Tampereen yliopisto

Konsta Boman

DATA-INTENSIIVISEN OHJELMISTON REFAKTOROINTI PYTHON-YMPÄRISTÖSSÄ

Tapaustutkimus

Informaatioteknologian ja viestinnän tiedekunta
Diplomityö
Tammikuu 2020

TIIVISTELMÄ

Konsta Boman: Data-intensiivisen ohjelmiston refaktorointi Python-ympäristössä
Diplomityö
Tampereen yliopisto
diplomi-insinöörin tutkinto-ohjelma
Tammikuu 2020

Python-ympäristöön on viimeisen vuosikymmenen aikana kehitetty monipuolinen kattaus tehokkaita ja suosittuja data-analyysityökaluja. Python on saavuttanut perinteisiä data-analyysityökaluja, kuten ohjelmointikieli-R:ää, niin suosiossa kuin toiminnallisuuden kattavuudessa. Työn tarkoitus on tutkia Pythonia data-analyysityökaluna ja löytää uusien data-analyysikirjastojen käyttötarkoituksia sekä hyviä ja huonoja puolia. Kerättyä tietoa hyväksi käyttäen tehdään tapaustutkimus, jossa kirjoitetaan uudelleen Pythonilla toteutettu vanha data-analyysisovellus. Vanha sovellus kaipaa uudelleentoteutusta, sillä sen ylläpito on työlästä ja sen suorituskyky on heikko. Näihin ongelmiin etsitään ratkaisua kirjoittamalla data-analyysi uudelleen uusia kirjastoja hyödyntäen.

Tapaustutkimuksen data-analyysisovellus analysoi testattavalta kosketuspaneelilta saatua dataa ja vertaa sitä robotilta saatuun referenssidataan. Datasta analysoidaan erilaisia virhearvoja, kuten lineaarivirhettä. Sovellus käyttää tiedon tallennukseen relaatiotietokantaa.

Uusi toteutus vastaa vanhan toteutuken toiminnallisuutta. Toteutuksia vertaillaan niiden suorituskyvyn ja ylläpidettävyyden perusteella. Suorituskykyä mitataan Python-profiloijalla ja ylläpidettävyyttä verrataan rivimäärällä ja huonoon lähdekoodiin viittaavia rakenteita etsimällä.

Uuden toteutuksen pääasiallisiksi työkaluiksi valikoituivat NumPy ja Pandas, jotka auttoivat ongelmakohtien ratkaisemisessa merkittävästi. Uusi toteutus on näiden työkalujen käytön myötä reilusti tehokkaampi ja tiiviimpi. Suorituskykyparannukset saavutettiin hyödyntäen NumPyn:n ja Pandasin optimoituja metodeja natiivien Python-metodien sijaan ja suurin syy koodikannan pienenemiselle on vastuunsiirto uusille kirjastoille monen toiminnallisuuden kohdalla.

Avainsanat: Data-analyysi, Python, Pandas, NumPy

ABSTRACT

Konsta Boman: Refactoring data-intensive software in Python environment
Master of Science Thesis
Tampere University
master of science programme
January 2020

Python environment has received a versatile, efficient and popular set of tools for data-analysis in the past decade. Python has reached more traditional data-analysis tools, such as the programming language R, in popularity and functionality. In this thesis, Python is researched as a data-analysis tool. The goal is to find the most efficient new libraries to develop data-analysis. The libraries are studied for their intented uses and in order to find their pros and cons. Based on the literature review, a case study is conducted. An old Python based data-analysis application is rewritten with new tools. The old application is in need of a reimplementation since it is troublesome to maintain and the performance is poor. The maintainability and performance issues are the two main problems that the thesis aims to solve.

The application studied in this case study analyses performance of touchpanels. Coordinates reported by the touchpanel are compared to reference values from a robot-drawn line. Error values such as linearity-error are calculated from the data. The application uses relational database for data storage.

The functionality of the new implementation is made to correspond the old application. Implementations are compared based on their maintainability and performance. Performance is measured with a profiler and maintainabilites are compared based on lines of code and structures that implicate poor design.

NumPy and Pandas were selected as the main tools for the new implementation. These libraries played a major role in solving the performance and maintainability issues. The new implementation has greatly improved performance and requires a lot smaller code base in comparison to the old implementation. Most of the performance improvement was achieved by utilizing optimized methods of NumPy and Pandas over native Python methods. Smaller code base is mainly due to the transfer of liability to NumPy and Pandas.

Keywords: Data analysis, Python, Pandas, NumPy

ALKUSANAT

Tämä diplomityö on tehty OptoFidelitylle, jota haluan kiittää työn mahdollistamisesta. Työni tarkastajina toimivat Timo Aaltonen ja Jouni Mäkitalo. Heitä haluan kiittää työni ohjauksesta, kommenteista ja kielioppivirheiden metsästämisestä. Kiitän myös esimiestäni Joni Piililää työaikani reilusta resursoinnista diplomityön tekemiseen. Lopuksi kiitän avopuolisoani Sannia oikoluvusta ja kaikesta tuesta.

Tampereella, 27. tammikuuta 2020

Konsta Boman

SISÄLLYSLUETTELO

TAULUKKOLUETTELO

OHJELMA- JA ALGORITMILUETTELO

LYHENTEET JA MERKINNÄT

Android	Käyttöjärjestelmä mobiililaitteille
cProfile	Suorituskykyprofiloija Pythonille
CSV	Comma-Separated Values, eli pilkulla erotellut arvot on datan tallennusmuoto
Excel	Taulukkolaskentaohjelmisto
HTML	Hypertext Markup Language on standardoitu kuvauskieli, jolla kuvataan pääasiassa internetissä esitettäviä dokumentteja
JSON	JavaScript Object Notation on datan tallennusmuoto
Mozilla Firefox	Avoimeen lähdekoodiin perustuva suosittu selain
MySQL	Relaatiotietokanta
NumPy	Numperical Python on Python-kirjasto numeeriseen laskentaan
ORM	Object-Relational Mapping on tapa kuvata tietokantoja korkean tason ohjelmointikielissä
Pandas	Python-kirjasto datan käsittelyyn
PostgreSQL	Relaatiotietokanta
Python	Ohjelmointikieli
Python-kirjasto	Lisäosa Python-ohjelmointikieleen
Relaatiotietokanta	Tietokanta, johon tallennetulla datalla on riippuvuuksia joiden perusteella dataa voidaan valikoida
SQL	Structured Query Language on kyselykieli, jolla tehdään kyselyitä relaatiotietokantaan
SQLAlchemy	Python-kirjasto SQL yhteyksien ja kyselyiden hallintaan, joka hyödyntää ORM:ää
SQLite	Relaatiotietokanta
sqlite3	Python-kirjasto, joka toimii rajapintana SQLite-relaatiotietokantaan
Stack Overflow	Verkkopalvelu ohjelmointiin liittyville kysymyksille ja vastauksille
Stack Overflow Trends	Verkkopalvelu, josta löytyy tietoa eri aihepiirien suosiosta Stack Overflow palvelussa
Suorityskyky profiloija	Mittaa ohjelmiston suorituskykyä
Tietokanta	Tietokoneelle tallennetua dataa, jota voidaan hakea, lisätä ja päivittää

TnT-alusta	Touch and Test -alusta on OptoFidelity:n kehittämä alusta robotin ohjaukselle
TPPT	Touch Panel Performance Tester on OptoFidelity:n tuote, jolla testataan kosketuspaneeleita

1 JOHDANTO

Python-ympäristöön on viime vuosien aikana kehitetty suuri määrä kirjastoja, jotka mahdollistavat tehokkaiden data-analyysisovellusten kehittämisen [23] [10] [28] [29]. Työn tarkoitus on tutkia näitä työkaluja ja niiden avulla löytää tapoja kehittää helposti ylläpidettävää ja tehokasta data-analyysiä Pythonilla. Ongelmaa lähestytään esimerkkitapauksen kautta. Esimerkkitapaus on olemassa oleva Pythonilla kirjoitettu data-analyysiratkaisu, joka laskee virhearvoja kosketuspaneelin raportoimasta kosketusdatasta. Ratkaisun koodikanta on vanha, se sisältää runsaasti itsetoteutettuja rakenteita ja alkuperäisten kehittäjien tietotaitoa on jäljellä vain vähän. Olemassa olevan ratkaisun ylläpito on näistä syistä työlästä. Työssä toteutetaan uudelleen osakokonaisuus esimerkkisovelluksesta. Uusi toteutus on kirjoitettu kirjallisuuskatsauksen perusteella lupaaviksi todettujen työkalujen avulla. Toteutuksia vertaillaan suorituskyvyn lisäksi niiden ylläpidettävyyden perusteella.

Työn rakenne on seuraava: Luvussa 2, *Datan käsittely Pythonilla*, esitetään teoriaa ja mahdollisia työkaluja, joiden avulla Pythonpohjaista datan käsittelyä on nykypäivänä järkevää tehdä. Luvussa tuodaan esiin erilaisten työkalujen tarkoitus, sekä niiden hyvät ja huonot puolet. Näiden tietojen pohjalta on toteutettu esimerkkiohjelmasta uusi versio, joka vastaa vanhan toteutuksen toiminnallisuutta. Tarkempi kuvaus esimerkkisovelluksesta ja sen ongelmista kuvataan luvussa 3, *Esimerkkitapaus*. Luvussa 4, *Arivointi ja vertailu -menetelmät*, käydään läpi kaksi työssä käytettävää mittaria: ylläpidettävyys ja suorituskyky. Ylläpidettävyyden määritelmä ja sen arviointimenetelmät sisältyvät lukuun 4. Uutta ja vanhaa ratkaisua arvioidaan ja vertaillaan luvussa 5, *arivointi ja vertailu*. Luku 5 on jaoteltu sovelluksen toiminnallisuuksien mukaisesti.

2 DATAN KÄSITTELY PYTHONILLA

Python data-analyysityökaluna on viime vuosina lisännyt suositotaan huimasti. Pythonin ympärille on rakentunut suuri määrä data-analyysikirjastoja, jotka yhdessä tarjoavat kattavan alustan data-analyysin tekemiselle [34]. Valtaosa Python-aiheisista data-analyysiä käsittelevistä kirjoista mainitsee vähintään NumPy- ja Pandas-kirjastot, jotka nähdään data-analyysin peruspilareina Python-ympäristössä [23] [15]. Näiden kirjastojen lisäksi erilaisille data-analyysiin liittyville tehtäville, kuten visualisoinnille ja koneoppimiselle, on lukuisia kirjastoja [10]. Kuva 2.1 esittää datan käsittelyyn Pythonille tarjolla olevien työkalujen laajaa tarjontaa. Kuva jaottelee datan käsittelyn osa-alueisiin ja kullekin osa-alueelle on lueteltu tarjolla olevia työkaluja.

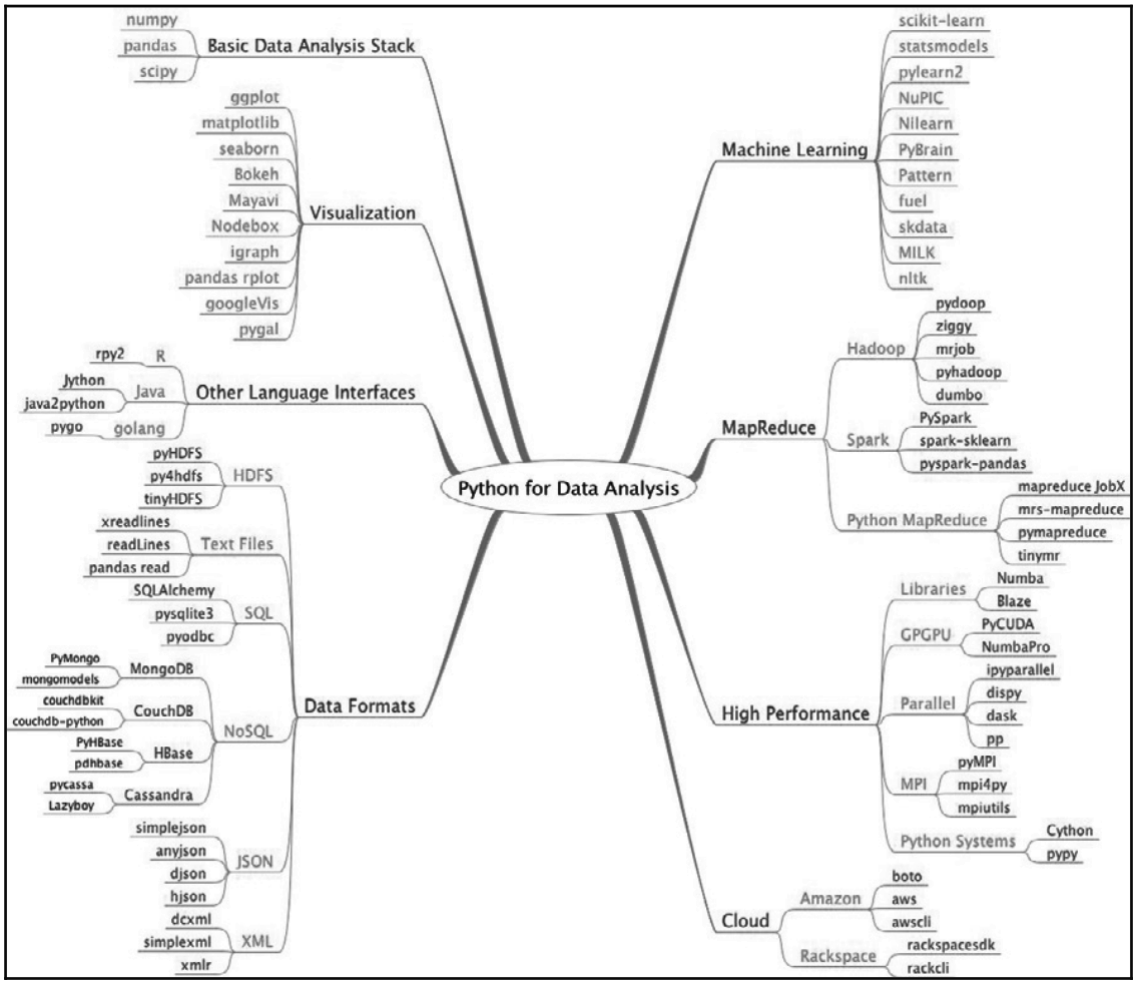
Tässä luvussa esitellään nämä peruskirjastot, jotka luovat perustan datan käsittelylle Pythonilla. Luku ei käy kattavasti läpi kirjastojen toiminnallisuutta, vaan antaa yleiskuvan tärkeimmistä kirjastoista ja niiden roolista data-analyysissä.

2.1 NumPy

NumPy tarjoaa tehokkaan moniulotteisen matriisimuotoisen tietorakenteen, jolla on laaja valikoima funktioita datan indeksointiin, pilkkomiseen ja numeeriseen laskentaan [26] [22]. Lähes kaikki korkeamman tason työkalut, mukaan lukien tässä työssä käytetyt kirjastot, ovat toteutettu NumPy:n avulla [22].

NumPy mahdollistaa tehokaan tavan laskea numeerista dataa Pythonissa. NumPy:ssä matriisien kaikki alkiot koostuvat aina samasta tietotyypistä ja niiden koko on ennalta määrätty. Koska jokainen alkio matriisin sisällä on samaa tyyppiä, voidaan hyödyntää niin sanottua *vektorisaatiota* eli laskutoimituksen suorittamista kaikille matriisin alkioille iteroimatta niitä korkealla tasolla. Nämä vektrorisoidut laskutoimitukset antavat merkittävän suorituskykyparannuksen verrattuna natiivisti Pythonilla toteutettuihin algoritmeihin, sillä NumPy-matriisi suorittaa laskutoimituksia optimoiduilla matalan tason ohjelmointikielillä toteutetuilla algoritmeillä. [38] [10]

Datan käsittely perustuu NumPy-matriiseihin, koska se tarjoaa korkean tason esityksen numeerisesta laskennasta vahingoittamatta suorituskykyä [38]. Datan käsittelyyn liittyvät korkeamman tason työkalut, kuten Pandas, ovat toteutettu NumPy:n avulla ja käyttävät NumPy-matriiseja sisäiseen datan käsittelyyn. On siis tärkeää ymmärtää NumPy-matriiseja tehokasta data-analyysiä kehittäessä.

