jamk.fi

# Matemaattisen tekstin tunnistaminen Tesseract-ohjelmiston avulla

Jyrki Takanen

Opinnäytetyö
Joulukuu 2020
Tietojenkäsittely ja tietoliikenne
Insinööri (AMK), tieto- ja viestintätekniikka

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

**Kuvailulehti**

| Tekijä(t) | Julkaisun laji | Päivämäärä |
|---|---|---|
| Takanen, Jyrki | Opinnäytetyö, AMK | Joulukuu 2020 |
| | Sivumäärä<br>41 | Julkaisun kieli<br>Suomi |
| | | Verkkojulkaisulupa myönnetty: x |

Työn nimi
**Matemaattisen tekstin tunnistaminen Tesseract-ohjelmiston avulla**

Tutkinto-ohjelma
Insinööri (AMK), tieto- ja viestintätekniikka

Työn ohjaaja(t)
Antti Häkkinen, Jouni Huotari

Toimeksiantaja(t)

Tiivistelmä

Opinnäytetyössä käsiteltiin tekstintunnistusta erityisesti matemaattisten lausekkeiden tunnistuksen näkökulmasta. Työn tavoitteena oli tutkia mikä tekee matemaattisten kaavojen tekstintunnistusongelmasta vaikean ja minkälaisia järjestelmiä tekstintunnistusohjelmistot ovat.

Tavoitteena oli toteuttaa prototyyppi, joka tunnistaa painettuja matemaattisia kaavoja ja tulostaa tunnistetun kaavan LaTeX-merkintäkielelle. Prototyypissä käytettiin black box -lähestymistavalla Tesseract-tekstintunnistusohjelmistoa merkkien tunnistamiseen. Prototyypin avulla haluttiin havainnollistaa matemaattisten kaavojen tunnistamiseen liittyviä ongelmia sekä motivoida mahdollisien ratkaisuiden löytämistä ja testata ideoiden toimivuutta.

Teoreettisessa viitekehyksessä tutkittiin tekstintunnistuksen vaiheita ja matemaattisen tekstin tunnistukseen liittyviä erityispiirteitä. Viimeaikaisia tekoälypohjaisia menetelmiä tarkasteltiin ja näitä verrattiin perinteisiin menetelmiin.

Toteutuksessa käytiin vaiheittain läpi perinteisen tekstintunnistusjärjestelmän arkkitehtuurin eri osa-alueet painottuen matemaattisten kaavojen tunnistukseen. Yksinkertaisia kaavoja onnistuttiin tunnistamaan.

Lopputuloksena prototyyppi onnistui siinä mihin se oli tehty, eli oppimisprojektina tekstintunnistusjärjestelmiin perehtymisessä. Työssä löydettiin osittainen vastaus esitettyyn kysymykseen ja pohdittiin mahdollisia tulevaisuuden näkymiä. Opinnäytetyön kirjallisessa osuudessa koostettua tietoa ja lähteitä voidaan käyttää lähtökohtana aiheen opiskelulle.

Avainsanat (asiasanat)
Matemaattinen tekstintunnistus, Tekstintunnistus, OCR, Tesseract, Neuroverkko

Muut tiedot (Salassa pidettävät liitteet)

jamk.fi

**Description**

| Author(s) | Type of publication | Date |
|---|---|---|
| Takanen, Jyrki | Bachelor's thesis | December 2020 |
| | | Language of publication: Finnish |
| | Number of pages: 41 | Permission for web publication: x |

Title of publication

**Recognition of mathematical formulas using Tesseract OCR**

Degree programme

Batchelor of Engineering, Information and Communication Technology

Supervisor(s)

Häkkinen, Antti; Huotari, Jouni

Assigned by


Abstract

The thesis covers the theory of Optical character recognition (OCR) with emphasis on recognition of mathematical formulas. The purpose was to determine what makes mathematical formula recognition problem difficult to solve and to study what is the typical architecture of an OCR system.

The goal was to implement a prototype designed to recognize and output formulas of typeset mathematics in to the LaTeX-markup language. Within the prototype, Tesseract OCR was used as a black box to recognize symbols. The prototype was used to illustrate problems related to recognition of mathematical formulas and to motivate finding and testing of possible solutions.

The theoretical framework explored the different stages of optical character recognition and specialized features of mathematical formula detection. State-of-the-art methods based on neural networks were studied and compared to the traditional methods.

In the implementation, the architecture of traditional OCR systems was explored step-by-step with the emphasis on mathematical formula recognition. Simple mathematical formulas were successfully recognized.

As the result, the prototype was successful as a learning tool to study recognition systems. Partial answer to the asked question was obtained and possible future avenues were explored. Theoretical section of the thesis and the references found there in may be used for introduction to optical character recognition of mathematical formulas.



Keywords/tags (subjects)

Mathematical formula recognition, Optical character recognition, OCR, Tesseract, Neural network


Miscellaneous (Confidential information)

# Sisältö

**Kuviot**

**Terminologia**

| | |
|---|---|
| CAS | Computer algebra system. Yhtälöitä symbolisesti ratkaiseva ohjelmisto |
| End-to-end arkkitehtuuri | Neuroverkkoja hyödyntävä arkkitehtuuri tekstin tunnistukseen, jossa kaikki vaiheet toteutetaan samalla neuroverkolla (Vrt. pipeline arkkitehtuuri) |
| Fourier-kuvaajat | Kuvankäsittelyssä, muuntaa kuva-avaruuden muotoja taajuusavaruuteen, mikä helpottaa muotojen luokittelua |
| Gaussian Blur | Kuvankäsittelyssä, silotusmenetelmä, jossa kuvaa silotetaan käyttäen Gaussin funktiota |
| HMM | Hidden Markov Model. Markovin piilomalli. Tilastollinen malli, jossa taustalla olevan systeemin oletetaan olevan Markovin prosessi, jonka parametreja ei tunneta |
| Kernelimenetelmät | Kokoelma koneoppimisen menetelmiä, joissa käytetään kernelifunktiota nopeuttamassa luokitteluongelman ratkaisemista. Tunnetuin näistä on SVM |
| LaTeX | Tekstinladontaohjelmisto |
| Neuroverkko | Neural Network. Joukko algoritmeja, jonka rakenne muistuttaa aivojen hermosolun verkkomaista rakennetta |
| OCR | Optical Character Recognition. Tekstintunnistus |
| OpenCV2 | Avoimen lähdekoodin konenäköalgoritmeja kokoava kirjasto |
| Pipeline arkkitehtuuri | Perinteinen tekstintunnistus arkkitehtuuri, jossa tekstintunnistuksen vaiheet suoritetaan peräkkäin (Vrt. end-to-end arkkitehtuuri) |
| Python | Tulkattava ohjelmointikieli |
| Segmentointi | Tekstintunnistuksessa, kuvassa esiintyvien merkkien erottaminen toisistaan |
| SVM | Support vector machine. Tukivektorikone. Lineaarinen luokittelumalli, jota voidaan soveltaa luokitteluun ja käyränsovitustehtävään |
| Syväoppiminen | Deep Learning. Menetelmä, jossa usean kerroksen neuroverkot opetetaan tunnistamaan esitystapoja ja luokituksia raakadatasta |
| Tekstintunnistus | Prosessi, jossa kuvan tai muun median sisältämä teksti palautetaan takaisin sähköisesti käsiteltävään muotoon |

| | |
|---|---|
| Tesseract | Avoimen lähdekoodin tekstintunnistusohjelmisto |

# 1 Johdanto

## 1.1 Tausta ja työn lähtökohdat

Viimeaikainen kiinnostus tekoälyä kohtaan ja halu saada tekoälypohjaiset järjestelmät ymmärtämään tieteellistä tietoa motivoi parantamaan myös matemaattisten kaavojen tunnistusta. Paremmasta kyvystä tunnistaa matemaattisia kaavoja voisi olla hyötyä tulevaisuudessa, kun rakennetaan entistä monimutkaisempia järjestelmiä, jotka koostavat, käsittelevät ja *ymmärtävät* tieteellistä tietoa.

Idea opinnäytetyölle lähti liikkeelle allekirjoittaneen kiinnostuksesta tekstin tunnistamiseen ja erityisesti matemaattisten lausekkeiden tunnistamiseen. Kiinnostus matemaattisen lausekkeiden tunnistamiseen syntyi osaltaan omakohtaisesta tarpeesta matematiikan opinnoissa ja toisaalta mielenkiinnosta tutustua aiheeseen liittyvään laajaan algoritmien ja menetelmien kirjoon. Myöskin halu ymmärtää, miksi tekstintunnistamisongelman ratkaiseminen on niin vaikeaa, oli eräs syy aiheen valintaan.

## 1.2 Aiheen rajaus

Johtuen aiheen tuntemattomuudesta, ei alusta alkaen ollut selvää, miten aihe tulisi rajata tai edes se mitä työssä tullaan lopulta tekemään. Kuitenkin työn edetessä aiheen rajaus tarkentui ja työn tavoitteet täsmentyivät.

Alkuperäisenä ajatuksena oli tutustua alan kirjallisuuteen ja selvittää minkälaisia tekstintunnistuskirjastoja, -työkaluja ja -ohjelmistoja oli ennestään olemassa. Tämän jälkeen piti päättää mitä itse työssä tehdään. Ensimmäinen ajatus oli tutustua alan kirjallisuuteen ja opitun perusteella kirjottaa ohjelma, jolla matemaattisten kaavojen tunnistus onnistuisi. Nopeasti kuitenkin tuli selväksi, että ongelman laajuuden takia tämä ei ole mahdollista opinnäytetyön puitteissa.

Lopulta päädyttiin ratkaisuun, jossa hyödynnettiin avoimeen lähdekoodiin perustuvia työkaluja ja kirjastoja nopeuttamaan työn tekoa. Käyttöön valikoitui avoimeen lähdekoodiin perustuva Tesseract-tekstintunnistusohjelma, sekä OpenCV2-kirjasto, joka kokoaa kuvankäsittelyyn, konenäköön ja kuvanmanipulointiin tarvittavia algoritmeja.

Kirjallisessa osuudessa keskityttiin tarkastelemaan pääosin tutkimuksia, joissa käsiteltiin yleistä tekstin tunnistusta sekä painettujen matemaattisten kaavojen tunnistusta. Erityisesti käsin kirjoitettujen kaavojen tunnistukseen liittyvät erityispiirteet rajattiin tarkastelun ulkopuolelle.

## 1.3 Tavoitteet

Opinnäytetyön tavoitteeksi asetettiin tutkia kysymystä ’Mikä tekee matemaattisten kaavojen tekstintunnistusongelman ratkaisemisesta vaikean?’ sekä selvittää mikä on matemaattisten kaavojen tunnistusohjelmien nykytila. Tavoitteena oli lisäksi tutustua tekstintunnistusjärjestelmien arkkitehtuuriin sekä rakentaa prototyyppi, jolla voitaisiin tunnistaa painettuja matemaattisia kaavoja ja tätä apuna käyttäen perehtyä matemaattisen tekstintunnistuksen yhteydessä törmättäviin ongelmiin. Tavoitteeksi asetettiin myös tutustua alan uusimpaan kirjallisuuteen ja viimeisimpään tutkimukseen sekä tutkimussuuntauksiin.

