu tapa, miten koneiden välinen kommunikointi toteutetaan.
Python	Python on yleiskäyttöinen korkean tason olio-orientoitu ohjelmointikieli
Ramppi	Ramppi on siirtymä lähtötasosta tavoitetasoon esimerkiksi tavoitenopeuteen kiihdytykseen käytetty aika.
Sigmoid	Neuroverkon neuronin laskentafunktio
Soketti	Soketti on kahden verkossa toimivan ohjelman kaksisuuntaisen tietoliikenneyhteyden päätepiste.
Säie	Säie on kevyt prosessi, joka käyttää jaettuja resursseja omien resurssien sijaan.
TCP/IP	TCP/IP eli transmission Control Protocol/Internet Protocol on yhdistelmä tietoliikenneprotokollia, joita käytetään internetin dataliikenteessä.
Tietokanta	Tietokoneella sijaitseva tietokokoelma, johon voidaan tallentaa erilaisia tietoja.
Versionhallinta	Versionhallinta pitää kirjaa ohjelmakoodiin tehdyistä muutoksista.
Yhteistyörobotti	Yhteistyörobotti on robotti, joka on suunniteltu soveltuvaksi työskentelemään ihmisten kanssa ilman erillisiä turva-aitoja.

1 JOHDANTO

1.1 Työn tausta

Työssä oli tarkoitus selvittää, kuinka pienellä datamäärällä on mahdollista optimoida hihnakuljettimen energiankulutusta ja läpimenoaikaa, kun optimoidaan vain kahta parametria kuljettimen taajuusmuuttajasta. Muutettavat parametrit, joita käytetään, ovat kiihdytysrampin pituus sekä kuljettimen nopeus.

1.2 Työn tavoite

Tällä hetkellä eletään aikaa, missä käydään maailmanlaajuisesti keskusteluja ilmastonmuutoksesta sekä hiilijalanjäljen pienentämisestä. Tällöin herää kysymys siitä, miten asiaa voisi lähestyä teknologian näkökulmasta hyödyntäen tekoälyn mahdollisuuksia.

Tarkoituksena on selvittää, kuinka paljon dataa tarvitsee kerätä hyödynnettävien tulosten saamiseksi. Tässä työssä selvitetään kuljetinhihnan energiankulutuksen optimointia neuroverkkoalgoritmilla

1.3 Työn rakenne

Toisessa luvussa käsitellään Python-ohjelmointikielen taustaa ja työssä käytettyjä python-moduuleja. Kolmannessa luvussa käydään läpi energiankulutukseen liittyviä asioita. Neljännessä luvussa esitellään GIT-versiohallinta, sen toimintaperiaatetta sekä komentoja. Viidennessä luvussa käsitellään neuroverkon perusteita. Kuudes luku koostuu käytännön työn osuudesta. Seitsemännessä luvussa koostetaan sekä käsitellään data. Kahdeksannessa luvussa käydään läpi neuroverkon opetukseen liittyvät asiat. Yhdeksännessä luvussa esitellään kuljetinhihnan energiankulutuksen optimointiin liittyviä tutkimuksia, joita on tehty aiemmin. Luvussa kymmenen esitellään tulokset ja pohditaan työtä.

1.4 Työn toimeksiantaja

Työ tehtiin Seinäjoen ammattikorkeakoulun Mixed reality & collaborative robotics -hankkeeseen demonstraationa.

Kuva 1. Kuvankaappaus videosta

2 PYTHON-OHJELMOINTIKIELI

Python on Guido van Rossumin luoma ohjelmointikieli, joka julkaistiin 1991. Python tukee mm. olio-orientoitua ohjelmointitapaa sekä sisältää kattavan kirjaston. (Goal-Kicker, [viitattu 10.04.2020].)

2.1 Soketti

Sokettia käytetään lähettämään viestejä verkon yli laitteelta toiselle (Jennings, [viitattu 10.04.2020]).

Palvelin–asiakas-kommunikointiin on konfiguroitava yhteyden molempiin päihin kaksisuuntainen kommunikaatio, vastaanotto sekä lähetys (Harsh, [viitattu 10.04.2020]).

Kuvassa 2 on malliesimerkki, miten soketti luodaan Python-ohjelmointikielellä.

```
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

Kuva 2. Soketin luominen Python-ohjelmointikielellä.

2.2 Scikit-learn

Scikit-learn on helppokäyttöinen Python-moduuli, jolla on helppo ottaa käyttöön tehokkaita koneoppimisalgoritmeja (Pedregosa. ym., [viitattu 10.04.2020]).

Scikit-learn on saanut alkunsa 2007 ”Google Summer of Code” -projektina David Cournapeaun toimesta. Projektissa, jossa Matthieu Brucher alkoi työskennellä osana tutkielmaansa (Boisberranger. ym., [viitattu 10.04.2020]).

2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort ja Vincent Michel tekivät projektista ensimmäisen julkaisun (Boisberranger. ym., [viitattu 10.04.2020]).

Kuva 3 Esittää, miten Sckit-learn-kirjastoa voidaan käyttää.

```
from sklearn.neural_network import MLPRegressor
import numpy as np


class Ai22:
    def __init__(self, x, y):
        self.clf = MLPRegressor(solver='lbfgs', alpha=1e-5,
                                hidden_layer_sizes=2, random_state=1)