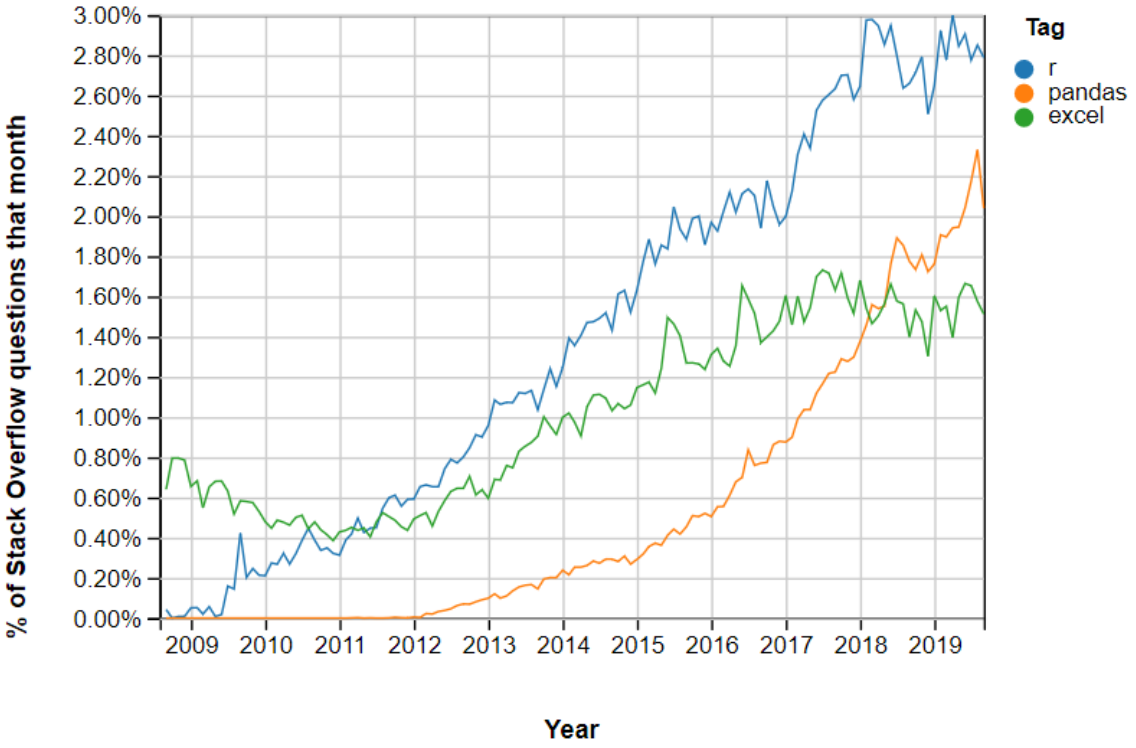


Kuva 2.1. *Pythonin data-analyysi-ympäristö vuodelta 2017. Data analyysi on jaoteltu osa-alueisiin ja kullekin alueelle on lueteltu tarjolla olevia Python kirjastoja. [10]*

2.2 Pandas

Pandas tarjoaa tehokkaita ja helppokäyttöisiä tietorakenteita ja työkaluja data-analyysin tarpeisiin Pythonilla [28] [22]. Pandas on hyvin suosittu ja monen lähteen mukaan yksi nykyaikaisen Pythoniin perustuvan data-analyysin peruspilareista [22] [23] [10] [30]. Pandasin suurta suosiota ja sen kasvua kuvastaa hyvin tunnetun *Stack Overflow* -yhteisön kysymysten ja vastausten määrä aiheesta Pandas. Stack Overflow on alusta, joka tarjoaa yhteisölle mahdollisuuden kysyä, vastata ja keskustella ohjelmointiin liittyvistä aiheista. *Stack Overflow Trends* on palvelu, joka tarjoaa visualisoitua dataa kysymysten määrästä kullekin ohjelmointiin liittyvälle aihepiirille [37]. Kuva 2.2 näyttää, kuinka Pandas on tällä vuosikymmenellä kasvattanut suosiotaan ja lähes saavuttanut kysymysten määrässä perinteisemmän R-ohjelmointikielen ja jopa ohittanut Excel-aiheisten kysymysyten määrän. Pandasilla on siis suuri käyttäjäkunta ja yhteisö, josta on suuri apu Pandasia käyttäessä.

Pandas koostuu kahdesta tietorakenteesta: *Series* ja *DataFrame*. Series on vektorimainen rakenne, jonka data on tallenntetu yksiulotteiseen NumPy-matriisin. Lisäksi Series sisältää niin sanotun *indeksin*, joka nimeää Seriesin kaikki datapisteet. Datapisteitä voi-



Kuva 2.2. *Stack Overflow Trends palvelusta haettu tulos 30.10.2019. Hakusanoina käytetty Pandas, Excel ja R.*

daan indeksin avulla käsitellä selkokielisten nimikkeiden avulla. DataFrame on kaksiulotteinen rakenne, joka sisältää sekä rivejä että sarakkeita. Sarakkeilla ja riveillä on kummallakin oma indeksinsä. DataFrame voidaan nähdä koostuvan Series-rakenteista, jotka jakavat saman rivi-indeksin. DataFrame on Pandasin ensisijainen tietorakenne. DataFrame ja Series rakenteille on tarjolla kattava lista metodeja, joita useimmissa datankäsittelysovelluksissa tarvitaan. Tietorakenteita voidaan pilkkoa, iteroida, yhdistää, muokata ja laskea valmiilla optimoiduilla metodeilla. [28] [22]

2.2.1 Datan luku ja kirjoitus

Yksi Pandasin vahvuuksista on monipuolinen tuki datan lukuun ja tallennukseen. Datan lataaminen DataFrame-tietorakenteeseen onnistuu monista lähteistä yksinkertaisilla komennoilla. Dataa voidaan lukea muun muassa CSV-, Excel-, JSON-, HTML- ja SQL-lähteistä. Pandas sijoittaa luetun datan suoraan omiin tietorakenteisiinsa. Samoihin datamuotoihin data voidaan tallentaa suoraan DataFrame-tietorakenteesta. Pandas soveltuu siis erinomaisesti esimerkiksi datan yhtenäistämiseen erilaisista tietolähteistä. Esimerkkiohjelma 2.1 demonstroi kuinka Pandasilla ja sqlite3:lla voidaan lukea ja kirjoittaa SQL-relaatiotietokantaan. Rivillä 6 luetaan SQL-kysely DataFrame tietorakenteeseen ja rivillä 10 DataFrame kirjoitetaan tietokantaan. [28]

```
1  import pandas as pd
2  import sqlite3
3
4  db_conn = sqlite3.connect('database.sqlite')
5  query = 'SELECT * FROM esimerkkitaulu'
6  dataframe = pd.read_sql_query(query, db_conn)
7
8  dataframe['summa'] = dataframe['esimerkkisarake'] + 10
9
10 dataframe.to_sql('summataulu', db_conn)
11
12 ikkunan_pituus = 10
13 dataframe['summa'].rolling(ikkunan_pituus).max()
```

Ohjelma 2.1. *Yksinkertainen Pandas esimerkki, jonka ainoa tarkoitus on esitellä Pandasin perustoiminnallisuutta.*

2.2.2 Datan jäsennys ja läpikäynti

Pandasin alkuperäisen kehittäjän yksi tärkeimmistä motiiveista oli sellaisen tietorakenteen luominen, jossa data joko automaattisesti tai vähintään selkeästi asettuu nimetyille akseleille [22]. Pandasissa data voidaan helposti datalähteestä riippumatta ladata nimetyille akseleille. Akseleita ja tietorakennetta voidaan tämän jälkeen jäsentää erilaisilla tehokkailla metodeilla kuten *groupby* ja *transpose*. Groupby jakaa DataFramen osiin määritellyn indeksi- tai sarake-nimikkeen perusteella ja transpose suorittaa datalle transpoosin eli peilaa rivit sarakkeiksi ja sarakkeet riveiksi [35]. Pandasin tietorakenteiden dataan päästään käsiksi esimerkiksi esimerkkiohjelman 2.1 rivillä 8, jossa DataFramen `esimerkkisarake` sarakkeen arvoihin lisätään luku 10 ja tallennetaan saadut arvot saman DataFrame:n uuteen sarakkeeseen `summa`. Luvun 10 lisäys tapahtuu vektorisoidusti kaikille DataFramen `esimerkkisarake` arvoille.

Hyvä esimerkki tapaustutkimuksessakin tarvittavasta datan tarkastelusta on *liukuva ikkuna*. Liukuva ikkuna tarkoittaa aineiston läpikäymistä tietynsuuruisia otoksia eli ikkunoita tutkimalla. Otokset koostuvat kulloinkin tarkasteltavan pisteen ympärillä sijaitsevasta datasta [23]. Pandas tarjoaa valmiin metodin, jolla dataa voidaan tarkastella liukuvan ikkunan avulla. Esimerkkiohjelman 2.1 rivillä 13 saraketta `summa` tarkastellaan 10 alkion mittaisen ikkunan avulla. Jokaisesta ikkunasta haetaan suurin arvo metodin `max` avulla. Liukuvasta ikkunasta voitaisiin laskea suoraan myös monia muita arvoja, joista lisää seuraavassa kappaleessa. Liukuvalle ikkunalle voidaan kutsua tarpeen vaatiessa myös räätälöityjä funktioita. [28]

2.2.3 Valmiiksi toteutettuja tilastomatematiikan sovelluksia

Pandasiin on sisäänrakennettu tilastomatematiikassa yleisesti käytettyjä menetelmiä, joita voidaan helposti suorittaa DataFrame:lle rivi- tai sarake-kohtaisesti. Yleisimmin tar-

vittavia tuettuja laskutoimituksia ovat esimerkiksi mediaani, keskihajonta ja minimi- tai maksimiarvon etsiminen. Kattava lista kaikista valmiista menetelmistä löytyy Pandasin dokumentaatiosta. [28]

2.2.4 Puuttuvat arvot

Koska puuttuvat arvot ovat yleinen asia data-analyysin parissa, yksi Pandasin alkuperäisistä tavoitteista oli tehdä puuttuvien arvojen käsittelystä mahdollisimman helppoa. Oletuksena Pandasin sisäänrakenntetuissa laskuissa puuttuva data jätetään huomiotta. Pandasin sisään rakennettuja laskuja voidaan siis suorittaa vaikka datassa olisi puutteita. Käyttäjän vastuulle jää arvioida puuttuvan datan merkitys suuremmassa mittakaavassa.

Puuttuvaa dataa voidaan huomiotta jättämisen lisäksi suodattaa tai täyttää Pandasin valmiilla metodeilla. Suodatus tarkoittaa puuttuvien data-alkioiden poistamista datasta kokonaan ja täyttäminen tarkoittaa puuttuvien alkioiden alustamista johonkin arvoon. Täytettävät arvot voivat olla käyttäjän määrittämiä tai Pandasin muuhun dataan sovittamia. Pandasilla on monia metodeja puuttuvan datan sovittamiseen olemassa olevaan dataan. [22] [28]

2.3 SQL

Tämä osio käsittelee relaatiotietokannan käyttöä Python-ympäristössä. Relaatiotietokanta sisältää dataa taulukkomuodossa ja taulukoiden väleillä voi olla riippuvuussuhteita. SQL eli Structured Query Language on kyselykieli, jolla relaatiotietokantaa käsitellään.

Esimerkkejä relaatiotietokannoista ovat: SQLite, MySQL ja PostgreSQL [36] [25] [32]. Pääpiirteiltään nämä tietokannat ovat samanlaisia. Tässä työssä keskitytään kevyeen SQLite tietokantaan, sillä tapaustutkimuksessa käsiteltävä sovellus käyttää kyseistä tietokantaa. SQLite on laajalti käytössä oleva tietokanta. Sitä käyttää muun muassa Mozilla Firefox ja suurin osa Android-sovelluksista. Pythonin vakiojakeluversioon on sisällytetty sqlite3 niminen kirjasto, joka toimii rajapintana Pythonin ja SQLite-tietokannan välillä.

Kuten jo aiemmassa Pandas-osiossa on käynyt selville, Pandas tarjoaa yksinkertaisen rajapinnan SQL-kyselyiden suorittamiseen ja datan lukemiseen DataFrame-tietorakenteeseen. Pandas kuitenkin tarvitsee erillisen kyselytoteutuksen kulloinkin kyseessä olevaan tietokantaan. Pandas osaa käyttää hyväkseen kaikkia käytetyimpiä kyselytoteutuksia, kuten edellä mainittua sqlite3:a [28]. Esimerkkiohjelmassa 2.1 rivillä 4 näkyy, kuinka sqlite3-yhteys luodaan ja sen jälkeen riveillä 6 ja 10 yhteys välitetään Pandasin SQL metodeille. Tämä tapa tehdä SQL-kyselyitä Pandasilla ja sqlite3:lla on kevyt ja yksinkertainen. SQL-kyselyiden abstrahointiin on olemassa Python-kirjastoja, jotka hyödyntävät niin sanottua ORM eli Object-relational mapping -tekniikkaa. ORM-tekniikka sitoo tietokannan taulut Pythonilla luotuihin luokkiin. Yksi suosittu ORM kirjasto on SQLAlchemy. [10] [15]

2.3.1 SQLAlchemy

SQLAlchemy on ORM-tekniikkaa hyödyntävä Python-kirjasto, joka luo relaatiotietokannan ja Pythonin välille abstraktiokerroksen. ORM mahdollistaa SQLAlchemyn sisäisen kuvauksen tietokannan relaatioista, jolloin ohjelmoijan tarve käyttää suoria SQL-kyselyitä poistuu. ORM:n avulla käyttäjä voi yhdellä SQLAlchemy-kyselyllä korvata monimutkaisia SQL-kyselyitä. Suorien SQL-kyselyiden sijaan käytetään SQLAlchemyn API:a ja tietokannan kuvaavia Python-luokkia. [10] [9] SQLAlchemyn avulla päästään siis eroon SQL-kyselyistä, jotka voivat joissakin tapauksissa olla hyvinkin monimutkaisia. [9] Toisaalta abstraktiokerroksen ja uuden API:n lisääminen koodiin heikentää suorituskykyä [10]. Jää myös tapauskohtaiseksi, milloin tietokannan kuvaaminen Python-luokilla luo enemmän haittaa kuin hyötyä. Erillisten kuvausluokkien luonti ja ylläpito eivät ole ilmaista. ORM:stä on usein eniten hyötyä, kun käsiteltävä tietokanta on monimutkainen ja sisältää useita riippuvuuksia.

2.3.2 Muita kirjastoja

Tässä osiossa esitellään lyhyesti muita hyödyllisiä data-analyysiin liittyviä Python kirjastoja, joita ei kuitenkaan syystä tai toisesta tarvittu tapaustutkimuksen sovelluksen uudelleen toteutuksessa. Esiteltävät kirjastot ovat SciPy, scikit-learn ja Spark.

SciPy on NumPyn:n päälle rakennettu avoimen lähdekoodin kirjasto, joka tarjoaa laajan valikoiman käyttäjäystävällisiä metodeja numeeriseen laskentaan. SciPy tarjoaa metodeja muun muassa integrointiin, interopolointiin, lineaarialgebraan ja signaalinkäsittelyyn liittyen. SciPy:n laaja valikoima toiminnallisuutta on jaoteltu niin dokumentaatiossa kuin kirjaston sisäisessä rakenteessa järkevästi eri osakokonaisuuksiin. Esimerkiksi intergrointiin liittyvät funktiot löytyvät dokumentaation osiosta *integrate* ja ohjelmassa funktiot esitellään seuraavasti `from scipy import integrate`. [34] SciPy:n toiminnallisuutta ei esimerkkitapausessa kuitenkaan tarvittu, vaan NumPy:n ja Pandasin matemaattiset funktiot riittivät tarvittavan toiminnallisuuden toteuttamiseen.

Scikit-learn on NumPy:ä ja SciPy:ä hyödyntävä avoimen lähdekoodin kirjasto, joka toteuttaa useita yleisiä koneoppimisalgoritmejä. Scikit-learn:in rajapinta on toteutettu mahdollisimman helpoksi käyttää vaikkei käyttäjällä olisi aiempaa osaamista koneoppimisesta [29]. Tapaustutkimuksessakin tarvittava regressio ja regressiosta laskettavat virhearvot olisi voitu laskea scikit-learn:in `LinearRegression` ja `Metrics` -luokilla. Scikit-learn jäi kuitenkin hyödyntämättä, sillä sen metodien sisäinen datan validointi luo esimerkkitapauksen kannalta turhaa työtä suorittimelle.

