Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Kandidaatintyö

Osku Grönberg

MERKITSEMISTYÖKALU KUVIEN MANUAALISEEN SEGMENTOINTIIN

Työn tarkastaja(t): Dosentti Tuomas Eerola

Työn ohjaaja(t): Dosentti Tuomas Eerola

TIIVISTELMÄ

Lappeenrannan teknillinen yliopisto
Teknistaloudellinen tiedekunta
Tietotekniikan koulutusohjelma

Osku Grönberg

Merkitsemistyökalu kuvien manuaaliseen segmentointiin

Kandidaatintyö

2016

30 Sivua, 6 kuvaa, 6 taulukkoa

Työn tarkastaja(t): Dosentti Tuomas Eerola

Hakusanat: annotointi, segmentointi, kuvankäsittely, MATLAB

Työn tavoitteena oli tuottaa MATLAB-pohjainen merkitsemistyökalu kuvien manuaaliseen segmentointiin. Ohjelmalla segmentoitujen kuvien on tarkoitus toimia opetusdatana oppivien konenäköalgoritmien kouluttamisessa. Ohjelma toteutettiin ohjelmistotuotannon vesiputousmallin mukaisesti. Käyttäjä pystyy segmentoimaan ja luokittelemaan kuvista objekteja annettujen piirtoelementtien avulla sekä tallentamaan segmentointisa tiedostoon. Käyttäjä pystyy muokkaamaan kuvan kontrastia, ajamaan MATLAB:in reunantunnistusfunktiota ja kohinanpoistofunktioita. Toteutettuina piirtoelementteinä ovat vapaa monikulmio, suorakulmia sekä ympyrä. Toteutettuina reunantunnistusfunktioina ovat Canny, Sobel, Prewitt, Robert ja LoG (Laplacian of Gaussian). Toteutettuina kohinanpoistofunktioina ovat Wiener- ja mediaanisuodatus.

ABSTRACT

Lappeenranta University of Technology
Faculty of Technology Management
Degree Program in Information Technology

Osku Grönberg

Annotation tool for manual image segmentation

Bachelor's Thesis

2016

30 pages, 6 figures, 6 tables

Examiner(s): Docent Tuomas Eerola

Keywords: annotation, segmentation, image processing, MATLAB

The purpose of the work was to create a MATLAB based image annotation tool. The purpose of the segmented images is to be used as a teaching data for learning algorithms. The project was executed according to the software engineering waterfall model. The user can segment and label objects from images with given shapes and save the annotations. The user can adjust the image contrast, run MATLAB given edge detection and image noise removal functions. Implemented segmentations shapes are free polygon, circle and a rectangle. Implemented edge detection functions are Canny, Sobel, Prewitt, Robert and Laplacian of Gaussian. Implemented noise removal functions are Wiener and median filters.

SISÄLLYSLUETTELO

1 JOHDANTO 6

1.1 TAUSTA 6

1.2 TAVOITTEET JA RAJAUKSET 7

1.3 TYÖN RAKENNE 7

2 TEORIA JA KIRJALLISUUSKATSAUS 8

2.1 KONENÄKÖJÄRJESTELMÄ 8

2.2 OPPIVAT ALGORITMIT 8

2.3 KUVAN SEGMENTOINTI JA REUNANTUNNISTUS 8

2.4 KUVAN KONTRASTIN MUOKKAUS 10

2.5 KUVAN KOHINANPOISTO 10

2.6 OHJELMISTOTUOTANTO 11

2.7 OLEMASSA OLEVIA ANNOTOINTITYÖKALUJA 11

3 RATKAISUMENETELMÄT 12

4 KÄYTÄNNÖNOSUUS 13

4.1 MÄÄRITTELYVAIHE 13

4.2 SUUNNITTELUVAIHE 17

4.3 TOTEUTUSVAIHE 17

4.4 TESTAUSVAIHE 23

5 MERKITSEMISTYÖKALU 25

6 JOHTOPÄÄTÖKSET 28

LÄHTEET 29

SYMBOOLI- JA LYHENNELUETTELO

GUIDE	Graphical User Interface Development Environment
LoG	Laplacian of Gaussian
RUP	Rational Unified Process

1 JOHDANTO

1.1 Tausta

Tämän kandidaatintyön tavoitteena oli tuottaa piirtotyökalu kuvien manuaaliseen segmentointiin, eli kuvissa olevien objektien merkitsemiseen. Työkalun dokumentointi on sisällytetty tähän raporttiin. Piirtotyökalun tuli olla helposti laajennettavissa, jotta sitä voidaan soveltaa tulevaisuudessa erilaisissa tutkimuksissa. Esimerkkitutkimuksina tästä ovat saimaannorppien automaattinen tunnistaminen kuvista [1] ja nanopartikkelien määrään laskeminen mikroskooppikuvasta [2].

Koneoppiminen on tietotekniikan osa-alue. Sen tarkoituksena on kehittää algoritmeja, jotka oppivat tekemään johtopäätöksen automaattisesti. Kuvien segmentointi liittyy konenäköön, jossa on usein tarkoituksena tunnistaa kuvasta tiettyjä objekteja. Käytännössä nämä ongelmat ratkaistaan erilaisten ohjatun oppimisen algoritmien avulla. Nämä algoritmit tarvitsevat usein valmiiksi segmentoitua testaus- ja opetusdataa, jolla algoritmi oppii tunnistamaan eri kuvista objekteihin liittyviä toistuvuuksia. Tämän kandidaatintyön tarkoituksena on ohjelmoida työkalu juuri näiden algoritmien opetusdatan luomiseen. Opetusdataa eli valmiiksi segmentoituja kuvia käytetään esimerkkeinä menetelmälle, jotta se oppii tekemään segmentoinnin itsenäisesti. Ohjelmalla luotua dataa voidaan käyttää myös muihin tarkoituksiin, kuten esimerkiksi testaamaan itsenäisesti kuvia segmentoivan ohjelman suorituskykyä. Ohjelmaa ja sen funktioita pystytään käyttämään pohjana ohjelmille, jotka segmentoivat itsenäisesti kuvista objekteja, esimerkkinä ohjelma nanopartikkelien laskemiseen.