        self.clf.fit(x, y)
```

Kuva 3. Esimerkki Scikit-learn-kirjaston käytöstä.

2.3 PostgrSQL

Postgresql-historia alkaa vuodesta 1977, jolloin kehitettiin vuoteen 1985 mennessä Ingress, jota pidetään PostgreSQL-tietokannan esi-isänä. Ingressistä kehitettiin myöhemmin kaupallisesti onnistuneita tietokantapalvelimia. Michael Stonebraker kehitti tiiminsä kanssa tietokantapalvelimen, jota kutsuttiin nimeltä Postgres, tämä tapahtui vuosina 1986 – 1994. Jolly Chen and Andrew Yu lisäsivät sql-ominaisuuksia Postgresiin Tämä projekti sain nimen Postgres95, tämä puolestaan tapahtui vuosina 1994-1995. 1996 nimi PostgreSQL otettiin käyttöön. PostgreSQL on avoimen lähdekoodin tietokantapalvelin. PostgreSQL käyttää asiakas-palvelin-mallia kommunikointiin. (Momjian 2000, 1-5.)

3 ENERGIANKULUTUS SUOMESSA

Johtuen turpeen sekä fossiilisten polttoaineiden käytön vähenemisestä Suomen energiankulutus oli vuonna 2019 yhden prosentin laskusuunnassa edelliseen vuoteen verrattuna. Samoin fossiilisten polttoaineiden kulutus laski 6 prosenttia vuodesta 2018 vuoteen 2019. Teollisuuden osuus energian käytöstä on puolet kokonaiskäytöstä. (Tilastokeskus, [viitattu 22.4.2020].)

Suomessa korkea energiankulutus johtuu pääsääntöisesti vientiteollisuuden rakenteesta, välimatkojen pituudesta sekä kylmästä ilmastosta. Sinkkitehdas kuluttaa omakotitalon vuoden lämmitysenergian viidessätoista minuutissa. (Hiilitieto, [viitattu 22.4.2020].)

3.1 Energiankulutuksen vähentäminen

Vuonna 2006 julkaistussa Euroopan unionin komission tiedoksiannossa määriteltiin tavoitteeksi primäärienergian vähentäminen 20 prosentilla vuoteen 2020 mennessä. (Euroopan yhteisöjen komissio 2006). Suomessa tavoitellaan kasvihuonepäästöjen vähentämistä 5-15% tasolle 1990 vuoden tasosta vuoteen 2050 mennessä. (Ympäristöministeriö, maa- ja metsätalousministeriö ja työ- ja elinkeinoministeriö 2015). Suomalaisten mielestä ydinvoima on hyvä keino torjua ilmastonmuutosta. Mielipide ei suuresti vaihtele eri ikäluokissa. (Sallinen 2020.)

4 GIT-VERSIONHALLINTA

Git on hajautettu versionhallintajärjestelmä, jonka toiminta poikkeaa useimmista versionhallintajärjestelmistä. Git poikkeaa siinä, miten data tulkitaan. Kun muissa järjestelmissä versiot tulkitaan siten, että uusi versio on aina jonkin tiedoston kohtainen muutos, niin Git-versionhallinnassa muutoksia käsitellään ikään kuin tilannekuvana, joka aina muutoksien tapahtuessa tallennetaan. (Chacon & Straub 2014, 1.3.)

4.1 Historia lyhyesti

Gitin alkuperäinen kehittäjä on Linus Torvalds, joka kehitti Git-versionhallintajärjestelmän, koska yhteistyö silloisen hajautetun versionhallintajärjestelmän BitKeeperin kanssa loppui vuonna 2005 (Chacon & Straub 2014, 1.2).

4.2 Haarauttaminen (branching)

Gitin haarauttamisen mallia pidetään Gitin parhaimpana ominaisuutena, ja se erottaakin Git-versionhallintajärjestelmän muista versionhallintajärjestelmistä. Gitin haarauttamisoperaatiot tapahtuvat tehokkaasti ilman suuria viiveitä. Haarat ovat ikään kuin tilannekuvia eri versioiden tilanteesta sillä hetkellä. (Chacon & Straub 2014, 3.1.)

4.3 Git-komentoja

Seuraavassa luettelossa esitellään keskeisimpiä Git-komentoja. Komennoilla saadaan erilaisia toimintoja aikaan Git-versiohallinnassa. Toiminnot, joita komennoilla saadaan aikaan, on esitetty luettelossa heti komennon esittelyn jälkeen.

Git init	Luo alihakemiston versiohallintaa varten ja repositorion luurangon.

Git add	Lisää parametrina annetun tiedoston versiohallintaan.
Git commit	Tallentaa muutokset.
Git clone	Tekee kopion parametrina annetussa osoitteessa olevasta repositoriosta.
Git merge	Yhdistää haarat toisiinsa.
Git fetch	Noutaa etä-repositoriosta projektin tiedot, joita ei vielä ole
Git pull	Hakee muutokset toisesta repositoriosta tai paikallisesta haarasta.
Git push	Siirtää tekemäsi muutokset repositorioon.

(Chacon & Straub 2014, 2.)

5 NEUROVERKKO

Neuroverkko-algoritmin rakenne jäljittelee ihmisen aivojen kaltaista h hermostojärjestelmää, jolla kyetään saavuttamaan mm. kognitiivisia tehtäviä esimerkiksi kasvojen tunnistaminen, puheentunnistus sekä vaativia luokittelutehtäviä (Chow & Cho 2007, 1-2).

Neuroverkko on algoritmi, joka jäljittelee ihmisen aivojen toimintaa matemaattisesti. Neuroverkko koostuu neuroneista, joita ovat mm. perceptron ja sigmoid. Algoritmille on mahdollista opettaa mm. tunnistamaan kuvasta muotoja (Nielsen 2015.)

Neuroverkko koostuu sisääntulokerroksesta ulostulokerroksesta, sekä piilokerroksista. Piilokerros tarkoittaa sitä, että siinä ei ole neuroverkon ulkopuolelta sisääneikä ulostuloja. (Nielsen 2015.)

5.1 Perceptron

Perceptron-neuroni koostuu useammasta binäärisestä sisääntulosta sekä yhdestä binäärisestä ulostulosta. Jokainen sisääntulo sisältää painokertoimen, jonka perusteella ulostulo määräytyy. Monimutkaisemmalla kokoelmalla perceptroneja saadaan aikaan ratkaisuja monimutkaisempiin ongelmiin. (Nielsen 2015.)

5.2 Sigmoid-neuroni

Rakenteeltaan sigmoid-neuroni on samankaltainen kuin perceptron. Sigmoid-neuronilla on useita sisääntuloja sekä yksi ulostulo. Suurin eroavaisuus perceptroniin tulee siinä, että sisääntulo voi olla mitä tahansa 0 ja 1 väliltä ja ulostulo on $\frac{1}{1+e^{(wx+b)}}$ ja tarkemmin ilmaistuna $\frac{1}{e^{-\sum_j w_j x_j - b}}$, missä painokerroin on $w_j = w_1, w_2, \ldots$ ja sisääntulo $x_j = x_1, x_2, \ldots$. (Nielsen 2015.)

5.3 Ensimmäiset neuroverkot

Hyvin aikainen neuroverkko oli McCullochin ja Pittsin malli vuonna 1943, jonka toiminta perustui usean eri painoarvon sisältävän sisääntulon summaukseen. Malli kykeni suorittamaan minkä tahansa aritmeettisen sekä loogisen funktion. (Chow & Cho 2007, 4-5.)

Viisikymmentäluvulla Rosenblatt ja Wignhtman esittelivät Mark I Perceptron -koneen, jonka kykyjä oli mm. yksinkertaisen merkkien tunnistaminen (Chow & Cho 2007, 4-5).

Näiden läpimurtojen jälkeen alkoi useiden tahojen toimesta ilmestyä erilaisia neuroverkkoja, mutta tämä innostus laantui, kun Minsky ja Papert todistivat matemaattisesti, että Perceptron malli ei kyennyt suorittamaan XOR-funktiota. (Chow & Cho 2007, 4-5.)

5.4 Monikerroksinen enteenpäinsyöttävä sekä takaisinkytketty ohjatusti oppiva neuroverkko

Monikerroksinen enteenpäinsyöttävä ohjatusti oppiva neuroverkko on yleisin käytetty neuroverkkoarkkitehtuuri. Opetus tapahtuu syöttämällä sisääntulo-ulostulo-pareja sisältävä opetusdata algoritmille. Algoritmi pyrkii säätämään painokertoimet siten, että ulostulo seuraa toivottua lopputulosta mahdollisimman hyvin. (Chow & Cho 2007, 14-15.)

Monikerroksinen takaisinkytketty ohjatusti oppiva neuroverkko poikkeaa aiemmasta siten, että siinä on yksi taikka useampi takaisinkytkentä, joka voi olla joko kerroksien taikka neuronien välillä (Chow & Cho 2007, 15-17).

5.5 Ohjaamattoman oppimisen neuroverkot

Ohjaamattoman oppimisen neuroverkossa opetusdata ei sisällä oikeita vastauksia, vaan toimii itse organisoidusti. Tyypillinen ohjaamaton neuroverkko sisältää sisääntulokerroksen ja kilpailullisen kerroksen. Kilpailullisen kerroksen neuronit kilpailevat

oppimissäännön mukaisesti saadakseen parhaiten kuvaavan ulostulon aikaiseksi. Ohjaamattoman oppimisen neuroverkoissa itseorganisoituva karttamalli on yleisimmin käytetty ohjaamaton neuroverkon malli. (Chow & Cho 2007, 17-18.)

5.6 Neuroverkon vahvuus mallinnuksessa

Neuroverkon vahvuuksia on kyky mallintaa järjestelmiä, joissa on useita sisään- sekä ulostuloja, ja sitä pidetäänkin yleisesti eräänlaisena kevyenä matemaattisena mallinnuskeinona (Chow & Cho 2007, 19).