## 1.4 Työn vaiheet

Työ jakautui karkeasti seuraaviin vaiheisiin:

- Aiheeseen tutustuminen ja alustava kirjallisuuskatsaus
- Sovelluksen suunnittelu ja toteutus
- Toinen kirjallisuuskatsaus
- Raportointi

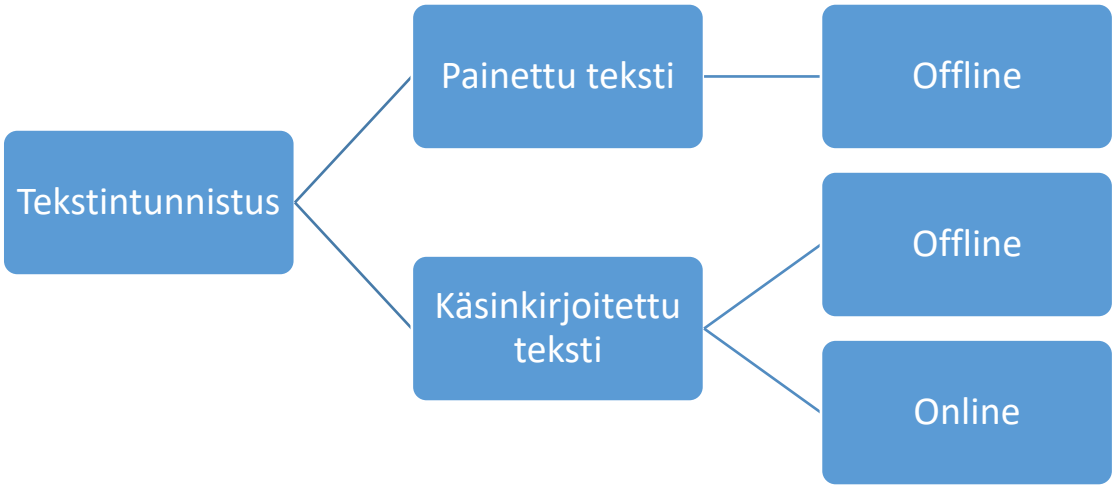
# 2 Tekstintunnistus

## 2.1 Historia

Ihmisen tavoite saada koneet toistamaan ihmisen tehtäviä on ollut haave jo muinaisilla ajoilla. Ensimmäiset konkreettiset yritykset tekstintunnistuksesta ovat kuitenkin 1900-luvun alkupuolelta, jolloin useat keksijät yrittivät kehittää tekstintunnistuksella apuvälineitä sokeille. Ensimmäiset nykymuotoiset tekstintunnistusjärjestelmät ovat kuitenkin 1940-luvulta, minkä mahdollisti samaan aikaan tapahtunut digitaalisten tietokoneiden kehitys. Siitä lähtien kehitystä on motivoinut mahdolliset sovellukset yritysmaailmassa. (Eikvil 1993, 8.)

## 2.2 Tekstin tunnistuksen lajit

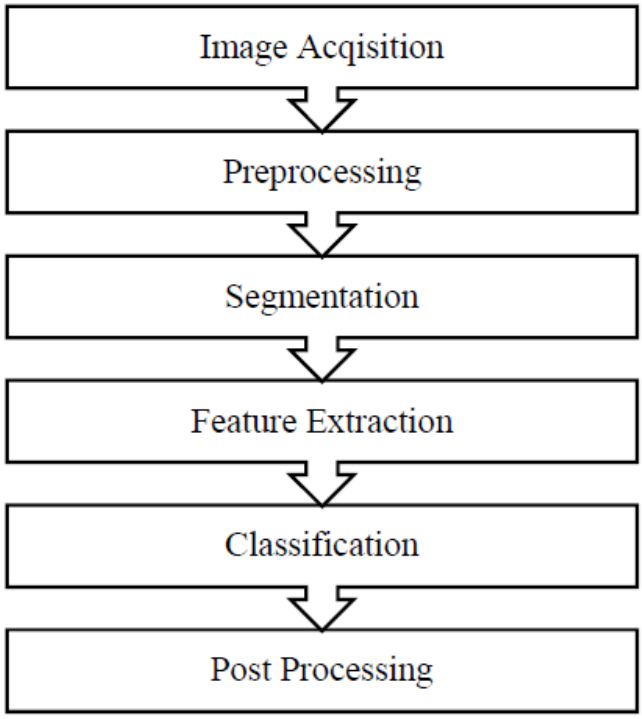
Tekstintunnistus voidaan jakaa useaan alalajiin, joista keskeisin on jako käsin kirjoitetun tekstin ja painetun tekstin välillä. (Ks. kuvio 1.) Käsin kirjoitetun tekstin tunnistus voidaan vielä jakaa reaaliaikaiseen (jossa tekstiä tunnistetaan sitä mukaa kun sitä kirjoitetaan, esimerkiksi kosketusnäytölle) sekä jälkikäteen tehtävään tunnistukseen. (Tapia 2005.)




Kuvio 1. Tekstintunnistuksen alalajit

## 2.3 Tekstin tunnistuksen vaiheet

Tekstintunnistus voidaan jakaa useaan osavaiheeseen, jotka suoritetaan järjestyksessä. Tätä kutsutaan pipeline arkkitehtuuriksi. Usein tekstin tunnistus jaetaan seuraavaan kuuteen eri vaiheeseen, jotka ovat kuvan muodostus, esikäsittely, segmentointi, ominaisuuksien tunnistaminen, luokittelu ja jälkikäsittely (Cheriet, Kharma, Liu & Suen 2007). Tarkastellaan jokaista vaihetta lyhyesti.




Kuvio 2. Tekstintunnistuksen vaiheet (Purohit & Chauhan 2016)

### 2.3.1 Kuvan muodostus

Ennen kuin minkäänlaista tunnistusta voidaan tehdä, on tunnistettavan tekstin sisältävä kuva saatava digitaaliseen muotoon. Tämä tapahtuu yleensä optisella skannerilla, mutta nykyään usein myös kameralla. Vaihtoehtoisesti kuva voidaan muodostaa jo valmiiksi digitaalisessa muodossa olevasta tiedostosta kuten pdf-tiedostosta.

Tunnistuksen haastavuus on paljolti riippuvainen siitä millä tavoin tarkasteltava kuva muodostetaan. Helpoimmassa tapauksessa kuva muodostetaan digitaalisesti julkaistusta materiaalista, joka on tuotettu tietokoneella (esimerkiksi LaTeX-ladontaohjelmalla). Tunnistuksen haastavuus lisääntyy, jos kuva on tuotettu fyysisestä materiaalista, skannaamalla tai valokuvaamalla. Tällöin kuvassa on usein erilaisia tahroja tai vinoumia, joita pitää korjata esikäsittelyvaiheessa. Myöskään kuvan kontrasti ei ole tällöin yhtä hyvä.

### 2.3.2 Esikäsittely

Ensimmäinen varsinainen tekstintunnistuksen vaihe on kuvan esikäsittely. Esikäsittelyssä kuvaa muokataan siten, että sitä on helpompi käsitellä myöhemmissä prosessoinnin vaiheissa. Tämä vaihe on myös keskeinen myöhemmin tapahtuvan tunnistuksen onnistumisen kannalta. Esikäsittelyyn voi kuulua useita eri vaiheita, joita kaikkia ei toteuteta kaikissa järjestelmissä, mutta yleensä kaikissa järjestelmissä toteutetaan ainakin seuraavat vaiheet: sivun suoristus, merkkien kallistuksen poisto, merkkien koon tasaus (normalisointi), tekstin silotus sekä kallistuksen poisto. Usein kuva myös muunnetaan harmaasävyiseksi tai mustavalkoiseksi, mikä yksinkertaistaa ja nopeuttaa datan prosessointia. (Cheriet, Kharma, Liu & Suen 2007.)

### 2.3.3 Segmentointi

Kuvan segmentoinnilla tarkoitetaan tekstissä esiintyvien merkkien erottamista toisistaan. Tämä voidaan tehdä käyttäen pikselihistogrammia. Useimmiten segmentointi tehdään hierarkkisesti irrottamalla ensin rivit toisistaan rivihistogrammilla, jonka jälkeen sanat erotetaan toisistaan sarakehistogrammilla ja lopuksi irrotetaan vielä merkit toisistaan. (Cheriet, Kharma, Liu & Suen 2007.)

### 2.3.4 Ominaisuuksien tunnistaminen

Ominaisuuksien tunnistuksessa erotelluista merkeistä etsitään ominaisuuksia, jotka parhaiten kuvaavat merkkiä luokitteluvaiheessa. Tämä vaihe nojaa vahvasti lukuisiin algoritmeihin ja matemaattisiin funktioihin, joilla kaksiulotteista kuvadataa muokataan muotoon, jossa sitä on helpompi käsitellä laskennallisin keinoin. Mahdollisia ominaisuuksia on paljon. Myös näitä voidaan luokitella tyyppien mukaan. Esimerkkejä ovat geometriset ominaisuudet kuten erilaiset momentit, histogrammit sekä suuntaperustaiset ominaisuudet. Rakenteellisia ominaisuuksia ovat muun muassa Fourier-kuvaajat sekä topologiset ominaisuudet. (Cheriet, Kharma, Liu & Suen 2007, 54-55.)

### 2.3.5 Luokittelu

Ominaisuuksien tunnistuksessa valittuja ominaisuuksia käytetään luokitteluvaiheessa syötteenä luokittelumenetelmille, joiden tavoitteena on luokitella ja nimetä merkit. Luokitteluun voidaan käyttää monia eri menetelmiä, joista yleisimpiä ovat: mallineeseen vertaaminen, tilastolliset menetelmät, neuroverkot, kernelimenetelmät kuten tukivektorikoneet (support vector machine, SVM) sekä näiden yhdistelmät. (Hamad & Kaya 2016.)

### 2.3.6 Jälkikäsittely

Jälkikäsittelyllä tarkoitetaan luokittelussa tapahtuneiden virheiden korjausta käyttämällä apuna muun muassa tietoa kielen ominaisuuksista esimerkiksi sanakirjoja tai Markovin ketjuja, joilla voidaan laskea peräkkäisten merkkien esiintymistodennäköisyyksiä. Myös muita monimutkaisempia, kontekstiin perustuvia menetelmiä on ehdotettu. (Hamad & Kaya 2016.)

# 3 Matemaattisen tekstin tunnistus

Matemaattisen tekstin tunnistuksella tarkoitetaan tekstissä esiintyvien matemaattisten kaavojen sekä symbolien tunnistusta. Painetun matemaattisen tekstin tunnistamisen sovelluskohteina voidaan mainita muun muassa erilaisten painettujen aineistojen (esimerkiksi integraalitaulukoiden) tuominen tietokoneen ymmärtämään muotoon, jolloin niitä voidaan edelleen käyttää esimerkiksi CAS-ohjelmissa sekä näkövammaisia auttavat teksti puheeksi järjestelmät.

## 3.1 Matemaattisen tekstin tunnistuksen erityispiirteet

Matemaattiset kaavat eroavat tavallisesta kirjoitetusta tekstistä usealla eri tavalla, riippuen mistä näkökulmasta näitä verrataan. Tässä vertailukohtana käytetään kirjoitusjärjestelmiä, joiden aakkoset muodostuvat merkeistä, jotka vastaavat kielen foneemeja (muun muassa latinalaiset, kyrilliset ja kreikkalaiset aakkoset).