Segmentointi tarkoittaa kuvan jakamista segmentteihin eli osiin. Tarkoituksena on yksinkertaistaa kuvaa sen analysoimisen helpottamiseksi. Yleensä segmentointia käytetään kappaleiden ja kappaleiden rajojen etsimiseen. Esimerkiksi tutkimuksessa, jonka tarkoituksena on laskea kuvasta nanopartikkelien määrää, pystytään valitsemaan kuvasta nanopartikkeleja vastaavat kuvapisteet, koska ne eroavat varsin selvästi taustan väristä. Tämän jälkeen kuvasta tulisi olla segmentoituna objekteja, jotka sisältävät yhden tai useamman nanopartikkelin. Ongelmaksi muodostuu päällekkäisten nanopartikkelien laskeminen ja tämä saattaa tilanteesta riippuen olla ihmisellekin hankalaa. Segmenttien pinta-alojen perusteella pystytään esimerkiksi arvioimaan kuvan nanopartikkelien määrää, mutta tämä ei vastaa niiden laskemista.

Ohjelma tuotetaan ohjelmistoprojektina. Ohjelman tuottamisessa pyritään noudattamaan pelkistettyä mallia Winston W. Roycen ohjelmistotuotannon vesiputousmallista [3]. Tällä menetelmällä pyritään varmistumaan tarvittavien dokumenttien luomisesta, sekä ohjelman yleisestä laadullisesta lopputuloksesta. Ohjelman tuottamiseen ei liity minkäänlaisia liiketoiminnallisia tavoitteita, eikä prosessi sisällä käytännössä mitään markkinointiin liittyvää.

1.2 Tavoitteet ja rajaukset

Kandidaatintyön tavoitteena on tuottaa MATLAB-ympäristöön merkitsemistyökalu kuvien manuaaliseen segmentointiin, käyttäen MATLAB:in GUIDE-kehitysympäristöä (Graphical User Interface Development Environment) [4]. Ohjelman tulee sisältää myös kattava dokumentointi sen ominaisuuksista. Opetusdatan luomista pidetään yleisesti hyvin työläänä ja epämukavana työnä. Kun tällaiseen tehtävään luodaan apuväline, tulee ottaa huomioon erityisesti ohjelman lopullisen käyttäjän tarpeet ja oletetut taidot.

Käyttäjän tulee pystyä avaamaan ohjelmaan kuvia ja merkitsemään kuviin objektien ääriviivoja. Ääriviivat merkitään lisäämällä kuvaan piirtoelementtejä. Käyttäjän tulisi myös pystyä muokkaamaan kuvaan lisäämiään piirto-elementtejä, kuten liikuttamaan monikulmioiden pisteitä ja poistamaan piirtoelementtejä. Käyttäjän tulee pystyä myös tallentamaan luodut merkinnät tiedostoon ja lataamaan merkinnät tiedostosta muokattaviksi.

Ohjelmaan liittyy myös hyödyllisiä lisäominaisuuksia, joita toteutetaan ohjelmaan projektin resursseista riippuen. Lisäominaisuuksia ovat kaikkien kansiossa olevien kuvien lataaminen, erimuotoisten valmiiden piirtoelementtien luominen, kuten suorakaide, ja lisäksi kuvan kontrastin kasvattaminen objektien rajojen etsinnän helpottamiseksi. Muita ominaisuuksia ovat myös semi-automaattiset segmentointimenetelmät sekä maskien luominen kuvaan. Lisäksi MATLAB sisältää kuville valmiita suodattimia, ja näiden suodattimien käyttäminen saattaa helpottaa segmentointia huomattavasti. Tietenkin käyttäjä pystyy ajamaan kuvan suodattimen läpi ennen kuin se avataan ohjelmassa, mutta ohjelman soveltamilla suodattimilla voidaan kuitenkin säästää aikaa.

Ohjelmakoodin rakenne tulee olla selkeä ja hyvin kommentoitu, jotta ohjelmaa pystytään helposti soveltamaan uusissa käyttötapauksissa. Esimerkiksi ohjelman käyttöliittymän rakenne tulee olla järjestelmällinen, jotta siihen pystytään lisäämään uusia ominaisuuksia. Toisin sanoen ohjelma tulee toteuttaa hyvien ohjelmointitapojen mukaisesti.

1.3 Työn rakenne

Kappale 2 käy läpi teoriaosuuteen liittyen teoriaa segmentoinnista, kohinanpoistosta, kontrastin muokkauksesta ja reunantunnistuksesta sekä ohjelmistotuotannosta.

Kappale 3 sisältää ratkaisumenetelmät ja kertoo miten ohjelma toteutus suunniteltiin.

Kappale 4 sisältää käytännönosuuden johon on dokumentoitu mitä suunniteltiin ja tehtiin. Pyritään vastaamaan myös kysymyksen miksi.

Kappale 5 käy läpi johtopäätöksiin liittyen yhteenvetoja lopputuotoksesta sekä käytetyn menetelmän arviointia.

2 TEORIA JA KIRJALLISUUSKATSAUS

2.1 Konenäköjärjestelmä

Tyypillisesti konenäköjärjestelmällä on viisi päävaihetta, jotka ovat kuvan hankinta, esikäsittely, segmentointi, piirreirrotus ja tunnistus. Kuvan hankinta tarkoittaa kuvan datan hankintaa, esimerkiksi kamerasta tai skannerista. Esikäsittely tarkoittaa kuvan käsittelemistä haluttujen objektien tunnistamisen helpottamiseksi, esimerkiksi kuvan kontrastin kasvattaminen tai kohinan poistaminen. Segmentointivaiheessa kuvasta rajataan ne alueet, joista ollaan kiinnostuneita. Piirreirrotus tarkoittaa kuvan segmentoitujen kohteiden analysointia, tarkoituksena on löytää kohteista piirteitä, kuten niiden koko, kaarevuus tai tummuus. Tunnistus vaiheessa segmentoidut alueet luokitellaan niistä löydettyjen piirteiden avulla määrättyihin luokkiin.[5]