6 TOTEUTUS

Tavoite hihnakuljettimen automatisoidusta energiakulutuksen optimoinnista tekoälyalgoritmin avulla aloitettiin miettimällä kokonaiskuvaa sekä määrittelemällä tarvittavat vaatimukset. Tarvittavien laitteistojen saatavuus oli varmistettava ennen työn aloittamista sekä selvitettävä oliko tarvittavat laitteet käytettävissä tähän tutkimukseen.

6.1 Laitteisto hankinta

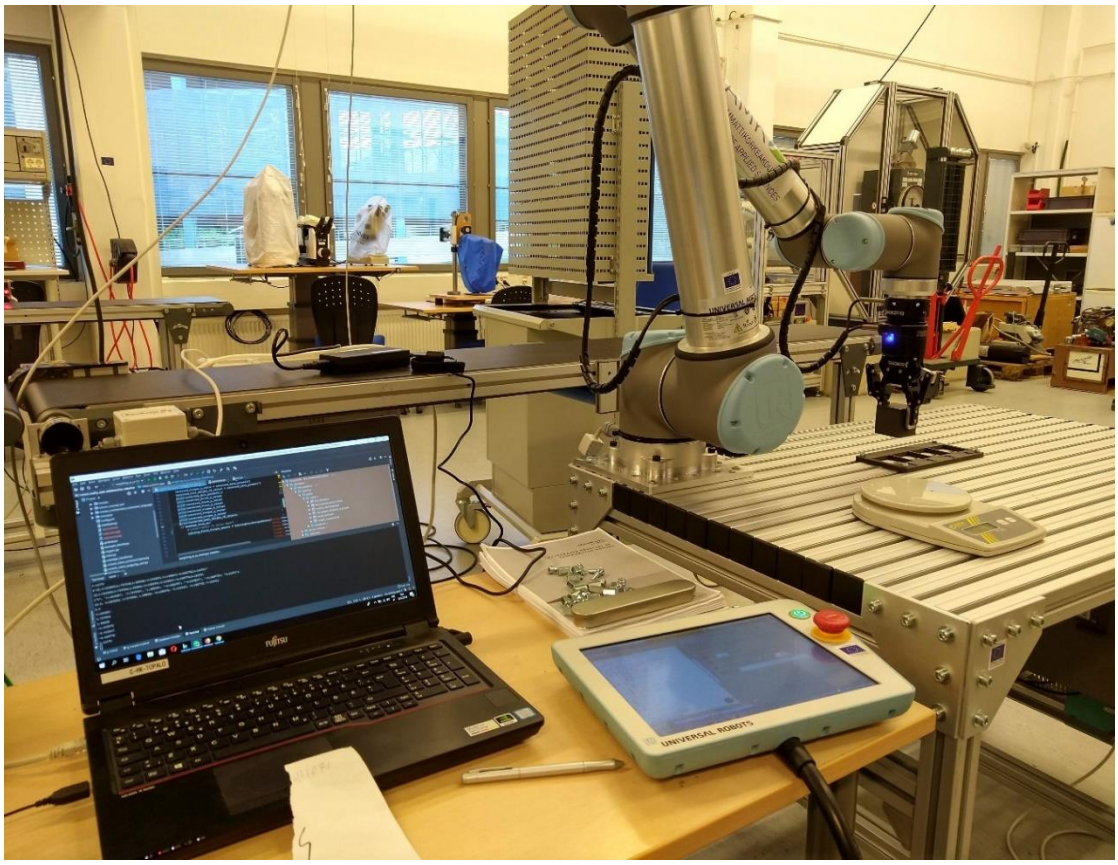
Hihnakuljettimen energiankulutuksen optimoinnin laitteisto oli tarkoitus koota Seinäjoen ammattikorkeakoululla olevista laitteista. Laitteisto on aktiivisessa opetuskäytössä, joten laitteiden saatavuus oli varmistettava ennen kuin oli mahdollista aloittaa tutkimuksen tekeminen.

6.1.1 Laitteiston valinta

Optimoinnin kohdelaitteeksi valittiin Seinäjoen ammattikorkeakoulun robotiikan laboratoriossa sijaitseva noin kaksi metriä pitkä hihnakuljetin. Hihnakuljetin oli tekohetkellä hyvin vähäisellä käytöllä, jolloin pitempiaikainen syventyminen opinnäytetyön tekoon oli mahdollista. Eikä ollut riskiä siitä, että joku tarvitsisi kuljetinta kesken opinnäytetyön käytännön osuuden.

Automatisointia varten valittiin robotiikan laboratoriossa sijaitseva Universal robots UR10 -yhteistyörobotti, joka oli käyttössä Mixed Reality & Collaborative robotics -hankkeessa. Robottiin oli liitetty Robotiq Force Torque FT300 -voima-anturi sekä Robotiq 2F-85 -kaksisormitarttuja. Siirrettävän kappaleen leveys oli alle 85 mm, joten tarttujan maksimitartuntaleveys 85 mm oli riittävä.

Kokonaisuuden ohjaamiseen päätettiin valita kannettava tietokone, johon tultaisiin ohjelmoimaan Pythonohjelmointikielellä ohjaussovellus. Kokonaisuus on esitetty kuvassa 4.



Kuva 4. Laitteistokokoonpano.

6.1.2 Laitteistonvalmiuden varmistaminen

Koska tarkoituksena oli kontrolloida koko kokonaisuutta tietokoneelta Ethernet-verkon välityksellä, niin vaatimuksena oli liitettävyys verkkokaapelilla, sekä mahdollisuus vaihtaa asetuksia samaa yhteyttä käyttäen.

Universal Robots UR10 -yhteistyörobotin liitettävyydestä hankkeessa oli jo aiempaa kokemusta ja tiedettiin sen toimivan saumattomasti yhteen aiemmin python-ohjelmointikielellä ohjelmoidulla TCP/IP- soketti-kommunikointipalvelimen kanssa.

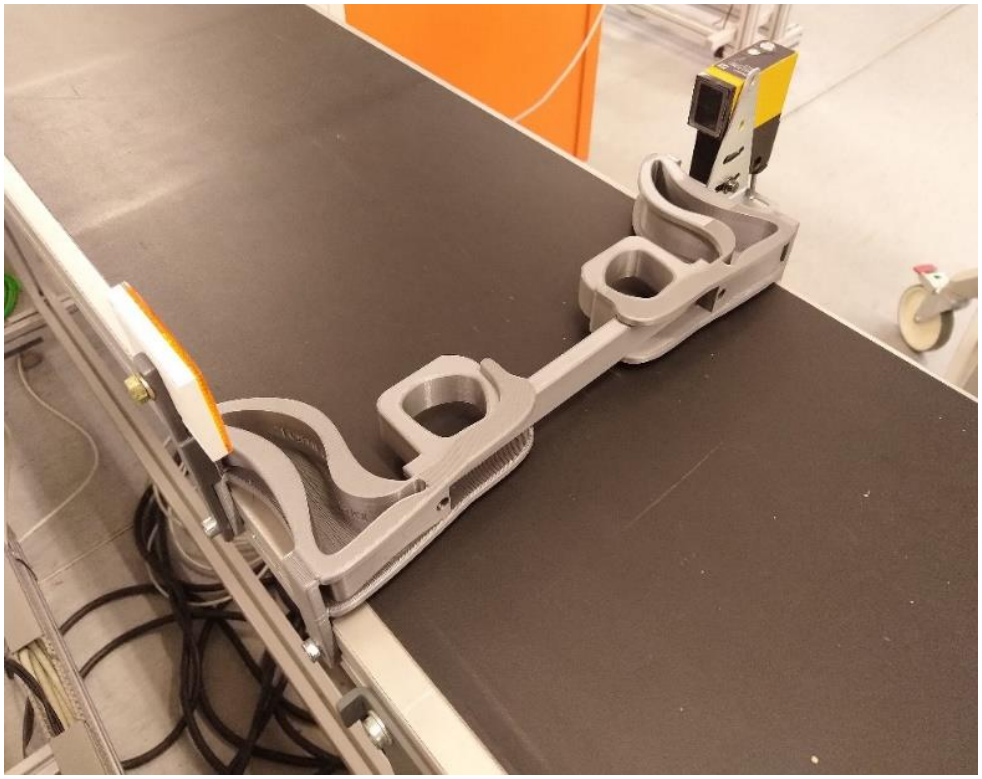
Kuljettimen kanssa tilanne oli eri, koska kuljettimessa käytössä olevassa taajuusmuuttajassa ei ollut mahdollisuutta verkkokytkentään, eikä näin myöskään mahdollisuutta verkon yli tapahtuvaan parametrien muuttamiseen, joten ongelma oli ratkaistava etsimällä tähän sopiva taajuusmuuntaja. Tästä enemmän myöhemmässä luvussa 6.2.2. Hihnakuljettimen loppupäähän oli myös lisättävä valokenno, josta

saadaan tieto kappaleen saapumisesta hihnakuljettimen päähän, sekä mitattua ajankohta, milloin kappale on saapunut hihnakuljettimen loppupäähän, kuva 5.

6.1.3 Kokonaisuuden rakenteellinen suunnittelu

Kokonaisuuden rakennetta suunnitellessa oli selvää, että robotin ulottuvuuden ollessa 1,3 m sijoitetaan se kuljettimen viereen keskikohdan tuntumaan, siten että kappale saadaan jätettyä kuljettimen alkupäähän sekä poimittua loppupäästä. Ajanoton pysäyttämisen toteuttamiseen oli tarpeen lisätä valokenno kuljettimen loppupäähän, sekä robotin pöydälle kappaleen asemointistoppari (Kuva 5).

Laitteiden välisen kommunikaation rakennetta suunnitellessa tultiin lopputulokseen, että kannettava tietokone toimii operaation kontrollerina, eli kaikki toiminnot lähtevät kootusti tietokoneelta. Ohjelmistolla on oltava yhteys kuljettimeen, yhteistyörobottiin sekä tietokantaan.



Kuva 5. 3D-tulostettu stoppari sekä valokenno

6.2 Kuljettimen moottorin ohjauslaite

Kuljettimen energiankäytön optimointia varten on oltava jokin laite, jolla moottoria voidaan ohjata. Taajuusmuuttaja on tähän tarkoitukseen sopiva. Muitakin mahdollisuuksia olisi, mutta taajuusmuuttaja on joustavin saatavilla olevista laitteista.

6.2.1 Kuljettimen taajuusmuuttajan vaatimukset

Kuljettimen taajuusmuuttajalla on oltava mahdollisuus muuttaa ajoon liittyviä parametreja, antaa käynnistys- sekä pysäytyskäskyt Ethernet-verkon yli. Taajuusmuuttajan on myös oltava sopiva hihnakuljettimen moottorin ohjaukseen.

6.2.2 Taajuusmuuttajan valinta

Kuljetinhihnassa valmiina olleessa taajuusmuuttajassa ei ollut Ethernet-liitäntää, joten se oli vaihdettava. Taajuusmuuttajaksi valittiin Vacon 100, koska se täytti kriteerit ja sellainen löytyi valmiiksi varastosta (kuva 6). Taajuusmuuttaja oli myös käytettävissä tutkimuksen ajan.

Kuva 6. Taajuusmuuttaja Vacon 100

6.2.3 Taajuusmuuttajan parametrointi

Ensimmäisenä taajuusmuuttajaan oli asetettava Ethernet-yhteyden mahdollistavat parametrit. Moottorin arvot sekä muut yleiset asetukset säädettiin tämän jälkeen Ethernet-verkon välityksellä python-ohjelmasta.

Toinen tärkeä asia oli selvittää, millainen viesti on taajuusmuuttajalle lähetettävä, että taajuusmuuttaja antaa virrat kuljettimelle ja näin ollen saattaa sen myös liikkeeseen. Tämän asian selvittämiseen meni ainakin päivän verran aikaa, ennen kuin

manuaalista löytyi oikea tapa lähettää Ethernetin välityksellä oikea käsky taajuusmuuttajalle, joka käynnistää kuljettimen. Tämä testattiin lyhyellä python-ohjelmapätkällä, joka ohjelmoitiin pelkästään testausta varten.

Taajuusmuuttajan parametreista oli myös etsittävä kaikki lähtökohdassa täydennettävät parametrit mm. moottorin tiedot. Näiden kaikkien lähtötietojen id-tunnukset oli otettava talteen myöhempää ohjelmallista alustusta varten, joka tultaisiin toteuttamaan Ethernet-verkon yli suoraan python-ohjelmasta. Taajuusmuuttajan muiden parametrien id-tunnukset poimittiin myös talteen tulevaisuuden tarpeita varten.

6.3 Robotin ohjelmointi

Robotin ohjelmointi päätettiin toteuttaa mahdollisimman modulaarisena hajauttaen kaikki mahdollinen toiminta erillisiin aliohjelmiin. Tämä mahdollisti nopeamman ja tehokkaamman ohjelmakoodin muuttamisen sekä jatkokäsittelyn. Tämän kaltaiset, rakenteelliset toteutukset oli jo aiemmissa projekteissa huomattu erittäin hyödyllisiksi, varsinkin tilanteissa, joissa joudutaan etsimään virhettä ohjelmakoodista.

6.3.1 Robotin ohjelmoinnin vaatimukset

Lähtökohtana oli, että ohjeet robotin toiminnoille tulevat soketti-viestinä numeromuodossa ja näitä numeroita vastaa aina tietty toimintokokonaisuus robotilla. Robotin on poimittava kappale asemointipaikasta, siirrettävä se kuljettimelle sekä poimittava kuljettimen loppupäästä ja tuotava takaisin, sekä asemoida kotiasemaan. Robottiin kytketyn kuljettimen loppupäähän sijoitetun valokennon tieto on myös välitettävä eteenpäin soketti-viestissä. Ohjelmakoodin on kyseltävä uusia ohjeita tietokoneen palvelinsovellukselta aina edellisen ohjelmakierron päätyttyä, sekä lähetettävä suoritetun ohjelmakierron numero soketti-viestin välityksellä.

6.3.2 Robotin ohjelmarakenne

Robotin ohjelmarakenteeksi päätyi valintarakenne (kuva 7) missä jokaisen valintavaihtoehdon alla on yksi liikesarja, tässä muutama esimerkki:

- vienti kuljettimen alkupäähän
- poiminta kotiasemasta
- poiminta kuljettimen loppupäästä.

Eri liikesarjat suoritetaan soketti-viestillä saapuvan numeron perusteella ja paluuviestinä lähetetään suoritettua liikesarjaa vastaava ennalta päätetty numero, sekä kuljettimella sijaitsevan robotin sisääntuloon kytketyn anturin tila. Näiden tietojen perusteella tietokoneen ohjelmakoodissa voidaan käsitellä, mikä tilanne koko toimintakierrossa milläkin hetkellä on.

