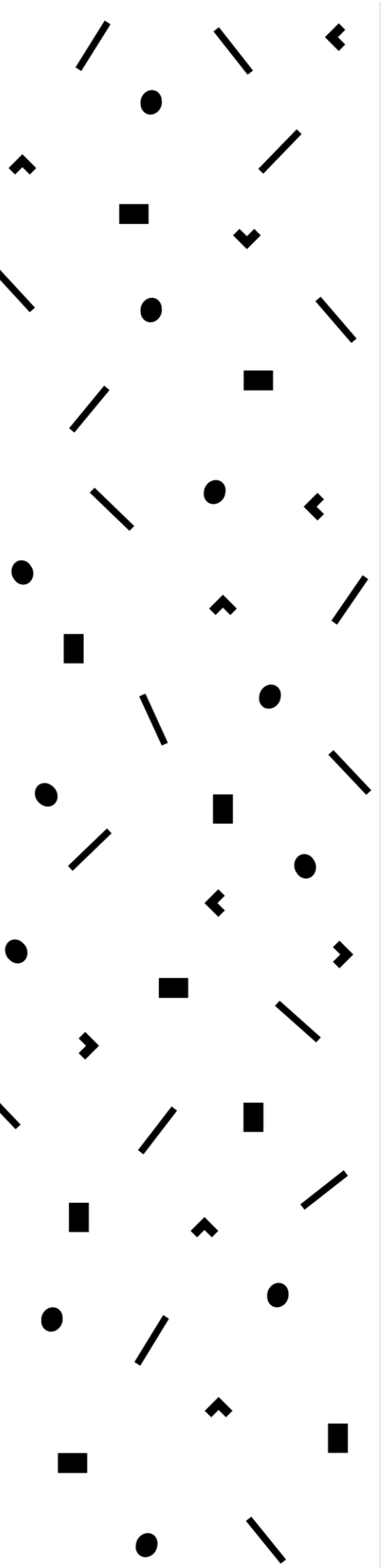


LAMK Lahden ammattikorkeakoulu
Lahti University of Applied Sciences

Tekstin tunnistus dokumentista

LAHDEN
AMMATTIKORKEAKOULU
Tekniikan ala
Tieto- ja viestintätekniikka
Ohjelmistotekniikka
Opinnäytetyö
Kevät 2018
Aki Puurula

Lahden ammattikorkeakoulu
Tieto- ja viestintätekniikka

PUURULA, AKI: Tekstin tunnistus dokumentista

Ohjelmistotekniikan opinnäytetyö, 46 sivua

Kevät 2018

TIIVISTELMÄ

SuperApp yrityksessä opinnäytetyötä varten suoritetun asiakasprojektin tavoitteena oli luoda ohjelma, joka kykenee lukemaan lomakkeiden kuvista tietoa koneluettavaan ja muokattavaan muotoon. Vaihtoehtoja työn toteuttamiseen oli luoda oma koneälymalli tai keskittyä johonkin valmiiseen kirjastoon sitä varten.

Työn tuloksena oli lopulta ohjelmisto, joka koostuu useammasta osasta. Yhtenä osista on iPhonella toimiva puhelinapplikaatio lomakkeiden kohdistamista ja kuvaamista varten sekä selaimella toimiva skripti, jolla lomakkeille voidaan luoda ulkoasutiedosto. Tärkeimpänä osana, johon opinnäytetyö myös keskittyy, on C++:lla toteutettu lomakkeet lukeva konsoliohjelma, joka kykenee tuottamaan JSON-muotoisen tiedoston sille syötetystä kuvasta.

Luotuun ohjelmaan jäi vielä parantamisen varaa saatujen tuloksien osalta, sillä niiden laatu vaihteli heikosta hyvään, riippuen lomakkeiden kuvien laadusta. Työ jäi kuitenkin tältä osin kesken työharjoittelun päättymisen takia, joten syvemmälle laadun parantamiseen ei ehditty menemään.

Asiasanat: Tesseract, OpenCV, OCR

Lahti University of Applied Sciences

Degree Programme in Information Technology

PUURULA, AKI: Text recognition from a document

Thesis in Software Engineering, 46 pages

Spring 2018

ABSTRACT

The thesis was commissioned by the SuperApp company. The objective of the customer project was to create a program that could convert text from pictures of forms into editable form. The options for the project were to either create a custom machine learning model or take the time to focus on a pre-made library that can already handle reading.

At the end of the project, the program consisted of three parts. The first part is an iPhone app, which has a guide to help focus the camera on the form and take a photo of it. The second part is a script running in browsers to help create a layout file for the forms. The final part, which this thesis focuses on, is the console application, written in C++, which actually reads the images fed to it and outputs the data in a JSON file.

As to the success of the project, the program needs to be improved as the quality varied from weak to good depending on the quality of the pictures fed to the program. The project remained unfinished at this point due to the internship ending, so there was no time to get further into improving the quality.

Key words: Tesseract, OpenCV, OCR

SISÄLLYS

1 JOHDANTO

Nykyään lomakkeita ja muita dokumentteja tallennetaan yleensä sähköisesti joko luonnon tai tilan säästämiseksi, tai paremman säilyvyyden takia. Kuitenkin paperisiakin dokumentteja tehdään vielä, senkin lisäksi, että vanhempiakin digitoimattomia dokumentteja löytyy vielä arkistoista. Digitoimattomien dokumenttien muuttaminen digitaaliseen muotoon on työlästä, ja vaikka ne voidaankin vain skannata, olisi parempi myös saada niiden sisältämä teksti digitaaliseksi.

Tätä tarkoitusta varten toteutettiin asiakasprojekti SuperApp-nimisessä yrityksessä, jonka kotipaikkana on Lahti. Yritys on ohjelmistotalo, joka tuottaa pääasiassa verkko- ja puhelinsovelluksia sekä erilaisia prototyyppejä.

SuperApp perustettiin vuonna 2015, ja sen liikevaihto on ollut nopeassa kasvussa. Toukokuussa 2016 yrityksen liikevaihto oli 79 000 euroa, mutta samaan aikaan vuotta myöhemmin liikevaihto on kasvanut 294 000 euroon. Yritys toimii Suomessa, jossa sen asiakaskuntaan kuuluvat erilaiset pörssiyritykset, PK-yritykset ja startupit. (Fonecta 2018; SuperApp 2018.)

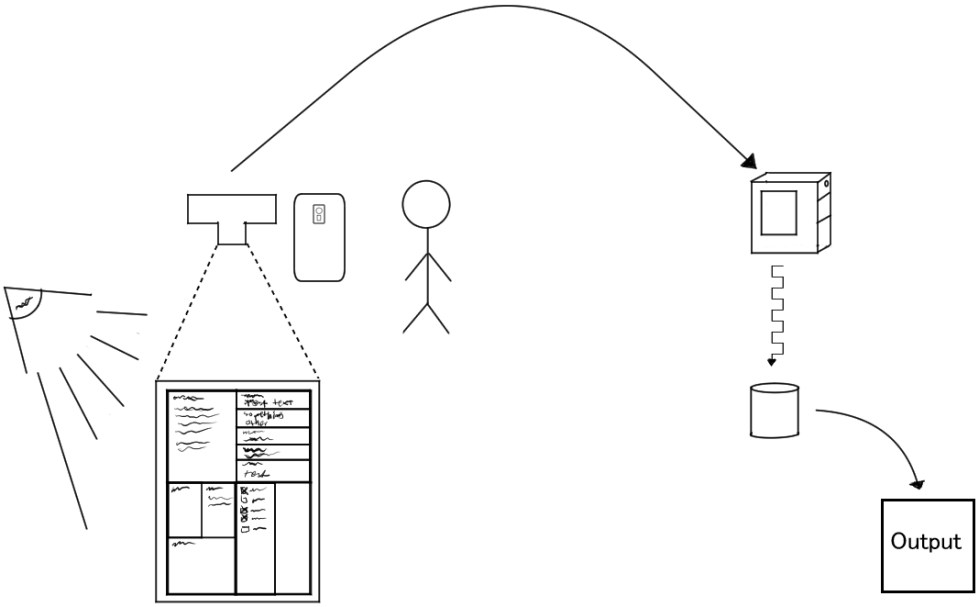
Asiakasprojektin tarkoituksena oli lomakkeen sisältämän tekstin muuttaminen digitaaliseen muotoon niin, että tiedot olisivat helposti käsiteltävissä. Tätä varten piti löytää sopiva kirjasto tekstin lukemista varten ja tutkia, millaisen esikäsittelyn luettavat kuvat tarvitsevat. Tämä lomakkeesta tekstiä lukeva ohjelma, sekä sen keskeiset komponentit ovat tämän työn tarkasteltavana kohteena.

2 OHJELMAN TOIMINTAYMPÄRISTÖ

2.1 Tarvittavat toimenpiteet ja laitteisto

Ohjelmiston käyttöä varten käyttäjä tarvitsee puhelimen, jolla voidaan ottaa kuva dokumentista sitä varten luodulla sovelluksella. Sovelluksessa on rajattu alue, jonka sisälle käyttäjän tulisi kohdistaa dokumentti. Itse ohjelma on palvelimella, joka tulostaa tunnistetun tekstin.

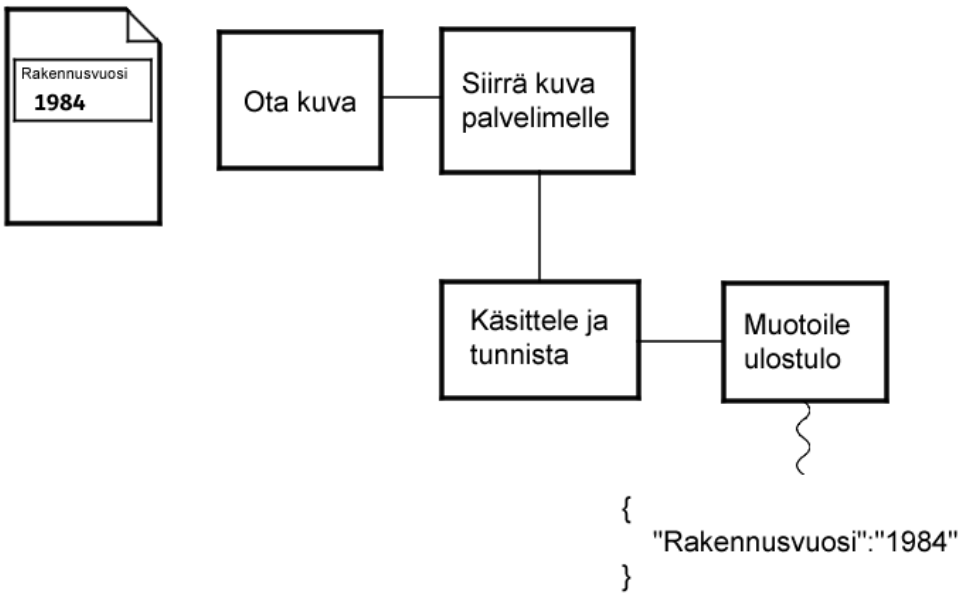
Jotta ohjelmisto lukee dokumenttia oikein, tulisi siinä olla selkeät rajat kenttien ympärillä. Tämä sitä varten, että myös tarvittavat kohdistuspisteet voidaan löytää lomakekenttien sijoittelua varten.



KUVIO 1. Toimintaympäristö

Toimintaympäristö koostuu lomakkeesta, joka vaatii myös sopivan valaistuksen, kuten kuvio 1 osoittaa. Käyttäjänä on henkilö, jolla on iPhone varustettuna sille suunnitellulla sovelluksella kuvan ottamiseen. Puhelimella tulee olla mahdollista yhdistää Internetiin, jonka kautta

voidaan siirtää kuva palvelimelle. Itse tekstin tunnistava ohjelma on palvelinkoneella, ja tämä ohjelma tuottaa JSON-muotoisen tiedoston.



KUVIO 2. Prosessin kulku

Kuvio 2 osoittaa prosessin eri vaiheet ja tapahtumat kuvauksesta JSON-tiedostoon. Käyttäjä ottaa ensin lomakkeesta kuvan, joka siirretään palvelimelle. Tämän jälkeen palvelimella suoritettu tekstin tunnistusohjelma lukee lomakkeesta tekstin ja muotoilee saadun tiedon käyttökelpoiseksi JSON-tiedostoksi.

2.2 Asiakasvaatimukset

Sovelluksen haluttiin pystyvän lukemaan asiakkaan antamilta lomakepohjilta tekstin siten, että se olisi jäsenneltynä ulostulossa kenttäkohtaisesti. Tämä edellytti tietojen tunnistusta muun tekstin joukosta, ja mahdollisuutta yhdistää otsikkotiedot niiden alle kuuluvien muiden tietojen kanssa.

Yhtiön rekisteröimispäivä		Hoitolainat / pvm		Rahoituslainat / pvm		Lainoista valtion asuntolainaa	
Valtion asuntolainaa ☐ Talokohtainen ☐ Henkilökohtainen		Käyttö- ja luovutusrajoitukset		Voimassa olevan yhtiöjärj. pvm		Osakekirjojen painaminen	
Asunnot kpl 12	Pinta-ala yhteensä 464	Osakkeiden lukumäärä 464	Liike ja muut huon. 5	Pinta-ala yhteensä 100.0	Osakkeiden lukum.	Huoneistoista yhtiön omistuksessa Asuntoja Liikehuon. 5 Muita	
Kaavoitetut autopaikat	Toteutetut autopaikat 10	Autotalli / hallipaikat	Muut autopaikat	Näistä osakkeina	Yhtiön omistuksessa		
Yhteiskäytössä olevat tilat ☒ Sauna ☐ Mankeli ☒ Pesutupa ☒ Ulkoiluvälinev.		☐ Askarteluhuone ☐	☐ ☐	Kiinteistönhoitojärjestelmä ☐ Väestönsuoja ☐ Ulma-allas			
Vakuutusyhtiö		Kiinteistövakuutukset		Muut vakuutukset			
Tontti Vuokra	Pinta-ala 1451	Rakennusten lkm 1	Porraskäytävien lkm 3	Kerrosluku 2	Kerrosala	Huoneistoala 464	Tilavuus 2670
Vuokra-aika 50 vuotta	Vuosivuokra	Vuokra-aika päättyy 31.12.2040	Vuokrantarkistusperuste		Vuokranantaja		Käyttämätön rak. oik.
Talotyyppi Kerrostalo		Valmistumisaika		Rakennusmateriaali Puu		Kattotyyppi ja kate Harjakatto ja pelti	
Lämmitysjärjestelmä ja lämmönjakotapa Kaukolämmitys		Ilmanvaihtojärjestelmä painovoimainen		Antennijärjestelmä Yhteisantenni		Hissit Ei	
Kulutustiedot vuodelta		Lämmönkulutus kWh / m3 / v		Vedenkulutus l / hlö / vrk		Sähkönkulutus kWh / m3 / v	

KUVIO 3. Esimerkki tunnistettavan lomakkeen sisällöstä

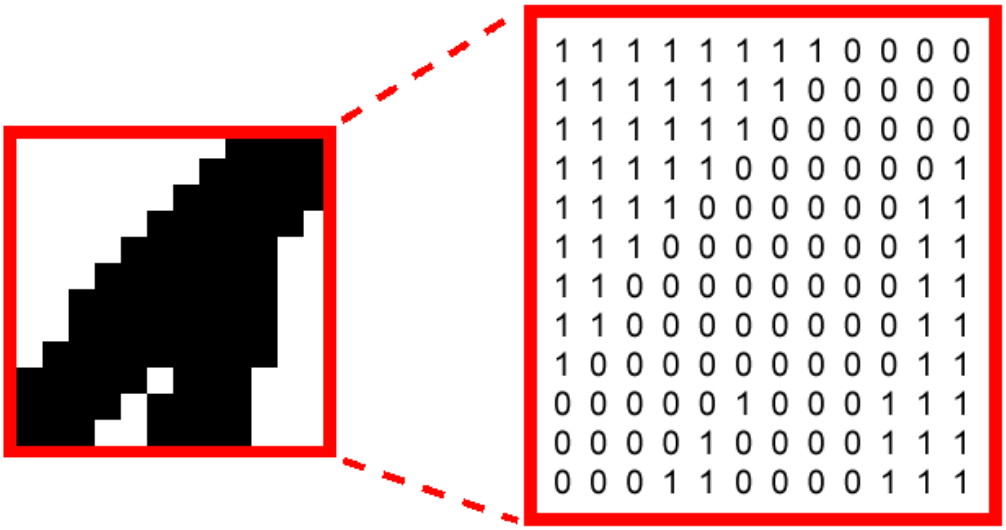
Kuviossa 3 nähdään, miten kirjasinkoko ja -tyyppi eroavat toisistaan tunnistettavassa lomakkeessa. Otsikoiden ollessa huomattavasti pienemmät kuin niiden alla olevat tiedot haluttiin ne erotella toisistaan tämän kokoeron sekä kirjasintyypin mukaan.