2.2 Oppivat algoritmit

Oppivat algoritmit voidaan karkeasti jakaa kolmeen luokkaan, ohjatun oppimisen algoritmeihin, ei-ohjatun oppimisen algoritmeihin ja puoliohjattuihin oppimisen algoritmeihin. Pääsääntöisesti ohjatun- ja puoliohjatun oppimisen algoritmit käyttävät valmiiksi luokiteltua opetusdataa algoritmin opettamiseksi. Ohjatussa oppimisessa tarkoituksena on usein kuoluttaa luokittelija, joka luokittelee saamansa datan määrättyihin luokkiin mahdollisimman hyvin. Luokittelijan koulutussessioissa luokittelija saa syötedataa ja syötedataan liittyvää haluttua tulostedataa.[6]

Puoliautomaattisen ohjatun oppimisen algoritmit ovat usein käytännöllinen vaihtoehto. Nämä algoritmit käyttävät luokitellun opetusdatan lisäksi luokittelematonta opetusdataa luokittelijan kouluttamiseksi. Tämä on käytännöllistä, koska luokittelematonta opetusdataa on yleisesti paljon helpommin saatavissa ja suuremmissa määrin, kuin luokiteltua opetusdataa. Usein hyvän luokittelijan luominen täysin ilman luokiteltua opetusdataa on hyvin hankala ja ei-ohjatun oppimisen algoritmit eivät sovellu hyvin tehtäviin joissa on tarkoitus tunnistaa objekteja kuvista.[6]

2.3 Kuvan segmentointi ja reunantunnistus

Semanttisen segmentoinnin päätavoite on jakaa kuva osiin jotka vastaavat kuvassa oikeasti olevia objekteja. Useimmat semanttisen segmentoinnin metodit voidaan jakaa 3 luokkaan, jotka ovat globaalit metodit, reunoihin perustuvat metodit, sekä aluepohjaiset metodit. Yksinkertainen globaali metodi on kynnystää kuvaa eli jakaa kuva osiin esimerkiksi kirkkauden tai värin mukaan. Tarkoituksena on vähentää kuvasta kuvapisteiden värien määrää. Aluepohjaiset metodit ovat käytännössä hyvin lähellä globaaleja metodeita. Kuva jaetaan osiin esimerkiksi kirkkauksien tai värien perusteella. Tämän jälkeen segmenttejä lähdetään laajentamaan viereisin saman kirkkauksisiin tai värisiin osiin. Lopullisena erona globaaliin segmentointiin on, ettei kaikkia saman värisiä/kirkkauksisia osia luokitella samaksi segmentiksi. [7]

Reunantunnistus tarkoittaa matemaattisia menetelmiä joiden tarkoituksena on löytää digitaalisesta kuvasta pisteitä joissa kuvan kirkkaus muuttuu terävästi tai sisältää epäjatkuvuuskohtia. Olemassa olevilla menetelmillä voidaan löytää kuvista epäjatkuvuuspisteitä jotka vastaavat kuvassa eroja kuvan pintojen syvyyksissä, suunnissa, materiaalien ominaisuuksissa ja taustavalotuksessa [7][8]. Reunojen tehokas löytäminen on usein hankalaa, koska kuvat sisältävät sumentuneita kohtia johtuen esimerkiksi kuvan objektien syvyyseroista ja ei pistemäisten valonlähteiden luomista varjoista. Useimmat reunantunnistusmenetelmät voidaan jakaa kahteen luokkaan, gradientin paikallisten maksimien etsimiseen perustuvat menetelmät [9] sekä toisen asteen derivaattojen nollakohtien etsimiseen perustuvat menetelmät [10]

Sobelin reunantunnistusmenetelmä on yksi yksinkertaisimmista algoritmeista. Menetelmässä kuvapisteelle lasketaan derivaatan approksimaatio x- ja y-suunnassa, joista puolestaan lasketaan Pythagoraan lauseella gradientin muutoksen suuruus. Jos kuvapisteen gradientin arvo on riittävän suuri, metodi merkitsee kuvapisteen reunapisteeksi. Derivaatan approksimaatio voidaan suorittaa konvoluutio operaatiolla. Konvoloimalla alkuperäinen kuva A 3x3 matriiseilla, saadaan laskettua Gx ja Gy jotka sisältävät derivaattojen approksimaatiot jokaiselle kuvan A kuvapisteelle [11] :

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \quad ja \quad Gy = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * A,$$

gradientin G suuruus sitä vastaavalle kuvapisteelle voidaan laskea Pythagoraan lauseella

$$G = \sqrt{Gx^2 + Gy^2}.$$

Prewitt reunantunnistusoperaatio on kuten Sobel operaatio, mutta se käyttää eri konvoluutio matriiseja derivaattojen approksimoimiseksi [12] :

$$Gx = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} * A \quad ja \quad Gy = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} * A.$$

Roberts reunantunnistusoperaatio on kuten Sobel operaatio, mutta se käyttää eri konvoluutio matriiseja derivaattojen approksimoimiseksi [13] :

$$Gx = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} * A \quad ja \quad Gy = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} * A.$$

Canny on yleisesti tunnettu ja hyväksytty reunantunnistusmenetelmä. Kuten Sobel-operaatio, menetelmä laskee kuvapisteille intensiteetistä gradientin. Gradienttia verrataan raja-arvoihin mahdollisten reunapisteiden löytämiseksi. Reunapisteistä poistetaan lopuksi heikon varmuuden pisteitä, jos niitä ei pystytä yhdistämään varmoihin reunapisteisiin. [14]

LoG (Laplacian of Gaussian) reunantunnistus perustuu kuvapisteiden intensiteettifunktion toisen derivaatan nollakohtien etsimiseen. Intensiteettifunktion toisen derivaatan nollakohdat kuvastavat intensiteettifunktion ensimmäisen derivaattafunktion lokaaleja maksimi ja minimikohtia, eli kohtia joissa intensiteettifunktio muuttuu voimakkaimmin. Valitettavasti toisen derivaatan nollakohdat ovat hyvin herkkiä kuvan kohinalle, joten kohinan poistaminen on usein tarpeellista. [9]

2.4 Kuvan kontrastin muokkaus

Kontrasti tarkoittaa kuvan kuvapisteiden värien tai kirkkauksien vaihtelua siten että kuvassa olevat objektit erottuvat toisistaan. Kontrastin kasvatuksella voidaan saada kuvassa olevia objekteja erottumaan toisistaan selvemmin. Kuvan kontrastin mittaamiselle ei ole olemassa yleistä standardia. Kuvan kontrastia voidaan muokata monella tapaa ja metodit voidaan yleisesti jakaa kahteen osaan, suoriin ja epäsuoriin metodeihin. Histogrammien tasoittaminen on eräs epäsuora metodi kontrastin muokkaamiselle. Tässä metodissa kuvapisteet jaetaan histogrammeihin valitun parametrin mukaan, esimerkiksi kirkkauden. Tämän jälkeen pyritään muokkaamaan kuvan histogrammeissa olevia kuvapisteitä siten, että histogrammien pylväistä tulee yhtä korkeita. [15]