Kaikki tähän mennessä käsitellyt kirjastot ovat keskittyneet paikalliseen laskentaan, eli laskentaan joka tapahtuu yhdellä järjestelmällä ja suorittimella. Paikallisen laskennan vastakohta on niin sanottu *klusteroitu*-laskenta. Klusterointi tarkoittaa ohjelman suorittamista usealla tietokoneella tietokoneiden muodostamassa klusterissa, eli verkossa [17]. Klusteroimalla saavutetaan parempi suorituskyky kuin yksittäisillä järjestelmillä on mah-

dollista saavuttaa. Klusterointia käytetään data-analyysissä silloin kun yksittäisen järjestelmän laskentateho ei riitä vastaamaan sovelluksen tarpeita. Tämä tapahtuu käytännössä silloin, kun analysoitavan datan määrä kasvaa liian suureksi yhdelle tietokoneelle. Klusterointia hyödyntäviä kirjastoja on saatavilla myös Pythonille. Yksi tällainen suosittu kirjasto on esimerkiksi PySpark, joka mahdollistaa Apache Spark nimisen klusterointiekosysteemin käytön Pythonilla. Esimerkkitapauksessa klusteroinnin käyttö ei nykyisillä vaatimuksilla ole tarpeen. Paikallinen laskentateho riittää sovelluksen käyttötarkoitukselle. Klusterointia hyödyntäviä kirjastoja ei tästä syystä tässä työssä ole enempää tutkittu.

3 ESIMERKKITAPAUS

Tapaustutkimuksessa käsiteltävä ohjelmisto on nimeltään TPPT eli Touch Panel Performance Tester. TPPT on kosketuspaneelien suorituskyvyn testaukseen ja mittaukseen käytettävä OptoFidelity:n kehittämä kaupallinen sovellus [27].

3.1 Yleiskuva

TPPT on rakennettu OptoFidelityn TnT eli Touch and Test alustalle, joka tarjoaa mahdollisuuden vuorovaikuttaa testattavan laitteen kanssa robotiikan avulla. Alustalla luodaan tarkkoja ja toistettavia eleitä kuten pyyhkäisyjä ja napautuksia testattavan kosketuspaneelin pinnassa [27]. TPPT käyttää näitä eleitä ja vertaa TnT:n ja kosketuspaneelin raportoimia koordinaatteja keskenään. Koordinaattidatasta analysoidaan sovelluskohtaisesti erinäisiä tuloksia kuten pyyhkäisyn *lineaarivirhettä* tai napautuksen tarkkuutta. Pyyhkäisyn lineaarivirhe tarkoittaa poikkeamaa pyyhkäisyn kosketusdataan sovitetusta suorasta ja napautuksen tarkkuus poikkeamaa robotin koskettamasta pisteestä x ja y -koordinaatteina.

TPPT on Pythonilla toteutettu ohjelmisto, joka jakautuu kahteen osuuteen: *testeihin* ja *analyysiin*. Testivaiheessa eleitä suoritetaan paneelin pinnassa ja raportoitu data tallennetaan relaatiotietokantaan. Kun halutut testit on ajettu, analyysiosuus voidaan käynnistää. Analyysiosuus lukee testien kirjoittamaa tietokantaa ja muodostaa sen perusteella verkkopohjaisen käyttöliittymän. Käyttöliittymästä valitaan haluttu testi, joka analysoidaan ja jonka tulokset esitetään.

Tässä työssä tullaan käsittelemään TPPT:n datan käsittelyä yhden esimerkkitestin avulla. Esimerkkitestiksi valikoitui pyyhkäisytesti, koska sen analysointi on monipuolista ollen kuitenkin yksinkertainen ele. Pyyhkäisyn analyysiin liittyy esimerkiksi kosketusdatan *koordinaatiston kierto* ja *lineaarinen regressioanalyysi*. Kosketusdatan koordinaatistoa kierretään niin, että pyyhkäisyn suunta on x-akselin mukainen. Näin tulosdatan x-koordinaatti kuvaa etäisyyttä pyyhkäisyllä ja y-koordinaatti tuloksen etäisyyttä varsinaisesta pyyhkäisystä. Koordinaatistoa kierretään origon ympäri *rotaatiomatriisin* avulla. Rotaatiomatriisin ja kosketuskoordinaatin *pistetulon* tulos on kosketuskoordinaatti kierretyssä koordinaatistossa. Pistetulo on matriiseille määritelty laskutoimitus [35]. Lineaarinen regressioanalyysi tarkoittaa suoran sovittamista aineistoon niin, että se parhaiten estimoi *tutkittavan muuttujan* arvoa *selittävän muuttujan* muuttuessa [24]. Tapaustutkimuksessamme selittävä muuttuja on sijainti robotin piirtämällä pyyhkäisyllä ja tutkittava muuttuja on poikkea-

ma tästä sijainnista. Pyyhkäisydataa analysoidaan myös liukuvan ikkunan avulla. Kuva 3.1 näyttää yhden pyyhkäisyn visualisoidun analyysin, joka sisältää 4 kuvaajaa. Points on regression line -kuvaaja esittää kosketuspisteet ja niihin sijoitetun regressiosuoran. Linear fit error calculated from regression line -kuvaaja esittää poikkeamaa regressiosuorasta ja kuvaajan viereen on laskettu maksimivirhe, virheen keskiarvo ja *neliöllinen keskiarvo* eli neliöityjen virheiden keskiarvon neliöjuuri [7]. Points on robot drawn line -kuvaaja visualisoi raportoidut kosketuspisteet ja robotin piirtämän pyyhkäisyn laitteen pinnassa. Jitter with a sliding window -kuvaaja esittää liukuvan ikkunan avulla lasketun *jitter*-arvon ja kunkin kosketuspisteen *offset*-arvon. Offset tarkoittaa raportoidun kosketuspisteen etäisyyttä robotin piirtämästä viivasta ja jitter suurimman ja pienimmän virheen erotusta kussakin ikkunassa. Kyseisessä pyyhkäisytestissä robotti on pyyhkäissyt testattavan kosketuspaneelin vasemmasta yläkulmasta oikeaan alakulmaan. Tapaustutkimus tulee keskittymään TPPT:n pyyhkäisytestin datan käsittelyn laskentaintensiivisiin osuuksiin. Nämä osuudet karkeasti jaoteltuna ovat:

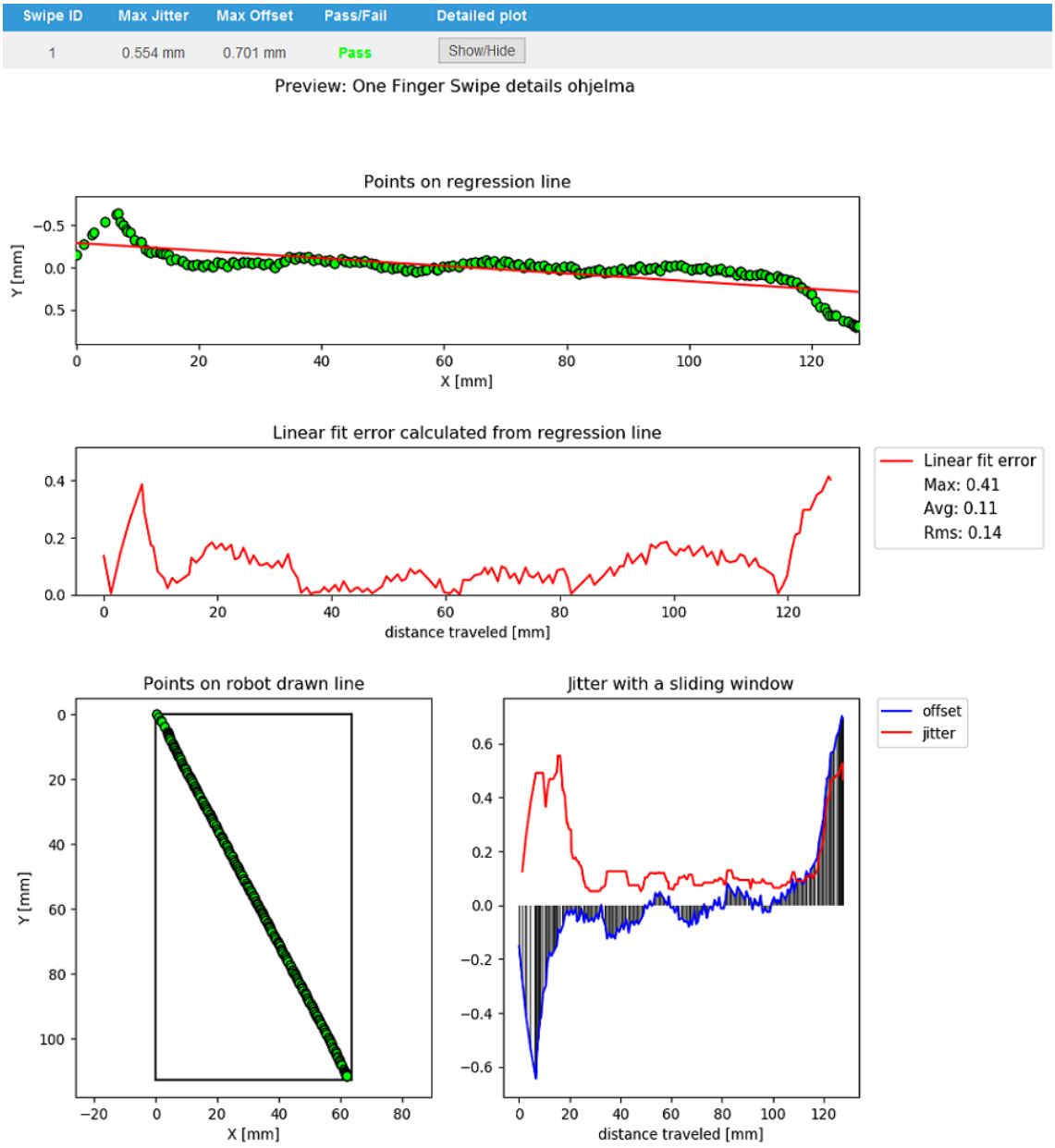
- Tulosdatan tallennus
- Tulosdatan luku ja suodatus
- Tulosdatan muokkaus käsiteltävään muotoon
- Haluttujen lukuarvojen, kuten lineaarivirheen laskenta

3.2 Nykyisen ratkaisun ongelmat

TPPT:n data-analyysi on osuus, joka vaatii lähes joka toimituskerralla joitakin muutoksia. Erilaisia kosketuspaneeleja ja niiden käyttötarkoituksia on lukematon määrä. Tämä tarkoittaa usein sitä, että testattavat eleet ja halutut analyysit saattavat vaihdella paljonkin TPPT-toimituksesta toiseen.

TPPT:n nykyinen koodikanta on vanhaa, eikä alkuperäisiä kehittäjiä tai kattavaa dokumentaatiota ole tarpeeksi. Nykyisessä koodikannassa on itsetehtyjä rakenteita, jotka voivat ensisilmäyksellä olla hyvinkin vaikealukuisia. Kehittäjien kesken keskustelua on herännyt myös käytössä olevasta SQL-kirjastosta *SQLAlchemy*:stä. SQLAlchemy luo sovelluksen ja tietokannan välille rajapinnan, jonka tarpeellisuutta TPPT:ssä tulee pohtia.

TPPT-analyysi kärsii myös suorituskykyongelmista. Tietokannan koon ja yksittäisen testin tuloskoon kasvaessa analysoinnin suoritusaika nousee usein kymmeniksi sekunneiksi ja pahimmissa tapauksissa minuuteiksi. Suuri osa suorituskykyongelmista on peräisin sovelluksen korkeamman tason ratkaisuista kuten siitä, että kaikki uudet testit analysoidaan heti sovelluksen avautuessa. Tällaisiin korkeamman tason ratkaisuihin ei tässä työssä kuitenkaan keskitytä. Suorituskykyä tullaan arvioimaan ja parantamaan matalammalla tasolla. Esimerkiksi, kuinka datan lataus on toteutettu ottamatta kantaa missä ohjelman vaiheissa dataa ladataan.



Kuva 3.1. *Erään pyyhkäisyn yksityiskohtaisen analyysin visualisointi TPPT sovelluksessa.*

4 ARVIOINTI JA VERTAILUMENETELMÄT

Tämä luku esittelee työssä käytettävät lähtokohdat, joiden perusteella esimerkkitapauksen uutta ja vanhaa toteutusta tullaan arvioimaan. Lähtökohdat ovat ajallinen suorituskyky ja ylläpidettävyys.

4.1 Suorituskyky

Suorituskyky ja skaalautuvuus ovat yleisiä vaatimuksia data-analyysin sovelluksille, sillä analyysiin liittyy usein suuret aineistot. Näin on myös työn esimerkkitapauksessa TPPT-:ssä. Kosketusdataa saattaa laitteesta ja testeistä riippuen tulla satojatuhansia rivejä, jolloin suorituskyvyn merkitys nousee huomattavaksi. Ajallista suorituskykyä mitataan niin sanotulla *profiloijalla*. Profiloija mittaa suoritetun ohjelman kunkin funktion kuluttaman suoritusajan. Ohjelman hitaat osiot on näin helppo löytää [19]. Yksi Python-profiloija on cProfile niminen kirjasto. CProfile sisältyy Pythonin vakiojakeluversioon ja on Pythonin dokumentaation useimpiin profilointitapauksiin suosittelema työkalu [33]. CProfile on saanut kritiikkiäkin, sillä sen avulla suoritusajan mittaaminen lisää ohjelman suoritusaikaa [13]. Tässä työssä vertaamme kuitenkin kahta eri toteutusta keskenään ja tärkeää on eroavaisuuksien löytäminen, eikä absoluuttisen suoritusajan mittaaminen.

Esimerkki cProfilen käytöstä on nähtävissä ohjelmassa 4.1. Esimerkkiohjelma luo Python-listan nimeltä `lista` ja täyttää sen 500 000:lla kokonaisluvulla. Muuttujan `lista` avulla luodaan vastaava NumPy-matriisi nimellä `np_lista`. Näiden kahden muuttujan kaikki alkiot jaetaan kahdella ohjelman funktioissa `np_jako` ja `python_jako`. Jako-operaatiohin kulutetut suoritusajat mitataan ja tulostetaan cProfile:n avulla. Rivillä 13 luodaan cProfile-instanssi, jonka metodien `enable` ja `disable` kutsujen välille jäävien ohjelmarivien suoritusajat mitataan. Lopulta rivillä 22 mittaustulokset tulostetaan. Tuloste on nähtävissä kuvassa 4.1. Tuloste on taulukko, jonka sarakkeet ovat *ncalls, tottime, percall, cumtime* ja *filename:lineno(function)*. Taulukon rivit kuuluvat mitatuille funktioille. Viimeinen sarake kertoo mille funktiolle ominaiset luvut on kirjattu millekin riville. Muiden sarakkeiden tarkoitukset ovat seuraavanlaiset:

- ncalls: funktiokutsujen lukumäärä
- tottime: funktiossa kulutettu suoritusaika
- cumtime: funktiossa ja sen sisällä kutsutuissa funktioissa kulutettu suoritusaika
- percall: yhteen funktiokutsuun käytetty suoritusaika

```
1  import cProfile
2  import numpy as np
3
4  def np_jako(np_lista):
5      return np_lista / 2
6
7  def python_jako(lista):
8      return [alkio / 2 for alkio in lista]
9
10 lista = range(0,500000)
11 np_lista = np.array(lista)
12
13 pr = cProfile.Profile()
14
15 pr.enable()
16
17 np_jako(np_lista)
18 python_jako(lista)
19
20 pr.disable()
21
22 pr.print_stats(sort='cumtime')
```

Ohjelma 4.1. *Esimerkki cProfile-profiloijan käytöstä.*

```
ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     1    0.000    0.000    0.058    0.058 esimerkki.py:7(python_jako)
     1    0.058    0.058    0.058    0.058 esimerkki.py:8(<listcomp>)
     1    0.002    0.002    0.002    0.002 esimerkki.py:4(np_jako)
```

Kuva 4.1. *Ohjelman 4.1 tuloste.*

Funktiossa `python_jako` ja sen alifunktioissa on siis kulunut 58 millisekuntia ja funktiossa `np_jako` 2 millisekuntia. Tulosteessa näkyy myös Pythonin sisäinen fuktio `<listcomp>`, jota Python kutsuu `python_jako` funktion for-silmukkarakennetta muodostettaessa.