3 TEORIA

3.1 Kuvaformaatit

Kuvaformaatteja on useita eri tyyppisiä, joista jotkin Holmesin (2013) mukaan sopivat paremmin tiettyihin käyttökohteisiin kuin toiset. Esimerkiksi PNG-muotoiset kuvat soveltuvat hyvin verkkosivuille pienen kokonsa ja laatunsa takia, ja vaikkakin niitä käytetään laajalti valokuvauksessa, ovat esimerkiksi TIFF-muotoiset kuvat siihen sopivampia. JPEG-kuvat PNG-kuvien lailla ovat myös sopivia verkkosivuille, mutta pakkauksensa takia ne eivät ole sopivimpia kuvien käsittelyyn. Toisin kuin JPEG, PNG-kuvat tukevat läpinäkyvyyttä. (Holmes 2013.)

Kuvatiedostot alkavat otsikkolohkolla, johon sisältyy usein tiedostomuodon allekirjoitus, kuvan koko, ja mahdollisesti myös määritelmä siitä, montako värikanavaa on käytössä. TIFF-kuvien tapauksessa nämä määritellään myöhemmin tiedostossa erilaisin tunnistein. (Maischein 1996.)



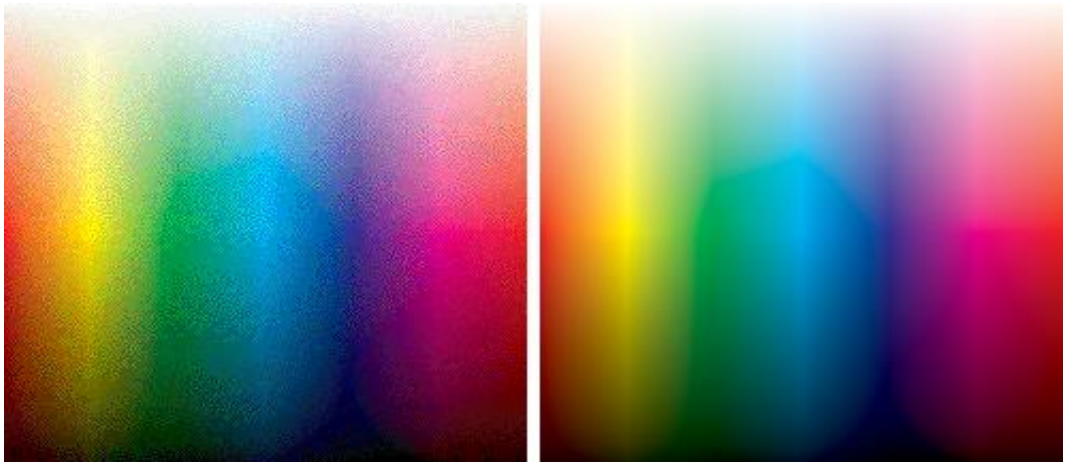
KUVIO 4. Binäärikuva

Yksinkertaisimmat digitaaliset kuvat ovat mustavalkoiset niin sanotut binäärikuvat, joissa jokaista pikseliä kuvaa tasan yksi bitti. Kuten kuvio 4 osoittaa, bitin arvo määrittää, onko pikseli musta vai valkoinen. Tässä tapauksessa pikseli on musta bitin ollessa nolla ja valkoinen sen ollessa yksi. Binäärikuvasta päästään harmaasävyisiin kuviin lisäämällä pikseliä kuvaavien bittien määrää. (Rochester Institute of Technology 2018.)

KUVIO 5. Binääri- ja harmaasävykuva

Esimerkiksi kuviossa 5 on vasemmalla binäärikuva, jossa on vain mustaa ja valkoista. Oikealla puolella taas on harmaasävyinen kuva, jossa jokaista pikseliä kuvaa kahdeksan bittiä, eli yksi tavu. Tässä tapauksessa väriä kuvaavia arvoja on 256, joka on myös harmaan sävyjen määrä.

Mustavalko- ja harmaasävykuvissa on vain yksi kanava pikselin sävyn määrittämiselle. Pikselin kahdeksan bitin ei kuitenkaan tarvitse kuvata vain harmaan sävyjä. Esimerkiksi Fultonin (2018) mukaan kuvalle voidaan luoda 256 värin väripaletti, jolloin voidaan luoda värikuvia.



KUVIO 6. Väripalettia käyttävän kuvan ja 24-bittisen värikuvan ero

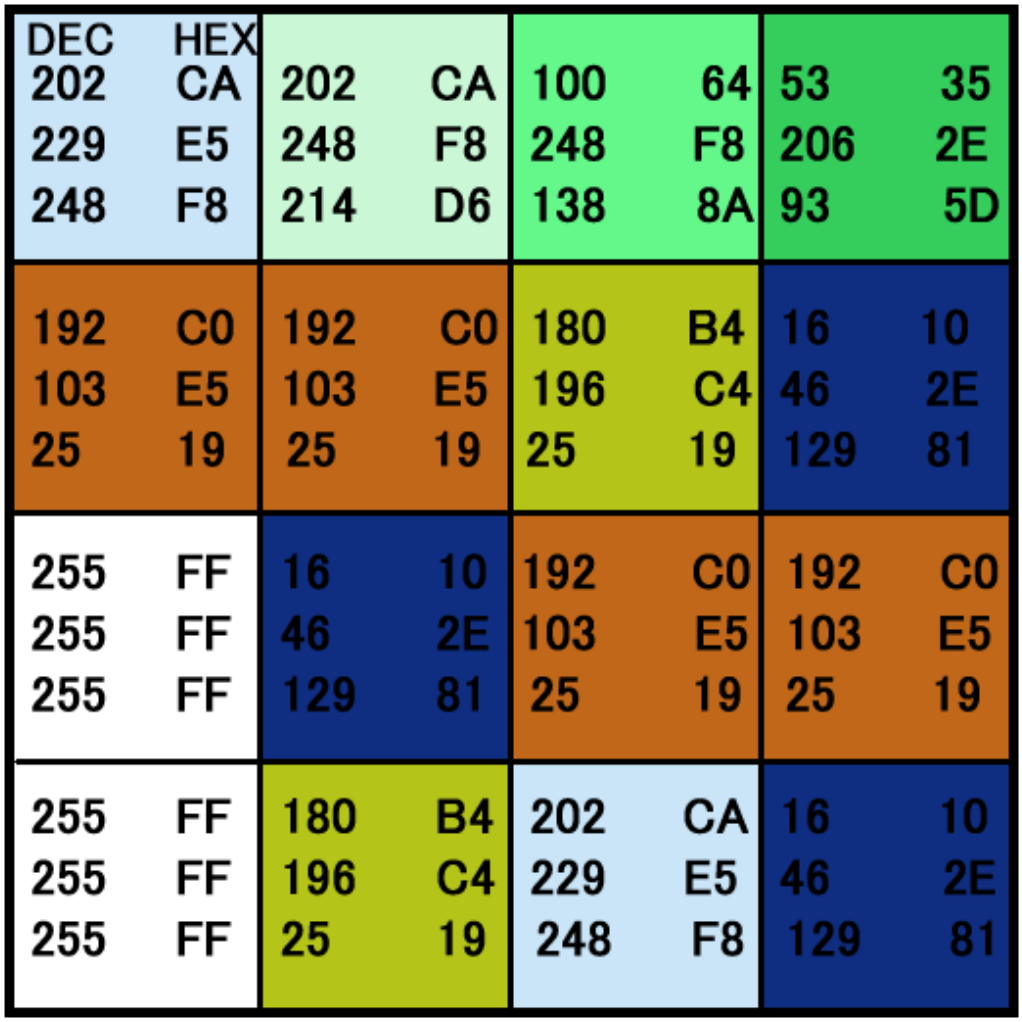
Kahdeksan bitin värikuvia käytettäessä kuitenkin tulee ottaa huomioon, että ne eivät kykene näyttämään korkealaatuisia ja moniväriisiä kuvia tarkasti. Kuten kuviosta 6 huomaa, on vasemmalla puolella olevassa väripalettia käyttävässä kuvassa selkeä laadullinen ero oikean puoleiseen kuvaan, jossa pelkästään punaisella, vihreällä ja sinisellä on omat värikanavansa, joita jokaista kuvaa yksi tavu. Kahdeksan bitin väripalettit soveltuvat siis paremmin kuville, joissa on vähän värejä.

24-bittisissä värikuvissa, joita käytetään laajemmin, on jokaiselle pikselille varattu kolme tavua. Tällöin käytössä voi olla kolme värikanavaa RGB-muotoisille kuville. Jokaisella värikanavalla voi olla arvo nollan ja 255 välillä, jolloin 24-bittiset kuvat voivat näyttää yli 16 miljoonaa väriä. Fultonin (2018) listalla seuraavana ovat 32-bittiset kuvat, jotka käyttävät CMYK-kanavia, lyhenteenä sanoista:

- cyan
- magenta
- yellow
- key (black).

Morrisonin (2018) mukaan, myös RGB-kuvat, joihin on lisättynä alfa-kanava läpinäkyvyydelle, ovat 32-bittisiä. Joillakin kuvatiedostoilla on tuki

48 bitille, jolloin RGB-kuvassa jokaiselle värikanavalle on varattu 2 tavua, mahdollistaen yli 281 biljoonaa eri väriä (Fulton 2018).

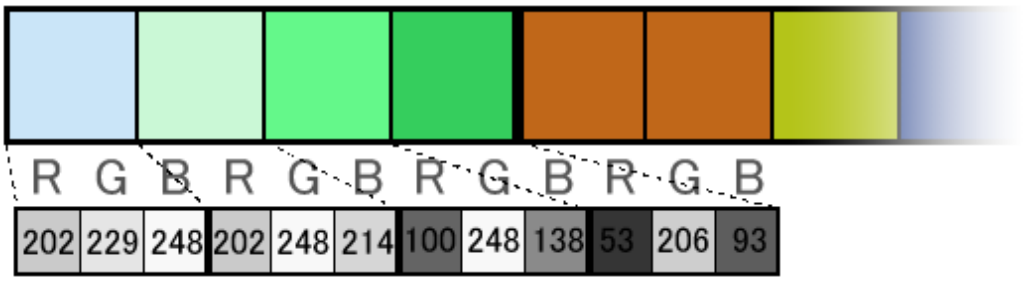


KUVIO 7. Esimerkkikuva väriarvoineen

Kuvio 7 sisältää suurennetun 24-bittisen esimerkkikuvan, jossa jokainen laatikko kuvaa yksittäistä pikseliä. Jokaisen pikselin väriarvot ovat lueteltuna desimaaleina ja heksadesimaaleina niiden päällä ylhäältä alaspäin järjestyksessä: punainen, vihreä, sininen. Kuva on siis RGB-muotoinen ja esitettynä kaksiulotteisessa muodossa, jossa ne normaalisti nähdään.

KUVIO 8. Pikselit tiedostossa yksinkertaistetusti

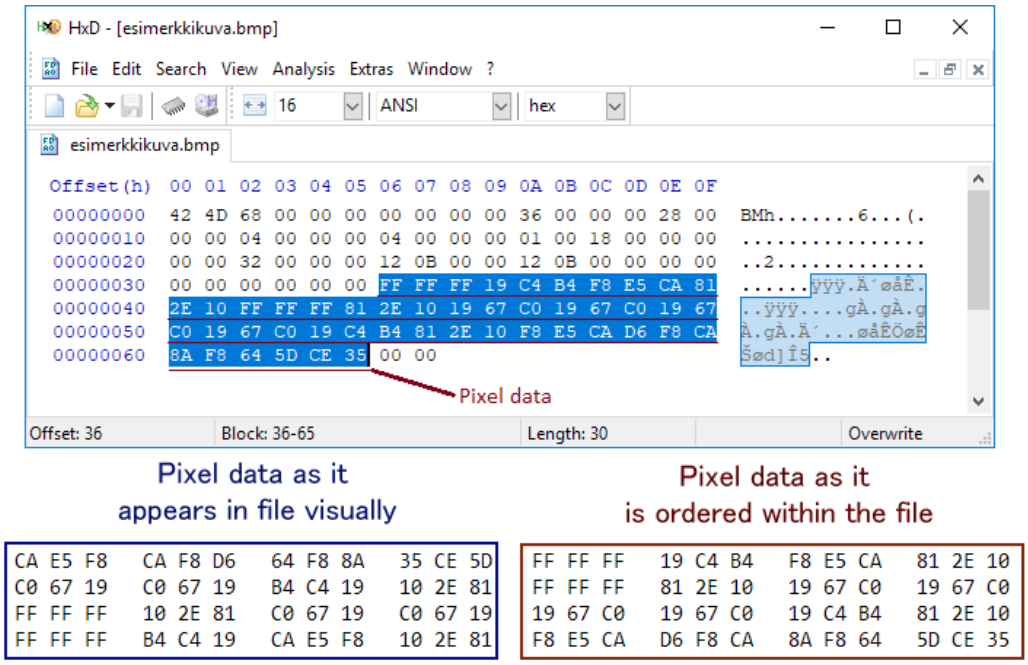
Kuvan sisältämä tieto pikseleistä ei ole suoraan kaksiulotteista, vaan sisäisesti pikselit saattavat olla peräkkäin jonossa. Kuva kyetään kuitenkin näyttämään oikein kaksiulotteisena tiedoston otsikossa tai tunnisteissa olevien määritelmien avulla. Kuviossa 8 näkyy esimerkki siitä, miten pikselit voisivat näyttää tiedostossa. Täytyy kuitenkin ottaa huomioon, että oikeasti pikselitietojan pakkauksesta ja suodatuksesta riippuen, ne todennäköisesti eivät esiinny suoraan kuvion 8 näyttämällä tavalla. Tiedostomuodosta riippuen pakkaus- tai suodatusmuoto käyvät ilmi esimerkiksi tiedoston otsikosta, jonka sisältämien muiden tietojen avulla kuva kyetään avaamaan oikealla tavalla (Maischein 1996). Kuitenkin esimerkiksi BMP-muodossa pikselit usein tallennetaan lähes suoraan sellaisenaan, vaikkakin hiukan eri järjestyksessä.