Paikallinen ja globaali kuvan kontrastin venytys on suora metodi kuvan kontrastin muokkaamiselle. Metodissa venytetään kuvapisteiden intensiteettiarvoja haluttuun mittaan. Kuvapisteille lasketaan uudet arvot uudessa mitassa kuvapisteen oman intensiteetin ja alkuperäisen kuvan minimi- ja maksimi-intensiteettien avulla. Kontrastin venytyksellä pyritään tekemän kuvan tummien ja kirkkaiden alueiden sisällöstä selvempiä. [16][17]

2.5 Kuvan kohinanpoisto

Kuvan kohina tarkoittaa satunnaista värien tai kirkkauden muuttumista kuvapisteissä, riippumatta kuvan sisällöstä. Kohinaa voi syntyä kuviin esimerkiksi sensorien tai digitaalisten kameroiden kennojen sähköisten signaalien seurauksena. Kohina on ei toivottu kuvien ottamisen sivutuote. Kohina tarkoitti alunperin ja tarkoittaa yhä myös radiosignaalien aallonpituuksien satunnaista muuttumista, joka aiheuttaa radion tuottaman äänen staattista sivuääntä. Kuvien kohina voidaan jakaa useaan eri kategoriaan. Gaussinen kohina tarkoittaa tasaista satunnaista kuvapisteiden kirkkauden tai värien muuttumista. Kohina on riippumaton itse kuvasta. Suola-pippuri kohina aiheuttaa kuvan kirkkaisiin alueisiin satunnaisia tummia kuvapisteitä ja kuvan tummiin alueisiin satunaisia kirkkaita kuvapisteitä, ikään kuin kuvan päälle oltaisiin ripoteltu suolaa tai pippuria. Poissonin kohina tekee kuvien tumista alueista rakeisia.

Kohinanpoisto tarkoittaa matemaattisia menetelmiä kuvan kohinan vähentämiseksi. Kohinaa poistamalla voidaan kuvasta saada paremmin selvää, sekä helpottaa sen analysointia. Mediaanisuodatuksella voidaan poistaa kuvasta kohinaa. Tässä metodissa käydään kuva läpi kuvapiste kerrallaan. Kuvapisteille lasketaan uusi arvo laskemalla kuvapisteen ja sen naapuruston kuvapisteiden mediaani. Naapurusto on erikseen määrätty alue. Yksinkertaisin naapurusto on kaikki kuvapisteen viereiset kuvapisteet. Naapurusto voi olla mielivaltaisen muotoinen, ja suodatuksen lopputulos riippuu myös naapuruston muodosta. Mediaanisuodatus on tehokas keino poistaa kuvasta suola-pippuri kohinaa. [18]

Wiener-suodatus on toinen esille nostettava keino poistaa kuvasta kohinaa. Tämä menetelmä pyrkii poistamaan kuvasta tasaista gaussista kohinaa ja kuvan sumentumia. Sumentumien poistaminen tarkoittaa käytännössä kuvan käänteistä suodatusta sumentuman aiheuttaneen kuvitteellisen suodattimen suhteen. Käänteinen suodatus on kuitenkin hyvin herkkä gaussiselle kohinalle. Wiener-suodatus pyrkii käänteisen suodatuksen ja gaussisen kohinan poistamisen parhaaseen suhteeseen. Wiener-suodatus on lineaarinen approksimaatio alkuperäisestä kuvasta ja se on pienimmän neliösumman mielessä optimaalinen. [19]

2.6 Ohjelmistotuotanto

Ohjelmistotuotannon puolelta löytyy paljon teoriaa ohjelmien tehokkaaseen tuottamiseen. Kuitenkin näiden prosessien pääsovelluskohteet ovet suuret projektit joita yksi ihminen ei pysty suorittamaan. Näitä menetelmiä ja prosesseja pystytään kuitenkin soveltamaan pienemmissä töissä. Ohjelman tuottamisprosessi jaetaan osiin suurimmassa osassa menetelmistä. Näitä menetelmiä kutsutaan vaihejakomalleiksi. Esimerkiksi vesiputousmalli sisältää usean vaiheen ja seuraavaan vaiheeseen siirrytään vasta kun edellinen vaihe on valmis. Prototyyppimallissa ohjelmasta luodaan ensin käyttöliittymä johon sen jälkeen aletaan lisäämän ominaisuuksia. Menetelmän etuna on että asiakas saa nähdä hyvin aikaisessa vaiheessa miltä ohjelma tulee näyttämään. RUP (Rational Unified Process) perustuu peräkkäisiin suunniteltuihin iteraatioihin ja kehittämien itse ohjelman jakautuu aloitus-, tarkennus-, rakennus- ja käyttöönottovaiheeseen. Pienemmät projektit suosivat usein ketteriä menetelmiä. Nämä menetelmät korostavat nopeita iteraatiosyklejä ja muutosten hallintaa. Tarkoituksena on korjata ja muuttaa kehityksen suuntaa mahdollisimman aikaisessa vaiheessa ohjelman tuottamista. [20]

2.7 Olemassa olevia annotointityökaluja

LabelMe [21] on online-ohjelma kuvien manuaaliseen segmentointiin, jonka tarkoitus on luoda kattava kuvapankki kuvista ja niiden sisältämistä objekteista. Kuvapankkia on tarkoitus käyttää konenäön tutkimisessa.

Szoter [22] on online merkitsemistyökalu kuvien manuaaliseen segmentointiin. Ohjelman käyttämiä piirtoelementtejä ovat janat, suorakaiteet, ympyrät ja nuolet. Käyttäjä pystyy lisäämään tekstejä piirtoelementeille. Ohjelman käyttämiä kuvankäsittelymenetelmiä ovat kuvan skaalaus, pyöritys ja leikkaus.