4.2 Ylläpidettävyys

Toteutusten toiseksi arvioitavaksi ominaisuudeksi on valittu ylläpidettävyys. Ylläpidettävyys tarkoittaa ohjemiston julkaisun jälkeisen muokkaamisen helppoutta ja tehokkuutta [18] [16]. Toisin sanoen ylläpidettävyyden laskiessa tarvittava työmäärä ohjelmiston muuttamiseen nousee. Ohjelmiston julkaisun jälkeiseen muokkaamiseen eli ylläpitämiseen sisältyy monesta eri syystä tarvittavia muutoksia. Ylläpidon eri osa-alueet voidaan

jakaa neljään seuraavasti:

- Mukautuvat muutokset: tarvittavat muutokset kun ohjelmistoympäristö muuttuu
- Uudet vaatimukset: tarvittat muutokset ja muu työ, kun ohjelmistolle asetetaan uusia vaatimuksia
- Korjaavat muutokset: tarvittava työ virheiden korjaamiseksi
- Ennaltaehkäisevä työ: työtä, jonka tarkoituksena on ehkäistä tulevia onglemia

Noin kolme neljäsosaa ylläpitotyöstä sijoittuu uusiin vaatimuksiin ja mukautuviin muutoksiin, viidesosa työstä on korjaavia muutoksia ja loput ennaltaehkäisevää työtä [20].

4.2.1 Miksi ylläpidettävyys on tärkeää

Ylläpito on iso ja tärkeä osa ohjelmiston elinkaarta [6]. Jopa 60–90 % ohjelmistojen kustannuksista muodostuu julkaisun jälkeisestä ylläpidosta [3]. Liiketoiminnan kannalta on siis kriittistä, että ohjelmiston ylläpidettävyys on mahdollisimman hyvällä tasolla. Suuren kustannusosuuden lisäksi, ylläpidettävyys on merkittävä tekijä uusien liiketoimintamahdollisuuksien hyödyntämisessä. Kun uusi asiakastarve ilmenee, hyvin ylläpidettävässä ohjelmistossa sen toteuttaminen on helpon muokattavuuden ansiosta nopeampaa ja halvempaa. Näin muodostuu sekä etua kilpaileviin sovelluksiin että asiakastyytyväisyyttä. [6]

Ylläpidettävyys on erityisen tärkeää TPPT-esimerkkitapauksemme yhtydessä. TPPT:tä sovelletaan lukuisille erilaisille ja eri käyttötarkoituksellisille kosketuspaneeleille, joten paneeleita halutaan usein testata asiakaskohtaisesti eri tavoin. Testattavat eleet ja niistä halutut lukuarvot voivat vaihdella tapauskohtaisesti. Tästä syystä useat TPPT toimitukset vaativat muutoksia sekä testeihin että analyysiin.

4.2.2 Kuinka ylläpidettävyyttä arvioidaan

Ylläpidettävyyden arvioiminen ei ole yksiselitteistä, sillä sitä ei voida yleispätevästi mitata millään lukuarvolla. Lukuisat tutkimukset ovat yrittäneet löytää tapoja ylläpidettävyyden numeeriseen mittaukseen mutta yleisesti hyväksyttyä tapaa ei ole löytynyt [2] [4] [8]. Lähdekoodista on kuitenkin perinteisesti laskettu useitakin lukuarvoja, kuten *Lines of Code*, *Cyclomatic complexity* [21] ja *Halsteadin mitat* [14]. Lines of Code tarkoittaa lähdekoodin rivien määrää. Cyclomatic complexity on lukuarvo, joka summaa erilliset lähdekoodipolut, joita pitkin ohjelma voi suorittua [21]. Lähdekoodipolkuja muodostuu useita esimerkiksi if-lauseiden myötä, jolloin ohjelmaan kehittyy vähintään yksi lisäpolku: If-lauseen toteutuessa suoritetaan eri osio lähdekoodista verrattuna if-lauseen ollessa epätosi. Halsteadin mitat ovat lukuarvoja ohjelman laajuudelle, haastavuudelle ja työmäärälle. Lukuarvot lasketaan ohjelmaan sisältyvien funktioiden ja muuttujien lukumäärillä. Perinteisiä lukuarvoja on myös yritetty yhdistää yhdeksi ylläpidettävyyttä kuvastavaksi arvoksi nimeltä Maintainability Index [39]. Tässä kappaleessa esitellään näiden numeeristen arvojen lisäksi niin

sanottu *code smells* -käsite. Code smells:t ovat mahdollisesti huonoon ylläpidettävyyteen viittaavia ohjelmarakenteita lähdekoodissa [12].

Perinteiset numeeriset mittarit

Lines of Code eli lähdekoodin rivimäärää on historiallisesti käytetty niin ohjelmoijien tuotteliaisuuden kuin ohjelmiston monimutkaisuuden mittana. Rivimäärä on helppo ymmärtää ja laskea, joten se on laajaltikin käytetty mittari [11]. On kuitenkin selvää, että uusien ohjelmointikielien ja abstraktiotasojen myötä ohjelmarivien määrän merkitys on kyseenalainen. Tämä ongelma todettiin jo 1970-luvun puolivälissä, jolloin esimerkiksi Halsteadin kompleksisuus -mitat kehitettiin [11]. Mikään rivimäärää kehittyneempi ohjelman ylläpidettävyyttä mittaava lukuarvo ei kuitenkaan ole yleisesti hyväksytyssä käytössä niin akateemisissa kuin teollisuudenkaan sovelluksissa. Lukuarvot ovat joko liian hankalia laskea, tai niiden merkitys on kiisteltyä [11].

Esimerkkiohjelman rivimäärää tullaan kuitenkin tarkastelemaan, sillä pienempi koodikanta on usein helpommin luettava ja lähestyttävä [31]. Tässä työssä rivimäärään tullaan laskemaan vain rivit, jotka sisältävät toiminnallista koodia. Kommentteja ja tyhjiä rivejä ei siis lasketa Lines of Code -arvoihin. Muita perinteisiä ylläpidettävyyden mittoja ei työssä lasketa.

Code smells

Vuonna 1999 esiteltiin ensikerran *code smells* -käsite. Se tarkoittaa ohjelmarakenteita, jotka tulisi refaktoroida, eli kirjoittaa uudelleen paremmin [5]. Tällaisiksi code smells:ksi lueteltiin muun muassa monistettu koodi, pitkät funktiot ja pitkät parametrilistat [5]. Code smells -listaa uudistettiin vuonna 2018 [12]. Tämä kappale esittelee uudistetun listan code smells:t, joita tässä työssä tullaan esimerkkitapauksen lähdekoodista paikantamaan ja analysoimaan. Kaikkia kirjallisuudessa esiteltyjä code smells:ä ei tässä työssä käsitellä, sillä niistä suuri osa on tarkoitettu laajemman kokonaisuuden tarkasteluun kuin tässä työssä käsiteltävään data-analyysiin.

Mysteerinen nimi

Selkeä nimeäminen on yksi helppolukuisen koodin tärkeimmistä osista. Funktioiden, luokkien, metodien ja muuttujien nimien tulee selkeästi kuvastaa niiden tarkoitusta ja kuinka niitä tulee käyttää. Mikäli nimeaämisessä käytetään lyhenteitä, tulee niiden olla dokumentoituja tai itsestään selviä. Jos hyvän nimen keksimien on vaikeaa, saattaa se viitata suurempiin ongelmiin ohjelmistossa, joita on syytä tutkia. [12]

Monistettu koodi

Monistettu koodi tarkoittaa koodia, joka on kirjoitettu ohjelmaan useaan kertaan joko täsmälleen tai lähes samalla tavalla [12]. Monistettu koodi heikentää ylläpidettävyyttä huo-

mattavasti, sillä usein kun tällaisia osia ohjelmasta muutetaan, tulee kaikki monistettu koodi tarkastella eroavaisuuksien löytämiseksi. Tämän prosessi on aikaavievää ja virhealtista. Muutokset joudutaan myös tekemään kaikkiin monistettuihin osiin, joka luonnollisesti lisää työmäärää. Toisaalta liiallinen koodin yhdistäminen saattaa heikentää sen luettavuutta, mikäli sama ohjelmaosio toteuttaa useita käyttötarkoituksia.

Pitkä funktio

Pitkät funktiot ovat usein vaikeampia ymmärtää kuin lyhyet, sillä funktion tarkoituksen selvittämiseksi on enemmän luettavaa ja muistettavaa. Pitkän funktion jakaminen lyhyempiin hyvin nimettyihin funktioihin auttaa lukijaa ymmärtämään ja jäsentämään kokonaisuutta. Hyvällä lyhyellä funktiolla on aina yksi selvä vastuualue. Parhaassa tapauksessa funktion tarkoitus selviää lukijalle vilauksella. Usein lyhyempien funktoiden sisäistä toteutusta ei ole tarvetta nähdä, jolloin luettavan lähdekoodin määrä ylläpitotehtäviä tehdessä vähenee. Vanhoilla ohjelmointikielillä funktioiden kutsuminen loi prosessorille suhteellisen paljon ylimääräistä työtä ja tästä syystä historiassa lyhyitä funktioita on kartettu. Nykyisin ohjelmointikielet ovat poistaneet tämän ylimääräisen työn ohjelman sisäisissä kutsuissa lähes täysin. [12]

Pitkä parametrilista

Pitkät parametrilistat ovat usein vaikealukuisia. Koodin luettavuus kärsii, kun pitkillä parametrilistoilla varustetut funktioiden kutsut ja esittelyt joudutaan pilkkomaan usealle riville häiritsevän pitkien rivien välttämiseksi. Pitkä parametrilista vaikeuttaa myös funktion ymärrettävyyttä, kun lukijan on muistettava useita muuttujanimiä ja niiden tarkoituksia. Luettavuutta helpottaa kun parametrit ovat jäsennelty esimerkiksi olioihin tai muihin tietorakenteisiin. [12]

Hajaantuva muutos

Hajaantuva muutos tarkoittaa, että muutosta tehtäessa joudutaan muuttamaan useita eri alueita lähdekoodista. Mitä hajautuvampia muutokset lähdekoodissa ovat, sitä työläämmäksi ylläpito muuttuu. [12] Hajautuvat muutokset lisäävät myös riskiä virheiden luomiseen muutoksia tehtäessä. Ohjelmoijan on joka kerta muistettava tehdä ja testata muutokset kaikissa paikoissa.

Silmukkarakenteet

Silmukkarakenteet ovat yksi ohjelmoinnin perusrakenteista. Monissa tilanteissa on kuitenkin mahdollista hyödyntää vektorisaatiota silmukkarakenteen käytön sijasta [12]. Varsinkin korkean tason ohjelmointikielissä, kuten Pythonissa, silmukkarakennetta luodessa on aina syytä tutkia, mikäli rakenteen käytöltä on mahdollista välttyä, sillä suurien aineistojen iteroiminen korkealla tasolla on suoritusajallisesti kallista. Toisaalta liiallinen vektori-

sointi saattaa joissakin tilanteissa heikentää lähdekoodin luettavuutta, jolloin tulee pohtia suorituskyvyn ja luettavuuden tärkeysjärjestystä.

5 TOIMINNALLISUUSKOHTAINEN ARVIOINTI JA VERTAILU

Teorian pohjalta uudelleen kirjoitettua TPPT-toteutusta vertaillaan vanhan toteutuksen kanssa toiminnallisuus kerrallaan.

5.1 Datan kirjoitus

Datan kirjoitus SQLite-tietokantaan tapahtuu TPPT:ssä jokaisen robotin tekemän pyyhkäisyn jälkeen. Pyyhkäisyn jälkeen testattavalta laitteelta saadaan lista kosketusdatasta, jonka pyyhkäisy on aiheuttanut. Kosketusdata esiintyy sekä uuden että vanhan toteutuksen lähdekoodissa nimellä `touchlist`. Muuttuja `touchlist` koostuu sisäkkäisistä Python listoista, joissa sisemmät listat muodostuvat yksittäisen kosketuspisteen tekijöistä, kuten x- ja y-koordinaateista.

Vanha toteutus

Vanha toteutus datan kirjoituksesta tietokantaan on nähtävissä ohjelmassa 5.1. Se käyttää hyväkseen SQLAlchemyä, joka vaatii ohjelman 5.2 mukaisen luokan, joka kartoittaa SQLAlchemylle käytetyn tietokantataulun. Näiden ohjelmien lisäksi vanha toteutus sisältää erillisen tietokantaluokan, jonka metodia addAll käytetään dataa kirjoittaessa. Kyseisen tietokantaluokan alustusfunktio ja addAll-metodi ovat nähtävissä ohjelmassa 5.3. Muuttuja `touchlist` iteroidaan vanhassa toteutuksessa Pythonin for-silmukalla ja jokainen yksittäinen kosketuspistedata alustetaan SQLAlchemyn vaatimaan tietokannan kartoittavaan luokkaan OneFingerSwipeResults. Tämän jälkeen data tallennetaan tietokantaan SQLAlchemyn avulla.

Uusi toteutus

Uusi toteutus käyttää hyväkseen Pandasia ja NumPyä. Tämä toteutus on nähtävillä ohjelmassa 5.4. Uusi toteutus käyttää tietokantaan kirjoittamisessa hyväkseen Pandasin tietorakenteen DataFrame metodia `to_sql`. Ennen `to_sql`:n kutsumista tulee data ensin muuttaa Pandasin DataFrameksi. DataFrame:n luomiseksi `touchlist` muutetaan ensin NymPy-matriisiksi rivillä 2. Muodostuva muuttuja `np_touchlist` on kaksiulotteinen NumPy-matriisi, josta karsitaan mahdollinen ylimääräinen data [:,:7] -notaatiolla. Notaa-

```
1  def save_measurement_data(self, line_id, touchlist):
2      dblist = []
3      for testresult in touchlist:
4          testresultswipe = OneFingerSwipeResults()
5          testresultswipe.panel_x = float(testresult[0])
6          testresultswipe.panel_y = float(testresult[1])
7          testresultswipe.sensitivity = float(testresult[2])
8          testresultswipe.finger_id = int(testresult[3])
9          testresultswipe.delay = testresult[4]
10         testresultswipe.time = testresult[5]
11         testresultswipe.event = testresult[6]
12         testresultswipe.swipe_id = line_id
13         dblist.append(testresultswipe)
14     self.db.addAll(dblist)
```

Ohjelma 5.1. *Vanha toteutus tulosten tietokantaan kirjoituksesta.*

```
1  class OneFingerSwipeResults( Base ):
2      __tablename__ = 'one_finger_swipe_results'
3      id = Column( Integer, primary_key = True )
4      swipe_id = Column(
5          Integer,
6          ForeignKey('one_finger_swipe_test.id',
7                     ondelete='CASCADE'),
8          nullable=False )
9      swipe = relation(
10         OneFingerSwipeTest,
11         backref = backref('one_finger_swipe_results',
12         order_by = id) )
13     panel_x = Column( Float )
14     panel_y = Column( Float )
15     sensitivity = Column( Float )
16     delay = Column( Float )
17     finger_id = Column( Integer )
18     time = Column( Float )
19     event = Column( Integer )
```

Ohjelma 5.2. *Vanhassa toteutuksessa vaadittava pyyhkäisydatan tietokantataulun esitys SQLAlchemylle.*

```
1 def initialize()
2     self.db = create_engine('sqlite:///' + path)
3     self.session = sessionmaker(bind=self.db, autoflush=False)
4
5 def addAll( self, data):
6     session = self.session()
7     session.add_all( data )
8     session.commit()
9     session.close()
```

Ohjelma 5.3. *Vanhan toteutuksen tietokantaluokan alustus ja datan tallennukseen käytettävä metodi.*

```
1  def save_measurement_data(self, swipe_id, touchlist):
2      np_touchlist = np.array(touchlist)[:,:7]
3      columns = [
4          'panel_x', 'panel_y', 'sensitivity',
5          'finger_id', 'delay', 'time', 'event']
6      df = pd.DataFrame(np_touchlist, columns=columns)
7      df['swipe_id'] = swipe_id
8      connection = sqlite3.connect(database_file)
9      df.to_sql(
10         'one_finger_swipe_results',
11         connection, index=False,
12         if_exists='append')
13     connection.close()
```

Ohjelma 5.4. *Uusi toteutus tulosten tietokantaan kirjoituksesta.*

tio pilkkoo matriisin niin, että sisemmistä alkioista, eli yksittäisten kosketuspisteiden attribuuteista, jää jäljelle vain seitsemän ensimmäistä alkiota. Vain nämä seitsemän ensimmäistä attribuuttia tallennetaan tietokantaan. DataFrame:n muodostamiseksi tarvitsemme tämän karsitun kaksiulotteisen NumPy-matriisin lisäksi listan, joka määrittelee DataFrame:n sarakkeiden nimet. Riveillä 3-5 luodaan `columns` niminen Python-lista, joka sisältää kaikille seitsemälle kosketusdatan attribuutille nimen. Nämä nimet ja niiden järjestys vastaavat tietokantataulua, johon kosketusdata kirjoitetaan. DataFrame muodostetaan rivillä 6 NumPy-matriisin ja kolumnilistan avulla. DataFrame:en lisätään rivillä 7 vielä kolumni `swipe_id`. DataFrame `df` vastaa nyt täydellisesti tietokannan taulua `one_finger_swipe_results`. Rivillä 8 luodaan tietokantayhteys sqlite3-kirjastolla. Tämän tietokantayhteyden avulla kosketusdata kirjoitetaan tietokantaan riveillä 9-12.