KUVIO 9. Kuvan värien arvot muistissa

Kun kuvaa luetaan muistista, esimerkiksi johonkin ohjelmaan, voi pikseleitä lukiessa tuloksena olla pitkä sarja numeroita, kuten esimerkiksi kuviossa 9. Näiden numerosarjojen tarkoituksena on siis kuvata eri värikanavia, ja kuviossa 9 ne ovat järjestyksessä punainen, vihreä ja sininen. Joissakin tapauksissa järjestys voi kuitenkin olla päinvastainen,

jonka lisäksi kuvassa saattaa olla useampia kanavia. Näissä tapauksissa, esimerkiksi kun halutaan kerätä pikselit järjestyksessä muistiin käsittelyä varten, on parempi varmistaa mihin järjestykseen värikanavat on luettu.



KUVIO 10. Kuva heksaeditorissa

Kuviossa 10 on esimerkki kuvion 7 esittämästä kuvasta BMP-muodossa heksaeditorissa. Kuva on Photoshopilla tallennettu 24-bittisenä Windowsin käyttämään muotoon pakkaamattomana. Kuviosta nähdään pikseleiden väritietojen sijainti ja sekä niiden järjestys, joka poikkeaa siitä miltä kuva visuaalisesti näyttää. BMP-tiedostossa värikanavat ovat järjesteltynä RGB-muodon sijaan takaperin BGR-muotoon. Tämän lisäksi kuvan viimeinen rivi on tallennettuna ensimmäisenä, kun taas ensimmäinen rivi on viimeisenä. Kuviosta 10 voidaan myös nähdä BMP-tiedoston allekirjoitus, joka on BM, heti kohdista 0-1. Heksamuodossa tämä on esitetty luvuilla 42 4D.

3.1.1 Portable Network Graphics

PNG-muoto on häviötön, tarkoittaen sitä, että tähän muotoon tallennettaessa tietoa ei häviä. Häviöttömänä muotona PNG-muotoon voidaan kuitenkin pakata ja suodattaa dataa, jolloin kyetään tallentamaan kuvatiedosto entistäkin pienempään muotoon. Lisäksi PNG tukee läpinäkyvyyttä, joka tekee kyseisestä muodosta erittäin sopivan verkkosivuille, jos halutaan tarkkoja kuvia. (Maischein 1996; Holmes 2013.)

3.1.2 Joint Photographic Experts Group

Joint Photographic Experts Group, lyhennettynä JPEG on toinen yleisimmistä tallennusmuodoista. JPEG itsessään yleensä viittaa käytettyihin pakkausalgoritmeihin, kun taas JFIF viittaa tiedostomuotoon, johon JPEG-pakatut kuvat yleensä tallennetaan. Tämän tiedostomuodon hyvä puoli on sen tiedostojen pieni koko. Pakkaaminen tähän muotoon kuitenkin tarkoittaa, että kuvat menettävät tarkkuutta, vaikka pakkauksen laatuun kyetään vaikuttamaan ohjelmasta riippuen. (Lilley 1996; Holmes 2013.)

3.1.3 Tagged Image File Format

TIFF on häviötön muoto, joka ei myöskään pakkaa kuvia pienempiin tiedostoihin. Tämän seurauksena tiedostot ovat myös isompia kuin muiden edellä mainittujen muotojen. Tästä syystä TIFF-muotoiset kuvat eivät ole suositeltavia verkkosivuille, mutta ne soveltuvat hyvin kuvankäsittelyyn juurikin niiden tarkkuuden takia. (Holmes 2013.)

3.2 OpenCV

OpenCV eli Open Source Computer Vision Library on konenäkökirjasto, joka on suunniteltu reaaliajassa toimiviin ohjelmiin, kuten esimerkiksi viivakoodien lukeminen. Kirjastoon löytyy API-rajapinnat C++-, Python- ja

Java-ohjelmointikielille, minkä lisäksi se tukee Windows-, Linux-, Mac OS-, iOs- ja Android-alustoja. (OpenCV 2018a.)

OpenCV käyttää modulaarista rakennetta, mikä tarkoittaa sitä, että pakkaus sisältää useita moduuleita eri käyttötarkoituksia varten. Joitakin moduuleita ovat mm. core, joka sisältää määritelmät perus tietorakenteille sekä perusfunktiot, joita muut moduulit käyttävät, video, jota käytetään videon analysointiin, gpu grafiikkaprosessorilla kiihdytettyihin algoritmeihin muista moduuleista, ja objdetect esineiden ja asioiden tunnistamiseen. Projektin kannalta tärkeimmät moduulit näistä ovat core ja imgproc, jota käytetään kuvien käsittelyyn. (OpenCV 2014.)

3.2.1 Muistin hallinta

OpenCV:ssä eri tietorakenteet sisältävät purkajia, jotka vapauttavat niiden käyttämän muistin tarvittaessa. Tämä siis tarkoittaa sitä, että jos tieto on jaettu useamman ilmentymän kesken, pidetään muisti varattuna. Jos yksi useasta samaan tietoon viittaavasta ilmentymästä poistetaan, purkaja vain laskee viittauksien määrää ylläpitävää laskuria yhdellä. Muisti vapautetaan vain ja ainoastaan silloin, kun viittauksien määrä laskee nollaan. Mikään muu rakenne ei siis viittaa enää kyseiseen puskuriin.

```
// create a big 8Mb matrix
Mat A(1000, 1000, CV_64F);

// create another header for the same matrix;
// this is an instant operation, regardless of the matrix size.
Mat B = A;
// create another header for the 3-rd row of A; no data is copied either
Mat C = B.row(3);
// now create a separate copy of the matrix
Mat D = B.clone();
// copy the 5-th row of B to C, that is, copy the 5-th row of A
// to the 3-rd row of A.
B.row(5).copyTo(C);
// now let A and D share the data; after that the modified version
// of A is still referenced by B and C.
A = D;
// now make B an empty matrix (which references no memory buffers),
// but the modified version of A will still be referenced by C,
// despite that C is just a single row of the original A
B.release();

// finally, make a full copy of C. As a result, the big modified
// matrix will be deallocated, since it is not referenced by anyone
C = C.clone();
```

KUVIO 11. Esimerkki muistin hallinnasta

Kuviossa 11 nähdään esimerkki siitä, miten muistin hallinta toimii Mat-tietorakenteen kohdalla. Jos Mat-rakenne halutaan kopioida toiseksi erilliseksi rakenteeksi, tulisi käyttää metodia copyTo(). Ilman sitä mitään dataa ei kopioida vaan sen sijaan viittauslaskuria kasvatetaan osoittamaan tiedon omistajien määrää. (OpenCV 2014.)

Muistin automaattisen vapauttamisen lisäksi, OpenCV kykenee myös varaamaan muistia funktion ulostuloparametreille. Jos funktio sisältää edes yhden sisääntulo- ja ulostulotaulukon, varataan ulostulotaulukoille muisti automaattisesti sisääntulojen koon ja tyypin mukaan. (OpenCV 2014.)

3.2.2 Koon muutokset

OpenCV tarjoaa muutaman erilaisen tavan muuttaa kuvan kokoa. Yksi tapa on käyttää kuvapyramideja, joita varten OpenCV tarjoaa pyrDown()- ja pyrUp()-metodit. PyrUp()-metodia käytettäessä kuvaa voidaan

suurentaa, ja oletuksena sen tuloksena on kuva, jonka mittasuhteet ovat kaksinkertaiset alkuperäiseen verrattuna. Lopuksi kuvaa vielä sumennetaan. PyrDown()-metodi taas toimii päinvastaisesti, eli se ensin sumentaa kuvaa, minkä jälkeen kuvan mittasuhteet pienennetään puoleen. (OpenCV 2017h.)

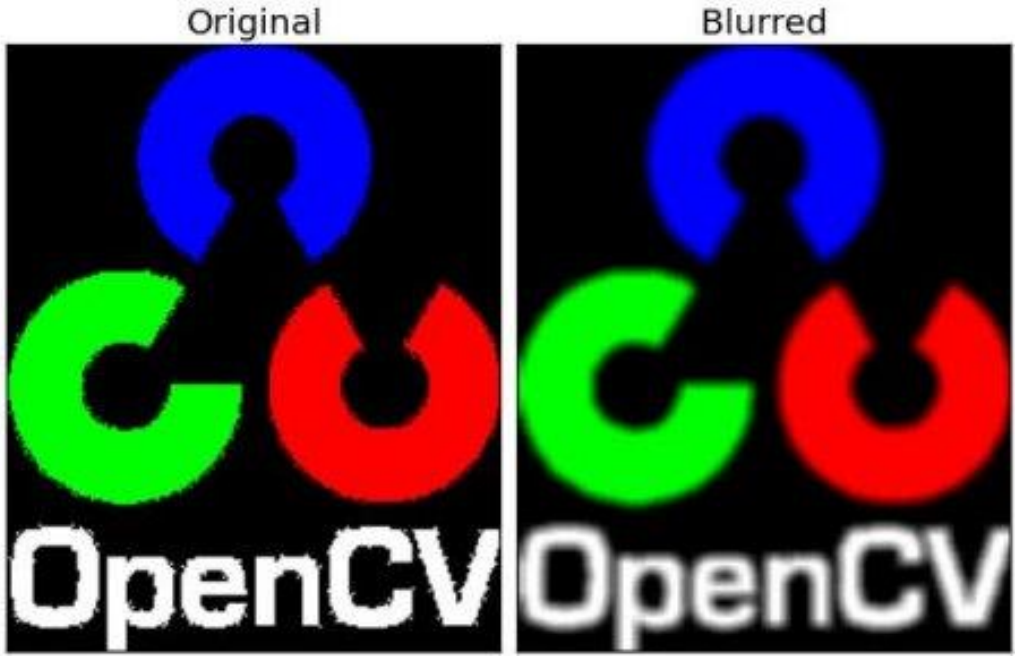
OpenCV:llä on kuitenkin mahdollisuus myös käyttää geometrisia muunnoksia, joiden joukosta löytyy resize()-metodi. Tämän avulla kuvaa voidaan pienentää tai suurentaa haluttuun kokoon ilman että sen mittasuhteiden tarvitsisi olla puolet tai kaksinkertaiset alkuperäiseen verrattuna. Tämän lisäksi voidaan vielä määrittää algoritmi interpolointia varten. (OpenCV 2018g.)

3.2.3 Suodattimet

Sen lisäksi, että OpenCV:llä kyetään muuttamaan kuvan kokoa, voidaan sillä suorittaa myös muita operaatioita kuvan laadun parantamiseksi tai muuhun tarkoitukseen.

Kuvan pehmennys

Kuvan pehmentämistä varten on OpenCV:llä useita eri tapoja. Yleisesti kuvan pehmennys toteutetaan yhdistämällä kuva alipäästösuodattimen jyvällä. Tämä tekniikka poistaa kohinaa ja reunoja kuvasta. OpenCV tarjoaa neljä erilaista pehmennystekniikkaa. (OpenCV 2017k.)

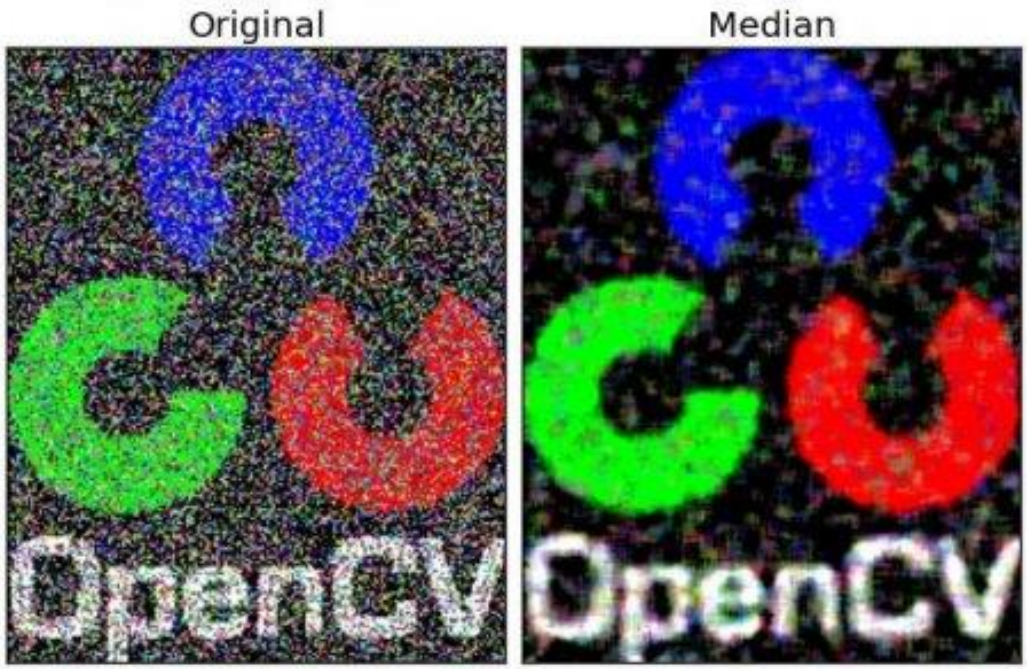


KUVIO 12. Keskiarvoistettu pehmennys (OpenCV 2017a)

Yksi näistä on keskiarvoistaminen, jolla siis lasketaan pikseleiden keskiarvo laatikon muotoisen jyvän alta. Keskimmäisen pikselin arvo korvataan sitten tällä keskiarvolla. Esimerkki tämän tyyppisestä pehmennyksestä on kuviossa 12 (OpenCV 2017k.)