Labellmg [23] on Python-pohjainen annotointityökalu kuvan objektien rajaamiseksi suorakulmioilla. Käyttäjä pystyy rajaamaan kuvista objekteja suorakulmioilla ja nimeämään niitä.

3 RATKAISUMENETELMÄT

Ohjelma toteutettiin ohjelmistoprojektina käyttäen Winston W. Roycen alkuperin luomaa vesiputousmallia. Mallin sisältämät vaiheet ovat järjestyksessä: vaatimusten määrittely, suunnittelu, toteutus, integraatio, testaus, asennus, ylläpito [3]. Ottaen huomioon toteutettavan ohjelman koon, mallista toteutettiin vain vaatimusmäärittely-, suunnittelu-, toteutus- ja testausvaihe. Käytännössä vaiheita suoritettiin päällekkäin samanaikaisesti. Esimerkiksi testausta suoritettiin ennen kuin kaikkea ohjelmointia oli saatu valmiiksi. Lisäksi vaiheiden aikana suoritettiin dokumentointia ohjelmasta. Vaiheiden samanaikainen toteuttaminen on tyypillistä vesiputousmallissa.

Koska ohjelman tuottamisen asiakasvaatimukset olivat selvät, ohjelman tekeminen voitiin aloittaa suorittamalla pikainen vaatimusmäärittely ohjelmalle. Koska ohjelma ei ollut kovinkaan kompleksinen ja yksi ihminen pystyi hahmottamaan sen toiminnan, ei ollut tarvetta massiiviselle vaatimusmäärittelylle ja suunnittelulle. Ohjelmointia suunniteltiin lähdettävän toteuttamaan top-down metodilla, ensin luotiin graafisen käyttöliittymän ranka johon alettiin sen jälkeen lisäämään erinäköisiä funktioita ja ominaisuuksia täyttämään ohjelman vaatimukset. Lopuksi suoritettiin ohjelmalle vielä testausta jotta kaikki ominaisuudet varmasti toimivat halutulla tavalla.

GUIDE-kehitysympäristöstä löytyy paljon tutoriaaleja MathWorks:in sivuilta, joissa käydään läpi graafisten käyttöliittymien luomista. MathWorks:in sivuilta löytyy myös kattava dokumentointi MATLAB:in funktioista sekä useita demoja. [4]

4 KÄYTÄNNÖNOSUUS

Toteutettavat vaiheet olivat tiedonkeruu, ohjelman määrittäminen, suunnittelu, toteutus, testaus ja dokumentointi.Ohjelman tekeminen aloitettiin kattavasta tiedonkeruusta. Tiedonkeruuta suoritettiin vielä lisäksi kaikkien muiden vaiheiden aikana ja suuri osa ajasta käytettiin kokonaisuudessaan tiedonkeruuseen.

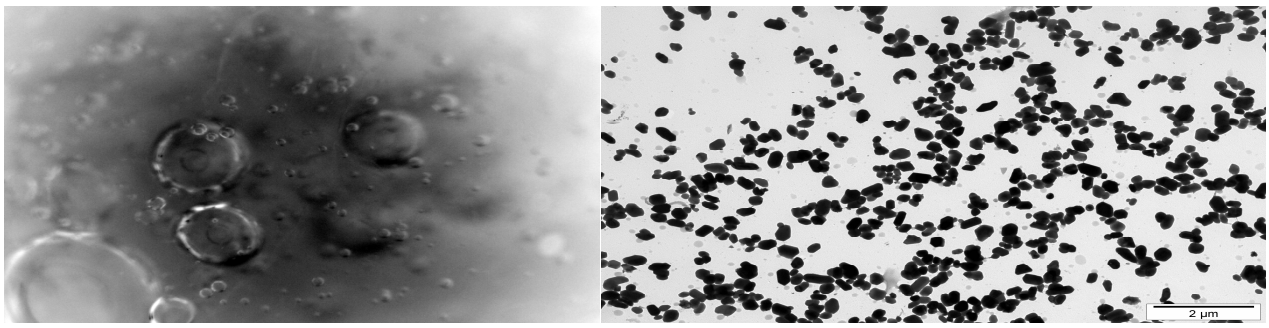
4.1 Määrittelyvaihe

Ohjelman tuottamiseen ei liittynyt konkreettisia liiketoiminnallisia tavoitteita. Ohjelma meni asiakkaalle tutkimuskäyttöön, eikä ohjelman tuottamiseen liittynyt minkään tasoista markkinointia. Asiakas ei ollut ilmoittanut ohjelmalle minkäänlaisia rajoitteellisia vaatimuksia, esimerkiksi ohjelman koon suhteen, mutta tämän kokoisessa projektissa yleisiä ohjelmallisia rajoitteita ei käytännössä tarvita. Esimerkiksi projektille määrätty aika piti huolen siitä ettei ohjelmasta tullut liian suurikokoista..

Pystyimme olettamaan, että ohjelman käyttäjäryhmä hallitsee ohjelman käyttöympäristön hyvin, eli käyttäjäryhmällä on paljon kokemusta MATLAB:in käyttämisestä. Valmis ohjelma sisälsi paljon MATLAB:in valmiita funktioita, joista löytyy dokumentointi MATLAB:in puolelta. Kuvien annotointia yleisesti pidetään hyvin ikävä ja vaivalloisena työnä joten olisi mukavaa jos annotointiin käytetty työkalu olisi laadukas ohjelma jota on tehokas käyttää. Tästä syntyi motivaatio merkitsemistyökalun tehokkaan käytön suunnitteluun. Projektin asiakas oli esittänyt esimerkkiohjelman siitä miltä valmis ohjelma voisi näyttää.

Ohjelmaa määritettäessä oli tarpeellista tarkastella ohjelman lopullista käyttötarkoitusta ja käyttötapauksia. Kuva 1 (a) liittyy tutkimukseen, jonka tarkoituksena on laskea kuvasta kuplien määrä ja selvittää kokojakauma [24]. Kuvan 1 objektien segmentoimiseen sopii vapaa monikulmio. Kuplat ovat kuitenkin pyöreitä joten monikulmioon tulee useita kulmia. Kenties kolmen merkityn pisteen kautta piirretty ympyrä on nopeampi merkitä ja soveltuu segmentoimiseen paremmin.