Logogrammeista muodostuvissa kirjoitusjärjestelmissä - kuten kiinan kielet - on tekstintunnistukseen liittyviä eroavaisuuksia, joita ei huomioida tässä.

Ladonnaltaan kirjoitettu teksti on huomattavasti matemaattista kirjoitusta yksinkertaisempaa. Siinä missä kirjoitettu teksti on yksinkertaista: muodostuen rakenteista, joissa yksittäiset merkit yhdistyvät sanoiksi, jotka edelleen jakautuvat lineaarisesti riveille, on matemaattinen lauseke usein rakenteeltaan puumaisempi (Ks. kuvio 3) ja merkkien keskinäiset sijainnit kaksiulotteisessa tasossa muodostavat erilaisia riippuvuussuhteita merkkien välille (Fateman & Tokuyasu 1996).

Erona on myös puun eri tasolla sijaitseviin objekteihin sisältyvä informaation määrä. Jos kirjoitettu teksti hahmotellaan puurakenteena, jonka eri tasoilla ovat rivi, sana ja merkki, on jokainen näistä tasoista yksinkertaisempi kuin vastaavat tasot matemaattisessa kaavassa.

Rivitasolla kirjoitettu teksti muodostuu melko yksinkertaisista syntaksisista rakenteista; lauseen jäseniä erotetaan pilkulla ja virke päättyy pisteeseen. Sanatasolla sanat muodostuvat peräkkäisistä merkeistä, joihin voidaan kohtalaisen helposti käyttää apuna esimerkiksi sanakirjoihin perustuvia menetelmiä ja Markovin ketjuihin perustuvia tilastollisia menetelmiä. (Ks. Cheriet, Kharma, Liu & Suen 2007.) Merkkitasolla monissa kielissä merkistö on myöskin melko suppea, sisältäen numeraalit ja isot ja pienet kirjaimet huomioiden yleensä noin 60-100 merkkiä. Myöskin tekstin muotoilu tavallisessa painetussa tekstissä on kohtalaiset yksinkertaista muodostuen yleensä kallistuksista ja lihavoinneista.

$$\Gamma_\epsilon(x) = [1 - e^{-2\pi\epsilon}]^{1-x} \prod_{n=0}^{\infty} \frac{1 - \exp(-2\pi\epsilon(n+1))}{1 - \exp(-2\pi\epsilon(x+n))}$$

Kuvio 3. Esimerkki matemaattisesta kaavasta

## 3.2 Matemaattisen tekstin tunnistuksen vaiheet

Matemaattisen kaavan tunnistaminen alkaa kuten tavallisen tekstin tunnistus, jota kuvattiin luvussa 2. Tunnistus alkaa tunnistettavan kohteen saattamisella digitaaliseen muotoon. Tämä voi tapahtua joko reaaliaikaisesti jonkin syöttölaitteen avulla tai jälkikäteen esimerkiksi skannaamalla tarkasteltava materiaali digitaaliseen muotoon.

Matemaattisen kaavan tunnistuksen erityispiirteet voidaan jakaa vaiheisiin, jotka ovat kaavan irrotus, merkkien tunnistus, rakenteellinen analyysi sekä tulkinta (Tapia 2005). (Ks. kuvio 4.) Käsitellään vaiheita yksitellen.

### 3.2.1 Kaavan irrotus

Kaavan irrotuksen tavoite on tunnistaa dokumentista alueet, joissa esiintyy matemaattisia kaavoja ja erottaa nämä muusta dokumentissa esiintyvästä tekstistä ja sisällöstä. Kaavan irrotuksen voidaan katsoa kuuluvan osaksi laajempaa dokumentin analyysin käsitettä. Kaavan irrotus on tärkeässä osassa, kun halutaan saattaa tieteellistä tietoa muotoon, jossa sitä voidaan hyödyntää automaattisesti koneoppimisen menetelmillä. (Bruce 2014.) Reaaliaikaisissa järjestelmissä, joissa syötettä luetaan suoraan esimerkiksi kosketusnäytöltä ei tätä vaihetta tarvita.

### 3.2.2 Merkin tunnistus

Kuvion 4. mukaisessa jaottelussa merkin tunnistus noudattaa luvussa 2 kuvattua prosessia, pois lukien esikäsittely, joka toteutetaan ennen kaavan irrotusta, sekä jälkikäsittely, joka suoritetaan myöhemmin rakenteellisen analyysin tai tulkinnan yhteydessä, kun apuna on rakenteellisesta analyysista saatavaa kontekstuaalista informaatiota. (Tapia 2005.)

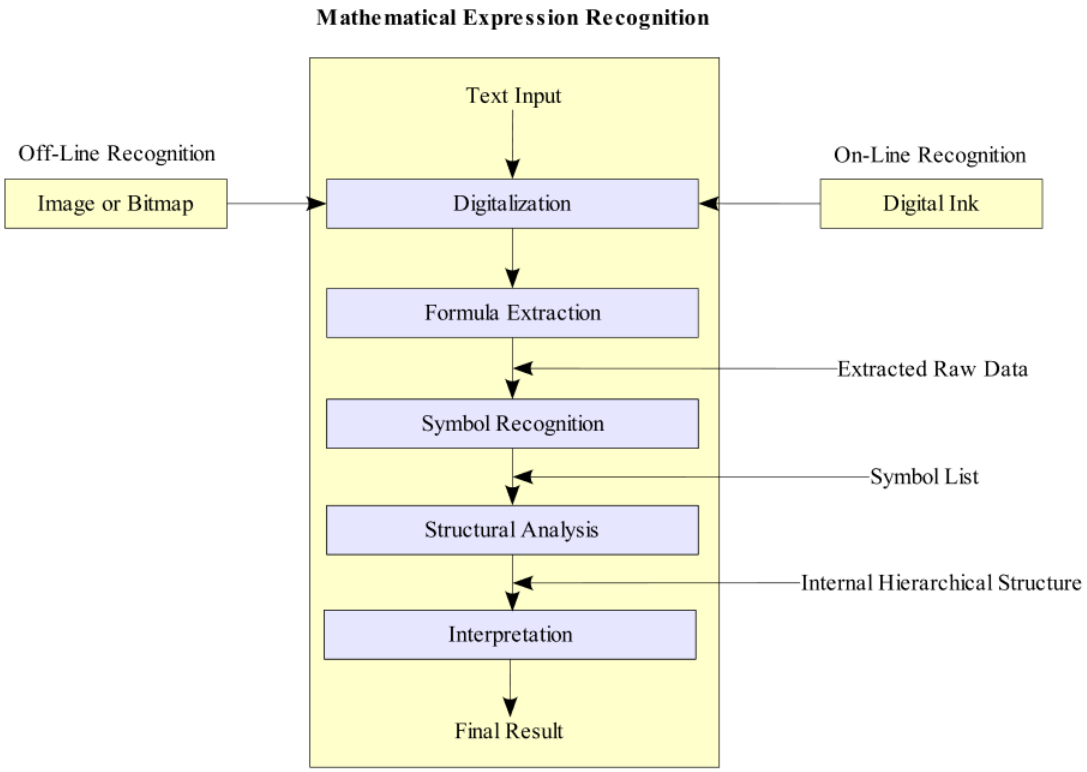
### 3.2.3 Rakenteellinen analyysi

Rakenteellisessa analyysissä, merkin tunnistuksessa kerättyä tietoa sekä merkkien sijaintitietoja apuna käyttäen muodostetaan kaavasta puurakenne, joka kuvaa

merkkien keskinäisiä suhteita tasossa. Usein merkkien suhteellisina sijainteina käytetään seuraavia sijainteja: ylävasen, ylä, yläindeksi, oikea, alaindeksi, ala, alavasen ja vasen. Puun juureksi voidaan valita mikä tahansa merkki ja puun yllä mainittujen sijaintien mukaan järjestetyt lehdet muodostuvat merkistä ja siihen liittyvästä numeerisesta datasta kuten sen sijainnista ja koosta. (Tapia 2005.) Kirjallisuudessa on ehdotettu myös monia muita menetelmiä lausekkeen rakenteelliseen analyysiin. (Ks. Tapia 2005; Blostein & Grbavec 1997.)

### 3.2.4 Tulkinta

Lopuksi saatu puu tulkitaan jollekin matemaattiselle kielelle, esimerkiksi LaTeX-merkintäkielelle. Ennen lopullista tulosta voidaan merkin tunnistuksessa tapahtuneita virheitä yrittää korjata rakenteellisesta analyysista saadulla kontekstuaalisella datalla. (Tapia 2005.) Erilaisia formaaleja ja sekä heuristisia menetelmiä on tarkasteltu kirjallisuudessa (Tapia, 18).




Kuvio 4. Matemaattisen kaavan tunnistusprosessi (Tapia 2005)

## 3.3 Tekoäly ja viimeaikaiset tutkimuskohteet

Viime vuosina neuroverkkoihin perustuvat syväoppimismenetelmät (deep learning) ovat olleet räjähdysmäisen kiinnostuksen kohteena sekä yliopistoissa että yrityksissä ja erilaisissa tutkimusorganisaatioissa. Vaikka neuroverkot itsessään eivät ole uusi asia, on niihin liittyvä kiinnostus kohtalaisen uutta. Tämä johtuu olennaisesti viimevuosina saavutetusta laskentatehosta, joka on mahdollistanut tarpeeksi syvät neuroverkot, jotka mahdollistavat kiinnostavien ongelmien ratkaisun (Introduction to Convolutional Neural Networks for Visual Recognition 2017).

Käytettäessä neuroverkkoja tekstintunnistusongelmaan voidaan lähtökohdaksi valita perinteiseen luvussa 2 kuvattuun prosessiin nähden täysin erilainen nk. ”end-to-end” menetelmä, jossa neuroverkolle syötetään kuvia ja saadaan LaTeX-kaavoja tuloksena. Menetelmä vaatii kuitenkin suuren määrän käytettävissä olevaa dataa, jolla neuroverkkoa opetetaan tunnistamaan kuvista kaavat. End-to-end menetelmää ovat käyttäneet muun muassa Deng, Kanervisto, Ling & Rush (2017).

Perinteisen ”pipeline” arkkitehtuurin ja ”end-to-end” menetelmän välille sijoittuvat erilaiset hybridimallit, joissa perinteisen tunnistusprosessin yksittäisiä komponentteja on korvattu neuroverkoilla. Tätä menetelmää ovat käyttäneet muun muassa Chang, Gupta & Zhang (2016).

Eräs ongelma, johon neuroverkkoja käytettäessä törmätään, on verkon opettamiseen tarvittavan datan suuri määrä. Tämä ongelma on kuitenkin suurilta osin ratkaistu ja datasettejä on nykyään vapaasti saatavilla. Esimerkkinä artikkelin (Deng, Kanervisto, Ling & Rush 2017) yhteydessä koottu IM2LATEX-100K (IM2LATEX-100K n.d) datasetti, joka on koostettu noin 60000 arXiv-palvelusta kerätystä teoreettisen fysiikan artikkelista. Kokoelma sisältää 103556 kaavaa LaTeX-muodossa, joista voidaan generoida opetukseen tarvittavat kuvatiedostot. Muita datasettejä ovat muun muassa (Infty Project Databases n.d.), (Marmot Dataset n.d.) sekä käsin kirjoitetun tekstintunnistuksen CROHME-kilpailuja järjestävän ICDAR (eli International Conference on Document Analysis and Recognition) datasetit, jotka sisältävät myös käsin kirjoitettuja kaavoja.