```
Switch NextMove
  Case 1
    'pick home'
    'Call is_objec_at_gripper'
    IsObjectAtGrip= False
    If  not IsObjectAtGrip
      Call to_waiting_point
      Call to_above_pick_pos
      Call gripper_open_fully
      Call to_pick_point
      Call gripper_close_fully
      Call to_above_pick_pos
      Call to_waiting_point
      message_to_send[0]=1
  Case 2
    'dropA'
    'Call is_objec_at_gripper'
    IsObjectAtGrip= True
    If IsObjectAtGrip
      Call to_waiting_point
      Call to_above_point_a
      Call to_point_a
      Call gripper_open_fully
      Call to_above_point_a
      Call to_waiting_point
      message_to_send[0]=2
  Case 3
    'pickA'
    'Call is_objec_at_gripper'
    IsObjectAtGrip= False
    If  not IsObjectAtGrip
      Call to_waiting_point
      Call to_above_point_a
      Call gripper_open_fully
      Call to_point_a
      Call gripper_close_fully
      Call to_above_point_a
      Call to_waiting_point
      message_to_send[0]=3
```

Kuva 7.Ote robottiohjelman valintarakenteesta

6.4 Versionhallinta

Versiohallintalla on tarkoitus pitää kirjaa ohjelmoinnin kulusta, sekä selkeyttää rakenteiden lisäämistä ja mahdollistaa helpomman ominaisuuksien testaamisen. Versionhallinnan käyttö on aiemmissa projekteissa todettu erittäin hyödylliseksi käytännöksi.

6.4.1 Versiohallinnan vaatimukset

Versiohallinnan vaatimukset tähän opinnäytetyöhön olivat seuraavat:

- Versiohallinnan on oltava integroitavissa ohjelmointiohjelmistoon (JetBrains Pycharm).
- Versiohallinnassa on oltava mahdollisuus asettaa projekti yksityiseksi.

6.4.2 Versiohallinnan valinta

Versionhallinnan vaihtoehtoja oli neljä kappaletta, jotka olivat suoraan Pycharm-yhteensopivia:

- Git
- Mercurial
- Perforce
- Subversion.

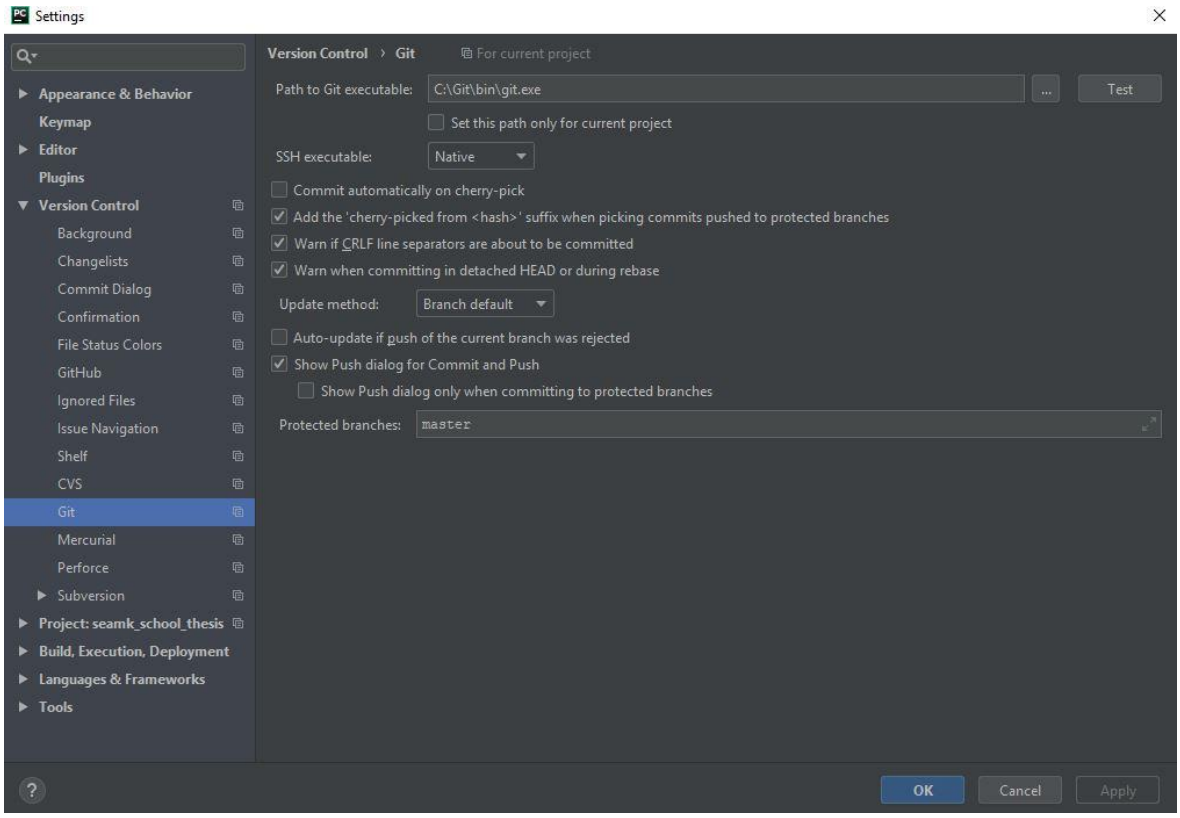
(Jetbrains, [viitattu 30.5.2020].)

Versiohallinnan valinnaksi selvisi Git, koska aiemmissa projekteissa oli kokemusta Git-versiohallinnan käytöstä ja tiedettiin sen integroituvan mutkattomasti JetBrains PyCharm -ohjelmointiohjelmiston kanssa.

6.4.3 Versiohallinnan käyttöönotto

Versiohallinnan käyttöönotossa oli ensimmäisenä asennettava Git-versionhallinta tietokoneelle, jonka jälkeen oli oltava tunnukset GitHub-sivustolle, GitHub-tunnukset oli hankittu jo aiemmin, joten jäljellä oli ainoastaan Git-versionhallinnan asentaminen koneelle.

Versiohallinnan integroiminen tapahtui PyCharm-ohjelmistossa asettamalla paikallisen Git-versiohallinnan asetukset, sekä versiohallinnan jakamisen GitHubin asetukset (Kuva 8).



Kuva 8. Git-versiohallinnan asetukset PyCharm-ohjelmistossa.

6.5 Tietokoneen ohjelmointi

Tietokoneen ohjelmointi suoritettiin Python-ohjelmointikielellä, jota oli jo aiemmissakin projekteissa käytetty. Pythonilla oli jo aiemmin ohjelmoitu soketti-kommunikointipalvelin, jonka arveltiin hyvin sopivan tähän kokonaisuuteen helpottaen ohjelmointitaakkaa, sekä nopeuttaen opinnäytetyön etenemistä. Tässä vaiheessa tiedettiin ohjelmoinnin vaativan paljon rivejä , jotta saadaan toteutettua täysin automatisoitu linjasto.

6.5.1 Vaatimukset

Python ohjelmakoodin on hoidettava seuraavat toiminnallisuudet:

- Kommunikaatio kuljettimen taajuusmuuttajan kanssa.
- Kommunikointi robotin kanssa.
- Tietojen tallentaminen PostgreSQL-tietokantaan.
- Ohjelmiston on myös pyöritettävä kokonaisuutta automaattisesti.
- Ohjelmiston on otettava vastaan robotin ohjaimesta määritetyt parametrit, kuten toimintakiertojen määrä.
- Ohjelmistosta on pidettävä versionhallintaa

Python-ohjelmakoodin on oltava rakenteeltaan suhteellisen selkeä, että mahdolliset jälkityöt sekä vian paikannukset on tehokkaammin hoidettavissa. Ohjelmistossa on pyrittävä pitämään koodin osalta samankaltaisuus muuttujien, metodien sekä luokkien nimeämisessä. Ohjelmakoodin tehokkuudella ei ole merkitystä, joten sen osalta ohjelmakoodin rakennetta ei tarvitse miettiä.

6.5.2 Tietokoneelle ohjelmoidun ohjelman ohjelmarakenne

Ohjelma sisältää soketti-kommunikointipalvelimen, joka hoitaa robotin ja tietokoneen välisen kommunikoinnin käyttäen säikeitä. Kommunikointipalvelin hoitaa kommunikoinnin vastaamalla lähettäjälle palvelimen luonnissa määritellyn käsittelymetodin toiminnan mukaisesti. Erillisessä säikeessä toimii modbus-kommunikaatio kuljetinhihnan sekä tietokoneen välillä. Molemmista kommunikointirajapinnoista tallennetaan tarvittavia tietoja tietokantaan, esimerkiksi kuljettimen lähtöaika sekä kuljettimen pysähdysaika. Tietojen tallennusta varten ohjelmoitiin ohjelmakoodi, joka tallentaa kaikki kommunikoinnissa sisälletyt tärkeät tiedot tietokantaan. Ohjelmakoodiin ohjelmoitiin myös tietokannan muokkaamiseen tarvittavia toimintoja siltä varalta, että tietokannan rakennetta on tarve muuttaa

6.5.3 Ohjelmointikielen sekä tietokannan valinta

Ohjelmointikieleksi valittiin Python, koska se oli tutuin ohjelmointikieli, sekä sillä on mahdollista saada aikaan hyvin luettavaa ohjelmakoodia. Lisäksi ohjelmakoodin luettavuus ja rakenteen selkeys oli jo vaatimuksissa esitetty, tämä tukee hyvin vaatimuksien linjaa. Samalla perusteella valittiin tietokannaksi PostgreSQL-tietokanta. Kuvassa 9 on esitetty osa tietokantaluokan sisällöstä.