Kuva 1 (b) liittyy tutkimukseen, jonka tarkoituksena on approksimoida kuvasta kaikkien nanopartikkelien määrää [2]. Partikkelit ovat osittain päällekkäin ja täten niiden absoluuttinen laskeminen on hyvin hankalaa. Kuvan 2 objektien segmentoimiseen soveltuu vapaa monikulmio. Lisäksi kuvaan voidaan merkitä alueita, joista testaavaa algoritmia ei rankaista jos se laskee partikkelien määrää väärin, koska useassa kohtaa kuvaa partikkelien määrää ei pystytä määrittämään tarkasti edes asiantuntijan toimesta.



(a) (b)

Kuva 1. Konenäkötutkimuksessa käytettyjä kuvia:
(a) Kuplakuva sellunvalmistusprosessista, (b) nanopartikkeleja

Kuva 2 (a) liittyy tutkimukseen, jonka tarkoituksena on tunnistaa kuvasta Saimaannorppa yksilö. Tunnistaminen tapahtuu norpan turkin kuvioiden perusteella. Tunnistamisen helpottamiseksi norppa halutaan segmentoida mahdollisimman tarkasti [1]. Koska norpat voivat olla kuvissa useissa asennoissa ja vain osittain näkyvissä, ei norppia pysty tunnistamaan pelkästään norpan ääriviivojen avulla. Norppa voidaan esimerkiksi rajata kuvasta suorakaiteen avulla ja pyrkiä selvittämään suorakaiteen sisällön parametrien avulla, löytyykö kuvasta norppaa.

Kuva 2 (b) liittyy tutkimukseen liikennemerkkien automaattiseen tunnistamisen ja kunnon arvioimiseen. Tarkoituksena on, että algoritmi automaattisesti tunnistaa kuvasta objekteja liikennemerkeiksi, jos sellaisia löytyy kuvasta, ja arvioi niiden kuntoa [25]. Käytännössä algoritmi tunnistaisi objekteja liikennemerkeiksi ja luokittelisi ne arvioidun kunnon mukaan. Liikennemerkin kunnon arvioiminen tarkoittaa että oppiva algoritmi tarvitsee sekä hyväkuntoisiksi ja huonokuntoisiksi luokiteltuja liikennemerkkejä opetusdatassaan. Liikennemerkeillä on määrättyjä muotoja, joten niiden tunnistamisessa voidaan hyödyntää merkin ääriviivoja. Tällöin tarkalla kappaleen ääriviivojen mukaan segmentoinnilla voidaan tehostaa merkkien tunnistamista. Lisäksi koska kyseessä on liikennemerkki, on uskottavaa väittää ettei se ole kuvassa vain osittain näkyvillä. Liikennemerkin muoto on yksi tärkeä merkin tunnistamiseen liittyvä ominaisuus. On hankala sanoa kannattaako merkkejä segmentoida kuvista vain rajaamalla ne suorakaiteen sisään vai tarkasti segmentoiden merkkien ääriviivojen mukaan. Jälkimmäisessä tapauksessa kaikki paitsi pyöreät merkit voidaan segmentoida kuvasta helposti vapaalla monikulmiolla. Kaikki pyöreät merkit eivät varmasti ole kuvassa suorassa. Tästä seuraa mahdollinen tarve ellipsille. Kuitenkin ympyrä todennäköisesti riittää hyvin suurelle osalle merkeistä, jos merkkejä on tarvetta segmentoida ääriviivojen mukaan.

(a) (b)

Kuva 2. Konenäkötutkimuksessa käytettyjä kuvia:
(a) Norppa, (b) Liikennemerkkejä

Suuri osa ohjelman käyttötapauksista ovat selviä ohjelman vaatimusten pohjalta. Ohjelman vaatimusten määrittäminen on esitetty taulukoissa 1-2.

Taulukko 1. Ohjelman vaatimuksia

Vaatimus	Kuvaus	Prioriteetti 1=hyvin tärkeä; 2=tärkeä 3=valinnainen
Ohjelma MATLAB ympäristöön	Ohjelma toteutetaan MATLAB:illa	1
Graafinen käyttöliittymä	Ohjelmaa käytettään täysin graafisen käyttöliittymän kautta	2
Kuvien lataaminen	Käyttäjä pystyy avaamaan ohjelmaan kuvia	1
Piirtoelementtien tallennus	Käyttäjä pystyy tallentamaan kuvaan lisäämänsä piirtoelementit.	1
Pisteiden lisäys	Käyttäjä pystyy merkitsemään kuvaan datapisteitä	1
Valmiit piirtoelementit	Käyttäjä pystyy piirtämän myös valmiita piirtoelementtejä kuten suorakaiteita	3
Piirtoelementtien lataus	Käyttäjä pystyy lataamaan tallentamansa piirtoelementit	1
Piirtoelementtien muokkaus	Käyttäjä pystyy muokkaamaan lisäämiään/lataamiaan piirtoelementtejä	1
Suodattimet	Ohjelmalla voidaan näyttää kuva suodattimen läpi (MATLAB:issa esimerkiksi valmiina Canny-funktio kuville)	3
Kansiossa olevien kuvien samanaikainen avaus	Käyttäjä pystyy lataamaan ohjelmaan kaikki kansiossa olevat kuvat.	3
Kuvan kontrastin kasvattaminen	Käyttää pystyy muokkaamaan kuvan kontrastia.	3
Semi-automaattiset segmentointimenetelmät	Käyttäjä voi ajaa semi-automaattisia segmentointimenetelmiä kuvalle	3
Luokitteleminen	Käyttäjä pystyy nimeämän eristämiään segmenttejä.	3
Usean kuvan käsittely	Ohjelmassa pystyy olemaan avoimena usea kuva samaa aikaa.	1
Kuvien välillä vaihtaminen	Käyttäjä pystyy vaihtamaan usean avatun kuvan välillä, mitä kuva haluaa muokata.	1
Kuvien sulkeminen.	Käyttäjä pystyy sulkemaan avattuja kuvia.	2
Piirtoelementtien värin vaihtaminen.	Käyttäjä pystyy globaalisti muuttamaan kaikkien piirtoelementtien väriä.	2
Kohinanpoisto	Ohjelmalla pystyy poistamaan kuvista kohinaa MATLAB:in kohinanpoistofunktioilla.	3
Värikartan valinta	Käyttäjä pystyy muuttamaan näytetyn kuvan värikarttaa.	3
Merkintöjä sisältämättömien alueiden merkitseminen	Käyttäjä pystyy merkitsemään kuvaan alueita jotka luokiteltu merkintöjä sisältämättömiksi	2