# 4 Prototyypin suunnittelu

## 4.1 Lähtökohdat

Työhön lähdettiin puhtaalta pöydältä käyttäen pohjana luvuissa 2 ja 3 kuvailtua arkkitehtuuria. Lisäksi apuna käytettiin kirjallisuudessa kuvattuja ideoita ja algoritmeja, joista pääosa on kuvattu lähteessä (Cheriet, Kharma, Liu & Suen 2007).

## 4.2 Tavoite

Lähtökohtana työn tekemiseen oli tavoite tutustua tekstintunnistukseen ja erityisesti matemaattisten kaavojen tunnistamiseen. Prototyypin tavoitteena oli kuroa umpeen kohtalaisen teoreettisen kirjallisuuden ja käytännön läheisen toteutuksen välistä kuilua. Ensimmäiseksi tavoitteeksi asetettiin toteuttaa prototyyppi, jolla saataisiin

tunnistettua edes yksinkertaisimpia kaavoja. Työlle asetettiin myös uusia tavoitteita sitä mukaa, kun aihekokonaisuus hahmottui. Myöhemmin asetettuja tavoitteita olivat muun muassa joidenkin kirjallisuudessa kuvattujen algoritmien toteuttaminen sekä valitun arkkitehtuurin ja tarvittavien tietorakenteiden toteutukseen liittyviin yksityiskohtiin paneutuminen, sillä näitä ei yleensä käsitellä kirjallisuudessa.

## 4.3 Työssä käytetyt teknologiat

Keskeisessä roolissa työn toteutuksessa olivat avoimeen lähdekoodiin perustuva Tesseract OCR-tekstintunnistuskirjasto sekä kuvankäsittelyyn soveltuvia algoritmeja sisältävä OpenCV2-kirjasto. Matemaattisten kaavojen ladontaan käytettiin LaTeX-tekstinladontaohjelmistoa. Ohjelmistokieleksi valikoitui Python. Käytetty versio oli 3.6.9.

## 4.4 Asetettuja rajauksia

Työn tarkentuessa ja tehtävän edetessä tuli selväksi, että sovellukselle oli asetettava erinäköisiä rajoituksia.

Ensimmäisiä tällaisia oli päätös siitä, että keskitytään tunnistamaan painettuja kaavoja. Käsin kirjoitetun tekstin tunnistamiseen liittyy vielä monia avoimia ongelmia (ks. Zhang, Du, Zhang, Liu, Hu, Hu, Wei & Dai 2017) ja oli selvää, että opinnäytetyön puitteissa näihin ei ehditä toteutusvaiheessa paneutua. Näin ollen työssä tarkasteltiin ainoastaan painettuja kaavoja.

Työssä toteutettiin luvussa 3 esitetyistä vaiheista symbolin tunnistaminen Tesseractin avulla, alkeellinen rakenteellinen analyysi, tulkintaa algoritmeilla ja sisäisen datan esittämiseen tarvittavia tietorakenteita. Erityisesti työssä ei toteutettu kaavan irrottamista, vaan oletuksena oli, että käytetyt testikuvat sisälsivät ainoastaan matemaattisia kaavoja.

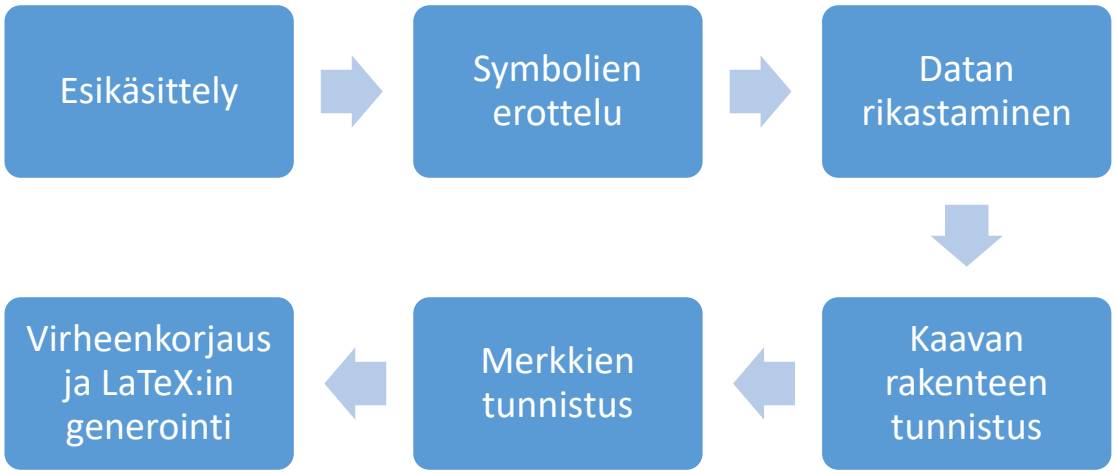
Usein vastaavissa sovelluksissa rajoitetaan tunnistettavaa merkistöä siten, että yritetään tunnistaa esimerkiksi ainoastaan ne symbolit, joihin törmätään

lukiomatematiikassa. Tässä emme rajoittuneet tarkastelemaan mitään tiettyä symbolien osajoukkoa, vaan tutkittavat symbolit rajautuivat sen mukaan, mitä symboleita käytetyissä testikuvissa esiintyi. Tarvittaessa sopiva osajoukko matemaattisista symboleista ja operaattoreista, joka kattaa suuren osan nykymatematiikasta voidaan löytää muun muassa LaTeX-ladontaohjelmiston dokumentaatiosta (Ks. Pakin 2020).

Ajan säästämiseksi toteutuksessa päädyttiin käyttämään Tesseract kirjastoa niin sanottuna mustana laatikkona. Vaihtoehtoisia tapoja olisivat olleet toteuttaa tietty osajoukko Tesseractin algoritmeista itse tai todennäköisemmin yhdistää olemassa olevia algoritmeja tai laajentaa Tesseractin omaa lähdekoodia. Työn edetessä kuitenkin todettiin, että on parasta keskittyä edellä kuvattuihin työvaiheisiin.

## 4.5 Ohjelman kuvaus ja rakenne

Ohjelmaa lähdettiin toteuttamaan pala kerrallaan niin sanotulla alhaalta ylös menetelmällä. Toisin sanoen tarvittavia komponentteja rakennettiin yksitellen ja näistä muodostettiin monimutkaisempi kokonaisuus. Tämä oli olennaisesti ainut mahdollinen lähestymistapa, koska aiheeseen tutustuttiin toteuttamisen edetessä. Tästä johtuen sovelluksen rakenne on myös toimintalogiikaltaan yksinkertainen, noudattaen kuvion 5 mukaista rakennetta.




Kuvio 5. Sovelluksen rakenne

Kuvataan seuraavaksi lyhyesti mitä kussakin kuvion 5 vaiheessa tehdään.

### 4.5.1 Esikäsittely

Esikäsittely aloitetaan kuvan lukemisella bittikartaksi, mikä mahdollistaa kuvan käsittelyn 2-ulotteisena taulukkona. Lukemisen jälkeen kuvaa voidaan skaalata tunnistuksen parantamiseksi sekä pehmentää merkkien reunoja niin kutsutulla Gaussian Blur:illa. Lopuksi kuva muunnetaan harmaansävyiseksi ja tästä edelleen mustavalkoiseksi ennalta valitun raja-arvon perusteella.

Pääosa esikäsittelyssä käytetyistä algoritmeista löytyy valmiina OpenCV2-kirjastosta. Työssä tarvittu esikäsittelyn määrä on verrattain vähäinen, koska suunnittelussa rajattiin kohtalaisen tarkasti millaisia kuvia tullaan käsittelemään. Toisin sanoen kuviksi valittiin ladontaohjelmalla luodusta painetusta tekstistä otetut kuvankaappaukset. Jos kuviksi olisi sallittu esimerkiksi kirjasta skannatut kuvat, olisi esikäsittelyssä tarvittu lisäksi ainakin kuvan suoristamiseen ja puhdistamiseen tarvittavia algoritmeja.

### 4.5.2 Symbolien erottelu

Symbolien erottelu toteutettiin käyttäen OpenCV-kirjastosta löytyvää findContours metodia, joka löytää ääriviivat binaarikuvasta. Sopivalla parametrillä algoritmi laskee ääriviivojen rajaaman alueen sisältymisen toisiinsa, jonka avulla datasta saatiin poistettua merkkien sisäiset ääriviivat, koska nämä eivät ole tunnistamisen kannalta olennaisia. Edelleen ääriviivoista laskettiin kyseistä merkkiä rajaavan suorakaiteen koordinaatit.

### 4.5.3 Datan rikastus

Datan rikastus vaiheessa olemassa olevista tiedoista lasketaan erilaisia apuna käytettäviä tietoja, kuten etsitään mille riville kaavaa kukin merkki kuuluu, sekä järjestellään merkit rivien, x- ja y-koordinaattien mukaisesti. Lisäksi lasketaan muun muassa symbolien kokojakauma myöhempää käyttöä varten.

### 4.5.4 Kaavan rakenteen tunnistus

Rakenteen tunnistusvaiheessa tunnistetaan kaavariviin sisältyviä erikoismerkkejä kuten jakoviivat, ala- ja yläindeksit, sekä symbolit, jotka muodostuvat useasta komponentista kuten yhtäsuuruus- ja erisuuruus merkit, kirjaimet i, j

### 4.5.5 Tekstin tunnistus

Tässä vaiheessa toteutetaan merkkikohtainen tunnistus Tesseract-kirjastoa apuna käyttäen. Yksittäisen symbolin tunnistus on kohtalaisen raskas prosessi vieden merkkiä kohti useamman sekunnin riippuen kuvan tarkkuudesta. Tunnistuksen nopeuttamiseksi käytettiin monisäikeistystä (multi-threading). Koska jokainen merkin tunnistus on riippumaton toisesta, saavutettiin näin huomattava nopeushyöty suoritusajassa. Tämä näkyi myös suoraan ohjelman kehityksen nopeutumisena.

Tesseract itsessään on suoritettava ohjelma, jonka käyttöliittymä toimii komentokehotteessa (command-line). Opinnäytetyössä käytettiin Pytesseract Python moduulia, kääreenä, jonka avulla Tesseract-ohjelmaa ajettiin.

Tunnistettavaksi syötettävän kuvan lisäksi Tesseractille syötettiin parametreinä käytettävä tunnistusmoottori, tunnistettavan tekstin kieli, sekä tekstin lohkomismuoto.

Tesseract ei itsessään esikäsittele kuvaa, mistä johtuen syötettävä kuva on oltava hyvälaatuinen tai tunnistuksen lopputulos on huono. Tesseractille voidaan myös syöttää kuvia, jotka sisältävät paljon tekstiä, mutta ohjelmassa on toimiva rakenteen tunnistus ainoastaan tavalliselle tekstiä sisältävälle sisällölle. Tästä johtuen kaavat päädyttiin syöttämään merkki kerrallaan, mikä vaikeuttaa tunnistamista huomattavasti. Tällaisen musta laatikko toteutuksen takia huomattava osa kehityksestä meni erilaisten asetusten testaamiseen, jotta tunnistustarkkuus saatiin mahdollisimman hyväksi.