Suorituskyky

Taulukko 5.1 esittää edellä kuvattuun datan kirjoitukseen kulutettuja suoritusaikoja. Tehdyissä testeissä on käytetty 18288 alkiota sisältävää listaa kosketusdatasta. Suoritusaikaa on mitattu sekä tarvittavasta työstä ennen varsinaista SQL-interaktiota, että varsinaisesta SQL-interaktiosta. Taulukon 5.1 rivi 'Datan valmistelu' kuvaa valmistelevaa työtä ja

Taulukko 5.1. *Uuden ja vanhan toteutuksen cProfilella mitatut datan kirjoitukseen käytetyt suoritusajat. Tallennettujen rivien määrä: 18288 kpl*

	Vanha toteutus (s)	Uusi toteutus (s)
Datan valmistelu	0,847	0,124
SQL – kirjoitus	3,405	0,095
Yhteensa	4,252	0,219

Taulukko 5.2. *Kummankin datan kirjoitus -toteutuksen LoC eli Lines of Code.*

	Vanha toteutus	Uusi toteutus
LoC	41	13

rivi 'SQL-kirjoitus' kuvaa SQL-interaktioon käytettyä suoritusaikaa.

Ylläpidettävyys

Taulukkoon 5.2 on laskettu kumpaankin toteutukseen tarvitun lähdekoodin rivimäärä eli Lines of Code. Uusi toteutus tietokantatauluun kirjoittamisesta on reippaasti tiivimmin ilmaistu. Koko uusi toteutus on tiiviyden ansiosta helposti esitetty samassa lyhyessä funktiossa, joka auttaa lukijaa ymmärtämään kokonaisuuden ilman tarvetta hyppimiselle lähdekoodin eri osien välillä. Vanhassa toteutuksessa ilmenee hajautuvan muutoksen ongelmaa, sillä tietokantaa kuvaavan luokan `OneFingerSwipeResults`:n muuttuessa, pitää muutos tehdä myös `save_measurement_data`-funktioon. Uudessa toteutuksessa vastaavan muutoksen tekeminen onnistuu yhdellä muutoksella. Ongelmaa pitkistä funktioista tai parametrilistoista ei ole kummassakaan toteutuksessa. Uuteen toteutukseen on muuttuja `line_id` nimi muutettu vastaamaan tietokannan vastaavan arvon nimeä `swipe_id` nimen vaihdoksen aiheuttaman hämmennyksen välttämiseksi.

Yhteenveto

Uusi Pandasia ja NumPy:ä hyödyntävä toteutus suoriutui tietokantatauluun kirjoituksesta yli kymmenen kertaa nopeammin. Suorituskykyparannus saavutettiin välttämällä hitaiden Python-rakenteiden käyttöä ja jättämällä SQLAlchemy:n käyttö pois toteutuksesta. Hitaiden python rakenteiden sijaan datan valmistelussa hyödynnettiin NumPy:n ja Pandasin vektorisoituja metodeja. SQLAlchemyn pois jättämisellä hävittiin mahdollisuus hyödyntää ORM tekniikan tuomaa helppoutta SQL-kyselyissä, mutta SQL-kyselyitä ei tallennetulle datalle TPPT:n testivaiheessa tehdä. Ylläpidettävyyden voidaan katsoa parantuneen huomattavasti, sillä uusi toteutus on paljon tiiviimpi, hajautuvan muutoksen ongelmista on päästy eroon eikä uusia varoittavia ohjelmarakenteita eli Code smells:ä ilmennyt.

```
1  class OneFingerSwipeTest( Base ):
2      __tablename__ = 'one_finger_swipe_test'
3      id = Column( Integer, primary_key = True )
4      test_id = Column(
5          Integer,
6          ForeignKey('test_item.id', ondelete='CASCADE'),
7          nullable=False )
8      test = relation(
9          TestItem,
10         backref = backref('one_finger_swipe_test',
11         order_by = id) )
12     start_x = Column( Float )
13     start_y = Column( Float )
14     end_x = Column( Float )
15     end_y = Column( Float )
16     through_x = Column( Float )
17     through_y = Column( Float )
18     radius_x = Column( Float )
19     radius_y = Column( Float )
```

Ohjelma 5.5. *Pyykäisytestin metadatan tietokantataulun kartoitusluokka SQLAlchemylle.*

```
1  dbsession = measurementdb.get_database().session()
2  dbswipes = dbsession.query(measurementdb.OneFingerSwipeTest).\
3      filter(measurementdb.OneFingerSwipeTest.test_id==self.test_id).\
4      options(joinedload('one_finger_swipe_results')).\
5      order_by(measurementdb.OneFingerSwipeTest.id)
6
7  for swipe in dbswipes:
```

Ohjelma 5.6. *Vanha toteutus datan luvusta ja iteroinnista.*

5.2 Datan luku ja iterointi

TPPT-analyysi lataa testien tietokantaan tallentaman datan. Pyyhkäisytestin metadata löytyy tietokantataulusta nimeltä `one_finger_swipe_test` ja laitteen raportoima kosketusdata tietokantataulusta nimeltä `one_finger_swipe_results`.

Vanha toteutus

Vanha toteutus käyttää datan lukemiseen SQLAlchemyä. Dataa luetaan kahdesta tietokantataulusta, joten kummallekkin taululle tarvitaan kuvausluokka. Luokat ovat nähtävissä ohjelmissa 5.2 ja 5.5. Itse datan lukevat ohjelmarivit ovat nähtävissä ohjelmassa 5.6. SQLAlchemyn avulla muuttujaan `dbswipes` ladataan yhdellä kyselyllä sekä pyyhkäisytestin metadata että laitteen raportoima kosketusdata.

```
1  con = sqlite3.connect(database_file)
2  query = """SELECT
3  swipe_id, start_x, start_y,
4  end_x, end_y, panel_x, panel_y
5  FROM
6  one_finger_swipe_results
7  INNER JOIN
8  one_finger_swipe_test ON
9  one_finger_swipe_results.swipe_id=one_finger_swipe_test.id
10 WHERE test_id={}""".format(test_id)
11
12 df_results = pd.read_sql_query(query, con)
13
14 for swipe_id, df_result in df_results.groupby('swipe_id'):
```

Ohjelma 5.7. *Uusi toteutus datan luvusta ja iteroinnista.*

Taulukko 5.3. *Uuden ja vanhan toteutuksen cProfilella mitatut datan lukuun ja iteroitiin käytetyt suoritusajat*

	Vanha toteutus (s)	Uusi toteutus (s)
Datan luku ja iterointi	2,281	0,371

Uusi toteutus

Uusi toteutus käyttää Pandasin `read_sql_query`-metodia, jonka avulla Pandasin DataFrame:en luetaan tiedot SQL-kyselyllä. SQL-kysely on luotu ohjelman 5.7 muuttujaan `query`. Kysely yhdistää kummastakin taulusta tarvitut datat käyttäen hyväksi SQL:n `INNER JOIN`-operaatiota, joka liittää kaksi tietokantataulua yhdeksi tulokseksi. Rivillä 12 DataFrame:en `df_results` latautuu näin sekä kaikkien pyyhkäisyjen tulosdata että niitä vastaavan pyyhkäisyn metadata. `df_results`:n läpikäymiseen käytetään DataFrame:n metodia `groupby`, joka jakaa DataFrame:n `swipe_id` sarakkeen mukaan osiin. Lopputulokseksi saadaan for-silmukka, jonka jokaisella iteraatiolla muuttujaan `df_result` on sijoitettu yhden pyyhkäisyn kaikki data.

Suorituskyky

Suorituskyky-mittaukseen on sisällytetty sekä datan lataaminen että sen pyyhkäisy kohtainen läpikäynti for-silmukoissa ohjelmien 5.7 ja 5.6 mukaisesti. Taulukkoon 5.3 on kirjattu mittaustulokset. Mittauksissa on käytetty 53005 kosketuspistettä sisältävää aineistoa. Kosketuspisteet ovat jakautuneet yhteensä 355 eri pyyhkäisylle.

Taulukko 5.4. *Kummankin datan luku ja iterointi -toteutuksen LoC eli Lines of Code.*

	Vanha toteutus	Uusi toteutus
LoC	44	12

Ylläpidettävyys

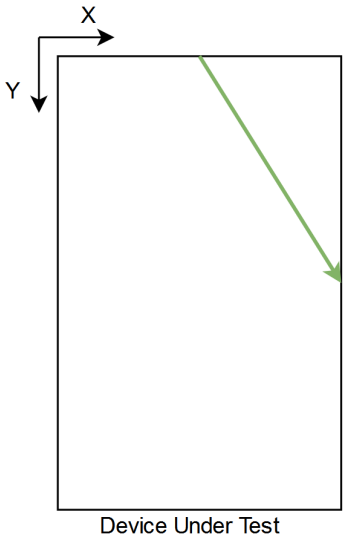
Lähdekoodien rivimäärät on kirjattu taulukkoon 5.4. Vanhan toteutuksen rivimäärää nostaa jälleen reilusti SQLAlchemyn vaatimat tietokannan kuvausluokat ja sen rivimäärä onkin jälleen reilusti suurempi. Suurin osa uuden toteutuksen rivimäärästä kuluu SQL-kyselyn esittämiseen mahdollisimman helposti luettavassa muodossa. Vanhasta toteutuksesta nähdään, kuinka tietokannan kuvausluokissa luotu relaatio testi- ja tulos-taulujen välillä mahdollistaa kummankin taulun yhdistämisen ilman pitkää SQL-kyselyä.

Yhteenveto

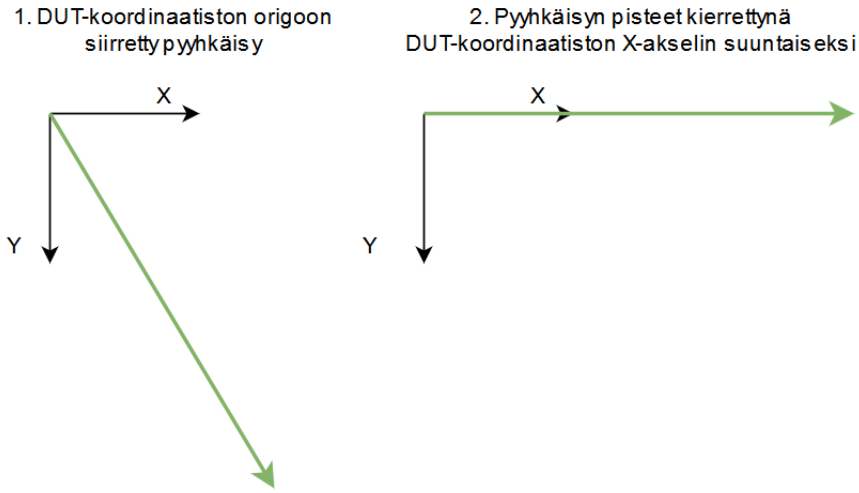
Uusi toteutus on moninkertaisesti nopeampi, eikä pitkiä tietokannan kartoitusluokkia tarvita. Tämän lisäksi uudessa toteutuksessa data on valmiiksi sijoitettu Pandasin tehokkaaseen DataFrame-tietorakenteeseen, josta on hyötyä tulevissa osissa ohjelmaa. SQLAlchemystä luovuttaessa voitettiin suorituskyvyssä, mutta hävittiin mahdollisuus hyödyntää SQLAlchemyn ORM-tekniikkaa, jonka avulla vanhassa toteutuksessa yhdellä funktiokutsulla päästiin käsiksi molempiin tietokantatauluihin. Mikäli tarve hyödyntää ORM:ää tulevaisuudessa ilmenee, voitaisiin SQLAlchemyä käyttää myös Pandasin kanssa.

5.3 Koordinaatistomuunnokset

Datan lataamisen jälkeen analyysi muokkaa raportoidun kosketusdatan muotoon, josta tarvittavat laskut on helppo laskea. Testattavalta laitteelta saatu koordinaattidata on ilmaistu pikseleinä, kun taas robotin koordinaattidata on ilmaistu millimetreinä. Kosketusdatan koordinaatit skaalataan vastaamaan millimetrejä testattavan laitteen mittojen ja resoluution avulla. Tämän jälkeen raportoidun kosketusdatan koordinaatisto siirretään ja kierretään niin, että x-koordinaatti vastaa kuljettua matkaa robotin piirtämällä viivalla ja y-koordinaatti vastaa poikkeamaa robotin piirtämästä viivasta. Koordinaatisto muokataan niin, että robotin piirtämä viiva vastaa kosketusdatakoordinaatiston x-akselia alkaen origosta. Kuva 5.1 havainnollistaa yhtä pyyhkäisyä kosketuspaneelin pinnassa. Kuvan laatikko on testattava kosketuspaneeli ja vihreä nuoli robotin tekemä pyyhkäisy. Robotin koordinaatistossa testattavan laitteen vasen yläkulma on origo, y-akseli kasvaa laitteen reunaa pitkin alaspäin ja x-akseli kasvaa laitteen reunaa pitkin oikealle. Kuvassa 5.2 on havainnollistettu kaksi muutosta, jotka tuloskoordinaateille tehdään. Ensin koordinaatiston origo siirretään vastaamaan robotin koordinaatiston origoa eli testattavan kosketuspaneelin vasenta yläkulmaa. Tämän jälkeen koordinaatisto kierretään x-akselin suuntaiseksi. Kyseistä koordinaatistoa tullaan työssä kutsumaan *pyyhkäisyn koordinaatistoksi*.