```
import psycopg2


# SELECT - extracts data from a database
# UPDATE - updates data in a database
# DELETE - deletes data from a database
# INSERT INTO - inserts new data into a database
# CREATE DATABASE - creates a new database
# ALTER DATABASE - modifies a database
# CREATE TABLE - creates a new table
# ALTER TABLE - modifies a table
# DROP TABLE - deletes a table
# CREATE INDEX - creates an index (search key)
# DROP INDEX - deletes an index

class DatabaseExecutor:
    def __init__(self, database_connection_parameters):
        self.params = database_connection_parameters
        self.conn = None
        self.cur = None

    def execute_database_command(self, command, value=None, auto_commit=False):
        """ execute database command"""
        self.conn = None
        executed = False

        try:

            self.make_connection_to_database()
            self.create_cursor()
            self.conn.autocommit = auto_commit
            self.execute_command(command, value)
            self.close_cursor()

            self.commit_the_changes()
            executed = True
            # print("database command executed")

        except (Exception, psycopg2.DatabaseError) as error:
            self.print_database_error(error)
            print("error occurred while executing database command", error)
        finally:
            connection_is_open = self.check_is_the_connection_open()
            if connection_is_open:
                self.close_connection()
        return executed
```

Kuva 9. Tietokantaluokka ohjelmakoodissa

6.5.4 Tietokoneohjelmiston ohjelmointi

Ohjelmiston ohjelmointi päätettiin aloittaa kaikkein tärkeimmästä, eli tietojen tallennuksesta. Tässä vaiheessa suunniteltiin, mitä tietoja tarvitaan tallennettavaksi tietokantaa, ja tehtiin tarvittavat ohjelmakoodit sen hoitamiseen. Samalla tehtiin alustavat ohjelmakoodit tietokannan tyhjentämiseen, näitä säädetään tarpeen vaatimalla tavalla myöhemmin.

Kommunikoinnin pohjatyöt kirjoitettiin niin ikään ennen kuin alettiin kytkeä koneita Ethernet-verkkoon. Tällöin luotiin siis soketti kommunikaatioon robotin ja tietokoneen välille sekä modbus kommunikaatioon kuljettimen taajuusmuuttajan ja tietokoneen välille. Kuvassa 10 on taajuusmuuttajan parametrien alustamisen ohjelmakoodia.