### 4.5.6 Virheenkorjaus ja LaTeX:in generointi

Tässä vaiheessa korjattiin tunnistuksessa tapahtuneita virheitä sekä muodostetaan sisäisestä datasta LaTeX-merkintäkielinen tuloste. Muodostuksessa käytetään apuna aiempia vaiheita, erityisesti rakenteellista analyysiä.

# 5 Toteutus

## 5.1 Teknologiat ja kehitysympäristö

Ensimmäisistä Javalla tehdyistä kuvan esikäsittelykokeiluista tultiin johtopäätökseen, että Java ei ole paras kieli ongelman ratkaisemiseksi. Toiseksi vaihtoehdoksi valittiin Python, joka osoittautuikin paremmin ongelmaan sopivaksi. Tähän syynä olivat Pythonin tiiviys, sekä lukuisat olemassa olevat kirjastot, jotka helpottivat työn tekoa.

Kehitysympäristöksi valikoitui virtuaalikoneessa pyörivä Ubuntu 18.04 LTS, johon asennettiin Pythonin versio 3.6.9. Sattumalta OpenCV2-kirjaston ja Tesseract-

ohjelmiston versio oli kumpaisellakin 4.1.1, vaikka näillä ei keskenään mitään tekemistä olekaan. Työssä tarvittiin lisäksi seuraavia Python kirjastoja: pytesseract, argparse, cv2 ja anytree. Näistä maininnan arvoinen on ehkäpä pytesseract, joka toimii kehyksenä komentokehotteessa ajettavalle Tesseract-ohjelmalle ja mahdollistaa näin ohjelman ajamisen Pythonin sisällä. Ohjelmointiympäristönä toimi JetBrainsin PyCharm.

### 5.1.1 Kehitysalustan käyttöönotto

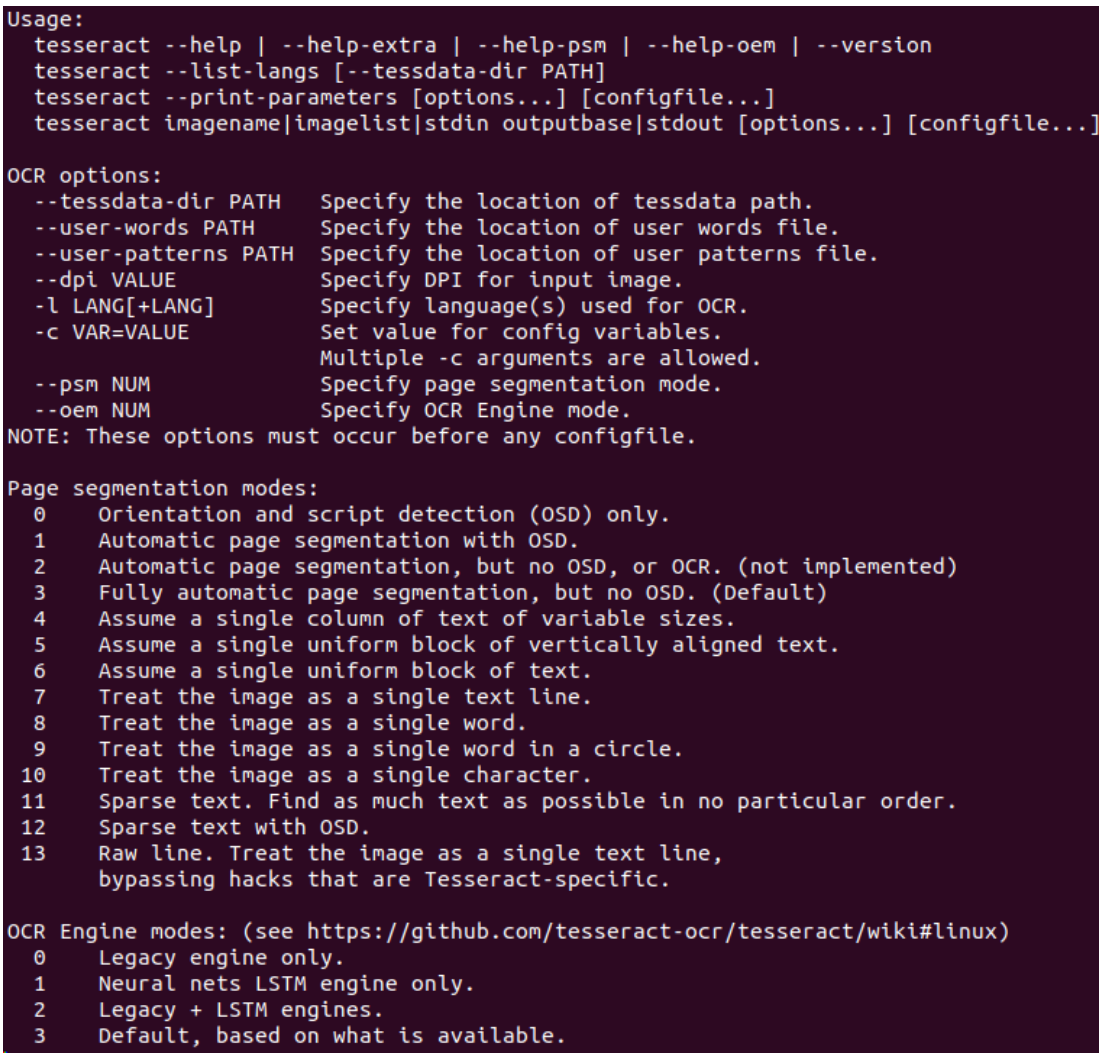
Kehitysympäristön pystytyksessä oli alkuun joitakin haasteita, koska kaikkia kirjastoja ei ollut suoraan saatavilla Ubuntun paketinasennusohjelman kautta. Oman haasteen alustan pystyttämiseen toi myös eri versioissa muuttuneet asennuskäytännöt. Tästä syystä ei myöskään tässä käydä tarkemmin asennusprosessia läpi, sillä on se todennäköisesti jo vanhentunut tätä luettaessa. Asennusohjeita löytyy parhaiten hakemalla internetistä.

Oman haasteensa toi myös Tesseractin uudemman neuroverkkopohjaisen LSTM-tunnistusmoottorin tarvitsemat opetuksen tulokset sisältävät tiedostot, joista tarvitaan erikseen omansa jokaiselle käytetylle kielelle. Tuetuiksi kieliksi valittiin englanti, suomi sekä kaavoissa esiintyviä erikoismerkkejä tukeva equation kieli. Ongelmia aiheutti erot ohjelman ja tiedostojen versioiden välillä sekä oikean asennussijainnin löytäminen, mutta lopulta ohjelma saatiin toimimaan halutuilla kielillä.

## 5.2 Sovelluskehitys

### 5.2.1 Alustus: Tesseractin testaaminen

Ensimmäinen vaihe ennen varsinaista sovelluskehitystä oli tutustua Tesseract-ohjelmaan sekä testata tunnistusta erilaisilla kuvilla sekä parametreillä. Tunnistettavan kuvan lisäksi Tesseractille voidaan syöttää lukuisia valinnaisia parametreja. Näistä tärkeimpiä ovat tunnistettavan tekstin kieli, tieto kuvan sisältämän tekstin muodosta (Page segmentation mode) sekä käytettävän tunnistusmoottorin valinta (OCR Engine mode). (Ks. kuvio 6.)




Kuvio 6. Tesseractin käyttöohje ja parametrien selitykset

Ohjelmalle voidaan viedä myös eräitä muita asetuksia, kuten tieto kuvantarkkuudesta (DPI), mutta tämä saadaan yleensä kuvasta automaattisesti eikä sitä tarvittu tässä työssä.

Tesseract oli käytössä ensimmäistä kertaa, joten käyttö aloitettiin perusteista, kokeilemalla miten kirjasto tunnistaa erilaista tekstiä sisältäviä kuvia. Osoittautuu että sekä englannin- että suomenkielinen painettu teksti tunnistui erittäin hyvin, ollen lähes virheetöntä. (Ks. esimerkki kuvioissa 7 ja 8.)

Testattaessa matemaattisten kaavojen tunnistusta huomattiin, että tunnistuksessa oli suuria ongelmia. (Ks. kuviot 7 ja 8.) Ongelmaan yritettiin etsiä ratkaisua

kokeilemalla erilaisia parametreja ja etsimällä tietoa internetistä. Osoittautui kuitenkin, että Tesseractia suunniteltaessa ei ole otettu huomioon matemaattisia kaavoja ja näin ollen pelkillä parametrien valinnoilla ongelmaan ei saataisi ratkaisua. Tästä pääteltiin, että työssä voidaan yrittää tehdä ainakin jotakin tunnistamistarkkuuden parantamiseksi.

missä $g : [a, b] \to \mathbb{R}$ on porrasfunktio, jos väli $[a, b]$ voidaan jakaa äärellisen moneen osaväliin siten, että $g$ on vakio kullakin osavälillä; porrasfunktion integraali on helppo määritellä: lasketaan vain porraspalkkien pinta-alat yhteen (portaan suunnan huomioiden): jos $a = x_0 \leq x_1 \leq x_2 \leq \cdots \leq x_k = b$ ovat sellaisia pisteitä, että $g(x) = c_j$, kun $x \in ]x_{j-1}, x_j[$, $j = 1, 2, \ldots, k$, niin

$$\int g = \sum_{j=1}^{n} c_j(x_j - x_{j-1}).$$

Kuvio 7. Tesseractin testaukseen käytetty, tekstiä sekä matemaattisia kaavoja sisältävä kuva

```
missä g : [a, d] — R on porrasfunktio, jos väli [a, b] voidaan jakaa äärellisen moneen
osaväliin siten, että g on vakio kullakin osavälillä; porrasfunktion integraali on
helppo määritellä: lasketaan vain porraspalkkien pinta-alat yhteen (portaan suun-
nan huomioiden): jos a = x9 < 11 < 19 <--- < 7j = b ovat sellaisia pisteitä, että
g(x) = cj, kun x € ]xj1,7;[, j = 1,2,...,k niin

/y = gcj(z]- — 51).
```

Kuvio 8. Kuvion 7 kuvasta Tesseractin tunnistama teksti. Käytetyt parametrit -l fin+equ –psm 3 ja –dpi 300

Perimmäinen ongelma kaavojen tunnistuksessa Tesseractilla on, että siinä ei ole toteutettu luvussa 3 kuvailtua matemaattisten kaavojen tunnistusta, vaan se on suunniteltu tunnistamaan tavallista kirjoitettua tekstiä (Smith 2016). Kaavat ovat kuitenkin lopulta tavallista tekstiä, kunhan ne pilkotaan tarpeeksi pieniin osiin, joten seuraava askel oli yrittää pilkkoa kaavoja pienempiin osiin ja käyttää Tesseractia näiden yksittäisten merkkien tunnistamiseen.

### 5.2.2 Kuvan lukeminen ja esikäsittely

Sovelluksen kehitys aloitettiin rakentamalla luvussa 4 kuvattuja osia. Aluksi kuva luettiin Pythonin ymmärtämään muotoon. Kuvan käsittely onnistui kohtalaisen helposti OpenCV-kirjastosta löytyvillä valmiilla funktioilla, joilla kuva saatiin luettua numpy-kirjaston taulukoksi. OpenCV-kirjastosta löytyi myös valmiit funktiot kuvan skaalaamiseksi ja harmaasävyiseksi sekä edelleen mustavalkoiseksi muuntamiseksi. Ennen harmaasävyiseksi muuntamista kuvaa sumennettiin Gaussian blur menetelmällä.