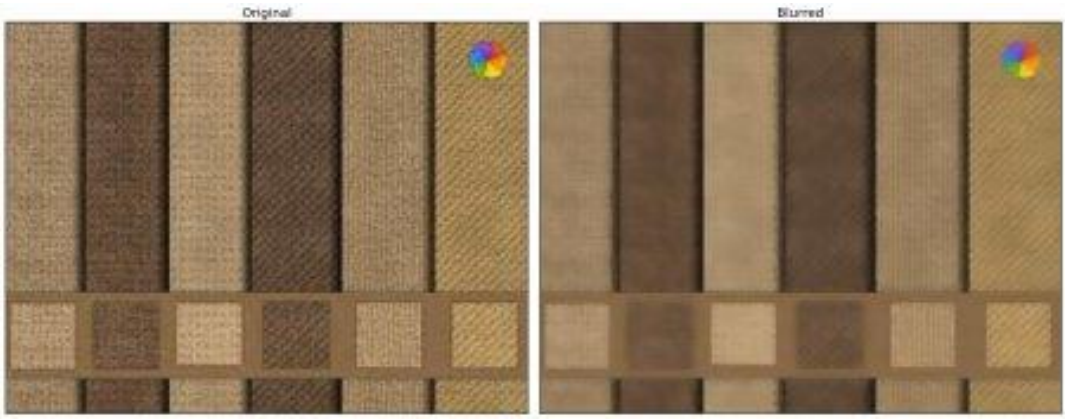
KUVIO 13. Gaussin pehmennys (OpenCV 2017e)

Gaussin pehmennyksessä, josta on esimerkki kuviossa 13, laatikkosuodattimeen sijaan käytetään gaussin jyvää, jossa leveys ja korkeus voivat olla eri kokoiset, kunhan niistä molemmat ovat positiivisia ja parittomia. Tämä on tehokas tapa poistamaan Gaussin kohinaa. (OpenCV 2017k.)



KUVIO 14. Mediaanipehmennys (OpenCV 2017j)

Mediaanipehmennyksellä lasketaan kaikkien jyvän alla olevien pikselien mediaani, jolla korvataan sitten keskimmäisen pikselin arvo. Jyvän koon tulee olla positiivinen ja pariton. Erona aiemmin mainituille menetelmille tässä on se, että jyvän keskimmäisen pikseli on aina jokin muu kuvan pikseli. Kuten kuvio 14 osoittaa, tällä menetelmällä kyetään pääsemään tehokkaasti eroon kohinasta. (OpenCV 2017k.)



KUVIO 15. Kaksipuoleinen suodatus (OpenCV 2017b)

Kaksipuoleinen suodatin käyttää hyödykseen Gaussin suodatinta, jossa se ottaa huomioon kaikki lähellä olevat pikselit. Tämän lisäksi kaksipuoleinen suodatin käyttää toista Gaussin suodatinta, joka varmistaa, että vain pikselit, joiden intensiteetti on lähellä keskimmäisen pikselin intensiteettiä, otetaan huomioon pehmennyksessä. Tällä tavoin kaksipuoleinen suodatin kykenee säilyttämään reunat, joilla yleensä on korkea intensiteettiero, kuten kuviosta 15 nähdään. (OpenCV 2017k.)

Mustavalko -muunnos

Mustavalkomuunnosta toteuttaessa kuva tulee ensin ladata muistiin suoraan harmaasävyiseksi tai jälkikäteen muuntamalla se cvtColor()-metodia käyttäen. Itse kuvan lataaminen toimii imread()-metodilla, johon liitettynä parametri IMREAD_GRAYSCALE lukee kuvan suoraan harmaasävyiseksi. Tästä kuva voidaan muokata kuva binäärikuvaksi käyttämällä raja-arvoja, joiden alta pikselin arvoksi asetetaan esimerkiksi musta, ja yläpuolella oleville arvoiksi valkoinen. (OpenCV 2017i.)

Kuluttaminen ja laajentaminen

OpenCV:llä voidaan laajentaa tai pienentää kirkkaita tai tummia alueita käyttämällä erode ja dilate-metodeja. Näillä operaatioilla voidaan poistaa kohinaa, eristää tai yhdistää kuvan elementtejä, tai löytää esimerkiksi reikiä kuvasta. Molemmat näistä käyttävät yleensä neliön tai ympyrän

muotoista jyvää, jolla tutkitaan alkuperäistä kuvaa. Jyvässä on myös ankkuripiste yleensä sen keskellä. Kuluttaessa jyvä laskee paikallisen minimin sen alueelta, ja korvaa ankkurin alla olevan pikselin minimillä, mikä saa aikaan vaaleiden alueiden kulumisen, tai pienenemisen. Toisinpäin, eli laajentaessa, jyvän alueelta lasketaan paikallinen maksimiarvo, jolla korvataan ankkurin alla oleva piste. Toisin kuin kuluttamalla, tällä menetelmällä kuvan vaaleat alueet taas laajenevat. (OpenCV 2017d.)

Kohinan poisto

Kohinaa voidaan poistaa OpenCV:llä aiemmin mainituilla menetelmillä, mutta OpenCV:stä löytyy tätä varten myös omat metodinsa:

- fastNlMeansDenoising
- fastNlMeansDenoisingColored
- fastNlMeansDenoisingMulti.

Nämä kaikki käyttävät ns. ei-paikallisen keskiarvon kohinan poistoa. Eli toisin kuin paikallinen keskiarvo, jossa tutkitaan kuvan tietyn alueen pikseleitä ja korvataan keskimmäinen pikseli niiden keskiarvolla, tutkitaan tässä pikseleitä kuvan koko alueelta. Pikseleitä korvataan etsimällä kuvasta kaikki samankaltaiset pikselit, ja lasketaan niiden keskiarvo. (Buades, Coll & Morel 2011.)

fastNlMeansDenoising-metodi on suunniteltu käyttettäväksi harmaasävyisillä kuvilla, kun taas fastNlMeansDenoisingColored on aiemmasta muokattu värikuville sopivaksi. Viimeisenä fastNlMeansDenoisingMulti on ensimmäisestä muokattu versio, joka toimii kuvasarjojen kanssa, jotka on otettu lyhyen ajan sisällä, kuten video. (OpenCV 2017c.)

Kaikkia näitä voidaan käyttää yhdessä, tai yksinään kuvien esikäsittelyyn. Ennen kuvan suurentamista voisi olla hyvä yrittää vähentää kohinaa mahdollisimman paljon, jonka jälkeen suurennusta toteuttaessa kohinan alla olevien pikseleiden määrä ei laajene, jolloin myös kohinaa poistaessa joudutaan tekemään enemmän työtä.

3.2.4 Levylle tallennus

OpenCV:ssä on esimerkiksi kuvien, vektoreiden ja matriisien säilyttämiseen Mat-luokka. Kuvien käsittely onnistuu tästä muodosta syöttämällä ne suoraan johonkin suodatinmetodiin, joka myös tuottaa kuvan matriisimuotoon. Itse kuvan tallennus levylle onnistuu imwrite()-metodilla, johon parametreiksi syötetään tiedoston sijainti ja nimi, sekä

tallennettava mat-rakenne. Tiedostomuoto riippuu nimessä määritellystä tiedostopäätteistä, joista tuettuja ovat aina mm. BMP ja DIB. JPG-muoto vaatii libjpeg-kirjaston, PNG vaatii libpng:n sekä TIFF vaatii libtiff-kirjaston. PNG-tiedostoja tallennettaessa on mahdollisuus tallentaa tiedosto alfa-kanavalla, jolloin kuvaan saadaan mukaan myös läpinäkyvyys. (OpenCV 2017g.)

3.3 Mikä on OCR

OCR eli optinen merkkien tunnistus on teknologia, jonka avulla voidaan muuntaa eri tyyppisiä dokumentteja, esimerkiksi skannattuja tai valokuvia, sellaiseen muotoon, että tekstiä kyetään muokkamaan. Käytännössä käyttäjä voisi ottaa puhelimellaan kuvan ja siirtää sen tietokoneelle. Kuvan sisältämää tekstiä ei kuitenkaan voida muokata millään tekstinkäsittelyohjelmalla. Tätä varten tarvitaan OCR-ohjelmisto, joka kykenee erittelemään kuvasta merkit ja tulostamaan ne käsiteltäväksi tekstiksi. (ABBYY 2018.)

Yksinkertaisesti sanottuna OCR-ohjelma lukee dokumentin tutkimalla sen tummia ja vaaleita alueita. Tämän avulla ohjelma kykenee yhdistämään dokumentin tekstin tunnettuihin merkkeihin, muuntaen sen näin digitaaliseen muotoon. (InStream 2014.)

3.4 Tesseract

Tesseract on OCR-moottori, joka kykenee tunnistamaan yli 100 eri kieltä. Tesseractin kehityksen aloitti Hewlett-Packard vuonna 1985, mutta vuodesta 2006 lähtien sitä on kehittänyt Google. (Google 2018b.)

Tesseractia voi käyttää sellaisenaan komentoriviohjelmana, tai sen voi sisällyttää kirjastona C tai C++-ohjelmaan. Osana Tesseractia on Leptonica-kirjasto, jonka PIX-luokkaan voidaan ladata kuvia. Tesseractia käytettäessä kirjastona, syötetään sille kuva tässä PIX-muodossa. Tulosteena Tesseract kykenee palauttamaan yksittäisiä merkkejä, sanoja tai kappaleita joko suoraan koneluettavana tekstinä, tai pdf, hOCR ja tsv-

muodoissa. Mm. hOCR-muodossa, jolloin ohjelma tuottaa hOCR-määritelmää seuraavaa XHTML-koodia, saadaan tekstille myös fontti, fonttikoko ja sijainti. (Google 2018a.)

Tesseract ei ole täydellinen OCR-kirjasto, koska se tuottaa myös virheellistä tekstiä, tai toisin sanottuna se saattaa lukea yksittäisen merkin useammaksi merkiksi, tai useamman merkin yhdeksi merkiksi. Esimerkkinä tästä voisi olla m-kirjaimen lukeminen rn-merkkiyhdistelmänä, tai toisinpäin. Vaikka Tesseract tekeekin virheitä, kyetään siihen vaikuttamaan syöttämällä sille mahdollisimman puhtaita ja selkeitä sivuja tai pyritään edes siistimään niitä etukäteen.

Vaikka kirjasto tarjoaa jo valmiiksi useita eri kieliä, on niitä mahdollista kouluttaa halutessa itse. Samalla kyetään myös kouluttamaan kirjastolle eri kirjasintyyppejä. (Google 2018b.)

3.4.1 Sivun osittelu

Riippuen kuvasta, josta teksti tunnistetaan, on hyödyllistä tuntea eri sivun osittelutavat. Tekstiä voi esiintyä yhdessä tai useammassa palstassa, ylhäältä alaspäin kirjoittettuna tai yksittäisinä sanoina tai lauseina sivun eri alueilla, kuten mm. lomakkeissa. Tesseractin eri osittelutavoilla voidaan määrittää siis miten se osittelee sivun sisältämän tekstin sen lukemista varten.

Ennen Tesseractin API:n alustusta, yksi sille määritettävistä valinnoista on sivun osittelu, joka kyetään asettamaan SetPageSegMode()-metodilla. Metodille annetaan parametrina PageSegMode-muotoinen muuttuja, johon on määriteltynä yksi osittelutavoista, jotka ovat esitettynä seuraavaksi.

PSM_OSD_ONLY

Tesseract pyrkii tunnistamaan dokumentin sisältämän kirjoitusjärjestelmän ja suunnan, muttei lue tekstiä (Smith 2010).

PSM_AUTO_OSD

Tesseract määrittää sivun osittelumenetelmän automaattisesti, minkä lisäksi se pyrkii tunnistamaan myös kirjoitusjärjestelmän, suunnan (Smith 2010).

PSM_AUTO_ONLY

Tesseract määrittää sivun osittelumenetelmän automaattisesti, mutta ilman kirjoitusjärjestelmän, suunnan ja tekstin tunnistusta (Smith 2010).

PSM_AUTO

Tesseract määrittää sivun osittelumenetelmän automaattisesti, sekä pyrkii tunnistamaan dokumentin sisältämän tekstin. Tähän ei kuitenkaan sisälly kirjoitusjärjestelmän tai suunnan tunnistusta (Smith 2010). Hyödyllinen, jos halutaan tunnistaa tekstiä, muttei olla varmoja sivun osittelusta, eikä välitetä kirjoitusjärjestelmästä tai tekstin suunnasta.

PSM_SINGLE_COLUMN

Tesseract tulkitsee kuvan sisältämän tekstin yksittäisenä sarakkeena (Smith 2010). Jos kuvassa on useampi sarake tekstiä, yhdistää Tesseract eri sarakkeiden rivit keskenään.

PSM_SINGLE_BLOCK_VERT_TEXT

Tesseract tulkitsee kuvan sisältämän tekstin yhtenäisenä palikkana ylhäältä alaspäin kirjoitettuna (Smith 2010). Kuitenkin niin, että tekstiä ei ole käännetty, vaan kirjaimet ovat vaakatasossa.

PSM_SINGLE_BLOCK

Tesseract tulkitsee kuvan sisältämän tekstin yhtenäisenä palikkana (Smith 2010).

PSM_SINGLE_LINE

Tesseract tulkitsee kuvan sisältämän tekstin yksittäisenä rivinä (Smith 2010). PSM_SINGLE_LINE soveltuu käytettäväksi yritettäessä lukea kuvia, jotka sisältävät yksittäisen rivin tekstiä.

PSM_SINGLE_WORD

Tesseract tulkitsee kuvan sisältämän tekstin yhtenä sanana (Smith 2010). PSM_SINGLE_WORD soveltuu käytettäväksi yritettäessä lukea yhden sanan sisältäviä kuvia.

PSM_CIRCLE_WORD

Tesseract tulkitsee kuvan sisältämän tekstin yhtenä ympyrän muotoon asetettuna sanana (Smith 2010).

PSM_SINGLE_CHAR

Tesseract tulkitsee kuvan sisältämän tekstin yksittäisenä kirjaimena. PSM_SINGLE_CHAR soveltuu käytettäväksi yritettäessä lukea yksittäisen merkin sisältämiä kuvia.

PSM_SPARSE_TEXT

Tesseract pyrkii lukemaan tekstiä mahdollisimman paljon, mutta ei välitä järjestyksestä (Smith 2010). Esimerkiksi jos kuvassa on useampi sarake tekstiä, ei Tesseract ota niitä tällä määritelmällä huomioon, vaan lukee tekstiä vasemmalta oikealle ja ylhäältä alas siinä järjestyksessä, kun sitä löytyy. PSM_SPARSE_TEXT soveltuu käytettäväksi esimerkiksi silloin, kun halutaan löytää erillisiä tekstialueita, jotka Tesseract yhdistäisi muuten muun tekstin joukkoon, esimerkiksi PSM_SINGLE_COLUMN-määritelmää käytettäessä.

PSM_SPARSE_TEXT_OSD

Tesseract toimii PSM_SPARSE_TEXT_OSD-määritelmällä kuten se toimisi PSM_SPARSE_TEXT-määritelmää käytettäessä, mutta sen lisäksi

Tesseract pyrkii myös tunnistamaan kirjoitusjärjestelmän ja tekstin suunnan (Smith 2010).