Tauluko 2. Ohjelman vaatimuksia

Vaatimus	Kuvaus	Prioriteetti 1=hyvin tärkeä; 2=tärkeä 3=valinnainen
Ei päällekkäisiä pisteitä.	Käyttäjä ei pysty piirtämään/siirtämään pistettä niin että se on toisen pisteen päällä.	2
Kuvien sulkemisen vahvistaminen.	Kysymysdialogi käyttäjälle kun hän yrittää sulkea kuvan.	3
Varoitus kun yritetään tallentaa ei mitään.	Kysymysdialogi käyttäjäll, kun hän yrittää tallentaa merkintöjä kuvasta, jossa ei ole merkintöjä.	3
Istuntojen tallennus.	Käyttäjä pystyy tallentamaan istuntoja tiedostoihin.	3
Istuntojen lataus.	Käyttäjä pystyy lataamaan ohjelmaan aikaisempia istuntoja.	3
Ohjelma pitää kuvan zoomauksen kuvan piirtoelementtejä muokatessa.	Käyttäjän muokatessa piirtoelementtejä, tulee kuva piirtää usein uudestaan. Ohjelma piirtää uuden kuvan alkuperäisen kuvan zoomauksen mukaan.	3
Kuvan kontrastin kasvattaminen.	Käyttäjä pystyy muokkaamaan kuvan kontrastia.	3
Kuvan palauttaminen.	Ohjelmassa toiminnallisuus, jolla kuva voidaan palauttaa alkuperäiseen muotoonsa.	3
Kaikkien merkintöjen poistaminen kuvasta.	Ohjelmassa toiminnallisuus kaikkien kuvaan tehtyjen merkintöjen poistamiseen.	3
Käyttöliittyvä venyvä.	Käyttäjä voi uudelleenmitoittaa käyttöliittymän kokoa.	2
Ohjelma vakaa	Ohjelma ei saa joutua normaalissa käytössä virhetilaan.	1
Ohjelman funktiot riittävän optimoituja.	Ohjelma ei saa sisältää käyttäjälle suoraan käyttökokemusta haittaavia huonosta optimoinnista johtuvia ominaisuuksia.	2
Kontrastin muuttaminen	Käyttäjä pystyy muuttamaan kuvan kontrastia.	3
Luokkien valinta valinta	Käyttäjä pystyy valitsemaan uudeksi luokaksi jo olemassa olevan luokan pudotusvalikosta.	2
Värit sidottuja luokkiin	Kaikki saman lluokan sisältävät piirtoelementit ovat samanvärisiä	3
luokilla ja väreillä pudotusvalikot	Merkintöjen luokilla ja värivaihtoehdoilla omat pudotusvalikot.	2
Värien ja luokkien valikoiden päivittyminen.	Värien pudotusvalikon arvo päivttyy valitun luokan mukaan ja luokan värit päivittyvät värien pudotusvalikon mukaan.	3
Datapisteiden tulostus Excel-taulukkoon	Ohjelma pystyy tallentamaan merkinnät luettavaan muotoon Excel-taulukkoon.	3

4.2 Suunnitteluvaihe

Merkitsemistyökalua lähdettiin tuottamaan Top-Down metodilla, jossa ohjelmalle ensin luodaan yleinen infrastruktuuri johon lisätään erilaisia ominaisuuksia yksi kerrallaan [19]. Tässä tapauksessa ohjelmalle toteutettiin ensin käyttöliittymä, jonka jälkeen käyttöliittymän objekteille luotiin toiminnallisuuksia. Tämän metodin hyvänä puolena oli että asiakkaalle pystytään hyvin aikaisessa vaiheessa näyttämään miltä lopullisen ohjelman käyttöliittymä tulee näyttämän. Ohjelman toteutus aikataulutettiin siten, että päivässä ohjelmaan lisättiin aina jokin toimiva ominaisuus käyttöliittymään ja päivän lopussa ohjelma toimi toteutettujen ominaisuuksien puolesta. Tällöin ohjelmasta oli olemassa aina päivän lopuksi toimiva kokonaisuus, josta oli hyvä jatkaa ohjelman kehittämistä. Tämänlainen toteutus muistuttaa suuresti ketteriä menetelmiä, joissa käytännössä tehdään sama asia, mutta vain suuremmassa mittakaavassa. Ketterissä menetelmissä ohjelmaa tuotetaan aina noin 24 tunnin pituisissa pyrähdyksissä, ja tarkoitus on saada ohjelma pyrähdyksen lopuksi aina toimivaan muotoon [19]. Hyvänä puolena tässä menetelmässä on, että ohjelmoijien työmoraali pysyy korkealla. Tämä johtuu siitä että pyrähdyksen lopuksi he pystyvät näkemän toimivasta ohjelmasta miten ohjelmointityö on edistynyt.

4.3 Toteutusvaihe

Ohjelman tekeminen aloitettiin yksinkertaisen käyttöliittymän luomisesta. Ensimmäinen toteutettu funktionaalisuus oli kuvan avaaminen ohjelmaan ja tallentaminen muistirakenteeseen. Ohjelman tulee vaatimusten mukaan pystyä avaamaan usea kuva samaan aikaan. Tästä seurasi vaatimus siitä että ohjelmalla tulisi pystyä käsittelemän useaa kuvaa samanaikaisesti. Ohjelma ylläpitää globaalia listaa osoittimista ohjelmassa avattuihin kuviin. Lisäksi ohjelma pitää kirjaa avattujen kuvien määrästä, jota päivitetään aina kuvia avatessa ja suljettaessa.

Ohjelman käyttöliittymää luotaessa nousi esille kysymys siitä miten usean eri kuvan välillä vaihtaminen tulisi toteuttaa. Ensimmäisenä ideana oli luoda jokaiselle kuvalle ohjelmaan oma välilehti, mutta välilehtien luominen oli suhteellisen uusi MATLAB:in ominaisuus eivätkä välilehdet olisi välttämättä yhteensopivia vanhempien versioiden kanssa. Useiden kuvien välillä vaihtaminen päätettiin toteuttaa selainmaisella ratkaisulla, jossa käyttäjä pystyy siirtymään kuvien välillä nappuloiden ja pudotusvalikon avulla.