### 5.2.3 Segmentointi ja datan rikastus

Merkkien segmentointi toteutettiin käyttämällä OpenCV-kirjaston findContour metodia, joka tunnistaa mustavalkoisesta kuvasta reunat. (Ks. kuvio 11.) findCountour palauttaa ulkoreunojen lisäksi myös merkkien sisäreunat, mutta valitsemalla reunojen hakumuodoksi (mode) cv.RETR_TREE saadaan mukaan tieto reunojen keskinäisestä hierarkiasta, jolloin sisäreunat voidaan siivota tuloksesta pois. Edelleen, OpenCV:n metodilla boundingRect saadaan merkkiä vastaavista ääriviivoista laskettua sitä rajaava suorakaide. (Ks. kuvio 12.)

Ohjelmaa testatessa ei törmätty merkkeihin, joiden tunnistamisessa olisi ollut ongelmia. Tämä ei kuitenkaan sulje pois mahdollisuutta, ettei ongelmia esiintyisi ja tästä syystä menetelmää olisi syytä testata suuremmalla testidatalla.

Käytetyn menetelmän rajoitteena ovat merkit, jotka muodostuvat useammasta komponentista, kuten i, j, =, ≤ jne. Komponenttien yhdistäminen osoittautui osittain epätriviaaliksi ongelmaksi. Yksinkertaisille kaavoille, joiden voidaan olettaa sijaitsevan yhdellä rivillä, tämä on melko helppoa, mutta jos kaava on monimutkainen sisältäen esimerkiksi monia sisäkkäisiä jakoviivoja, ei ole aivan selvää, miten merkkejä tulisi yhdistää.

Työssä rakennettiin alkeellinen menetelmä yhtäsuuruusmerkkien, i ja j kirjaimien yhdistämiseksi (ks. kuvio 12), mutta menetelmä ei vaikuta yleistyvän helposti monimutkaisempiin tapauksiin. Ongelma vaikuttaa siltä, että ratkaisuun kannattaisi

käyttää jotakin oppimalla toimivaa algoritmia, sillä korjattavia symboleita ei ole mahdottoman paljon tai vaihtoehtoisesti mallineeseen vertaaminenkin saattaisi toimia. Käytetty, sijaintiin perustuva menetelmä ei ollut paras ratkaisu ongelmaan.

Kun merkkejä rajaavat laatikot saatiin selvitettyä, oli seuraava askel selvittää merkkien keskinäistä sijaintia. Kehityksen lähtökohdaksi valittiin kuvion 10 mukainen kuva, joka muodostui useammasta kaavasta, jotka olivat omilla riveillään. Erityistä perustelua kyseisen formaatin valinnalle ei ollut, mutta kyseinen tapaus valittiin lähtökohdaksi.

Testikuvasta erotettiin kaavarivit kuvion 9 mukaisella funktiolla. Tämän jälkeen merkeille laskettiin mille riville kukin merkki kuuluu.  Funktiota tulisi kehittää sillä sen toimivuus kuvasta toiseen riippui valitun split_thresholdin suuruudesta. Liian pienellä arvolla rivi jakautui virheellisesti kahdeksi riviksi ja liian suurella arvolla rivit eivät erottuneet toisistaan.

```
# Separates lines of text which have empty row of white pixels between them
# Returns list containing the lines in the following format:
# [iy, fy, height, dist_to_prev, dist_to_next]
# If line is first or last, dist_to_prev and dist_to_next are -1 accordingly
# split_threshold in pixels
def detect_text_lines(image, split_threshold=5):
    j = 0
    summed = np.sum(cv2.bitwise_not(image), axis=1)  # sum values of horizontal
    # line of pixels and return array of size image.height
    lines = []
    line = False
    for i in range(len(summed)):
        if summed[i] > 0 and not line:
            line = True
            d_to_prev = -1
            if len(lines) != 0:
                b = lines.pop()
                d_to_prev = i - b[1]
                b[4] = d_to_prev
                lines.append(b)
            lines.append([i, -1, -1, d_to_prev, -1])
        elif summed[i] == 0 and line:
            j = j+1
            if j <= split_threshold:
                continue
            line = False
            a = lines.pop()
            a[1] = i
            a[2] = a[1] - a[0]
            lines.append(a)
            j = 0
    return lines
```

Kuvio 9. Pikselihistogrammia hyödyntävä erotusfunktio

### 5.2.4 Kaavan rakenteellinen analysointi

Työssä toteutettiin joitakin alkeellisen rakenteen analysoinniksi luettavia ominaisuuksia. Koska tarkasteltavat kaavat olivat yksirivisiä (ks. kuvio 10) käytettiin merkkien sijaintia kaavarivillä apuna päättelyyn, oliko tarkasteltava merkki ala- tai yläindeksissä. Tieto tallennettiin kyseisen merkin tietorakenteeseen ja tietoa käytettiin myöhemmin apuna LaTeX-kielen generoinnissa. Myös yksinkertaisia menetelmiä jakoviivan tunnistamiseksi merkin dimensioiden perusteella kokeiltiin, mutta tätä ei ehditty työn yhteydessä saattaa toimintavalmiiksi.

Työn lopussa kokeiltiin vielä rakentaa yksinkertaista puumaista rakennetta, jossa tietorakenteena käytettiin Pythonin anytree moduulia, mutta aikarajoitteet tulivat vastaan ennenkö toteutusta saatiin valmiiksi.

### 5.2.5 Merkkien tunnistus

Yksittäisten merkkien tunnistaminen toteutettiin pytesseractin image_to_string funktiolla, jolle syötettiin parametrina merkin sisältävä osa kuvadatasta, sekä tesseractille vietävät parametrit. Parhaaksi psm asetukseksi osoittautui 10, eli yksittäistä merkkiä kuvaava asetus (ks. kuvio 6), kuten luonnollista on. Testatessa ohjelmaa tuli kuitenkin kokeiltua myös psm asetuksen vaikutusta tunnistuksen laatuun.

Tämä olikin yksi haasteellisimmista asioista työssä. Tesseract tuntui olevan äärimmäisen herkkä asetuksien muutoksille. Tunnistustarkkuus riippui voimakkaasti valituista esikäsittelyasetuksista sekä parametreistä. Myöskin merkkiä rajaavan suorakaiteen koon muuttaminen saattoi muuntaa tuloksen täysin toiseksi.

Ylivoimaisesti hitain osa ohjelman suorituksessa oli Tesseractin ajaminen. Kun ohjelman testaamisen hitaus alkoi käydä hermoille, päätettiin selvittää, onnistuuko Tesseract prosessien ajaminen rinnakkain. Tämä onnistuikin kohtalaisen helposti ja ainut päänvaivaa aiheuttanut osuus oli selvittää kuinka rinnakkain suoritettujen prosessien palauttama data tulisi käsitellä. Tämä onnistui kohtalaisen helposti käyttäen apuna Pythonin multiprocessing moduulia, jonka Pool luokan map metodilla saatiin Tesseract ajettua taulukollisella parametreja siten että tulokset palautuivat vastaavassa järjestyksessä taulukkoon. Näin ollen ainoaksi ongelmaksi jäi parametrien vieminen alkuperäisestä tietorakenteesta taulukkoon ja vastaavasti saadun syötteen vieminen takaisin alkuperäiseen tietorakenteeseen. (Ks. liite 1.)

### 5.2.6 Virheenkorjaus ja LaTeX:in generointi

Huolimatta siitä, että Tesseract ajettiin käyttäen matemaattisia symboleita tunnistavaa equ kielikirjastoa, ei esimerkiksi integraalisymbolit tahtoneet tunnistua oikein. Integraali tunnistui kuitenkin järjestelmällisesti joko symboliksi ’/’ tai ’J’. Näin

ollen päädyttiin yksinkertaiseen ratkaisuun, jossa kyseiset merkit korvataan LaTeX:in integraalia kuvaavalla \int koodilla. Tässä on tietysti ongelmana mahdollisten oikeiden J symboleiden esiintyminen kaavoissa.

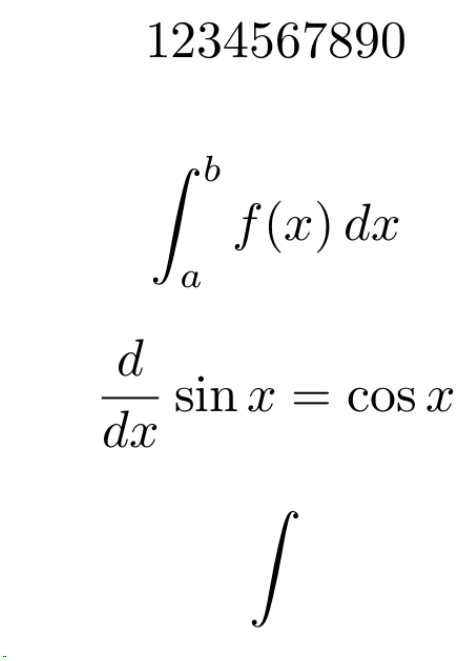
Mahdollisia ratkaisuja tälle olisi verrata merkkien keskinäistä kokoa tai merkkiä rajaavan laatikon kuvasuhdetta. Integraalimerkit ovat tyypillisesti suurempia kuin kirjain ’J’ tai merkki ’/’ ja tätä tietoa voitaisiin käyttää hyödyksi virheenkorjauksessa.

Myöskin merkkien rajaamien suorakaiteiden kuvasuhteet antavat mahdollisuuden erotella merkit toisistaan ja mahdollistaen siten paremman virheen korjauksen. Negatiivisena puolena jälkimmäisessä on kuvasuhteiden riippuvuus kaavan fontista ja luonnollisesti näin yksinkertainen menetelmä ei toimi käsin kirjoitetulle tekstille tai jos tekstissä on skannauksen aiheuttamia vääristymiä.

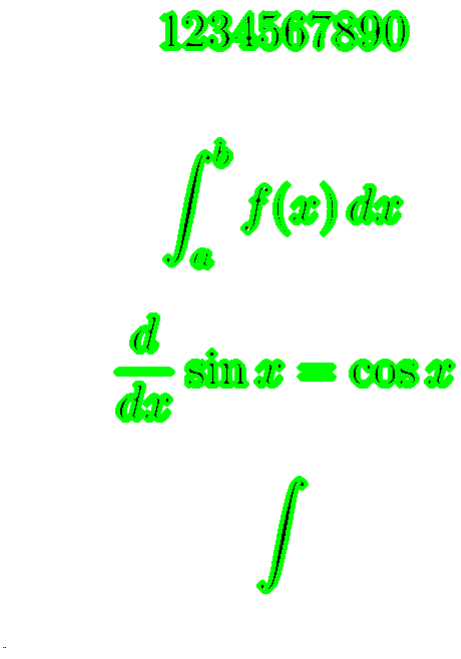
Käyttämällä rakenteellisessa analyysissä saatua tietoa muodostettiin tietorakenteesta LaTeX-merkkikielinen versio. Käytännössä työssä toteutettiin lähinnä ylä- ja alaindeksien käsittely, joka nojasi merkkien sisältämään tietoon, olivatko ne ylä- tai alaindeksejä sekä merkkien keskinäiseen järjestykseen tietorakenteessa. Saatu tulos esimerkkikuvan (kuvio 10) tapauksessa on esitetty kuviossa 13.