```
def write_init_values_to_vacon(self, client):

    deceleration_time_1_scaled = self.deceleration_time * 10
    motor_nominal_frequency_scaled = self.motor_nominal_frequency * 100
    motor_nominal_current_scaled = self.motor_nominal_current * 10
    motor_nominal_power_scaled = self.motor_nominal_power * 10

    print("Table 27: Motor nameplate parameters")
    self.set_parameter_value(client, self.motor_normal_voltage_id, value=self.motor_nominal_value)
    motor_normal_voltage_value = self.get_parameter_value(client, self.motor_normal_voltage_id)
    print("motor norm voltage changed to ", motor_normal_voltage_value)

    self.set_parameter_value(client, self.motor_nominal_frequency_id, value=motor_nominal_frequency_scaled)
    motor_nominal_frequency_value = self.get_parameter_value(client, self.motor_nominal_frequency_id)
    print("motor norm freq changed to", motor_nominal_frequency_value)

    self.set_parameter_value(client, self.motor_nominal_speed_id, self.motor_nominal_speed)
    motor_nominal_speed_value = self.get_parameter_value(client, self.motor_nominal_speed_id)
    print("motor norm speed changed to", motor_nominal_speed_value)

    self.set_parameter_value(client, self.motor_nominal_current_id, motor_nominal_current_scaled)
    parameter_value = self.get_parameter_value(client, self.motor_nominal_current_id)
    print("motor nominal current changed to", parameter_value)

    self.set_parameter_value(client, self.motor_cos_phi_id, self.motor_cos_phi)
    parameter_value = self.get_parameter_value(client, self.motor_cos_phi_id)
    print("Motor CosPhi changed to", parameter_value)

    self.set_parameter_value(client, self.motor_nominal_power_id, motor_nominal_power_scaled)
    parameter_value = self.get_parameter_value(client, self.motor_nominal_power_id)
    print("Motor Nominal Power", parameter_value)

    print("Table 28: Motor control settings 0 is for freq")
    self.set_parameter_value(client, self.motor_control_type_id, self.motor_control_type)
    parameter_value = self.get_parameter_value(client, self.motor_control_type_id)
    print("Motor control type set to", parameter_value)

    self.set_parameter_value(client, self.motor_type_id, self.motor_type)
    parameter_value = self.get_parameter_value(client, self.motor_type_id)
    print("Motor type set to", parameter_value)

    print("Table 33: Start/stop setup menu")
    self.set_parameter_value(client, self.remote_control_place_id, self.remote_control_place)
    parameter_value = self.get_parameter_value(client, self.remote_control_place_id)
    print("Remote Control place set to", parameter_value)

    print("Table 38: Ramp 1 setup")

    self.set_parameter_value(client, self.ramp_1_shape_id, self.ramp_1_shape)
    parameter_value = self.get_parameter_value(client, self.ramp_1_shape_id)
    print("Ramp_1_Shape_set_to", parameter_value)

    acceleration_time = 5
    self.set_acceleration_time(acceleration_time, client)