Kuva 5.1. *Testattavaa kosketuspaneelia havainnollistava kuva. Vihreä nuoli kuvaa robotin piirtämää pyyhkäisyä.*



Kuva 5.2. *Kuvassa 5.1 esitetyn pyyhkäisyn tuloskoordinaateille tehtävät muutokset: 1. siirto 2. kierto.*

Vanha toteutus

Vanhan toteutuksen koordinaatistomuunnokset ovat jakautuneet kolmeen tiedostoon. Muunnoksia kutsuva lähdekoodi kutsuu `analyzers` nimisen tiedoston metodeita `panel_to_target`, `robot_to_target` ja `target_to_swipe`. Nämä metodit kutsuvat vuorostaan `Transform2D`-luokan metodeja `offset`, `rotate_radians`, `transform` ja `__add__`. Koordinaattien skaalaus tapahtuu funktiossa `panel_to_target`, jossa kosketusdata iteroidaan for-silmukalla ja koordinaatit kerrotaan testattavan laitteen resoluution ja mittojen suhteella. Kosketuskoordinaattien muuntaminen pyyhkäisyn koordinaatistoon alkaa funktiosta `target_to_swipe`, jossa luodaan `Transform2D`-instanssi ja kutsutaan tämän metodia `transform`. `Transform2D`-instanssi luodaan ensin kahdella eri tavalla, jonka jälkeen instanssit yhdistetään niin, et-

```
1  import analyzers
2  for swipe in dbswipes:
3      panel_points = [(p.panel_x, p.panel_y) \
4          for p in swipe.one_finger_swipe_results]
5
6      target_points = analyzers.panel_to_target(
7          panel_points, dutinfo)
8
9      swipe_start, swipe_end = analyzers.robot_to_target(
10         [(swipe.start_x, swipe.start_y),
11          (swipe.end_x, swipe.end_y)],
12         dutinfo)
13
14     swipe_points = analyzers.target_to_swipe(
15         target_points, swipe_start, swipe_end)
```

Ohjelma 5.8. *Vanha toteutus kutsuista koordinaatistomuunnoksiin.*

tä yhdistetyn instanssin muunnosmatriisi kattaa sekä koordinaatiston siirron origoon että sen kierron x-akselin suuntaiseksi. Kosketuskoordinaatiston ja muunnosmatriisin pistetulo suoritetaan manuaalisesti for-silmukassa `transform`-funktiossa. Vanhan toteutuksen lähdekoodi on nähtävissä ohjelmissa 5.8, 5.9 ja 5.10.

Uusi toteutus

Uusi toteutus hyödyntää mahdollisimman paljon vektorisaatiota. Koordinaattien skaalaaminen ja origoon siirtäminen suoritetaan ennen pyyhkäisyjen iterointia, jolloin laskutoimitukset voidaan suorittaa koko aineistolle kerralla. Skaalaaminen tapahtuu funktiossa `scale_results` ja pyyhkäisyn siirto origoon funktiossa `swipe_to_origin`. Kosketuskoordinaatiston kiertäminen pyyhkäisyn koordinaatistoon suoritetaan funktiossa `to_swipe`, jossa muodostetaan rotaatiomatriisi ja käytetään NumPy:n vektorisaatiota hyödyntävää `dot`-metodia pistetulon laskemiseen. Uuden toteutuksen lähdekoodi on nähtävissä ohjelmissa 5.11 ja 5.12.

Pyyhkäisyjä iteroidessa uusi toteutus luopuu Pandasin DataFrame-tietorakenteesta. Ohjelman 5.11 rivillä 5 pyyhkäisyn datan sisältävä DataFrame muutetaan vastaavaksi NumPy-matriisiksi. Prosessissa hävitään mahdollisuus hyödyntää Pandasin nimikoitua dataa, jonka avulla lähdekoodia on helpompi ymmärtää. Esimerkiksi origoon siirrettyjen koordinaattien valitseminen Pandasin avulla näyttää ohjelman 5.11 rivin 15 sijaan tältä:

```
df_result[['from_origin_x', 'from_origin_y']]
```

Pandasin nimikoidun datan käyttäminen ei kuitenkaan ole yhtä tehokasta kuin NumPy:n numeroidun datan käyttö. Datan valikoiminen DataFrame:sta nimettyjen sarakkeiden mukaan luo hieman ylimääräistä työtä prosessorille. Tämä työ moninkertaistuu pyyhkäisyjen lukumäärän funktiona, sillä datan valinta joudutaan tekemään pyyhkäisyjä iteroivan silmukan sisällä. Tästä syystä uusi toteutus muuttaa pyyhkäisydatan NumPy-matriisiksi, jonka

```
1  import transform2d
2  def panel_to_target(points, dutinfo):
3      dimensions = dutinfo.dimensions
4      resolution = dutinfo.digitizer_resolution
5      sx = float(dimensions[0])/float(resolution[0])
6      sy = float(dimensions[1])/float(resolution[1])
7      target_points = [(row[0]*sx, row[1]*sy) for row in points]
8      return target_points
9
10 def robot_to_target(points, dutinfo):
11     # Currently does nothing
12     return points
13
14 def target_to_swipe(points, swipe_start, swipe_end):
15     """Maps swipe (target) coordinates - (x,y) list of tuples
16     to swipe (length, offset) coordinates."""
17     return target_to_swipe_transform(swipe_start, swipe_end)\
18         .transform(points)
19
20 def target_to_swipe_transform(swipe_start, swipe_end):
21     """ Returns the Transform2D object that does the transform
22      from swipe (target) coordinates to swipe (length, offset)
23      coordinates """
24     direction = (swipe_end[0] - swipe_start[0],
25      swipe_end[1] - swipe_start[1])
26     angle = math.atan2(direction[1], direction[0])
27     transform = transform2d.Transform2D.offset(-swipe_start[0],
28     -swipe_start[1]) + transform2d.Transform2D.rotate_radians(-angle)
29     return transform
```

Ohjelma 5.9. *Vanhan toteutuksen analyzer-tiedoston koordinaatistomuunnosfunktiot.*

numeropohjainen datan valikointi on paljon tehokkaampaa.

Suorituskyky

Taulukkoon 5.5 on kirjattu koordinaatistomuutoksiin kuluneet suoritusajat. Suoritusajat on jaettu kolmeen osaan: Datan jäsennys, skaalaus sekä koordinaatiston siirto ja kierto. Datan jäsennys tarkoittaa tarvitun kosketus- ja pyyhkäisydatan valmistelua. Vanhassa toteutuksessa tämä tapahtuu ohjelmassa 5.8 riveillä 3 ja 9 ja uudessa toteutuksessa ohjelman 5.11 riveillä 5, 10, 11 ja 15.

Ylläpidettävyys

Vanhan toteutuksen muunnosmatriisit ja pistetulon toteuttava luokka `Transform2D` on itsetoteutettujen pistetulo- ja summa-laskujen kanssa varsin massiivinen. Uudessa toteutuksessa vastaavasta luokasta on luovuttu ja pistetulo lasketaan NumPy:n avulla. Näistä syistä uusi toteutus on noin puolet tiiviimpi vanhaan verrattuna. Lähdekoodien rivimäärät

```
class Transform2D:
    def __init__(self, matrix):
        '''Initialize the transformation from a 2x3 matrix.
        Argument must be of form [[x1, x2, x3], [y1, y2, y3]].'''
        self.matrix = matrix

    @classmethod
    def offset(cls, x, y):
        '''Translation (i.e. simple move) of the coordinate system.'''
        return cls([[1, 0, x], [0, 1, y]])

    @classmethod
    def rotate_radians(cls, angle):
        '''Rotate the coordinate system. Positive direction is
        counter-clockwise. Angle is in radians.'''
        return cls([[math.cos(angle), -math.sin(angle), 0],
                    [math.sin(angle), math.cos(angle), 0]])

    def transform(self, points):
        '''Transform a group of points, or a single point.
        Argument can be list of tuples [(x,y), (x,y), (x,y)]
        or a single tuple (x,y).'''

        if isinstance(points[0], Iterable):
            return [self.transform(p) for p in points]
        else:
            x, y = points
            x2 = self.matrix[0][0]*x + self.matrix[0][1]*y +
                 self.matrix[0][2]
            y2 = self.matrix[1][0]*x + self.matrix[1][1]*y +
                 self.matrix[1][2]
            return (x2, y2)

    def __add__(self, other):
        # p2 = R1 * p + T
        # p3 = R2 * p2 + T2
        # => p3 = R2 * R1 * p + R2 * T + T2

        a1, b1, c1, d1 = self.matrix[0][0], self.matrix[0][1],
                         self.matrix[1][0], self.matrix[1][1]
        a2, b2, c2, d2 = other.matrix[0][0], other.matrix[0][1],
                         other.matrix[1][0], other.matrix[1][1]
        tx, ty = other.transform((self.matrix[0][2], self.matrix[1][2]))
        new = Transform2D([
            [a1 * a2 + b2 * c1, a2 * b1 + b2 * d1, tx],
            [c2 * a1 + d2 * c1, c2 * b1 + d2 * d1, ty]])
        return new
```

Ohjelma 5.10. *Vanhan toteutuksen transform2d-luokka.*

```
df_results = analyzers.scale_results(df_results, dutinfo)
df_results = analyzers.swipe_to_origin(df_results)

for swipe_id, df_result in df_results.groupby('swipe_id'):
    np_result = df_result.values

    # Matriisin sarakket ovat seuraavassa järjestyksessä:
    # swipe_id, start_x, start_y, end_x, end_y
    # jolloin pyyhkäisyn alku ja loppu luetaan seuraavasti:
    swipe_start = (np_result[0,1], np_result[0,2])
    swipe_end = (np_result[0,3], np_result[0,4])

    # Valitaan matriisista 2 viimeistä saraketta, jotka ovat
    # swipe_to_origin-funktiossa lisätyt origoon siirretyt koordinaatit
    np_from_origin = np_result[:,-2:]

    np_swipe = analyzers.to_swipe(np_from_origin, swipe_start, swipe_end)
```

Ohjelma 5.11. *Uusi toteutus kutsuista koordinaatistomuunnoksiin.*

```
def scale_results(df_results, dutinfo):
    x_scale = dutinfo.dimensions[0]/dutinfo.digitizer_resolution[0]
    y_scale = dutinfo.dimensions[1]/dutinfo.digitizer_resolution[1]
    df_results['panel_x'] = df_results['panel_x'].values * x_scale
    df_results['panel_y'] = df_results['panel_y'].values * y_scale
    return df_results

def swipe_to_origin(df_results):
    df_results['from_origin_x'] = \
        df_results['panel_x'] - df_results['start_x']
    df_results['from_origin_y'] = \
        df_results['panel_y'] - df_results['start_y']
    return df_results

def to_swipe(np_from_origin, swipe_start, swipe_end):
    direction = (swipe_end[0] - swipe_start[0],
                 swipe_end[1] - swipe_start[1])
    angle = math.atan2(direction[1], direction[0])

    rotation_matrix = np.array(
        [[math.cos(angle), -math.sin(angle)],
        [math.sin(angle), math.cos(angle)]])

    np_swipe = np.dot(np_from_origin, rotation_matrix)
    return np_swipe
```

Ohjelma 5.12. *Uuden toteutuksen koordinaatistomuunnosfunktiot.*

Taulukko 5.5. *Uuden ja vanhan toteutuksen koordinaatistomuunnoksiin kulutetut suoritusajat.*

	Vanha toteutus (s)	Uusi toteutus (s)
Datan jasennys	0,082	0,057
Skaalaus	0,011	0,002
Koordinaatiston siirto + kierto	0,276	0,012
Yhteensa	0,369	0,071

Taulukko 5.6. *LoC eli Lines of Code -arvot koordinaatistomuunnoksille.*

	Vanha toteutus	Uusi toteutus
LoC	60	29

on kirjattu taulukkoon 5.6.

`Transform2D`-luokka aiheuttaa hajautuvan muutoksen ongelmaa siinä määrin, että luokan rajapinnan muuttuessa joudutaan muutos tekemään myös rajapintaa kutsuviin funktiohin. Vanhan toteutuksen ohjelman 5.9 rivit 27 ja 28 ovat varsin vaikealukuiset. Rivit voidaan nähdä pitkinä parametrilistoina `Transform2D`:n `__add__` metodille. Vanhaan toteutukseen on jäänyt kutsu funktioon `robot_to_target`, jolla ei ole toiminnallisuutta. Funktiolle on luultavasti suunniteltu jotakin toiminnallisuutta, jota ei lopulta ole tarvittu tai ehditty kehittämään. Tällaisen funktion kutsuminen luo turhaa hämmennystä ylläpitotöitä tehdessä.

Uuden toteutuksen ylläpidettävyyttä heikentää numeropohjainen datan valikointi NumPy-matriisista. Lukijan on tiedettävä mihin esimerkiksi `np_result[0,1]` tai `np_result[:,-2:]` viittaavat. Ongelmaa on korjattu kommenttien avulla.

Yhteenveto

Pandasin ja NumPy:n avulla toteutettu uusi versio on jälleen moninkertaisesti nopeampi ja paljon tiiviimmin esitetty. Suorituskykyparannus saavutetaan vektorisaatiota hyödyntämällä ja tiivimmän esityksen mahdollistaa esimerkiksi NumPy:n valmiiksi toteutetun pistetulofunktion käyttö. NumPy:n pistetulonfunktion käytön ansiosta myös ylläpidettävyyttä heikentävästä transform2d-luokasta on uudessa toteutuksessa päästy eroon.

5.4 Lineaarisuus-analyysi

Pyyhkäisyjen lineaarisuutta analysoidaan lineaarisen regression avulla. Pyyhkäisyn koordinaatistossa sijaitseville koordinaateille sovitetaan kosketuspisteitä estimoiva suora. *Lineaarivirhe* vastaa kosketuspisteiden lyhintä etäisyyttä sovitesuorasta. Pisteen lyhin etäisyys sovitesuorasta lasketaan kaavan 5.3 avulla [1]. Kaikille pisteille lasketaan lineaarivirhe ja näiden virheiden avulla määritellään:

- Lineaarinen maksimivirhe, eli kaikkien kosketuspisteiden suurin poikkeama lineaa-

risovitteesta.

- Lineaarinen keskivirhe, eli pyyhkäisyn poikkeamien keskiarvo.
- Neliöidyn lineaarisen keksivirheen neliöjuuri, eli niin sanottu *neliöllinen keskivirhe*.

Vanha toteutus

Lineaarisovitteen luomiseksi käytetään hyväksi NumPy:n metodia `polyfit`. Funktioon syötetään x- ja y-koordinaatit NumPy-matriiseina ja palautusarvoksi saadaan tekijät polynomille, joka on sovitettu syötettyyn dataan. Kolmas metodille syötettävä arvo on halutun sovitefunktion aste. Metodia käytetään lineaarisovitteen löytämiseksi, joten aste-luvuksi syötetään luku yksi. Paluuarvoiksi saadaan sovitesuoran tekijät a ja c kun suoran funktio on esitetty ratkaistussa muodossa:

$$y = a * x + c, \tag{5.1}$$

jossa y on funktion arvo kohdassa x, a on muuttujan x tekijä ja c funktion vakiotermi [1]. Pisteen etäisyyden eli lineaarivirheen laskemiseksi suoran funktio pitää muuttaa sen normaalimuotoon:

$$a * x + b * y + c = 0, \tag{5.2}$$

jossa esiintyy ratkaistun muodon merkkien lisäksi merkki b eli muuttujan y tekijä [1]. Ratkaistu muoto voidaan muuttaa normaalimuotoon kun y:n tekijä b on yhtä kuin -1. Sijoitetaan y:n tekijä suoran ratkaistuun muotoon ja ratkaistaan nollakohta:

$$\begin{aligned} -1 * y &= a * x + c \\ 1 * y &= -a * x - c \\ a * x + 1 * y + c &= 0 \end{aligned}$$

Ratkaistu muoto vastaa siis suoran normaalimuotoa kun b = -1. Pisteen etäisyys suorasta voidaan nyt laskea seuraavalla kaavalla:

$$d = \frac{|a * x_0 + b * y_0 + c|}{\sqrt{a^2 + b^2}}, \tag{5.3}$$

jossa d on jonkin pisteen (x_0, y_0) kohtisuora etäisyys suorasta [1]. Etäisyydet kaikille kosketuspisteille voidaan laskea sillä arvot a, b ja c ovat selvillä. Lasketuista etäisyyksistä etsitään suurin arvo NumPy:n metodilla `max` ja virheiden keskiarvo lasketaan NumPy:n metodilla `mean`. Neliöllisen keskiarvon laskemiseen tarvitaan näiden lisäksi neliöjuurimetodia `sqrt` ja asteenkorotusmetodia `power`. Metodille `power` annetaan toiseksi parametriksi luku

2, sillä korotamme lineaarivirheet toiseen potenssiin.

Vanha toteutus on nähtävissä ohjelmassa 5.13. Ohjelman riveillä 13-17 koordinaattidata erotellaan erikseen x- ja y-koordinaatit sisältäviksi Pythonlistoiksi. Lineaarisovitteen tekijät ratkaistaan rivillä 21 ja etäisyydet suoraan lasketaan riveillä 32-33. Samoilla riveillä joudutaan myös muuttamaan Pythonlistat x ja y NumPy:matriiseiksi. Lopulta halutut lukuarvot lasketaan riveillä 35-37. Vanhassa toteutuksessa on tuntemattomasta syystä laskettu rivillä 22 myös y-koordinaattien arvot sovitesuoralla. Tälle laskulle ei kuitenkaan analyysin missään vaiheessa ole käyttöä.