PSM_SINGLE_LINE	IN	palauttamaan yksittäisiä merkkejä, sanoja tai kappaleita joko suoraan koneluettavana tekstinä, tai pdf, hOCR ja tsv -muodoissa. Mm. hOCR -
	OUT	EZBL'ZZZZÄTZEZSS2353553553iiiššišfifjféfšä,
	IN	palauttamaan yksittäisiä merkkejä, sanoja tai kappaleita joko suoraan
	OUT	pamuttamaan yksmawsla merkkaa, Sanoja tal kappmena joko suoraan
PSM_SINGLE_WORD	IN	palauttamaan yksittäisiä merkkejä, sanoja tai kappaleita joko suoraan koneluettavana tekstinä, tai pdf, hOCR ja tsv -muodoissa. Mm. hOCR -
	OUT	EZBL'ZZZZÄTZEZSS2353553553iiiššišfifjféfšä,
	IN	palauttamaan yksittäisiä merkkejä, sanoja tai kappaleita joko suoraan
	OUT	pamuttameanykswltawslamerkkeje, sanojatalKappaWeltajokosuorean
	IN	palauttamaan
	OUT	pamuttamaan
PSM_SINGLE_CHAR	IN	palauttamaan
	OUT	m
	IN	n
	OUT	n

KUVIO 16. Muutaman osittelumenetelmän tuloksia

Kuviossa 16 on esillä esimerkkejä muutamasta edellä mainitusta osittelumenetelmästä. IN-kentässä on nähtävissä Tesseractin luettavaksi annettu kuva, ja OUT-kentässä on Tesseractin tulostama teksti vasemmalla määriteltyä osittelumenetelmää käyttäen. Esimerkeissä on käyttetty muutamaa eri tyyppistä tekstiä, jotta nähdään miten ne vaikuttavat tuloksiin.

PSM_SINGLE_LINE soveltuu paremmin yksittäiselle riville, joten sen toisen esimerkin tulos on selvästi parempi, kuin ensimmäisen, jossa kuvassa oli kaksi riviä. Kahden rivin tekstiä käytettäessä tuloksena oli tässä tapauksessa epäselvä merkkijono.

PSM_SINGLE_WORD-määritelmää käytettäessä voidaan huomata, että Tesseract tuotti yhden pitkän merkkijonon ilman välilyöntejä sen yrittäessä lukea rivin tekstiä, joka sisältää useamman sanan. Tulokset olivat kuitenkin tässä tapauksessa identtiset käytettäessä samaa kahden rivin tekstiesimerkkiä, kuin PSM_SINGLE_LINE-määritelmän kohdalla.

PSM_SINGLE_CHAR-määritelmän kohdalla Tesseract tuotti yhden merkin sille syötetystä kuvasta, joka ei pidemmän merkkijonon kohdalla toimi.

3.4.2 Merkkien rajaus

Ennen APIn alustusta Tesseractille voidaan määrittää mitä merkkejä halutaan tunnistaa, käyttämällä muuttujia tessedit_char_whitelist ja tessedit_char_blacklist.

tessedit_char_whitelist

Määrittää merkit, jotka halutaan sallia tunnistettaviksi.

tessedit_char_blacklist

Määrittää merkit, joita Tesseractin ei haluta tunnistavan.

Vaikka merkkien tunnistusta on rajattu, yrittää Tesseract silti hakea jokaiselle löydetylle merkille lähimmän vastineen sallituista merkeistä, mikä tulisi ottaa huomioon näitä käytettäessä. Esimerkiksi sallittaessa vain numerot, voi Tesseract tunnistaa kuvasta löytyvän I-kirjaimen merkiksi 1.

3.4.3 Kuvan asettaminen, rajaaminen ja resoluutio

Halutun kuvan asettaminen Tesseractille onnistuu metodilla setImage(), johon syötetään parametriksi vain Leptonican PIX*-muotoon ladattu kuva.

Jos kyseinen kuva sisältää jonkin tietyn alueen, jota halutaan tutkia, voidaan sen rajaamista varten käyttää Tesseractista löytyvää SetRectangle()-metodia. Metodiin syötetään vain parametreiksi halutun alueen vasemman ylänurkan x- ja y-koordinaatit, sekä leveys ja korkeus, jolloin Tesseract tietää mitä aluetta tutkitaan. Tällä tavalla kuvaa ei tarvitse erikseen pilkkoa ensin palasiksi, vaan Tesseractille voidaan syöttää koko kuva tarvittavien tietojen kanssa, mahdollistaen saman kuvan käyttämisen useampaan kertaan eri alueiden lukemisessa. (Google 2012.)

Jollei Tesseract löydä kuvatiedostosta sen tarkkuustietoja, olettaa se automaattisesti tarkkuuden olevan 72 pistettä tuumalla. Tämä voidaan kuitenkin määrittää myös manuaalisesti ohjelmassa SetSourceResolution()-metodilla, jolloin Tesseract käyttää asetettua arvoa. Tässä tapauksessa TIFF-muotoista tiedosta käyttäessä, osaa Tesseract lukea tiedoston meta-tiedoista kuvan tarkkuuden. (Google 2012.)

3.5 Ulostulomuodot yleisesti

XML ja JSON-muodot ovat yleisimpiä verkossa käytettäviä tiedonvaihtomuotoja. Näiden rinnalle voidaan ottaa vertailukohdaksi tavallinen teksti.

XML on muodoltaan hyvin samankaltainen HTML:n kanssa, mutta erona on niiden käyttötarkoitus. XML on suunniteltu tiedon siirtämiseen, kun taas HTML on suunniteltu sen näyttämiseen. XML:ssä ei myöskään ole ennalta määriteltyjä tunnisteita, vaan XML-dokumentin luoja itse määrittelee ne. (w3schools 2018a).

```xml
<note>
  <to>Jane</to>
  <from>John</from>
  <heading>Meeting reminder</heading>
  <body>Remember our meeting today at 2 pm</body>
</note>
```

KUVIO 17. Esimerkki XML-tiedoston sisällöstä

Kuviossa 17 on esimerkki yksinkertaisesta XML-dokumentista, josta käy hyvin ilmi kuinka itsensä kuvaileva se on. Dokumentissa kuvataan muistilappua, joka käy ilmi note-tunnisteista. Näiden sisällä on määritelty muistilapun otsikko, sisältö, lähettäjä ja vastaanottaja omille tunnisteilleen.

```json
{
  "note": {
    "to": "Jane",
    "from": "John",
    "heading": "Meeting reminder",
    "body": "Remember our meeting today at 2 pm"
  }
}
```

KUVIO 18. Esimerkki JSON-tiedoston sisällöstä

Vaihtoehtoisesti XML:n sijaan voidaan tietoa siirtää myös JSON-muodossa. Kuten XML, JSON on myös itseään kuvaava. Toisin kuin XML, JSON ei käytä päätetunnisteita ja siinä kyetään käyttämään taulukoita. JSON on myös lyhyempi. (w3schools 2018b.)

Esimerkiksi kuviossa 18 esitetty JSON-muotoinen versio kuvion 17 muistiosta tiivistettynä yhdelle riville on 119 merkkiä pitkä, kun taas XML-versio yhdellä rivillä on 140 merkin pituinen. Siirrettävän tiedon määrän kasvaessa ero XML:n ja JSON:n välillä kasvaa juurikin edellä mainittujen päätetunnisteiden ja taulukoiden takia.

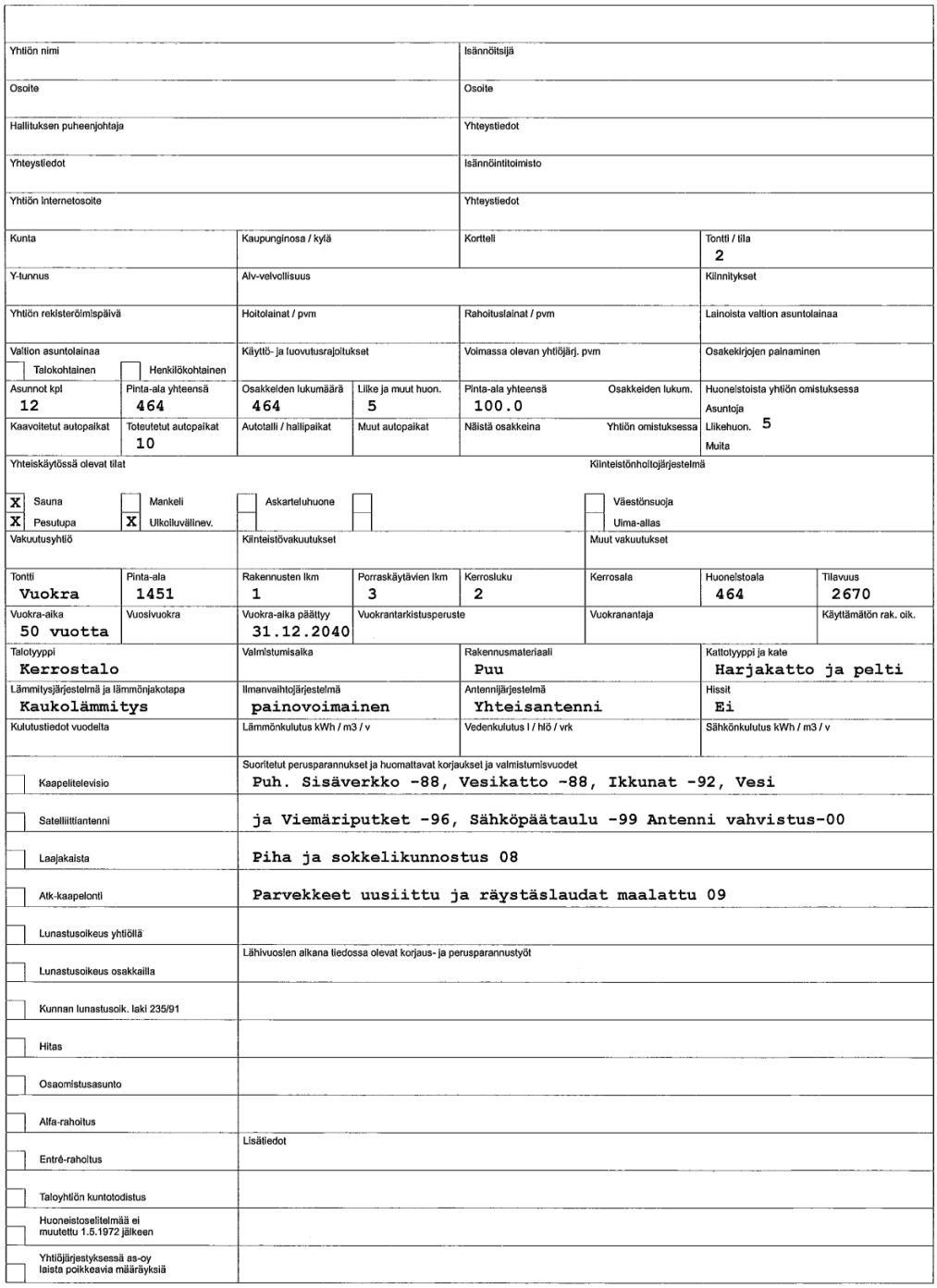
```
1   note
2   to Jane
3   from John
4   heading Meeting reminder
5   body Remember our meeting today at 2 pm
```

KUVIO 19. Esimerkki muistilapusta tekstinä

Tavalliseen tekstitiedostoon verrattuna sekä XML että JSON ovat kuitenkin tehokkaampia tiedon siirtämiseen, sillä tekstimuotoisessa tiedostossa ei ole minkäänlaisia tunnisteita erottelemaan eri tietoja toisistaan, kuten kuviosta 19 näkyy. Vaikka tiedosto olisikin lyhyempi tekstimuodossa, tällaista tiedostoa varten tarvitsee erikseen luoda jäsentäjä, jolla tieto kyetään purkamaan. Kaiken lisäksi tiedostolle tulee määrittää standardi, jota seurata. JSON:lle ja XML:lle löytyy kuitenkin valmiita ratkaisuja purkamista varten, joten niitä käyttäessä säästetään myös aikaa (w3schools 2018b).

3.6 Ulkoasutiedoston tarve ja muoto

Ulkoasutiedosto määrittää dokumentin kenttien otsikon, mutta myös niiden sijainnin prosentteina. Jos dokumenttipohjia olisi vain yksi, ei ulkoasutiedosto olisi välttämätön. Koska dokumenttipohjia on useampia, sekä yhdessä dokumentissa on useampi toisistaan täysin erinäköinen sivu, on käytännöllisempää luoda jokaista pohjaa varten oma ulkoasutiedosto, kuin yrittää ohjelmoida jokin yleinen sääntö joka toimii kaikille dokumenttipohjille.



Yhtiön nimi		Isännöitsijä		
Osoite		Osoite		
Hallituksen puheenjohtaja		Yhteystiedot		
Yhteystiedot		Isännöintitoimisto		
Yhtiön internetosoite		Yhteystiedot		
Kunta	Kaupunginosa / kylä	Kortteli	Tontti / tila: 2	
Y-tunnus	Alv-velvollisuus		Kiinnitykset	
Yhtiön rekisteröimispäivä	Hoitolainat / pvm	Rahoituslainat / pvm	Lainoista valtion asuntolainaa	
Valtion asuntolainaa: ☐ Talokohtainen ☐ Henkilökohtainen	Käyttö- ja luovutusrajoitukset	Voimassa olevan yhtiöjärj. pvm	Osakekirjojen painaminen	
Asunnot kpl: 12	Pinta-ala yhteensä: 464	Osakkeiden lukumäärä: 464	Liike ja muut huon.: 5	Pinta-ala yhteensä: 100.0 Osakkeiden lukum.
Huoneistoista yhtiön omistuksessa: Asuntoja				
Kaavoitetut autopaikat	Toteutetut autopaikat: 10	Autotalli / hallipaikat	Muut autopaikat	Näistä osakkeina Yhtiön omistuksessa
Liikehuon. 5 Muita				

Yhteiskäytössä olevat tilat

Kiinteistönhoitojärjestelmä

X Sauna	☐ Mankeli	☐ Askarteluhuone	☐	☐ Väestönsuoja		
X Pesutupa	X Ulkoiluvälinev.	☐	☐	☐ Uima-allas		

Vakuutusyhtiö		Kiinteistövakuutukset		Muut vakuutukset		
Tontti: Vuokra	Pinta-ala: 1451	Rakennusten lkm: 1	Porraskäytävien lkm: 3	Kerrosluku: 2	Kerrosala	Huoneistoala: 464 / Tilavuus: 2670
Vuokra-aika: 50 vuotta	Vuosivuokra	Vuokra-aika päättyy: 31.12.2040	Vuokrantarkistusperuste	Vuokranantaja		Käyttämätön rak. oik.