    self.set_parameter_value(client, self.deceleration_time_1_id, deceleration_time_1_scaled)
    deceleration_time_1_scaled = self.get_parameter_value(client, self.deceleration_time_1_id)
    print("Deceleration Time 1 is set to", deceleration_time_1_scaled)
```

Kuva 10. Ohjelmakoodi taajuusmuuttajan parametrien alustamiseen.

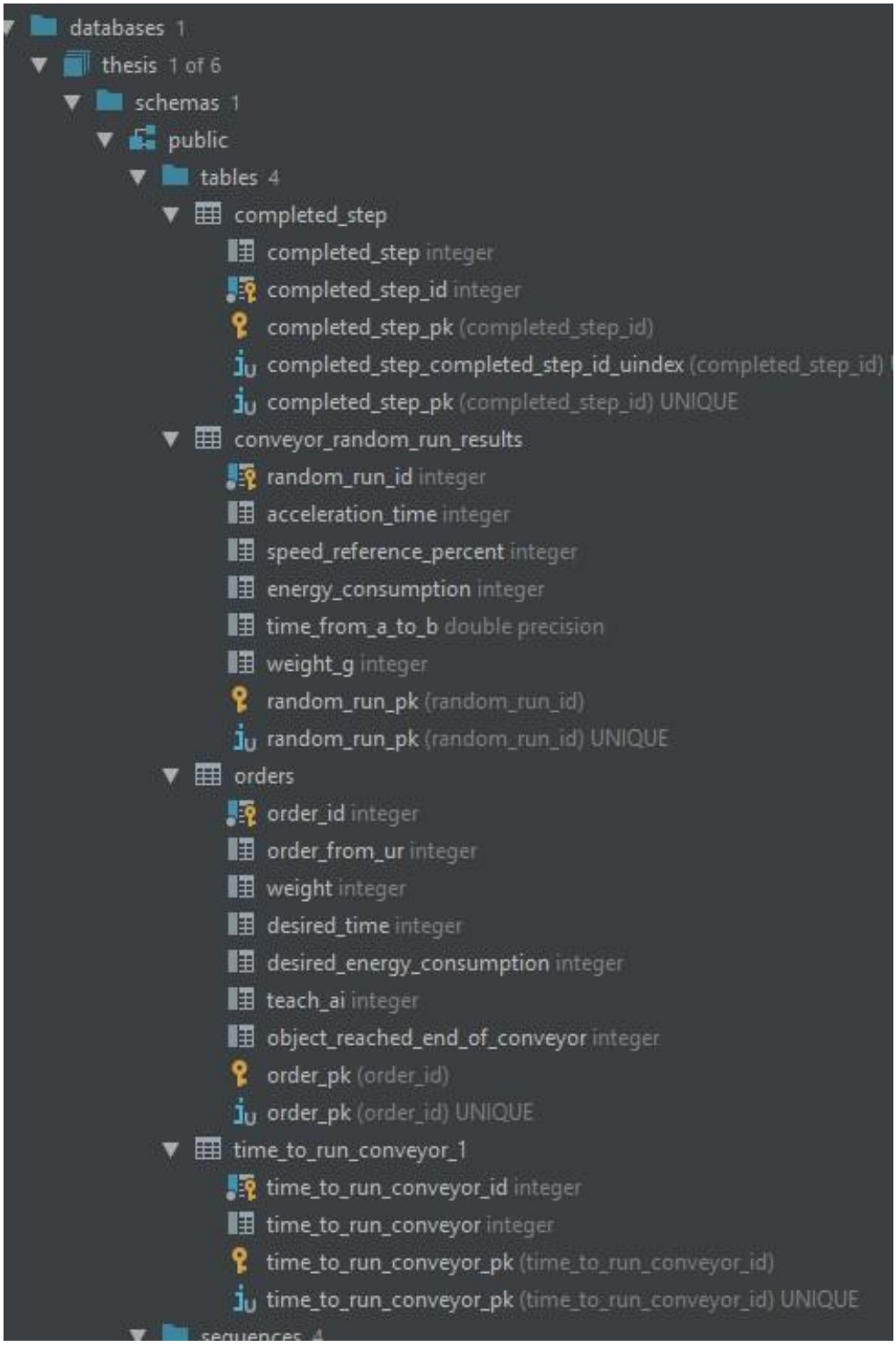
7 DATAN KOOSTAMINEN

Datan koostamisen ohjelmakierroissa valittiin muutettaviksi parametreiksi nopeus sekä kiihdytysrampin pituus. Kerättävät tiedot olivat kiihdytysrampin pituusparametrin arvo, nopeusparametrin arvo, energian kulutusmittarin arvo ja aika, jonka kappale kulkee hihnakuljettimella sekä kappaleen paino. Kuvasta 11 selviää tietokannan rakenne, minne tiedot tallennetaan.

7.1 Datan koostamisen ohjelmakierrot

Ensimmäisillä analysointiajoilla testattiin kokonaisuuden toimivuutta ja huomatiin ongelmakohta, joka oli soketti-kommunikoinnista johtuva pieni viive, joka saattoi nopeilla ajoilla aiheuttaa sen, ettei kuljetin pysähtynyt oikeaan kohtaan, vaan kappale meni hieman ohitse. Robotin kohdalla tämä vaikeutti poiminta poimintavarmuutta, joten päädyttiin lisäämään stoppari, joka suunniteltiin itse sekä tulostettiin 3D-tulostimella.

Stopparin asentamisen jälkeen datankeruuajojen testit sujuivat hyvin, mutta toista vaakaa testatessa huomattiin satojen grammojen ero. Kuitenkin hyvin pian tuli selväksi, että vaa’assa oli vaihtunut yksikkö, mahdollisesti oli painettu taaranappia liian pitkään, jolloin yksikkö vaihtuu.



Kuva 11. Tietokannan rakenne

Taajuusmuuttajan energiankulutuksen pienimmän yksikön ollessa 1 Wh ja moottorin energian kulutus n. 1 Wh per 9 kierrosta oli selvää, että kuljettimen ajokertoja

täytyy olla vähintään kymmeniä per energianmittausajo, että mahdolliset energiankulutusten erot saadaan paremmin näkyviin.

Ensimmäisistä datankeräyksistä sekä mittauksista selvisi, että 1 Wh:n kulutus tuli 2–10 kierron välein, joten tultiin siihen tulokseen, että 100 kiertoa per mittaus on pienin määrä, jolla kulutusten erot ovat riittävän selkeästi havaittavissa.

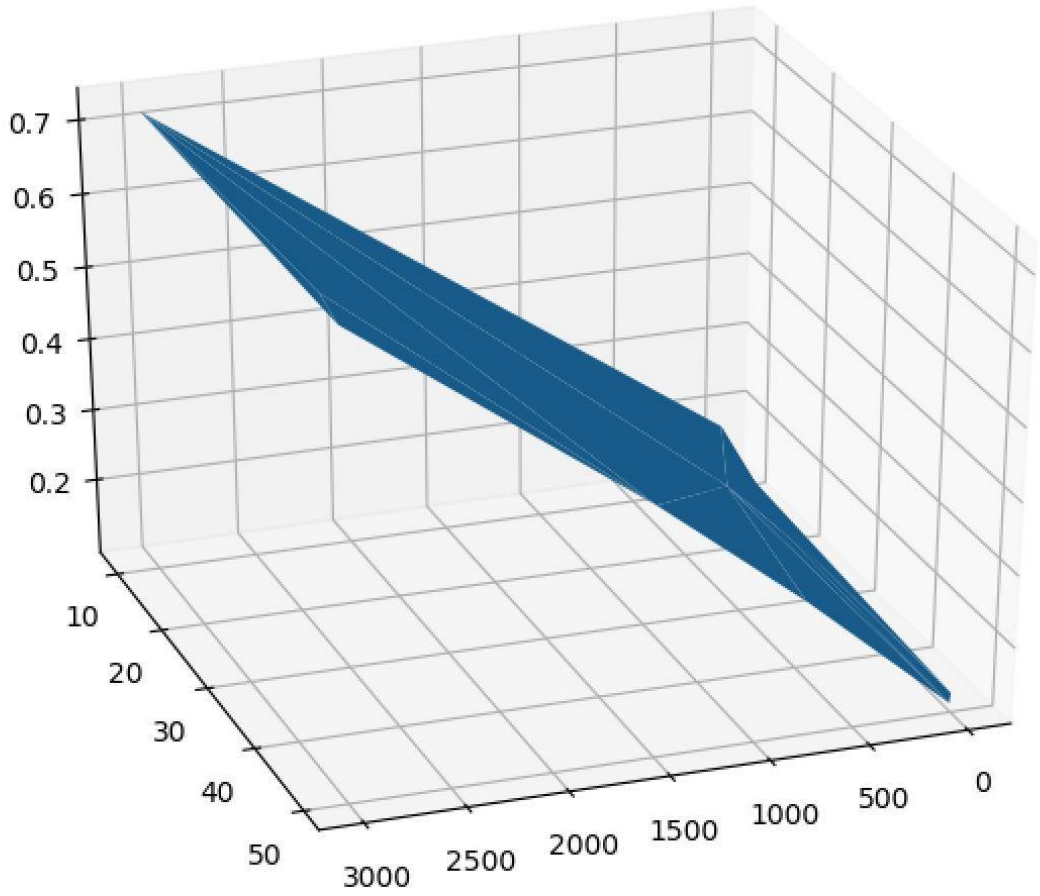
Ensin suoritettiin datan keruun ääriarvoilla ja niiden kaikilla kombinaatioilla, sen jälkeen suoritettiin datan keruu minimi- ja maksimiarvojen keskeltä, sekä lopuksi tietokoneen arpomilla satunnaisilla parametriarvoilla.

7.2 Datan käsittely

Ennen kuin dataa voitiin käyttää hyödyksi, oli dataa käsiteltävä sellaiseen muotoon, siten että sitä voitiin käyttää opetusdatana neuroverkkoalgoritmille. Datassa oleva energian kulutus piti laskea, koska tallennettu kulutus oli mittaushetkellä oleva kokonaiskulutuksen kertymä, tätä varten oli ohjelmoitava ohjelmakoodi, mikä käsittelee, kokoaa ja laskee datasta käytettävät arvot (kuva 12).

```
i = 0
energy_consumption_list_result = []
while i < len(energy_consumption_list):
    energy_consumption_list_result.append(
        (energy_consumption_list_2nd[i] - energy_consumption_list[i]) / run_counts_results[i])
    i += 1
```

Kuva 12. Yhden kierron energiankulutuksen laskenta ohjelmakoodissa.



Kuva 13. 3D-kuvaaja energiankulutuksesta: ramppi 0–3000, nopeus 0–50 pystyakselilla yhden kierron energiankulutus

8 NEUROVERKON OPETUS

8.1 Datan muotoilu neuroverkolle

Aiemmin käsitelty data oli vielä noudettava tietokannasta muokattava neuroverkkoalgoritmille syötettävään muotoon. Data oli myös syötettävä kirjaston vaatiman rakenteen mukaisesti neuroverkkoalgoritmille. Tämä toteutettiin käytännössä keräämällä tiedot kahteen erilliseen listaan, jotka sisälsivät erillisiä listoja. Tämän jälkeen algoritmin annettiin prosessoida kuljettimelta kerätty ja muotoiltu data. Kuvassa 14 on kuvakaappaus ohjelmakoodin kohdasta, jossa neuroverkko opetetaan.