Uusi toteutus

Uusi toteutus lineaari-analyysille ei merkittäviltä osin eroa vanhasta. Kosketusdata on jo funktioon saavuttaessa NumPy-matriisissa, joten muunnosta Pythonlistasta ei tarvita ja lasku pisteiden etäisyydestä suoraan sievenee muotoon:

```
np.absolute(a*x + b*y + c) / math.sqrt(a**2 + b**2)
```

X- ja y-koordinaattien erotteleminen kosketusdatasta suoritetaan ilman for-silmukkarakennetta seuraavalla tavalla:

```
x = df_swipe['distance'].values

y = df_swipe['offset'].values
```

jossa `df_swipe` on pyyhkäisynkoordinaatistossa sijaitseva kosketusdata. 'distance' -avaimen taakse on tallennettu x-koordinaatti ja 'offset'-avaimen taakse y-koordinaatti.

Uudesta toteutuksesta on tämän lisäksi jätetty pois vanhan toteutuksen 5.13 rivillä 22 lasketut y:n arvot sovitesuoralla. Uuteen toteutukseen on myös lisätty kommentti rivin 27 b:n arvon asettamisen selittämiseksi. Kommentti avaa lyhyesti syyn b:n arvolle -1.

Lineaarivirhe olisi voitu laskea myös scikit-learn-kirjaston lineaariregressio-luokan avulla, jolloin manuaalisia pisteenetäisyys-laskuja ei olisi tarvittu. Lisäksi lineaarivirheen keskiarvo ja neliöity keskivirhe olisi voitu laskea scikit-learn-kirjaston metrics-luokan avulla. Ohjelma olisi näin vieläkin tiiviimpi ja matemaattisten ohjelmarivien sijaan käytettäisiin selkokielisiä funkntiokutsuja scikit-learn-kirjastoon. Scikit-learn:ia ei kuitenkaan hyödynnetty uudessa toteutuksessa sen huonomman suorituskyvyn takia. On myös kyseenalaista ottaa käyttöön uusi kirjasto vain yhden toiminnallisuuden takia. Suorituskyky on scikit-learnissa huonompi kahdesta syystä. Ensinnäkin scikit-learn suorittaa aina sille syötetylle datalle omia validointeja virhetilanteilta välttyäkseen. Tällaiset validoinnit ovat kuitenkin raskaita suorittaa esimerkkitapauksemme pyyhkäisyjä iteroivan silmukkarakenteen sisällä. Toisekseen joudutaan silmukkarakenteen sisällä luomaan jokaiselle pyyhkäisylle oma lineaariregressio-olio. Olion luonti ei ole täysin ilmaista, joten suorituskyvyssä kärsitään erikseen jokaisen pyyhkäisyn kohdalla.

```
1  def analyze_swipe_linearity(points):
2      """
3      Calculates linear fit for a single swipe line
4      Determine max, avg and rms errors from linear fit
5      """
6      if len(points) == 0:
7          results = {'linear_error': [],
8                     'lin_error_max': float('nan'),
9                     'lin_error_avg': float('nan'),
10                    'lin_error_rms': float('nan'),
11                    'linear_fit': []}
12         return results
13     x = []
14     y = []
15     for point in points:
16         x.append(point[0])
17         y.append(point[1])
18
19     # Linearfit to fit the data
20     # Linearcoef has slope (1) and intercept (2)
21     linearcoef = np.polyfit(x, y, 1)
22     linearfit = np.polyval(linearcoef, x)
23
24     # Starndard form of linear equation
25     # ax + by + c = 0
26     a = linearcoef[0]
27     b = -1
28     c = linearcoef[1]
29
30     # Max deviation calc: orthogonal distance
31     # from fit line to data set
32     lin_error = np.absolute(a*np.array(x) + b*np.array(y) + c)
33                 / math.sqrt(a**2 + b**2)
34
35     lin_error_max = max(lin_error)
36     lin_error_avg = np.mean(lin_error)
37     lin_error_rms = np.sqrt(np.mean(np.power(lin_error, 2)))
38     results = {'linear_error': lin_error.tolist(),
39                'lin_error_max': lin_error_max,
40                'lin_error_avg': lin_error_avg,
41                'lin_error_rms': lin_error_rms,
42                'linear_fit': linearfit}
43     return results
```

Ohjelma 5.13. *Vanha toteutus lineaarisuus-analyysistä*

Taulukko 5.7. *Uuden ja vanhan toteutuksen lineaari-analyysiin kulutetut suoritusajat.*

	Vanha toteutus (s)	Uusi toteutus (s)
Lineaarisuus – analyysi	0,142	0,101

Taulukko 5.8. *LoC eli Lines of Code -arvot lineaari-analyyseille.*

	Vanha toteutus	Uusi toteutus
LoC	26	23

Suorituskyky

Uuden ja vanhan toteutuksen suoritukseen kuluneet ajat ovat listattu taulukkoon 5.7. Mittauksesta on jätetty pois vanhan toteutuksen y-koordinaatti-arvojen lasku sovitesuoralla vertailukelpoisuuden säilyttämiseksi. Uusi toteutus on hieman nopeampi, sillä koordinaattien erotteluun ei tarvita Pythonin for-silmukkarakennetta, eikä koordinaattidataa tarvitse erikseen muuttaa NumPy-matriisiksi.

Ylläpidettävyys

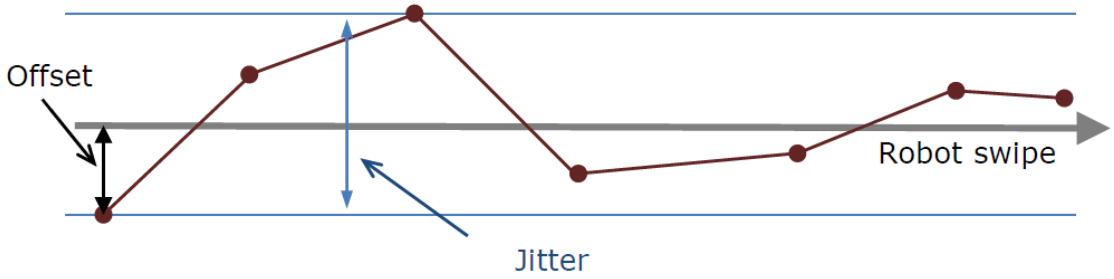
Toteutuksiin tarvitut lähdekoodin rivimäärät on kirjattu taulukkoon 5.8. Tarpeeton y-koordinaattien lasku sovitesuoralla on jälleen jätetty pois laskuista vertailukelpoisuuden säilyttämiseksi. Uusi toteutus on kolme riviä tiiviimpi, sillä koordinaattidatan jako erikseen x- ja y-koordinaatteihin tapahtuu NumPy:n avulla ilman for-rakennetta. Muuten ratkaisut ovat pituudeltaan toisiaan vastaavat.

Vanhan toteutuksen y-koordinaattien lasku sovitesuoralla on paitsi turha, on sen tulokset myös sijoitetty mysteeriseen nimeen `linearfit`. Nimi linearfit esiintyy funktion kommenteissa eri merkityksellä kuin lähdekoodissa. Ohjelman 5.13 kommentissa rivillä 4 mainitaan tulosten laskeminen *linear fit*:n avulla. Tuloksia ei kuitenkaan lasketa muuttujan `linearfit` avulla, vaan lasku tapahtuu sovitesuoran tekijöiden avulla.

Analyysin yhdestä silmukkarakenteesta on uudessa toteutuksessa päästy eroon, sillä koordinaattidatan erottelu onnistuu NumPy:n avulla.

Yhteenveto

Vanhassa toteutuksessa on jo hyödynnetty etäisyyksien laskentaan NumPy-matriiseja, joten suurta eroa uudella ja vanhalla toteutuksella ei ole. Vanhassa toteutuksessa on kuitenkin ensin jouduttu muuttamaan kosketusdata Python-listasta NumPy-matriisiksi. Uudessa toteutuksessa tätä vaihetta ei tarvita, sillä kosketusdata on NumPy-matriisissa jo valmiiksi. Vanhan toteutuksen turhaan tehdyt laskut heikentävät sen luettavuutta ja ymmärrettävyyttä kaikkein eniten.



Kuva 5.3. *Jitter- ja offset-termejä havainnollistava kuva.*

5.5 Offset ja jitter

Offset tarkoittaa kosketusdatapisteiden etäisyyttä robotin tekemästä pyyhkäisystä. Kosketusdatan koordinaatit on jo muutettu pyyhkäisyn koordinaatistoon eli muotoon, jossa x-koordinaatti vastaa etäisyyttä pyyhkäisyllä ja y-koordinaatti etäisyyttä pyykäisystä. Offset-arvo kaikille yksittäisille pisteille on siis jo laskettu. Analyysi etsii vielä *max_offset*-arvon eli suurimman yksittäisen offset-arvon pyyhkäisykohtaisesti.

Jitter tarkoittaa etäisyyttä suurimman ja pienimmän offset-arvon välillä. Jitter kuvaa siis väliä, johon offsetit mahtuvat. Kuva 5.3 havainnollistaa jitter- ja offset-arvoja. Kuvan harmaa viiva kuvaa robotin tekemää pyyhkäisyä, punaiset pisteet laitteen raportoimia kosketuspisteitä, musta väli offset-arvoa ja sininen väli jitter-arvoa. Jitter lasketaan 10 millimetrin pituisille osuuksille pyyhkäisystä. Jitter-arvoja lasketaan liukuvan ikkunan avulla ja ikkunan näytekoon määrittää pyyhkäisyllä kuljettu matka eli x-koordinaatti pyyhkäisyn koordinaatistossa.

Vanha toteutus

Suurimman offset-arvon löytämiseksi vanha toteutus iteroi pyyhkäisyn koordinaatistoon muunnetut kosketuspisteet ja muodostaa listan y-koordinaattien itseisarvoista. Y-koordinaattilistalle eli offset-listalle kutsutaan Pythonin sisäänrakennettua `max`-funktiota, joka etsii listasta suurimman alkion.

```
offset = max([abs(p[1]) for p in swipe_points])
```

Jitter-arvojen laskeminen liukuvalla ikkunalla on toteutettu for-silmukalla ja *deque*-rakenteella, jota käytetään dataikkunan käsittelyyn. Deque on Pythonin sisäinen tietorakenne, jonka alku- ja loppu-päistä alkioiden poistaminen tai lisääminen on tehokasta [33]. Vanha toteutus laskee jitter-arvojen lisäksi pisteet, jotka ovat edelliseen pisteeseen verrattuna kulkeneet pyyhkäisyllä takaisin päin tai pysyneet paikallaan. Toistuneita tai väärään suuntaan kulkeneita pisteistä ei kuitenkaan syystä tai toisesta käytetä analyysin missään osassa. Toteutus on nähtävissä ohjelmassa 5.14. Liukuvan ikkunan koko määräytyy riveillä 37-38. Muuttuja `window` on deque-tietorakenne, jonka loppupäähän lisätään kullakin iteraatiolla tarkasteltava piste rivillä 36. Pisteen ikkunaan lisäämisen jälkeen rivillä 37 tarkastellaan ikkunan alkupään kosketuspisteitä ja verrataan pisteiden x-koordinaattia, eli

etäisyyttä pyyhkäisyllä, tarkasteltavan pisteen x-koordinaattiin. Mikäli ikkunan alkupään pisteen ja tarkasteltavan pisteen x-koordinaattien ero on suurempi kuin ikkunan koko `window_length`, ei alkupään piste mahdu enää ikkunaan ja se poistetaan deque:n metodilla `popleft` rivillä 38. Pisteiden poistamista jatketaan kunnes ikkuna on tarpeeksi lyhyt. Tämän jälkeen ikkunasta lasketaan riveillä 41-43 suurimmat ja pienimmät offset-arvot, joiden erotus rivillä 44 vastaa tarkasteltavan ikkunan jitter-arvoa.

Uusi toteutus

Suurin offset-itseisarvo haetaan uudessa toteutuksessa NumPy:n funktioilla `max` ja `absolute`.

```
np.max(np.absolute(np_swipe[:,1]))
```

Jitter-arvot on laskettu hyödyntäen Pandasin sisäänrakennettua liukuvan ikkunan toteutusta nimeltä `rolling`. Pandasin liukuvan ikkunan kokoa ei voida suoraan määritellä x-koordinaattien millimetrien perusteella, niin kuin vanhassa toteutuksessa on tehty. Ikkunan koko Pandasissa voidaan määrittää kahdella tavalla. Koko voi olla joko kiinteä lukuarvo, jolloin jokainen ikkuna sisältää tietyn määrän alkioita, tai aikamääreeseen perustuva, jolloin ikkunan kooksi muodostuu tiettyyn aikamääreeseen mahtuvat alkiot. Jälkimmäinen tapa vaatii, että data on järjestetty jonkin aikamääreen mukaan. Kosketusdata voidaan järjestää Series-tietorakenteeseen niin, että x-koordinaattien sijaintidata vastaa näennäistä aikamäärettä. Tällä tavalla voimme käyttää mukautuvaa ikkunan kokoa pandasin `rolling` toiminnallisuudella.

Uusi toteutus on nähtävissä ohjelmassa 5.15. Kosketuksen sijainti pyyhkäisyllä muutetaan sekunneiksi Pandasin `TimedeltaIndex`-muotoon rivillä 6. TimedeltaIndex on aikamääreeseen perustuva indeksi. Kosketusdatan offset-arvoista ja luodusta TimedeltaIndex:stä muodostetaan Series-tietorakenne rivillä 7. Tietorakenteen metodia `rolling` kutsutaan kahdesti riveillä 8-11. Ensimmäisellä kerralla ikkunista lasketaan offsetin maksimiarvot ja toisella kertaa minimi-arvot. Parametreiksi kutsuille annetaan haluttu ikkunan pituus ja argumentti `min_periods`, joka määrittelee ikkunan alkioiden vähimmäislukumäärän. Ikkunan pituus annetaan merkkijonona, joka koostuu ikkunan pituutta kuvaavasta lukuarvosta ja sen yksiköstä. Millimetrit ovat tässä tapauksessa muutettu sekunneiksi, joten muuttujaan `window_length` lisätään yksiköksi sekuntia kuvaava `'s'`-merkki. Jitter-arvot lasketaan rivillä 12 vähentämällä minimi-arvot maksimeista. Suurin jitter-arvo etsitään NumPy:n metodilla `nanmax`, joka etsii joukon suurimman alkion piittaamatta puuttuvista arvoista.