Talotyyppi: Kerrostalo	Valmistumisaika	Rakennusmateriaali: Puu	Kattotyyppi ja kate: Harjakatto ja pelti
Lämmitysjärjestelmä ja lämmönjakotapa: Kaukolämmitys	Ilmanvaihtojärjestelmä: painovoimainen	Antennijärjestelmä: Yhteisantenni	Hissit: Ei
Kulutustiedot vuodelta	Lämmönkulutus kWh / m3 / v	Vedenkulutus l / hlö / vrk	Sähkönkulutus kWh / m3 / v

☐ Kaapelitelevisio	Suoritetut perusparannukset ja huomattavat korjaukset ja valmistumisvuodet: Puh. Sisäverkko -88, Vesikatto -88, Ikkunat -92, Vesi
☐ Satelliittiantenni	ja Viemäriputket -96, Sähköpäätaulu -99 Antenni vahvistus-00
☐ Laajakaista	Piha ja sokkelikunnostus 08
☐ Atk-kaapelonti	Parvekkeet uusiittu ja räystäslaudat maalattu 09
☐ Lunastusoikeus yhtiöllä	
☐ Lunastusoikeus osakkailla	Lähivuosien aikana tiedossa olevat korjaus- ja perusparannustyöt
☐ Kunnan lunastusoik. laki 235/91	
☐ Hitas	
☐ Osaomistusasunto	
☐ Alfa-rahoitus	
☐ Entrè-rahoitus	Lisätiedot
☐ Taloyhtiön kuntotodistus	
☐ Huoneistoselitelmää ei muutettu 1.5.1972 jälkeen	
☐ Yhtiöjärjestyksessä as-oy laista poikkeavia määräyksiä	

KUVIO 20. Esimerkki lomakepohjasta

Kuviossa 20 on esimerkki yhdestä lomakkeesta, jolle ulkoasutiedosto voidaan luoda. Koska paperilomakkeessa kentät ovat usein ennalta määriteltyjä, tulee ulkoasutiedoston toinen hyötypuoli esille. Esimerkiksi

kuvion 20 sisältämästä lomakkeesta voidaan ulkoasutiedoston avulla tarkastella vain tiettyjä kenttiä, jolloin tämän alueen sisältämät tiedot voidaan liittää tutkittavaan kenttään ohjelmallisesti. Tällä kyetään myös vähentämään Tesseractin tuottamien virheiden määrää tekstiä luettaessa. Tämä siitä syystä, että joskus Tesseract saattaa lukea rajaviivoja osaksi tekstiä, tai tietyissä tilanteissa yhdistää tekstiä rajaviivojen yli, jolloin jotkin kentät saavat sisällökseen väärää tietoa.


```
{
    "documentSize":{
       "width":5488,
       "height":7544
    },
    "elements":[
       {
          "name":"Yhtiön nimi",
          "type":"text",
          "size":{
             "width":48.870262390670554,
             "height":2.9029692470837754
          },
          "location":{
             "top":2.982502651113468,
             "left":0.09110787172011661
          }
       },
       {
          "name":"Sauna",
          "type":"checkmark",
          "size":{
             "width":2.0043731778425657,
             "height":1.4846235418875928
          },
          "location":{
             "top":38.2423117709438,
             "left":0.10932944606413994
          }
       },
       {
          "name":"Pesutupa",
          "type":"checkmark",
          "size":{
             "width":2.0043731778425657,
             "height":1.4448568398727466
          },
          "location":{
             "top":39.74019088016967,
             "left":0.10932944606413994
          }
       },
```


KUVIO 21. Ulkoasutiedoston rakenne

Ulkoasutiedosto täytyy luoda dokumenttipohjia varten käsin, mutta sen helpottamiseksi projektia varten luotiin yrityksen palvelimelle yksinkertainen JavaScript-sovellus, johon ladataan halutun dokumentin kuva, jonka päälle voidaan vetää halutun tyyppisiä laatikoita, tai tässä tapauksessa kenttiä. Jokaiselle kentälle voi ohjelmassa määrittää oman nimen, jonka mielellään tulisi olla kentän sisältöä kuvaava, kuten esimerkiksi dokumentissa oleva kentän otsikko. Kun ulkoasu on valmis, voidaan se ladata JSON-muotoisena tiedostona.

Ulkoasutiedoston rakenne käy ilmi kuviosta 21. Alkuun on määriteltynä documentSize, joka on siis dokumentin koko pikseleinä. Tämä on laskettu dokumentin rajaamasta neliöstä, joka tulee määrittää ulkoasutiedostoa luotaessa. Rivillä 6 alkaa elements-lista, johon sisältyy kaikki määritellyt kentät. Jokainen kenttä sisältää neljä ominaisuutta; nimen, tyypin, koon ja sijainnin. Nimi on kenttää määriteltäessä sille annettu nimi. Tyyppi taas määrittää sisältääkö haluttu kenttä tekstiä tai valintalaatikoita. Kentän koko määrittää sen leveyden ja korkeuden prosentteina koko dokumentin mitoista, kun taas sijainnissa määritellään vasemman ylänurkan etäisyys prosentuaalisesti yläreunasta ja vasemmasta reunasta. Prosentteja tässä tapauksessa käytetään siksi, että kuvat, joita tekstin tunnistukseen käytetään, voivat olla hyvinkin eri kokoisia, joten kenttien koko- ja sijaintitietoja ei ole kannattavaa määrittää pikselimitoilla. Prosenttien avulla voidaan ohjelmassa nopeasti laskea kentän sijainti ja koko, kun tiedetään kohdistinpisteiden sijainti.

4 CASE: DOKUMENTIN LUKIJA

4.1 Suunnitelma

Alunperin tarkoituksena oli luoda prototyyppi asiakasta varten, jotta voidaan näyttää mihin tekstin tunnistuksella kyetään. Asiakas halusi ohjelman, jolla kyettäisiin lukemaan erilaisista isännöitsijäntodistuksista tietoja.

Koska alunperin tarkoituksena oli kehittää vain prototyyppi, ei toteutuksella ollut rajoitteita. Projekti päätettiin alunperin toteuttaa Swiftiä käyttäen Applen Vision viitekehyksellä, joka sisältää jo valmiiksi tekstialueiden tunnistuksen. Osana Visionia on myös Core ML-moduuli, jonka avulla voidaan käyttää valmista koneälymallia dokumenttien lukuun. Koneälymallin luomisessa on omat haasteensa, minkä takia päädyttiin tutkimaan valmiita kirjastoja. Ensimmäisenä testattavana oli SwiftOCR, joka ei kuitenkaan kykene lukemaan kuin yhden rivin tekstiä kerralla. Tesseractia testattaessa todettiin tulosten olevan riittävät, joten projekti muutettiin Swift-kielestä C++-kieleen.

Asiakkaalle esitettiin myöhemmin prototyyppiversio, jota sitten lähdettiin kehittämään eteenpäin. Projektiin tuotiin myöhemmin mukaan ominaisuuksina ulkoasutiedostojen luonti ja kuvien ottaminen, sekä niiden perspektiivin korjaus omina osinaan.

4.2 Miksi Tesseract

Ennen projektin alkua tutkittiin oman konenälymallin luomista. Kyseistä mallia olisi kyetty käyttämään tässä projektissa, jolloin se olisi mahdollisesti toteutettu Swift 4.0-kieltä käyttäen Applen Core ML-viitekehyksessä. Koska tulokset eivät omalla koneälymallilla parantuneet tarpeeksi ajan kanssa, päätettiin siirtyä käyttämään valmista kirjastoa tekstin tunnistusta varten. Tesseract oli tullut jo aiemmin esiin eri hakujen yhteydessä, kuten myös tässäkin kohtaa, joten se otettiin ensiksi käyttöön pientä testailua varten. Testien perusteella Tesseract suoriutui

huomattavasti paremmin kuin omien koneälymallien luominen, joten se otettiin tämän perusteella käyttöön.

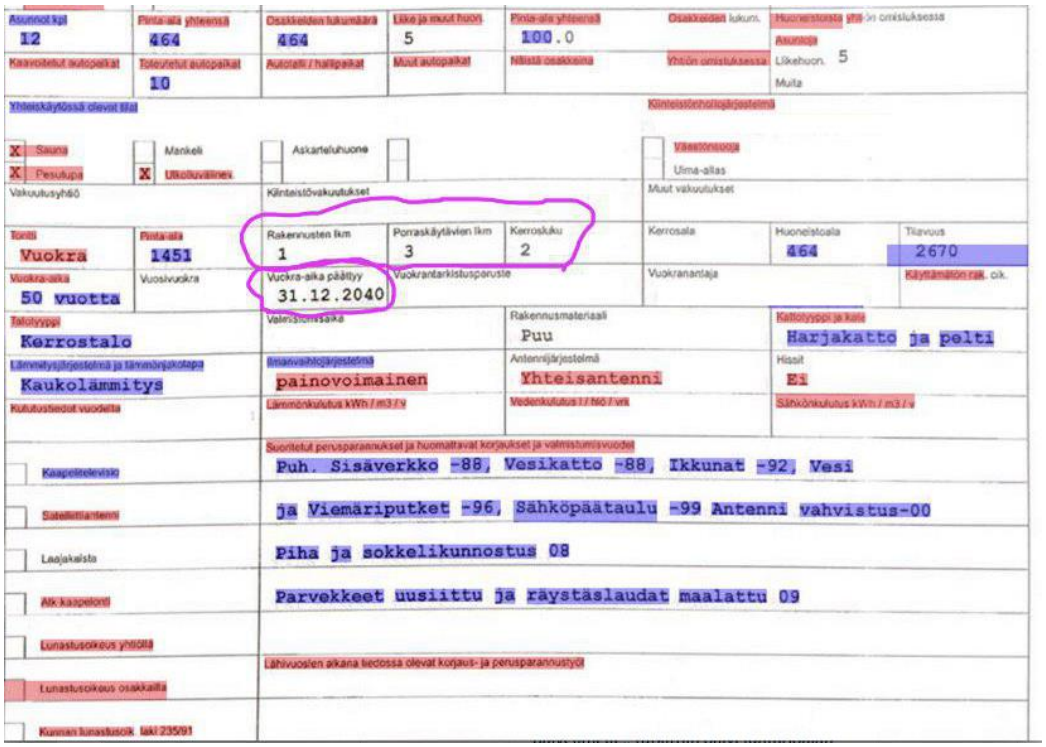
4.3 Toteutus

Projekti toteutettiin kahdessa vaiheessa. Ensimmäinen vaihe oli prototyyppi, jonka jälkeen asiakkaan ollessa siihen tyytyväinen, kehitettiin siitä eteenpäin lopullista ohjelmaa. Projektin päätteeksi ohjelma kykeni lukemaan lomakkeita ja tuottamaan niistä halutunlaisen JSON-tiedoston.

4.3.1 Prototyyppi

Prototyyppiä lähdettiin alunperin työstämään Swift-projektina, ennen kuin Tesseractiin siirryttäessä vaihdettiin kieli myös C++:ksi, jolle Tesseract-kirjasto on myös kirjoitettu. Prototyypissä ohjelma ei käyttänyt ulkoasutietoja dokumentin lukemiseen, vaan se luki dokumentin yhdellä kertaa. Ohjelmaa rakennettiin aluksi siis siten, että se kykenisi määrittämään dokumenttien elementit niiden koon ja etäisyyksien perusteella. Tämän takia prototyyppi oli tässä vaiheessa täynnä monia monimutkaisia ja vaikeasti seurattavia tarkastusmetodeja määrittämässä elementtien tyypit ja miten elementit kytketään keskenään. Myöhemmässä versiossa tästä luovuttiin sen monimutkaisuuden takia, koska ohjelman olisi tarkoitus pystyä tulevaisuudessa lukemaan muitakin kuin yhden tyyppisiä dokumentteja, mikä tällä tyylillä monimutkaistaisi asioita entisestään.

Dokumentin lukemisen jälkeen tunnistettujen tekstien fonttikokoja vertailtiin ja niille asetettiin raja, jonka alapuolella olevat tekstikentät määriteltiin otsikoiksi, ja loput tietokentiksi. Vaikka raja joissakin tapauksissa oli hyvinkin selkeä, ei se kuitenkaan toiminut täydellisesti, sillä Tesseract saattoi lukea tekstikenttien fonttikoon väärin, riippuen käytetystä kuvasta. Tämä aiheutti sen, että jotkin tekstikentät merkittiin väärän tyyppiseksi.



KUVIO 22. Tekstialueiden tunnistus prototyypissä

Kuviossa 22 on esimerkki tekstin alueiden tunnistuksesta. Tesseractin tuottamien tietojen mukaan lomakekuvan päälle kyettiin piirtämään alueet, joilta Tesseract on löytänyt tekstiä. Punaisella värjätyt alueet ovat niitä, jotka ohjelma on määrittänyt otsikoiksi, kun taas sinisellä värjättyjen alueiden on tarkoitus olla otsikoiden alle kuuluvia tietoja. Koska tunnistettu fonttikoko ei aina välttämättä ole oikein, on sitä epäluotettavaa käyttää tekstityypin määrittelemiseen. Kuviosta 22 pystytään myös näkemään, että Tesseract ei aina löydä kaikkia tekstialueita, kuten esimerkiksi keskeltä kuvaa ympäröidyt alueet.

Tesseract ei aina myöskään lukenut lauseita kokonaisina, vaan saattoi katkaista ne sanoiksi, joten ohjelmaa kirjoittaessa piti ottaa huomioon, että ohjelman tulisi itse myös osata yhdistellä pätkityt tiedot yhdeksi kokonaisuudeksi. Tämä toi mukanaan omat ongelmansa, kuten sen, että ohjelma yhdisti väärin tunnistettuja tietoja keskenään. Tällöin esimerkiksi otsikko ja tietokenttä tunnistettiin yhdessä yhdeksi elementiksi, tai

huonommissa tapauksissa lähekkäin sijaitsevien tietokenttien tekstit saatettiin yhdistää yhtenäiseksi elementiksi. Viimeisin mainittu tapahtui esimerkiksi silloin, kun Tesseract tunnisti tekstialueet isommiksi kuin ne olivat.

Tulosteena prototyyppi luetteli elementit konsolille ”otsikko – tieto”-pareiksi, jonka lisäksi se merkitsi dokumentin kuvaan tunnistetut elementit. JSON tulostusmuotona otettiin käyttöön vasta myöhemmin lopulliseen versioon, kun tulosteeseen haluttiin saada elementeistä enemmän tietoja.

Asiakas oli tässä vaiheessa tyytyväinen prototyyppiin, joten siitä alettiin kehittämään lopullista ohjelmaa.

4.3.2 Lopullinen ohjelma

Koska tuli esiin, että kaikki dokumenttipohjat eivät välttämättä tule olemaan samanlaisia, piti ohjelmaa muokata. Luotiin ohjelma, jolla kyetään luomaan ulkoasutiedosto manuaalisesti piirtämällä laatikoita dokumentin päälle. Tämä kuitenkin tarkoitti sitä, että jokaiselle dokumenttipohjalle täytyisi luoda ulkoasutiedosto manuaalisesti. Itse projektia muokattiin siten, että se kykenee lukemaan luodun ulkoasutiedoston ja sen avulla tunnistamaan tekstiä yksittäisistä kentistä.