## 5.3 Testaus

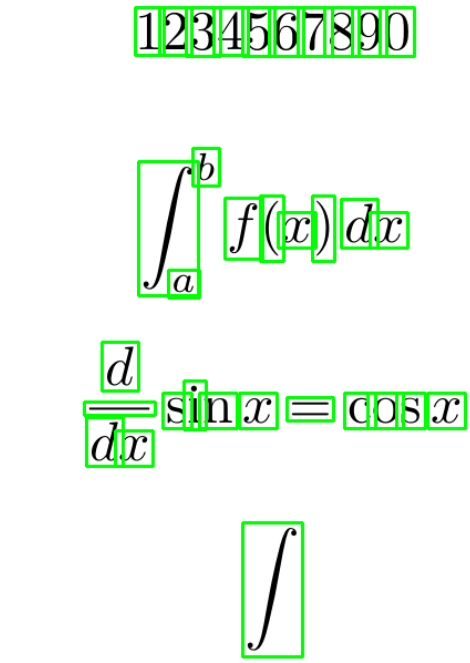
Ohjelmaa testattiin syöttämällä sille kuvia yksinkertaisista painetuista kaavoista (Ks. kuvio 10). Debuggaamisen helpottamiseksi kuvaan piirrettiin suorituksen aikana merkeistä tunnistetut reunat sekä rajaavat laatikot (Ks. kuviot 11 ja 12).




Kuvio 10. Esimerkki syötteenä käytetystä testikuvasta




Kuvio 11. Kuvasta tunnistetut reunat



Kuvio 12. Merkkien ulkoreunoja rajaavat laatikot (useasta komponentista muodostuvat merkit yhdistetty).

```
1234567890
\int_{a}^{b}f(x)dx
_{-d}^{d_{}x}S1nx=C0:Sx
\int
```

Kuvio 13. Kuvaa vastaava tulos

Kuviota 10 vastaava LaTeX kaava on esitetty Kuviossa 13. Kuvasta nähdään, että ensimmäisen rivin numeraalit tunnistuivat oikein ja toisen rivin integraali tunnistui myös oikein. Sen sijaan rivillä 3 esiintyi useita virheitä. Johtuen puutteellisesta rakenteen analyysistä ei jakoviiva tunnistunut oikein vaan osa merkeistä tunnistettiin ala ja yläindekseiksi. Myös matemaattisten operaattorien sin ja cos merkkien tunnistuksessa oli ongelmia. Nämä johtuivat Tesseractin virheellisestä tunnistustuloksesta. Tulos, johon olisi haluttu päästä on esitetty kuviossa 14.

Ongelmaa operaattoreiden sin ja cos merkkien kanssa yritettiin korjata muuttamalla rajaavien suorakaiteiden kokoa, jolloin Tesseractin saaman kuvan koko muuttui.

Tämä auttoikin joidenkin symboleiden kanssa, mutta rikkoi tunnistuksen toisille symboleille. Älykkäämpi tapa muuntaa merkkiä ympäröivää marginaalia merkkikohtaisesti saattaisi auttaa tunnistuksessa, mutta tätä ei työn puitteissa tutkittu. Kaavoja olisi myös voinut korjata käsin kaavojen tunnistuksessa, esimerkiksi korvaamalla merkkijonon ’S1n’ merkkijonolla ’\sin’, mutta tämä tehtiin jo integraalille, eikä ratkaisu vaikuttanut tyydyttävältä.

```
1234567890
\int_{a}^{b} f(x) dx
\frac{d}{dx} \sin x = \cos x
\int
```

Kuvio 14. LaTeX-koodi, joka generoi kuvion 10 lausekkeet

# 6 Johtopäätökset

Toteutuksen avulla saatiin osittainen vastaus opinnäytetyön alussa esitettyyn kysymykseen: ’miksi matemaattisten kaavojen tunnistaminen on vaikeaa?’.

Vaikka työn käytännön osuudessa ei päästy kovin pitkälle ongelman ratkaisussa, törmättiin työssä kuitenkin yksinkertaisempiin versioihin ongelmista, jotka ovat aiheuttaneet pään vaivaa myös tutkijoille viime vuosikymmenten aikana. Osa pohdituista ongelmista oli jopa samoja, joita myöhemmin huomattiin tarkastellun tutkimusartikkeleissa alan alkuvuosina. (Ks. Anderson 1967.)

Osoittautui, että matemaattisten kaavojen tunnistuksen haastavuus johtuu suurilta osin luvussa 3 kuvatuista erovaisuuksista tavalliseen tekstiin nähden. Olennaisimpana näistä ovat merkkien keskinäisten riippuvuussuhteiden monimutkaisuus sekä matemaattisten lausekkeiden monimutkaisempi riippuvuus kontekstuaalisesta datasta.

Työssä Tesseract-tekstintunnistusohjelmaa käytettiin black box ajatuksella tunnistamaan yksittäisiä merkkejä. Toisin sanoen ohjelmalla oli tunnistuksen apuna mahdollisimman vähän kontekstia. Toisaalta vaikka lähestymistapa olisi ollut erilainen ei lopputulos olisi välttämättä ollut merkittäviltä osin parempi, johtuen matemaattisten kaavojen kontekstuaalisen datan monimutkaisuudesta. Paremmalla rakenteellisella analyysillä voitaisiin mahdollisesti saada hyödyllistä kontekstuaalista dataa merkintunnistuksen avuksi.

Työn aikana havaittiin myös black box lähestymistavan rajoittuneisuus. Tesseractin suorittaman merkintunnistuksen avuksi ei voida viedä ohjelmassa kerättyä tietoa, joka voisi auttaa yksittäisten symboleiden tunnistamisessa, vaan tunnistustarkkuuteen voidaan vaikuttaa ainoastaan Tesseractin omilla parametreillä sekä muokkaamalla merkin sisältävää kuvadataa.

Näin ollen mahdollisesti parempi lähtökohta olisi ollut rakentaa kaavojen tunnistuksen komponentteja Tesseractin sisään. Tämä ei kuitenkaan työn alussa vaikuttanut realistiselta ajatukselta tekijän lähtötason huomioon ottaen. Voidaan myös ajatella, että jos Tesseractiin olisi helppo lisätä rakenteellinen analyysi matemaattisille kaavoille, olisi joku sen jo tehnyt.

Mielenkiintoisena seikkana voidaan todeta, että kirjallisuudessa kuvatun perinteisen tekstintunnistusjärjestelmän arkkitehtuuri ei ole olennaisesti muuttunut vuosikymmenten saatossa, vaan on pysynyt suurilta osin samana. Toisin sanoen useimmissa työn lähteissä kuvatuissa järjestelmissä on edelleen löydettävissä kuvioiden 2 ja 4 mukaiset vaiheet.

Aivan viime vuosina tähän on kuitenkin tullut muutos, kun laskentatehon kasvu on mahdollistanut monimutkaisien ongelmien ratkaisun käyttäen hyväksi neuroverkkoja. Luvussa 3.3 kuvatuilla tekoälypohjaisilla menetelmillä on jo päästy hyviin lopputuloksiin, jotka vastaavat ja joissain tapauksissa ohittavat perinteisillä

menetelmillä saavutetut tulokset, ainakin kun käytössä on synteettinen (keinotekoinen) testidata (Deng, Kanervisto, Ling & Rush 2017).

Työssä toteutetun prototyypin ja osaltaan myös kirjallisuuteen tutustumisen perusteella saatiin kuva, että matemaattisten kaavojen tunnistus on edelleen haastava ongelma. Tämä on varmasti totta, varsinkin kun tarkasteluun otetaan mukaan reaalimaailman muuttujat, jolloin tunnistusta vaikeuttaa muun muassa fyysisiin materiaaleihin ja näiden skannaamiseen liittyvät laadulliset ongelmat.

Toisaalta näin saadaan vain osa totuudesta, sillä esimerkiksi Mathpixin (Mathpix 2020) kaltaiset kaupalliset sovellukset tunnistavat painettuja matemaattisia kaavoja huomattavan hyvällä tarkkuudella. Tutkimuksen näkökulmasta kaupallisista suljetun lähdekoodin sovelluksista ei kuitenkaan ole hyötyä. Eikä nämä sovellukset yleensä sovellu opinnäytetyön alussa kuvattujen järjestelmien tarvitseman datan keräämiseen.

Markkinoilta löytyy myös lukuisia suuremman tietomäärän tunnistamiseen tarkoitettuja ratkaisuja, joilla voidaan saattaa digitaalisesti käsiteltävään muotoon esimerkiksi yrityksien arkistoja tai muita suuria tietolähteitä. Esimerkkeinä voidaan mainita Aquaforestin AutoBahn DX (AutoBahn DX 2020) ja SimpleOCR:n sovellukset kuten ABBYY (SimpleOCR 2020). Näille on kuitenkin tyypillistä korkea hinta sekä soveltuvuus ainoastaan painetulle tekstille tai taulukkomuotoiselle datalle.

Lopuksi todettakoon vielä, että toteutuksessa kuvatuista menetelmistä päädyttiin johtopäätökseen, joka vastasi mielestäni kirjallisuuden perusteella saatua kuvaa matemaattisten kaavojen tunnistuksen tilasta perinteisiä algoritmeja käyttäen. Tällä tarkoitetaan, että käytettäessä perinteisiä algoritmeja ja menetelmiä päädytään helposti kappaleessa 5.2.6 kuvattuun tilanteeseen, jossa päädytään tarkastelemaan loputtomasti erikoistapauksia ja korjaamaan käsin yksittäisiä virhetilanteita.

# 7 Pohdinta ja jatkokehitys

## 7.1 Pohdinta

Työssä asetettuihin tavoitteisiin päästiin ainakin siinä mielessä, että kirjallisuudessa käytetty termistö ja osajärjestelmien rajaukset näkyivät myös toteutetussa työssä.

Sivuhuomiona todetaan, että työssä toteutetun sovelluksen kaltaisilla rajoituksilla parempi tapa mahdollistaa sisällön käsittely jälkikäteen tietokoneella, olisi sisällyttää jo luontivaiheessa ladottuun tiedostoon mukaan metadatana automaattisesti käsiteltävissä oleva muoto samasta datasta. Käytännössä esimerkiksi ladontaohjelmiston käyttämä versio tekstistä. Tämä mahdollistaisi muun muassa kaavojen lukemisen ääneen ja prosessoinnin automaattisesti. Koska nykyjärjestelmät eivät tätä kuitenkaan yleensä tee, on myös työn rajoitteiden mukaiselle ohjelmalle käyttöä. Ja toisaalta järjestelmää voidaan myös kohtalaisen helposti laajentaa tukemaan myös yleisempää kuvadataa (esimerkiksi skannattuja kuvia).

### 7.1.1 Lähdeaineistosta

Haasteen mille tahansa teknologia-alan tutkimuksen tarkastelulle, aiheuttaa se, että iso osa viimeisimmästä tutkimustiedosta on olemassa ainoastaan yrityksien sisällä ja tutkimusryhmissä. Tässä mielessä myös toimeksiantajasta, jolla olisi ollut ennakkotietoa aiheesta ja jonka kanssa olisi voinut keskustella ideoista olisi ollut suuri apu työtä tehdessä. Lopulta huomattava aika työstä meni sopivien lähdemateriaalien etsimiseen. Osaan artikkeleista, joista olisi ollut hyötyä työtä tehdessä ei ollut pääsyä maksumuurin takia.