Toistuvia tai väärään suuntaan kulkeneita pisteitä ei uudessa toteutuksessa etsitä, sillä kyseistä dataa ei analyysissä käytetä. Mikäli tulevaisuudessa tällaisia pisteitä haluttaisiin etsiä, voitaisiin se tehdä esimerkiksi NumPy:n *ediff1d*-funktiolla. Ediff1d palauttaa syötetyn matriisin peräkkäisten arvojen erotukset. Erotuksen ollessa nolla, piste on sama kuin edellinen ja erotuksen ollessa negatiivinen, piste on kulkenut väärään suuntaan.

```
def analyze_swipe_jitter(points, window_length=10.0):
    """ Analyzes jitter for swipe points that have been
    transformed to coordinate system (swipe-direction,
    perpendicular-to-swipe) with origo at swipe begin and
    positive x-axle to swipe direction. Jitter is calculated
    with sliding window of length window_length
    Returns dictionary {'jitters' (list), 'max_jitter',
    'backwards_points' (count), 'repeated_points' (count)}

    First point jitter is always None, and in case of
    backwards movement or repeated measurements jitter is
    float('nan') to signal failed point """
    jitters = []
    previous_x = None
    previous_y = None
    backwards_points = 0
    repeated_points = 0
    window = deque()
    max_jitter = None
    for point in points:
        if previous_x is None:
            previous_x = point[0]
            previous_y = point[1]
            jitters.append(None)
            window.append(point)
        elif point[0] < previous_x:
            # Backwards movement
            backwards_points += 1
            jitters.append(float('nan'))
        elif point[0] == previous_x and point[1] == previous_y:
            # Repeated measurement
            repeated_points += 1
            jitters.append(float('nan'))
        else:
            # Moving forward or at least not backwards...
            window.append(point)
            while window[0][0] <= (point[0] - window_length):
                window.popleft() # Can never remove the point itself

            # Find out the minimum and maximum offsets in window
            offsets = [p[1] for p in window]
            window_min = min(offsets)
            window_max = max(offsets)
            jitter = window_max - window_min  # Peak-to-peak
            jitters.append(jitter)
            if max_jitter is None or jitter > max_jitter:
                max_jitter = jitter
    results = {'jitters': jitters,
               'max_jitter': max_jitter,
               'backwards_points': backwards_points,
               'repeated_points': repeated_points}
    return results
```

Ohjelma 5.14. *Vanha toteutus Jitter-analyysistä*

```
1  jitter = []
2  max_jitter = None
3
4  # Jitter can only be calculated if there is 2 or more data points
5  if len(np_swipe) > 1:
6      index = pd.TimedeltaIndex(np_swipe[:,0], 's').sort_values()
7      s_offset = pd.Series(np_swipe[:,1], index=index)
8      offset_max = s_offset.rolling(
9          str(window_length)+'s', min_periods=2).max().values
10     offset_min = s_offset.rolling(
11         str(window_length)+'s', min_periods=2).min().values
12     jitter = offset_max - offset_min
13     max_jitter = np.nanmax(jitter)
14
15 results = {'jitters': jitter,
16           'max_jitter': max_jitter}
17
18 return results
```

Ohjelma 5.15. *Uusi toteutus jitter-analyysistä*

Taulukko 5.9. *Uuden ja vanhan toteutuksen jitter- ja offset-analyyseihin kulutetut suoritusajat.*

	Vanha toteutus (s)	Uusi toteutus (s)
Offset	0,015	0,009
Jitter	0,221	0,586
Yhteensa	0,236	0,595

Suorituskyky

Taulukkoon 5.9 on kirjattu suurimman offset-arvon etsimiseen ja jitter-analyysiin kuluneet suoritusajat. Pandasin `rolling`-metodia joudutaan uuden toteutuksen jitter-analyysissä kutsumaan kahdesti, sillä valmista funktiota ikkunan suurimpien ja pienimpien arvojen erottamiselle ei Pandasiin ole toteutettu. Tämän lisäksi suoritusaikaa kuluu Series-tietorakenteen luontiin. Näistä syistä uuden toteutuksen jitter-analyysiin kulunut suoritusaika on vanhaa toteutusta heikompi.

Ylläpidettävyys

Kumpaankin toteutukseen tarvitut lähdekooodin rivimäärät on kirjattu taulukkoon 5.10. Vertailukelpoisuuden säilyttämiseksi vanhan toteutuksen rivejä, jotka liittyvät toistuvien tai väärään suuntaan kulkeneiden pisteiden etsimiseen, ei oteta laskussa huomioon.

Vanhan totetutuksen jitter-analyysin liukuva ikkuna on toteutettu manuaalisesti käyttäen kahta for-silmukkaa ja yhtä while-silmukkaa. Toteutus on sisäkkäisten silmukoiden ja tar-

Taulukko 5.10. *LoC eli Lines of Code -arvot jitter-analyyseille.*

	Vanha toteutus	Uusi toteutus
LoC	24	14

peettomien laskutoimitusten syystä vaikeasti lähestyttävä. Uusi toteutus käyttää Pandasin toteuttamaa liukuvaa ikkunaa, jonka ansiosta ohjelmarivien määrä lähes puolittuu. Suuri osa uuden toteutuksen rivimäärästä kuluu Series-tietorakenteen luomiseen, jolle Pandasin `rolling`-metodia voidaan kutusa. Silmukkarakenteita ei uudessa toteutuksessa tarvita lainkaan.

Yhteenveto

Uusi toteutus on reilusti tiiviimpi ja helpommin luettava. Toisaalta suorituskyvyssä on tällä kertaa jouduttu joustamaan vanhaan toteutukseen verrattuna. Millimetrien perusteella määräytyvä liukuvan ikkunan koon määrittely on jouduttu tekemään muuntamalla millimetrit sekunneiksi. Tämä heikentää muuten selkeän ja tiiviin uuden toteutuksen ymmärrettävyyttä.

Uusi toteutus voidaan tarpeen vaatiessa tehdä myös vanhaa toteutusta vastaavalla tavalla. Tällöin kaikista silmukka- tai deque-rakenteista ei päästä eroon mutta suorituskyvyssä ei hävitä. Toisaalta uuden toteutuksen liukuvan ikkunan analyysiä on nykyisellään helppo laajentaa. `Rolling`-metodin avulla on helppo analysoida esimerkiksi pisteiden keskihajontaa ja muita Pandasin valmiiksi toteuttamia tilastomatematiikan sovelluksia.

6 YHTEENVETO

Työssä toteutettiin uudelleen TPPT-pyyhkäisytestin data-analyysi. Uudessa toteutuksessa käytettiin hyväksi tärkeimmiksi todettuja data-analyysityökaluja NumPy:ä ja Pandasia. Vanhassa toteutuksessa käytetty SQLALchemy on jätetty pois uudesta toteutuksesta. Lopputulos on toimiva, tiivis ja tehokas osakokonaisuus TPPT-analyysistä, joka vastaa toiminnallisuudeltaan vahaa toteutusta.

Taulukkoon 6.1 on kirjattu yhteenveto toteutusten suoritusajoista. Uusi toteutus on lähes kertaluokkaa nopeampi Pandasin ja laajemman NumPy:n hyödyntämisen ansiosta. Huomattavaa suoritusajoissa on reilun parannuksen lisäksi se, että jitter ja offset laskuihin kulutettu suoritusaika kattaa lähes puolet uuden toteutuksen kokonaissuoritusajasta. Mikäli kyseinen osio jätettäisiin huomiotta tai toteutettaisiin ilman Pandasin valmista `rolling`-metodia, olisi suorituskykyparannus vieläkin huomattavampi. Toisaalta tällöin hävittäisiin ylläpidettävyydessä rivimäärän ja oman koodikannan kasvaessa.

Taulukkoon 6.2 on kirjattu yhteenveto toteutuksiin tarvituista rivimääristä. Huomataan, että uusi toteutus on osakokonaisuuden kaikissa osissa tiiviimpi. Tämä johtuu suurelta osin siitä, että valtaosa varsinaisesta toteutuksesta on jätetty Pandasin ja NumPy:n vastuulle. Uuden toteutuksen koodikannassa kutsutaan valmiiksi toteutettuja rakenteita, kuten Pandasin SQL-metodia `read_sql` tai NumPy:n pistetulo-metodia `dot`. Hyvä puoli vastuunsiirrolla kolmannen osapuolen kirjastoille on se, että oman sovelluksen koodikanta ja ylläpitoon tarvittava työ pienenee. Vastaavasti huono puoli vastuunsiirrolle on täydellisten räätälöintimahdollisuuksien häviäminen. Esimerkiksi tapaustutkimuksen jitter-lasku Pandasin `rolling`-metodilla häviää suorituskyvyssä ja millimetrit on jouduttu hämäävästi muuttamaan sekunneiksi, jotta metodi on saatu taipumaan esimerkkisovelluksen vaatimuksia vastaavaksi.

Tapaustutkimus osoittaa kuinka Pythonilla on nykyaikaisten työkalujen avulla mahdollista toteuttaa monipuolista ja tehokasta data-analyysiä. Erityisen tärkeää on kuitenkin oikeiden työkalujen käyttö. Ennen kaikkea NumPy osaaminen on tärkeää, sillä se mahdollistaa tehokkaiden dataintensiivisten sovellusten kehityksen Pythonilla. Pelkillä natiiveilla Python-rakenteilla ei voida kehittää kilpailukykyistä data-analyysiä, sillä se on moninkertaisesti hitaampaa matalan tason ohjelmointikieliin verrattuna. Kaikki korkeamman tason työkalut kuten Pandas perustuvat NumPy:yn. Pandas tarjoaa suuren määrän tehokasta ja helppolukuista toiminnallisuutta NumPy:n ohelle. Tapaustutkimuksessa suurin hyöty Pandasista oli SQL-rajapintana ja datan jäsentämisessa kulloinkin tarvittavaan muotoon. Merkittävä etu Pandasin käytössä on myös se, että Pandas on laajalti käytössä

Taulukko 6.1. *Yhteenveto uuden ja vanhan toteutuksen cProfilella mitatuista suoritusajoista.*

	Vanha toteutus (s)	Uusi toteutus (s)
Datan kirjoitus	4,252	0,219
Datan luku ja iterointi	2,281	0,371
Koordinaatistomuunnokset	0,369	0,071
Lineaarisuus – analyysi	0,142	0,101
Jitter ja offset	0,236	0,595
Yhteensa	7,280	1,357

Taulukko 6.2. *Yhteenveto tarvituista rivimääristä.*

	Vanha toteutus	Uusi toteutus
Datan kirjoitus	41	13
Datan luku ja iterointi	44	12
Koordinaatistomuunnokset	60	29
Lineaarisuus – analyysi	26	23
Jitter ja offset	24	14
Yhteensa	195	91

oleva työkalu. Internetissä on paljon tukea Pandasiin liittyvissä kysymyksissä. Tukea löytyy niin Pandasin dokumentaatiosta kuin kolmannen osapuolen sivustoilta kuten Stack Overflow:sta. Toisekseen, tunnetuilla työkaluilla kirjoitettuun sovellukseen on helpompi löytää jatkokehittäjiä, kuin esimerkkisovelluksen kaltaiselle laajalti itsetoteutetulle ja heikosti dokumentoidulle sovellukselle.

Tapaustutkimusta voidaan tulevaisuudessa jatkaa data-analyysin tulosten visualisoinnin tutkimiseen ja uudelleen toteutukseen. Kuten kappaleen 3 kuvasta 2.1 käy ilmi, datan visualisointiin on olemassa useita työkaluja, joita tutkia. Myös Pandas tarjoaa toiminnallisuutta datan visualisointiin. Jatkotutkimus olisi myös TPPT:n kehityksen kannalta merkittävää, sillä samat suorituskyky ja ylläpidettävyys ongelmat ovat esillä myös datan visualisointi -osuudessa sovellusta.

LÄHTEET

[1] R. A. Adams. *Calculus a complete course. 6th edition*. English. Addison Wesley Longman, 2007.

[2] K. K. Aggarwal, Y. Singh ja J. K. Chhabra. An integrated measure of software maintainability. *Annual Reliability and Maintainability Symposium. 2002 Proceedings (Cat. No.02CH37318)*. Tammikuu 2002, 235–241. DOI: 10.1109/RAMS.2002.981648.

[3] A. A. Alain April. *Software Maintenance Management: Evaluation and Continuous Improvement*. 2008.

[4] Ash, Alderete, Yao, Oman ja Lowtber. Using software maintainability models to track code health. *Proceedings 1994 International Conference on Software Maintenance*. Syyskuu 1994, 154–160. DOI: 10.1109/ICSM.1994.336779.

[5] K. Beck, J. Brant, M. Fowler, W. Opdyke ja D. Roberts. *Refactoring: Improving the Design of Existing Code*. English. 1. painos. Addison-Wesley Professional, 1999.

[6] K. H. Bennett ja V. T. Rajlich. Software Maintenance and Evolution: A Roadmap. *Proceedings of the Conference on The Future of Software Engineering*. ICSE '00. Limerick, Ireland: ACM, 2000, 73–87. ISBN: 1-58113-253-0. DOI: 10.1145/336512.336534. URL: http://doi.acm.org/10.1145/336512.336534.

[7] A. Butterfield ja J. Szymanski. *root-mean-square*. 2018.

[8] D. Coleman, D. Ash, B. Lowther ja P. Oman. Using metrics to evaluate software system maintainability. *Computer* 27.8 (elokuu 1994), 44–49. DOI: 10.1109/2.303623.

[9] R. Copeland ja J. Myers. *Essential SQLAlchemy, 2nd Edition*. English. O'Reilly Media, Inc, 2015. ISBN: 1491916540;9781491916544;

[10] A. Fandango. *Python data analysis: data manipulation and complex data analysis with Python*. English. Second. Birmingham: Packt Publishing, 2017. ISBN: 9781787127920;1787127923;

[11] N. E. Fenton ja M. Neil. Software metrics: successes, failures and new directions. English. *Journal of Systems and Software* 47.2 (1999), 149–157.

[12] M. Fowler. *Refactoring: Improving the Design of Existing Code*. English. 1. painos. Addison-Wesley Professional, 2018.

[13] M. Gorelick ja I. Ozsvald. *High Performance Python, 2nd Edition*. English. O'Reilly Media, Inc, 2020. ISBN: 9781492055013;1492055018;

[14] M. H. Halstead. *Elements of software science*. English. Vol. 2.;2; New York: Elsevier, 1977. ISBN: 0444002057;9780444002051;

[15] I. Idris. *Python data analysis: learn how to apply powerful data analysis techniques with popular open source Python modules*. English. 1st. Birmingham: Packt Publishing, 2014.

[16] IEEE Standard Glossary of Software Engineering Terminology. *IEEE Std 610.12-1990* (joulukuu 1990), 1–84. DOI: 10.1109/IEEESTD.1990.101064.

[17] D. Ince. *computer cluster*. eng. 2013-09-19. URL: http://www.oxfordreference.com/view/10.1093/acref/9780191744150.001.0001/acref-9780191744150-e-4121.

[18] *Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models*. Standard. International Organization for Standardization, maaliskuu 2011.

[19] G. Lanaro. *Python high performance programming: boost the performance of your Python programs using advanced techniques*. English. 1st. Birmingham, UK: Packt Pub, 2013. ISBN: 1783288469;9781783288465;9781783288458;1783288450;

[20] B. P. Lientz ja E. B. Swanson. *Software Maintenance Management*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1980. ISBN: 0201042053.

[21] T. J. McCabe. A Complexity Measure. *IEEE Transactions on Software Engineering* SE-2.4 (joulukuu 1976), 308–320.

[22] W. McKinney. *Python for Data Analysis*. English. 1. painos. US: O'Reilly, 2013;2012;

[23] W. McKinney. *Python for data analysis: data wrangling with pandas, NumPy and IPython*. English. Second. Sebastopol: O'Reilly, 2017.

[24] D. C. Montgomery, E. A. Peck ja G. G. Vining. *Introduction to linear regression analysis*. English. Fifth. Vol. 821. Hoboken, New Jersey: John Wiley Sons Ltd, 2012;2015;

[25] *MySQL 8.0 Reference Manual*. English. URL: https://dev.mysql.com/doc/refman/8.0/en/.

[26] *NumPy Documentation*. URL: https://numpy.org/doc/.

[27] *OptoFidelity homepage*. English. URL: https://www.optofidelity.com/.

[28] *Pandas Documentation*. URL: https://pandas.pydata.org/pandas-docs/stable/.

[29] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot ja E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.

[30] M. V. Persson ja L. F. Martins. *Mastering Python data analysis: become an expert at using Python for advanced statistical analysis of data using real-world examples*. English. Birmingham, UK: Packt Publishing, 2016.

[31] D. Posnett, A. Hindle ja P. Devanbu. A Simpler Model of Software Readability. *Proceedings of the 8th Working Conference on Mining Software Repositories*. MSR '11. Waikiki, Honolulu, HI, USA: ACM, 2011, 73–82. ISBN: 978-1-4503-0574-7. DOI: 10.1145/1985441.1985454. URL: http://doi.acm.org/10.1145/1985441.1985454.

[32] *PostgreSQL Documentation*. English. URL: https://www.postgresql.org/docs/.

[33] *Python Documentation*. URL: https://www.python.org/doc/.

[34] *SciPy ecosystem*. URL: https://www.scipy.org/.

[35] L. E. Spence, A. J. Insel ja S. H. Friedberg. *Elementary linear algebra: a matrix approach*. English. Upper Saddle River (NJ): Prentice Hall, 2000.

[36] *SQLite Documentation*. English. URL: https://www.sqlite.org/docs.html.

[37] *Stack Overflow Trends*. English. URL: https://insights.stackoverflow.com/trends.

[38] S. van der Walt, S. C. Colbert ja G. Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation. *Computing in Science Engineering* 13.2 (maaliskuu 2011), 22–30. DOI: 10.1109/MCSE.2011.37.

[39] K. D. Welker. The software maintainability index revisited. *CrossTalk* 14 (2001), 18–21.