Ulkoasutiedostoon kyetään myös määrittämään kentän tyyppi, jolloin esimerkiksi valintaruudut pystytään erikseen tutkimaan ja toteamaan ovatko ne tyhjiä vai eivät. Näiden kohdalla ruudun sisällöstä tulee tarkastella pikseleiden värejä, mutta sitä ennen ruudun sisältö muutetaan mustavalkoiseksi. Koska tekstit käyttävät mustaa fonttia, ja koska dokumenttipohjan elementit eivät aina osu pikselilleen kohdakkain ulkoasutiedoston kanssa, ei ruutua voida merkitä tyhjäksi tai täytetyksi sen mukaan sisältääkö se mustia pikseleitä vai ei. Valintaruutu määritellään tässä vaiheessa mustien pikseleiden peittämän alueen koon mukaan. Tässä kohtaa raja-arvoksi mustan määrälle on valittu 18 % ruudun pikseleistä, jonka yläpuolella ruutu merkitään täytetyksi. 18 % valittiin

muutamien testien perusteella, koska se osoittautui toimivaksi, mutta sitä ei ehditty tämän työn aikana testata pidemmälle.

Ennen kuin ulkoasutiedostoihin määriteltiin kenttien otsikot manuaalisesti, pyrittiin käyttämään Tesseractin tunnistamia otsikoita, joista pyrittiin ohjelmallisesti korjaamaan kirjoitusvirheitä. Dokumentin ulkoasutiedostoihin sisällytettyihin otsikoihin siirryttiin, koska ne olivat aina samat samassa dokumenttipohjassa.

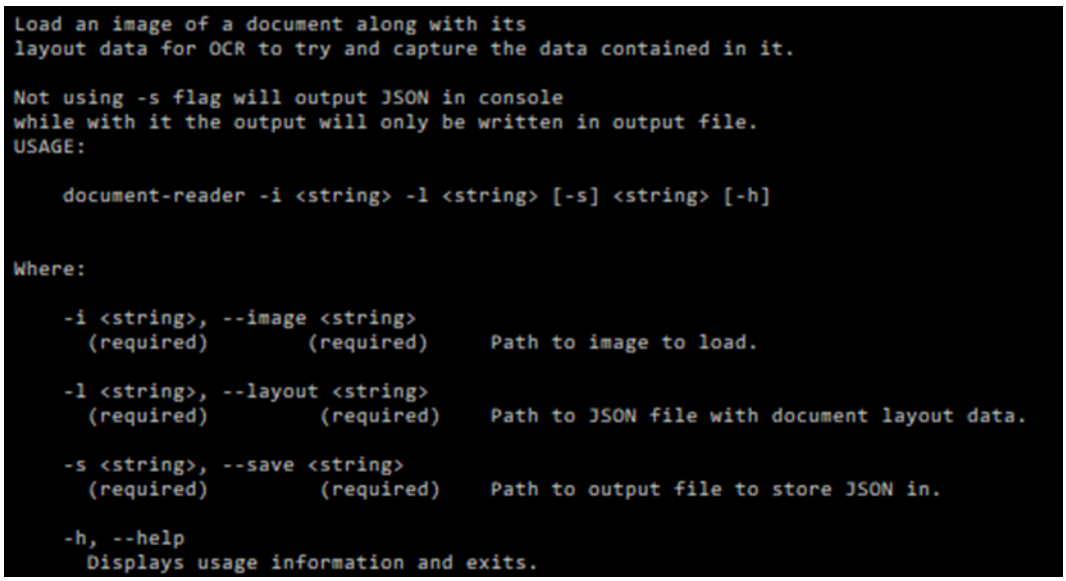
Tunnistettujen otsikoiden korjaamista varten ohjelmalle luotiin otsikoista lista, jota ohjelma käyttäisi Levenšteinin etäisyyttä, toisinsanottuna editointietäisyyttä. Levenšteinin etäisyydellä kyetään laskemaan kuinka monta operaatiota esimerkiksi jollekin sanalle täytyy tehdä, jotta siitä saadaan jokin toinen sana. Tällä siis pystyttiin vertaamaan sitä kuinka monta operaatiota tunnistettuun tekstiin täytyi tehdä, jotta se vastaisi listattuja otsikoita. Operaatioita tässä tapauksessa ovat merkkien lisäys, poisto ja korvaaminen. Variaationa tästä kokeiltiin myös Damerau-Levenšteinin etäisyyttä, jossa on lisäksi operaatio kahden vierekkäisen merkin paikan vaihdolle. Tuloksista valittiin korjatuksi otsikoksi se, johon kului vähiten operaatioita. (Manning, Raghavan & Schütze 2009; Stern 2012.)

Tekstin korjaamisen ongelmana oli kuitenkin se, että heikoimmissa tapauksissa korjattava teksti oli täysin epäselvä tai se oli operaatioiden määrän kannalta yhtä lähellä tai lähempänä väärää otsikkoa. Lisäksi erikoisemmat merkit, kuten å, ä ja ö, tuli ottaa huomioon, koska ne ovat tyyppiä wchar_t ja niiden leveys on kaksi tavua, kun taas tyypin char leveys on yksi tavu (Microsoft 2018). Merkkien leveys johti siihen, että kyseiset merkit luettiin kahdeksi merkiksi, mikä vääristi tuloksia. Jotta merkit luettaisiin oikein käytettiin väliaikaisena ratkaisuna å, ä ja ö-merkkien muuttaminen a ja o-merkeiksi säilyttäen samalla niiden sijainnit, ja muuttamalla ne takaisin jälkikäteen. Tätä merkkien muuntamista ei kuitenkaan käytetty pitkään ennen kuin tekstin korjauksesta luovuttiin, koska otsikot olivat muutenkin listattuna erilliseen tiedostoon. Lopulliseen

ohjelmaan lisättiin myös mahdollisuus luoda JSON-tiedosto tekstin tunnistuksesta, sekä aputiedot sen käyttöä varten

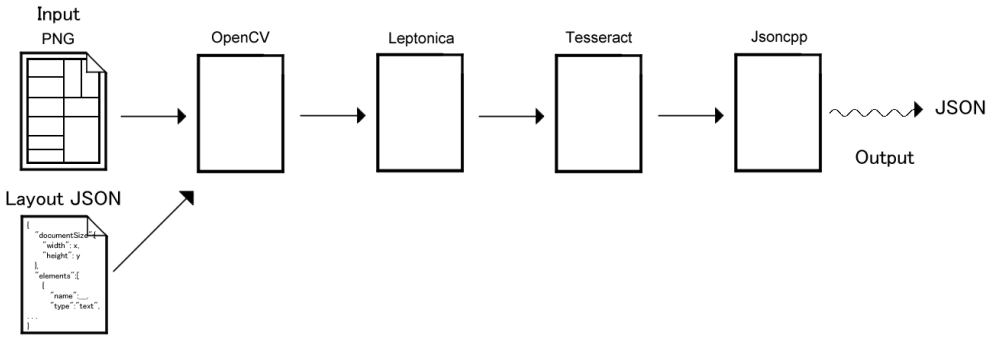
4.4 Prosessin kulku

Ohjelma lähtee liikkeelle komentorivistä, josta ohjelma saa argumentteina luettavan kuvan ja ulkoasutiedoston sijainnin, sekä JSON-tiedoston tallennussijainnin.



KUVIO 23. Ohjelman apuvalikko

Kuviossa 23 nähdään ohjelman apuvalikko, josta käy ilmi vaaditut argumentit ja se mitä jokainen niistä tekee. Kuten kuviosta näkee, JSON-tiedoston tallennusargumentti ei ole pakollinen, ja sen puuttuessa ohjelma tulostaa JSON:n vain suoraan konsoliin.



KUVIO 24. Yksinkertaistettu prosessin kulkukaavio

Ohjelman tarkastaessa vaaditut argumentit ja tiedostojen sijainnit, luetaan ulkoasutiedosto std-kirjaston avulla ja kuva OpenCV:tä käyttäen muistiin mat-rakenteeseen. Kuvion 24 mukaisesti kuva on tässä vaiheessa OpenCV:n käsiteltävänä ennen seuraavaan vaiheeseen siirtymistä. Jotta vältetään kohinan paheneminen kuvan kokoa muuttaessa, suoritetaan kohinan poisto tässä vaiheessa jos kuva on alle 2180 pikseliä leveä. 2180 pikselin leveys valittiin tähän vain testitiedostojen mukaan, sillä isommissa tiedostoissa kohinan poisto kestää aina kauemmin pikselimäärän ollessa suurempi. Kohinan poisto suoritetaan käyttämällä fastNlMeansDenoisingColored()-metodia.

Kuvan kokoa muutetaan tämän jälkeen OpenCV:n resize()-metodilla kuvasuhteet säilyttäen siten, että kuvan leveys tulee olemaan 4000 pikseliä, jotta siinä olevat pienemmät merkit olisivat vielä selkeät. Tämän jälkeen kuva täytyy muuntaa Leptonican PIX-muotoon, mutta koska tämä ei onnistu suoraan, tallennetaan kuva levylle ohjelman kansioon käyttämällä OpenCV:n imwrite()-metodia. Leptonicalla kyetään lukemaan kuva pixRead()-metodilla.

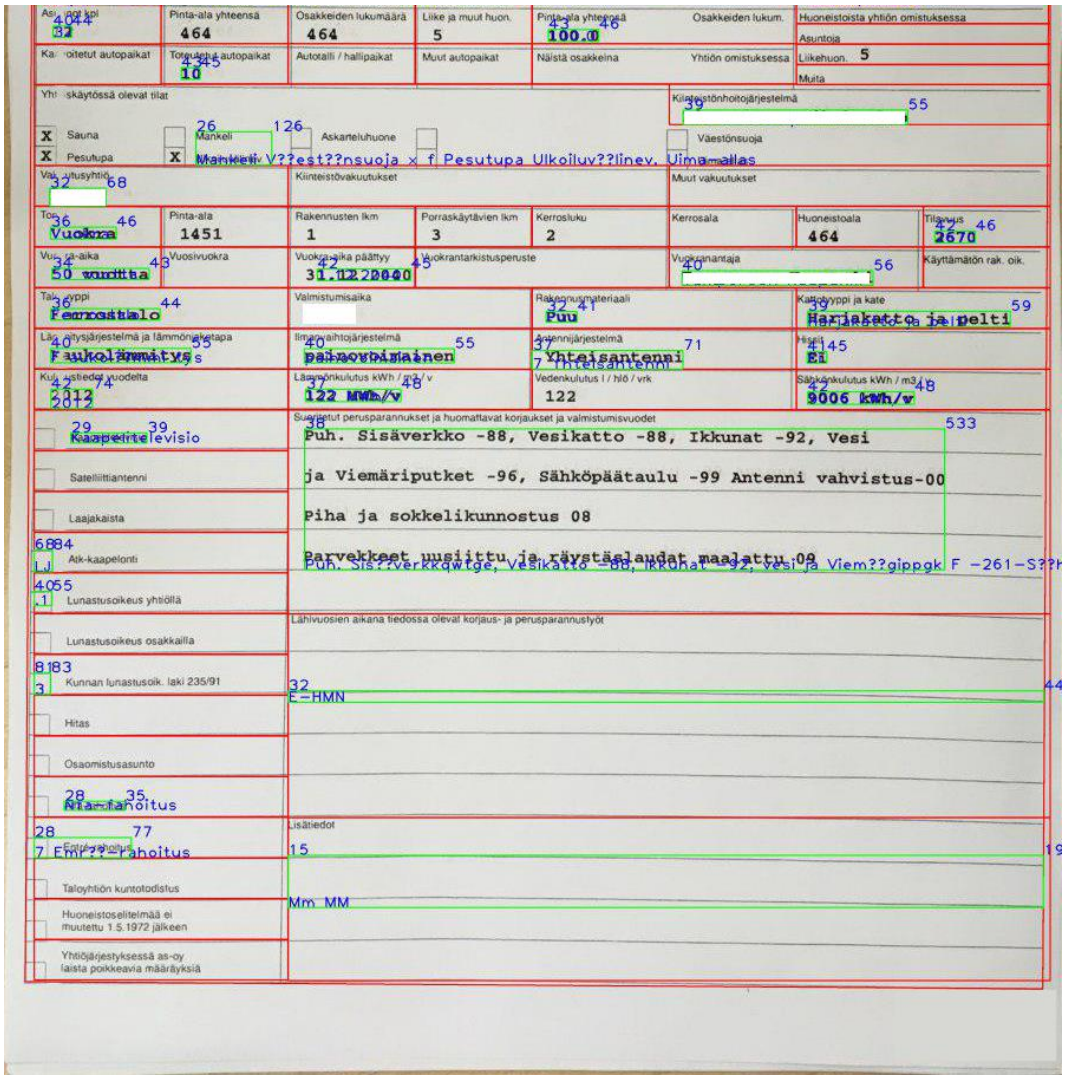
Mat-muotoista kuvaa käytetään vielä kohdistuspisteiden löytämiseen, ja tätä varten kuva täytyy muuntaa mustavalkoiseksi binäärikuvaksi. Binäärikuvasta kyetään hakemaan vaaka- ja pystysuorat elementit getStructuringElement()-metodia käyttäen, jolle voidaan syöttää eri kokoisia nelikulmion muotoisia elementtejä, joita etsitään. Tässä

tapauksessa haettavat elementit ovat kapeita pysty- ja vaakaviivoja. Koska kuva on binäärimuodossa, voidaan siitä hakea liitoskohdat vaaka ja pystyviivojen risteyskohdista, joista otetaan talteen neljä ulommaista pistettä.

```
630     // Initialize Tesseract OCR
631     api.Init(datapath, language, tesseract::OEM_DEFAULT);
632     api.SetVariable("tessedit_char_whitelist", "abcdefghijklmnopqrstuvwxyzääö
633         ABCDEFGHIJKLMNOPQRSTUVWXYZÅÄÖ""0123456789,.@-+()/"); // Set detectable characters
634     api.SetVariable("language_model_penalty_non_freq_dict_word", "0.1");
635     api.SetVariable("language_model_penalty_non_dict_word", "0.15");
636     api.SetPageSegMode(pagesegmode);
637     api.SetImage(pix);
638     api.SetRectangle(cell.boundingBox.x, cell.boundingBox.y,
639         cell.boundingBox.width, cell.boundingBox.height);
640     api.SetOutputName("output");
641     api.SetSourceResolution(100);
642     api.SetVariable("debug_file", "tesseract.log");
643 
644     int lcount = 1;
645 
646     // Recognize text from the image
647     api.Recognize(0);
```

KUVIO 25. Tesseractin alustus

Tässä vaiheessa ulkoasutiedostolle kyetään asettamaan pikselimitat kohdistuspisteiden avulla ja sen määrittämiä alueita voidaan käydä kuvasta läpi yksi kerrallaan Tesseractissa. Ennen tunnistusta Tesseract tulee kuitenkin alustaa ennen kuin sitä voidaan käyttää. Kuviossa 25 on esitettynä Tesseractin ja sen eri muuttujien alustus. Tärkeimpinä näistä ohjelman kannalta ovat Init(), SetPageSegMode(), SetImage() ja SetRectangle(). Näiden avulla Tesseract osaa käsitellä kuvan oikeaa aluetta ja lukea sitä halutulla osittelutyylillä, joka tässä projektissa on PSM_SPARSE_TEXT_OSD.