## 7.2 Jatkokehitys

Työssä toteutetut menetelmät ja näitä käsittelevät lähteet sijoittuvat nykynäkökulmasta vähän menneeseen aikaan. Nämä antavat kyllä hyvät pohjatiedot ja lähtökohdan aiheen tutkimiseen, mutta erityisesti työn käytännön osuudessa ei

tutustuttu aivan uusimpiin menetelmiin ja nykyaikana tärkeisiin neuroverkkoihin ja muihin tekoälypohjaisiin menetelmiin.

Luonnollinen jatko opinnäytetyölle olisi tutustua viimeisen 5-10 vuoden aikana tehtyyn tutkimukseen, jota kuvattiin luvussa 3.3 ja tutustua vastaaviin moderneihin menetelmiin sekä tarkastella tekstintunnistusongelmaa käyttäen näitä menetelmiä.

## 7.3 Kokemukset

Prototyyppiä tehdessä uusin lähde, johon tekijä oli aiheesta törmännyt, oli vuodelta 2007. Näin ollen käytännön osuudessa ei tutkittu uudempia end-to-end arkkitehtuureja tai tekoälypohjaisia menetelmiä, joihin tutustuttiin raportin teon yhteydessä tarkastellussa kirjallisuudessa. Vaikka luvussa 1.4 kuvattu työjärjestys ei ollut paras mahdollinen, oli perusteltua lisätä työhön katsaus viimeaikaisesta kehityksestä lisäämään työn ajankohtaisuutta.

Opiskelun näkökulmasta kirjallisuuteen palaaminen oli myös hyödyllistä. Jälkeenpäin tarkasteltuna olisi ennen prototyypin toteutusta pitänyt tutustua kirjallisuuteen paremmin, ja kun prototyyppiä alettiin toteuttamaan, jäätiin ehkä turhan pitkäksi aikaa miettimään yksinkertaisia ongelmia, kun hyödyllisempää olisi ollut aina välillä tutustua uudempaan kirjallisuuteen.

Osoittautui, että prototyyppi, jossa testattiin monenlaisia ideoita, oli hyvä tapa tutustua aiheeseen, eikä se, että tulokset jäivät kohtalaisen vaatimattomiksi, ole ongelma, kun huomioidaan, että käytetyt menetelmät eivät jälkikäteen tarkasteltuna olleet nykymittapuulla hirveän relevantteja. Tästä syystä olikin ehkä parempi, ettei yksittäisen osa-alueen kanssa piipertämiseen käytetty liikaa aikaa.

Jälkikäteen tarkasteltuna uuden aihekokonaisuuden opiskelu opinnäytetyön puitteissa – ilman toimeksiantajaa tai ohjaajaa, jolla olisi ollut tietoa aiheesta — ei ollut paras idea. Tästä johtuva epävarmuus opinnäytetyön valmistumisesta meinasi välillä käydä hermoille.

## 7.4 Tulevaisuus

Tekijälle jäi lopulta epäselväksi mikä menetelmä tulee loppujen lopuksi olemaan paras tapa tunnistaa matemaattisia lausekkeita. Perinteisten menetelmien tutkiminen vaikuttaisi olevan lähes menneen talven lumia, mutta on vaikeaa sanoa missä määrin esimerkiksi kaupalliset järjestelmät soveltavat ”pipeline” arkkitehtuuria ja missä määrin tekoälyä. Turvallista on kuitenkin sanoa, että tulevaisuudessa tekoäly tulee olemaan yhä suuremmassa roolissa myös tekstintunnistukseen liittyvien ongelmien ratkaisussa.

Tekoälypohjaisten järjestelmien etuna on niiden suhteellinen yleisyys: samoja neuroverkkoja voidaan käyttää monen erilaisen sovelluskohteen ratkaisemiseen ilman, että tarkasteltavaa ongelmaa tarvitsee tuntea yhtä tarkasti kuin perinteisiä ”pipeline” arkkitehtuureja käytettäessä. Tämän osoittaa niin kutsuttujen ”end-to-end” järjestelmien toimivuus. Tästä johtuen on myös odotettavissa, että neuroverkkojen jatkuva kehitys ja käytettävissä olevan laitteiston kehittyminen tulee automaattisesti auttamaan myös tekstintunnistuksessa.

Toisaalta tekoäly ja neuroverkkopohjaiset ratkaisut tuovat mukanaan myös uudenlaisia haasteita. Neuroverkon opetuksessa käytettävä data on keskeisessä roolissa hyvän tunnistustarkkuuden saavuttamisessa. Toisaalta koska neuroverkoissa tehtävien muutosten vaikutukset nähdään yleensä epäsuorasti ja uudelleenopettaminen on suhteellisen työlästä, on näihin menetelmiin perustuvien järjestelmien testaaminen usein hankalaa ja ovat yksittäisistä muutoksista saatavat kehitysaskeleet yleensä pieniä.

Tulevaisuudessa tulee olemaan jännittävää seurata korvaavatko YOLO:n (Kamal 2019) kaltaiset yleiset kappaleentunnistusalgoritmit nykyisen kaltaiset erikoistuneet sovellukset.

# Lähteet

Anderson, R. H. 1967. Syntax-directed recognition of hand-printed two-dimensional mathematics. In Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium (pp. 436-459).

Autobahn DX. 2020. Viitattu 4.12.2020
https://www.aquaforest.com/en/Autobahn.asp

Blostein, D. & Grbavec, A. 1997. Recognition of mathematical notation. In Handbook of character recognition and document image analysis (pp. 557-582).

Bruce, J. R. 2014. Mathematical expression detection and segmentation in document images. Master's thesis. Virginia Tech. Viitattu 3.12.2020
https://vtechworks.lib.vt.edu/bitstream/handle/10919/46724/Bruce_JR_T_2014.pdf

Chang, J., Gupta, S. & Zhang, A. 2016. Painfree LaTeX with Optical Character Recognition and Machine Learning. Loppuraportti. CS229 Stanford University. Viitattu 1.12.2020. http://cs229.stanford.edu/proj2016/report/ChangGuptaZhang-PainfreeLatexWithOpticalCharacterRecognitionAndMachineLearning-report.pdf

Chaudhuri, A., Mandaviya, K., Badelia, P. & Ghosh S. K. 2017. Optical Character Recognition Systems for Different Languages with Soft Computing. Springer.

Cheriet, M., Kharma. N., Liu. C.-L. & Suen C. Y. 2007. Character Recognition Systems – A Guide for Students and Practitioners. Wiley.

Deng, Y., Kanervisto, A., Ling, J. & Rush, A. M. 2017. Image-to-markup generation with coarse-to-fine attention. In International Conference on Machine Learning (pp. 980-989). PMLR.

Eikvil, L. 1993. OCR – Optical Character Recognition. Luentomuistiinpanot.

Fateman, R. J. & Tokuyasu, T. A. 1996. Progress in recognizing typeset mathematics. In Document Recognition III (Vol. 2660, pp. 37-50). International Society for Optics and Photonics.

Hamad, KA. & Kaya, M. 2016. A detailed analysis of optical character recognition technology. International Journal of Applied Mathematics, Electronics and Computers 4.1 (2016): 244-249.

IM2LATEX-100K. N.d. Viitattu 1.12.2020.
https://www.icst.pku.edu.cn/cpdp/sjzy/index.htm

Infty Project Databases. N.d. Viitattu 1.12.2020.
http://www.inftyproject.org/en/database.html

Introduction to Convolutional Neural Networks for Visual Recognition. 2017. CS231n Convolutional Neural Networks for Visual Recognition. Luento. Stanford University. Viitattu 3.12.2020. https://www.youtube.com/watch?v=vT1JzLTH4G4

Jensen, A. & Marklund, H. 2017. Turning Equations into Latex: Pipeline vs. End-to-End. Loppuraportti. CS229 Stanford University Viitattu 1.12.2020. http://cs229.stanford.edu/proj2017/final-reports/5243453.pdf

Kamal, A. 2019. YOLO, YOLOv2 and YOLOv3: All You want to know – Medium. Viitattu 3.12.2020. https://medium.com/@amrokamal_47691/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899

Marmot Dataset. N.d. Viitattu 1.12.2020. https://www.icst.pku.edu.cn/cpdp/sjzy/index.htm

Mathpix. 2020. Viitattu 4.12.2020. https://mathpix.com/

Pakin, S. 2020. The Comprehensive LaTeX Symbol List. Viitattu 3.12.2020. http://tug.ctan.org/info/symbols/comprehensive/symbols-a4.pdf

Purohit, A. & Chauhan, S. S. 2016. A Literature Survey on Handwritten Character Recognition. IJCSIT) International Journal of Computer Science and Information Technologies, 7(1), 1-5.

SimpleOCR. 2020. Viitattu 4.12.2020. https://www.simpleocr.com/

Smith, R. 2016. Tesseract Blends Old and New OCR Technology. DAS2016 Tutorial – Santorini – Greece. Viitattu 30.11.2020. https://github.com/tesseract-ocr/docs/tree/master/das_tutorial2016

Tapia, E. 2005. Understanding Mathematics: A System for the Recognition of On-Line Handwritten Mathematical Expressions. Väitöskirja. Viitattu 3.12.2020. http://dx.doi.org/10.17169/refubium-16590

Wu, W., Li, F., Kong, J., Hou, L. & Zhu, B. 2006. A Bottom-Up OCR System for Mathematical Formulas Recognition. In: Huang DS., Li K., Irwin G.W. (eds) Intelligent Computing. ICIC 2006. Lecture Notes in Computer Science, vol 4113. Springer, Berlin, Heidelberg.

Zhang, J., Du, J., Zhang, S., Liu, D., Hu, Y., Hu, J., Wei, S. & Dai, L. 2017. Watch, attend and parse: An end-to-end neural network based approach to handwritten mathematical expression recognition. Pattern Recognition, 71, 196-206.

# Liitteet

Liite 1. Säikeistyksen toteutus merkkien tunnistukselle

```
def ocr():
    dx = s['dx']
    dy = s['dy']
    origH = d['image_origH']
    origW = d['image_origW']
    # OCR glyph
    for glyph in d['glyphs']:
        x, y, w, h = glyph["x"], glyph["y"], glyph["w"], glyph["h"]
        line = algorithm.get_parent(glyph)

        if y > dy and x > dx and y + h + dy < origH and x + w + dx < origW:
            roi = d["image"][y - dy:(y + h) + dy, x - dx:(x + w) + dx]
            cv2.rectangle(d["original"], (x - dx, y - dy),
                          (x + w + dx, y + h + dy), (0, 255, 0), 2)
            glyph['symbol_input'] = [roi, [100], [10], [3]]
        else:
            roi = d["image"][y:(y + h), x:(x + w)]
            cv2.rectangle(d["original"], (x, y), (x + w, y + h), (0, 255, 0), 2)
            glyph['symbol_input'] = [roi, [100], [10], [3]]
    data =[]
    for glyph in d['glyphs']:
        data.append(glyph['symbol_input'])

    temp = worker_pool(data)

    for glyph, t in zip(d['glyphs'], temp):
        if 'symbol2' in glyph.keys():
            glyph['symbol'] = glyph['symbol2']
        else:
            glyph['symbol'] = t[0]['symbol']


def worker_pool(arg):
    with Pool(16) as p:
        return p.map(algorithm.img_to_string, arg)
```