```
ai = Ai(1)
teached_ai = ai.teach_ai(x, y)
```

Kuva 14. Opetetun neuroverkon luonti.

8.2 Neuroverkon toimivuuden testaaminen

Kun neuroverkko oli opetettu, oli vuorossa testaus. Testaus suoritettiin syöttämällä toivottu läpimenoaika sekunteina, sekä toivottu sähkönkulutus wattitunteina per läpiajo. Ensimmäisten testausten jälkeen verrattiin ehdotettujen parametrien paikkansapitävyyttä 3D-visualisoituun dataan (kuva 13).

Testien perusteella neuroverkkojen piilotettujen tasojen hyväksi määräksi osoittautui kolme kappaletta.

9 HIHNAKULJETTIMEN ENERGIANKULUTUKSEEN LIITTYVÄT TUTKIMUKSET

Kuljetinhihnan energiankulutuksen optimoinnista on tehty useita tutkimuksia. Salawun, Brightin ja Onunkan (2020) toimesta on tehty tutkimus, jossa kuljettimen optimaalisen tehokkuuden saavuttamiseksi tehtiin simulointimalli suunnitteluvaiheessa. Simulointimallissa mallinnettiin hihnakuljettimen suunnitteluparametreja kappaleiden siirrossa eri tuotantovaiheiden välillä.

Ji, Miao ja Li X (2020) ovat tähdänneet tutkimuksessaan hihnakuljettimen energian kulutuksen optimoinnin toteuttamiseen luomalla mallin hihnakuljettimen energiankulutuksesta, jossa on otettu huomioon hihnan nopeus, materiaalin määrä sekä monia muita parametrejä. Tutkimuksessa todettiin, että mallilla on suuri arvo mm. hiili-, satama-, energia-, kaivos-, metalli- ja kemikaaliteollisuudessa sekä muussa teollisuudessa. Tutkimus antaa strategian energian säästöön hihnakuljettimella. Tutkimuksessa huomioidaan myös materiaalin kasaantumisen mahdollisuudet ongelmat ajettaessa optimaalisella nopeudella. (Ji J. ym., [Viitattu 20.04.2020].)

10 TULOKSET JA POHDINTA

1. Opinnäytetyön tuloksena syntyi ohjelmisto, jolla voidaan automatisoida hihnakuljettimen energiankulutuksen optimointi.
2. Lopullisessa tarkastelussa 13 datariviä riitti poistamaan lähtökitkasta johtuvan ylimääräisen energiankulutuksen.
3. Opinnäytetyön yhteydessä syntyi demonstraatiovideo. Kuvakaappaus dokumentista löytyy kuvasta 1.

Työn tavoitteet saavutettiin hyvin, koska jo pelkästään 13 datasetin avulla saatiin optimoitua kuljettimen energiankulutusta, eli tekoälyalgoritmia voi hyödyntää tuloksellisesti muutaman päivän datan keruulla lähtökohdasta, jossa dataa ei ole.

Datan kerääminen jatkuu päättötyön jälkeen ainakin jonkin aikaa. Tämä voisi mahdollistaa useamman parametrin säätämisen tekoälyn avulla sekä eripainoisten kappaleiden painon huomioimisen algoritmissa.

Työssä olisi voinut myös ottaa mukaan myös muita asioita optimoitavaksi kuljettimessa, kuten esimerkiksi, melu, moottorin lämpötila, moottorin värinät, laakerivian tunnistaminen yms.

Koska tavoitteena oli automatisoida koko prosessi verkon yli, aiheutui suurin työmäärä toimintaa ohjaavan python-ohjelman kirjoittamisesta, johon rivejä kertyikin n. 3700. Käsikäyttöinen datankeruu olisi ollut huomattavasti yksinkertaisempi, mutta mahdollisuus myöhempään ison datamäärän keräämiseen ei tällöin olisi järkevää, koska se vaatisi ihmisen läsnäoloa ja toimintaa kokoaikaisesti. Automatisoituna oli mahdollista jättää kokonaisuus päiväksi sekä illaksi keräämään itsenäisesti dataa, sekä samalla oli mahdollista jatkaa muita töitä sekä lopun ohjelmakoodin ohjelmointia.

LÄHTEET

Boisberranger, J., Van den Bossche, J., Estève, L., Fan, T., Gramfort, A., Grisel, O., Halchenko, Y., Hug, N., Jalali, A., Lemaitre, G., Metzen, J., Mueller, A., Niculae, V., Nothman, J., Qin, H., Thirion, B., Dupré la Tour, T., Varoquaux, G., Varoquaux, N., Yurchak, R., Blondel, M., Brucher, M., Buitinck, L., Cournapeau, D., Dawe, N., Du, S., Dubourg, V., Duchesnay, E., Fabisch, A., Fritsch, V., Ghosh, S., Gollonet, A., Gorgolewski, C., Grobler, J., Holt, B., Joly, A., Jones, T., Kastner, K., Kumar, M., Layton, R., Li, W., Losi, P., Louppe, G., Michel, V., Millman, J., Passos, A., Pedregosa, F. Prettenhofer, P., Rajagopalan, R., Schreiber, J., Vanderplas,J., Warde-Farley, D. & Weiss, R. Ei päiväystä. About us. [Verkkosivu]. [Viitattu 10.04.2020] Saatavana: https://scikit-learn.org/stable/about.html

Chacon, S. & Straub, B. 2014. Pro Git. Apress. [Verkkokirja]. [Viitattu 22.4.2020] Saatavana: https://git-scm.com/book/en/v2

Chow, T. & Cho, S-Y. 2007. Neural Networks and Computing Learning Algorithms and Applications. Imperial College Press. [Verkkokirja]. [Viitattu 10.4.2020]. Saatavana: https://epdf.pub/neural-networks-and-computing-learning-algorithms-and-applications-series-in-ele.html

Euroopan yhteisöjen komissio. 2006. Komission tiedonanto - Energiatehokkuuden toimintasuunnitelma: Mahdollisuuksien toteuttaminen. [Verkkojulkaisu]. [Viitattu 22.4.2020]. Saatavana https://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=COM:2006:0545:FIN:FI:PDF

GoalKicker. Ei päiväystä. Python Notes for Professionals. [Verkkojulkaisu]. [Viitattu 10.4.2020]. Saatavana: https://books.goalkicker.com/PythonBook/

Harsh, S. Ei päiväystä. Essentials of Python Socket Programming. [Verkkosivu]. [Viitattu 10.4.2020]. Saatavana https://www.techbeamers.com/python-tutorial-essentials-of-python-socket-programming/

Hiilitieto. Ei päiväystä. Energiankulutus. [Verkkosivu]. [Viitattu 22.04.2020]. Saatavana: https://www.hiilitieto.fi/hiilitietoa/hiili-suomessa/energiankulutus/

JetBrains. Ei päiväystä. Version control. [Verkkosivu]. [Viitattu 30.5.2020]. Saatavana: https://www.jetbrains.com/help/pycharm/settings-version-control.html

Jennings, N. Ei päiväystä. Socket Programming in Python (Guide). [Verkkosivu]. [Viitattu 10.4.2020]. Saatavana: https://realpython.com/python-sockets/

Ji, J., Miao, C. & Li, X. 2020. Research on the energy-saving control strategy of a belt conveyor with variable belt speed based on the material flow rate. [Verkkoartikkeli]. PLoS ONE. [Viitattu 20.4.2020]. Saatavana: https://doi.org/10.1371/journal.pone.0227992

Momjian, B. 2000. PostgreSQL Introduction and Concepts. [Verkkokirja]. Addison–Wesley. [Viitattu 22.4.2020] Saatavana: http://www.foo.be/docs-free/aw_pgsql_book.pdf

Nielsen, M. 2015. Neural Networks and Deep Learning. [Verkkosivu]. Determination Press. [Viitattu 10.4.2020]. Saatavana: http://neuralnetworksanddeep-learning.com

Pedregosa, F., Varoquaux,G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J, Passos, A., Cournapeau, D., Brucher, M, Perrot, M. & Duchesnay, E. 2011. Scikit-learn: Machine Learning in Python. [Verkkoartikkeli]. Journal of Machine Learning Research 12 (2011) 2825-2830. [Viitattu 10.4.2020] Saatavana: http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

Tilastokseskus. 2020. Fossiilisten polttoaineiden kulutus väheni 6 prosenttia vuonna 2019. [Verkkojulkaisu]. Viitattu 22.4.2020]. Saatavana: https://www.stat.fi/til/ehk/2019/04/ehk_2019_04_2020-04-17_tie_001_en.html

Salawu, G., Bright, G. & Onunka, C., 2020. Modelling and Simulation of a Conveyor Belt System for Optimal Productivity. [Verkkoartikkeli]. International Journal of Mechanical Engineering and Technology, 115-121. [Viitattu 20.4.2020]. Saatavana: http://www.iaeme.com/IJMET/issues.asp?JType=IJMET&VType=11&IType=1

Sallinen, P. 2020. Ydinvoimalla päästöjä vastaan. [Verkkoartikkeli]. Energia uutiset. [Viitattu 22.4.2020]. Saatavana. https://www.energiauutiset.fi/tuotanto/ydinvoimalla-paastoja-vastaan.html

Ympäristöministeriö, maa- ja metsätalousministeriö ja työ- ja elinkeinoministeriö. 2015. Suomen ilmastopolitiikka – kohti vähähiilistä ja energiatehokasta yhteiskuntaa. [Verkkojulkaisu]. [Viitattu 22.4.2020]. Saatavana: https://ilmastoopas.fi/ilocms-portlet/article/8a54c390-fed4-42da-a2c2-4bab74993ebd/r/193d4fa4-49ed-4639-aab0-a30e52813f8e/suomen_ilmastopolitiikka_cmyk.pdf