KUVIO 26. Heikompi tekstin tunnistus visualisoituna

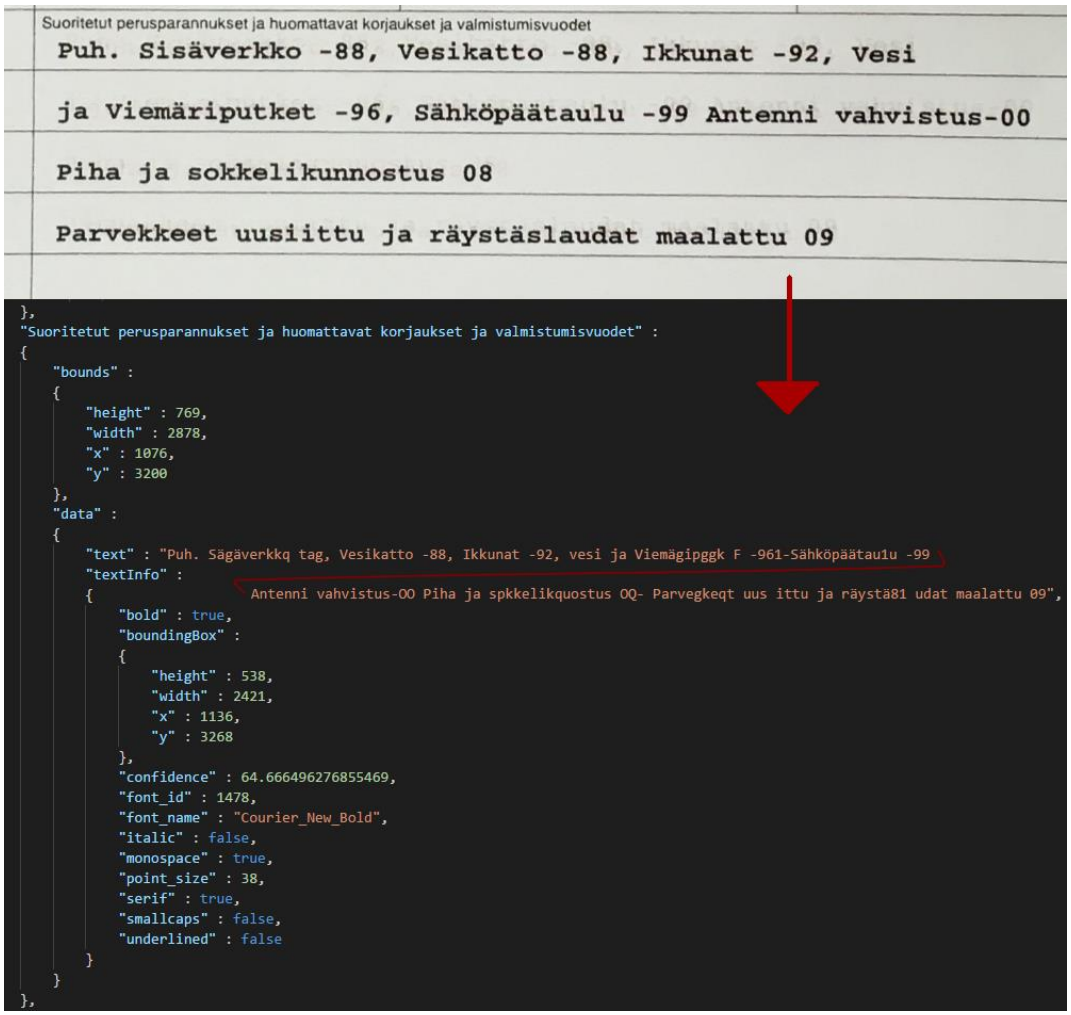
Tulokset käydään ennen tallennusta yksi kerrallaan läpi, jotta voidaan poistaa tyhjät rivit sekä tyhjät merkit tekstin alusta ja lopusta. Samalla suodatetaan joukosta pois myös löydetyt tekstialueet, joiden korkeus ei sovi ennalta määrätylle välille. Tämä siksi että Tesseract saattaa joskus tunnistaa esimerkiksi lomakkeen reunaviivoja tekstiksi. Esimerkkinä tästä on kuvion 26 oikean alareunan kentissä, joissa tekstiä on tunnistettu alueilta, joissa sitä ei oikeasti ole.

Koska Tesseract saattaa tunnistaa erilliset sanat omiksi tekstialueiksi, tarkistetaan tuloksista lähekkäin ja samalla tasolla olevat tekstit, jotta ne voidaan yhdistää keskenään. Tämän jälkeen saadut tiedot tallennetaan

tulokset sisältävään vektoriin. Koska otsikkokentät sisältyvät ulkoasutiedoston alueeseen, poistetaan ne vielä tuloksista jälkikäteen, sillä otsikkotiedot ovat jo määriteltynä ulkoasutiedoissa.

Tekstin tunnistuksen ohella, lomakkeesta pyritään myös tunnistamaan onko valintaruudussa rasti vai ei. Tämä on toteutettu yksinkertaisesti muuttamalla ulkoasutiedoston määrittämän valintaruutualueen sisältö binäärikuvaksi, josta sitten tarkistetaan mustien pikseleiden suhde pikseleiden kokonaismäärään. Koska lomake saattaa olla vähän vinossa esimerkiksi taitoksien takia, saattavat ruudun reunaviivat sisältyä tutkittavaan alueeseen. Reunaviivat lisäävät mustien pikseleiden määrää, joten tämä pitää ottaa huomioon asettaessa rajaa. Asetettu raja määrittää onko ruutu merkitty vai tyhjä riippuen siitä ylittääkö mustien pikseleiden prosentuaalinen määrä sen.

Viimeisenä vaiheena on tietojen jäsentely JSON-muotoon Jsoncpp-kirjastoa käyttäen. Jokaisen tuloksen tiedot käydään yksitellen läpi, ja ne lisätään Jsoncpp:n Value-rakenteeseen, johon tietoja lisätessä määritetään jokaisen kentän otsikko ja sisältö. Lopuksi tarkistetaan haluttiinko tulos tallentaa tiedostoon vai haluttiinko se vain konsoliin, suoritetaan haluttu toimenpide ja suljetaan ohjelma.



KUVIO 27. Esimerkki tekstin tunnistuksesta

Lopuksi tuotetusta JSON-tiedostosta on esimerkki kuviossa 27, josta nähdään vielä vertailuksi alkuperäinen teksti tunnistukseen käytetystä kuvasta. Itse tuloksena saatu teksti ei ole täydellinen, vaikka Tesseract on joitakin sanoja silti onnistunut lukemaan oikein. JSONin bounds-kenttä sisältää ulkoasutiedostosta lasketun sijainnin tekstikentälle. TextInfo-kentän alta löytyvistä tiedoista kaikki paitsi confidence-arvo ovat taas Tesseractin löytämiä ominaisuuksia tekstille. Confidence, eli Tesseractin luottamus siihen että tunnistettu teksti on oikein. Luku on keskiarvo Tesseractin varmuudesta kaikkien tekstin sanojen kohdalla, ja se on ilmoitettu tapauksessa nollan ja sadan väliltä siten, että pienempi arvo tarkoittaa heikompaa luottamusta

5 YHTEENVETO

Työn tavoitteena ollut ohjelmisto toteutettiin onnistuneesti haluttuun pisteeseen, jossa se kykeni tuottamaan tietoa halutussa muodossa. Tuotettu ohjelma siis kykenee käsittelemään kuvia lomakkeista ja tuottamaan niistä JSON-muotoisen tiedoston, jota voidaan käyttää muuhun tarkoitukseen.

Ohjelmassa on silti isoimpana rajoitteena tekstin lukemisen kannalta valaistus, sen tasaisuus, ja kuvan laatu, sekä niiden eri yhdistelmistä johtuva toisinaan heikko tekstin tunnistus. Laatua toki voidaan parantaa varmistamalla että dokumentin on valaistu tarpeeksi hyvin ja tasaisesti, jotta sen sisältö erottuisi selkeästi. Laatua voidaan parantaa edelleen käyttämällä laadukasta kameraa.

Muuta kautta laadun parantamiseksi yhtenä ratkaisuna voisi ajatella Tesseractin kouluttamisen vain dokumenteissa käytettyihin kirjasintyyppeihin. Tämä tietenkin itsessään tuo oman ongelmansa, joka liittyy uusien dokumenttityyppien lukemiseen. Jos uudessa dokumentissa käytetään huomattavasti erilaista kirjasintyyppiä, täytyy silloin käyttää aikaa sen opettamiseen Tesseractille. Vaihtoehtoisesti aikaa voisi käyttää myös enemmän Tesseractin eri ominaisuuksien tutkimiseen mahdollisia parannuksia varten.

Toisena mahdollisesti parempana ratkaisuna olisi vaihtoehtoisesti testata Tesseractin 4.00 alpha-versiota, joka käyttää LSTM (Long Short-Term Memory)-neuroverkkoja, joiden pitäisi kyetä tarkempiin tuloksiin. Tämä saattaisi johtaa ohjelman uudelleen kirjoittamiseen, mutta jos tulokset olisivat huomattavasti parempia myös heikommalla valaistuksella, olisi tämä varteenotettava vaihtoehto.

Viimeisimpänä vaihtoehtona olisi tarkemmin tutkia muiden tekstin tunnistuskirjastojen tarkkuutta ja kokeilla niiden käyttämistä Tesseractin sijaan. Kuten edellä, tämä tulisi vaatimaan ohjelman uudelleenkirjoitusta, mutta tuotetun tekstin laatu on tällaisissa ohjelmissa tärkeässä asemassa.

LÄHTEET

ABBYY 2018. What is OCR and OCR Technology. [viitattu 1.4.2018]. Saatavissa: https://www.abbyy.com/en-eu/finereader/what-is-ocr/

Buades, A., Coll, B. & Morel, J. 2011. Non-Local Means Denoising, Image Processing On Line. Tutkimus [viitattu 2.4.2018]. Saatavissa: http://www.ipol.im/pub/art/2011/bcm_nlm/

Fonecta 2018. SuperApp Oy:n yritystiedot. [viitattu 27.3.2018]. Saatavissa: https://www.finder.fi/IT-konsultointia+IT-palveluja/SuperApp+Oy/Lahti/yhteystiedot/3074414

Fulton, W. 2018, Color Bit-depth & Memory Cost of Images. Verkkosivu [viitattu 27.3.2018]. Saatavissa: https://www.scantips.com/basics1d.html

Google 2012. Tesseract: Advanced API. Verkkosivu [viitattu 28.3.2018]. Saatavissa: https://zdenop.github.io/tesseract-doc/group___advanced_a_p_i.html

Google 2018a. Command Line Usage. [viitattu 27.3.2018]. Saatavissa: https://github.com/tesseract-ocr/tesseract/wiki/Command-Line-Usage

Google 2018b. Tesseract OCR. [viitattu 27.3.2018]. Saatavissa: https://github.com/tesseract-ocr/tesseract/

Holmes, N. 2013. What’s the difference between GIF, PNG, JPEG and TIFF. Blogi [viitattu 27.3.2018]. Saatavissa: http://www.smartimage.com/whats-the-difference-between-gif-png-jpeg-and-tiff/

InStream 2014. OCR Basics. Blogi [viitattu 6.4.2018]. Saatavissa: https://instreamllc.com/2014/12/03/ocr-basics/

Lilley, C. 1996. JPEG JFIF. [viitattu 27.3.2018]. Saatavissa: https://www.w3.org/Graphics/JPEG/

Maischein, M. 1996. File format list. Lista [viitattu 31.3.2018]. Saatavissa: http://www.fileformat.info/mirror/corion/original.htm#lst

Manning, C., Raghavan, P. & Schütze, H. 2009. Dictionaries and tolerant retrieval. Cambridge University Press. Oppikirja [viitattu 3.4.2018]. Saatavissa: https://nlp.stanford.edu/IR-book/pdf/03dict.pdf

Microsoft 2018. Multibyte and Wide Characters. [viitattu 3.4.2018]. Saatavissa: https://msdn.microsoft.com/en-us/library/z207t55f.aspx

Morrison, J. 2018. The difference between PNG24 and PNG32. Blogi [viitattu 28.3.2018]. Saatavissa: https://deepbluesky.com/news/-/the-difference-between-png24-and-png32_49/

OpenCV 2014. Introduction. [viitattu 16.3.2018]. Saatavissa: https://docs.opencv.org/3.0-beta/modules/core/doc/intro.html

OpenCV 2017a. Average blurring [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/d13/tutorial_py_filtering.html

OpenCV 2017b. Bilateral filtering [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/d13/tutorial_py_filtering.html

OpenCV 2017c. Denoising. [viitattu 1.4.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d1/d79/group__photo__denoise.html

OpenCV 2017d. Eroding and Dilating. [viitattu 1.4.2018]. Saatavissa: https://docs.opencv.org/3.3.0/db/df6/tutorial_erosion_dilatation.html

OpenCV 2017e. Gaussian blurring [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/d13/tutorial_py_filtering.html

OpenCV 2017f. Geometric Image Transformations. [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/da/d54/group__imgproc__transform.html

OpenCV 2017g. Image file reading and writing. [viitattu 1.4.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/da8/group__imgcodecs.html

OpenCV 2017h. Image Filtering. [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/d86/group__imgproc__filter.html

OpenCV 2017i. Image Thresholding. [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d7/d4d/tutorial_py_thresholding.html

OpenCV 2017j. Median blurring [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/d13/tutorial_py_filtering.html

OpenCV 2017k. Smoothing Images. [viitattu 31.3.2018]. Saatavissa: https://docs.opencv.org/3.3.0/d4/d13/tutorial_py_filtering.html

OpenCV 2018. OpenCV. [viitattu 3.4.2017]. Saatavissa: https://opencv.org/

Rochester Institute of Technology, 2018. Binary Images. Esitelmä [viitattu 29.3.2018]. Saatavissa: http://www.cis.rit.edu/people/faculty/pelz/courses/SIMG203/res.pdf

Smith, R. 2010. publictypes.h. Lähdekoodi [viitattu 19.3.2018]. Saatavissa: https://github.com/tesseract-ocr/tesseract/blob/master/ccstruct/publictypes.h

Stern, K. 2012. Damerau-Levenshtein Edit Distance. Blogi [viitattu 3.4.2018]. Saatavissa: http://software-and-algorithms.blogspot.fi/2012/09/damerau-levenshtein-edit-distance.html

SuperApp 2018. Etusivu. [viitattu 27.3.2018]. Saatavissa: http://www.superapp.fi/

w3schools 2018a. Introduction to XML. [viitattu 1.4.2018]. Saatavissa: https://www.w3schools.com/xml/xml_whatis.asp

w3schools 2018b. JSON vs XML. [viitattu 2.4.2018]. Saatavissa: https://www.w3schools.com/js/js_json_xml.asp