Esille nousi kysymys kuvien sulkemisesta. Tämä ominaisuus ei sisältynyt suoraan asiakasvaatimuksiin, mutta on kuitenkin yleisesti hyödyllinen toiminnallisuus. Tämä ominaisuus päätettiin toteuttaa. Kuvia sulkevasta funktiosta ei pitäisi jäädä kummittelemaan muistin turhia datarakenteita. Ohjelmointi MATLAB:lla on käytännössä korkeamman tason ohjelmointia Javalla, ja Java pitää varsin hyvin huolta siitä ettei viittaamattomia muuttujia jää kummittelemaan muistiin. Kuvaa suljettaessa ohjelma automaattisesti vaihtaa seuraavana ladattuun kuvaan, jos sellainen on olemassa.

Kaikkien kansiossa olevien kuvien lataaminen toteutettiin yksinkertaisesti antamalla käyttäjälle mahdollisuus valita useita eri kuvia vain yhden sijaan. Viimeisenä ladattu kuva asetetaan näkyviin. Käyttäjä pystyy halutessaan varsin vaivattomasti valitsemaan kaikki kansiossa olevat kuvat ja lataamaan ne. Tähän liittyi pohdintaa siitä miten tarpeellista on, että ohjelma sisältää ominaisuuden, jolla käyttäjä pystyy suoraan avaamaan kaikki kansiossa olevat kuvat. Kyseinen ominaisuus vaatisi käytännössä ohjelmaan oman funktionsa ja jos ohjelman toimintaa haluttaisiin automatisoida, voi syntyä tarve tälle ominaisuudelle. Kuitenkin ohjelmaa ei ollut lähtökohtaisesti suunniteltu toisten prosessien käytettäväksi, tämä oli varsin selvää jo siitä, että ohjelmalla tuli olla graafinen

käyttöliittymä. Pystyttiin toteamaan, ettei ohjelmassa ollut tarvetta erilliselle funktiolle, joka lataisi kaikki kansiossa olevat kuvat.

Lisättiin ohjelmaan ominaisuus, joka pitää kuvan zoomauksen, jos piirrettävä kuva on sama kuin korvattava kuva. Toisin sanoen, kun käyttäjä muokkaa kuvan piirtoelementtejä, hän ei menetä kuvalle asettamaansa zoomausta. Lisättiin pisteiden piirtämisfunktioon ominaisuus, joka estää pisteiden piirtämisen liian lähelle toisiaan, tässä etäisyyden tarkastuksessa käytetään suhteellista etäisyyttä, joten kuvan zoomaus ei vaikuta pisteiden lisäämiseen.

Lisättiin ohjelmaan toiminnallisuus kuvan piirtoelementtien tallentamiseen ja lataamiseen. Piirtoelementit tallennetaan mat-tiedostoon. Tiedoston muokattavaksi oletusnimeksi laitettiin kuvan nimi. Esimerkiksi image.jpg:n tallennettavien piirtoelementtien oletusnimeksi tulisi image.mat. Piirtoelementtien lataamisessa käyttäjä valitsee mat-tiedoston, jonka jälkeen ohjelma korvaa valitun kuvan piirtoelementit tiedoston sisältämillä piirto-elementeillä. Lisättiin myös kysymysdialogi käyttäjälle, kun hän yrittää tallentaa merkintöjä kuvasta, johon ei ole tehty merkintöjä. Käyttäjä saa valita dialogista, että piirtoelementtien tallennus perutaan tai että tallennus suoritetaan. Lisättiin kysymysdialogi käyttäjälle, kun hän yrittää sulkea kuvan, ettei käyttäjä sulje kuvaa vahingossa ja menetä kaikkia merkintöjään. Vaikka tapahtuma onkin epätodennäköinen, saattaa se olla varsin ikävä, jos kuvaan on tehty paljon merkintöjä.

Valitettavasti useiden merkintöjen samanaikainen lataaminen eri kuviin oli hyvin hankala toteuttaa toimivasti, koska ohjelman tulisi onnistuneesti spesifioida, mitkä merkinnät menisivät millekin kuvalle. Tämä tapahtuma sisälsi suuren määrän asioita, jotka voivat mennä pieleen. Vaihtoehtoisena ratkaisuna käyttäjälle voitiin tarjota mahdollisuus tallentaa ohjelman istunto tiedostoon ja ladata istunto tiedostosta. Tässä tapauksessa mahdollisten virhetilanteiden määrä oli suotavalla tasolla, mutta tallennettavasta tiedostosta tuli huomattavasti isompi, koska tiedosto sisälsi myös itse kuvat. Istuntojen tallentaminen oli kuitenkin todennäköisesti hyvä ominaisuus, koska se salli segmentoinnin jatkamisen sujuvasti siitä mihin se oli viimekerralla päätetty. Lisättiin ohjelmaan toiminnallisuus istuntojen lataamiseen ja tallentamiseen. Samalla tavalla kuin merkintöjen tallentaminen tapahtuu, tapahtuu myös istuntojen lataaminen ja tallentaminen. Ainoana erona on että mat-tiedostoon tallennetaan kaikkien kuvien merkinnät, itse kuvat ja valittu merkintöjen väri, sekä indeksi, jolla asetetaan oikea kuva näkyviin kuvia ladatessa.

Toteutettiin käyttöliittymään toiminnallisuus kuvan kontrastin muokkaamiselle. Käyttäjä valitsee ohjelman valikosta vaihtoehdon kuvan kontrastin muuttamiselle, jolloin ohjelma avaa käyttäjälle dialogin. Ohjelman valikkoon lisättiin vaihtoehto kuvan palauttamiseksi alkuperäiseen muotoon. Kuvan alkuperäisen muodon palauttaminen tapahtuu luodusta kuvan varmuuskopiosta. Ohjelman valikkoon lisättiin myös vaihtoehto reunantunnistus funktion ajamiseksi kuvalle. Kyseinen vaihtoehto avaa käyttäjälle dialogin josta käyttäjä sai valita mitä reunantunnistusmenetelmää hän haluaa käyttää. Vaihtoehtoina käyttäjälle ovat Canny, log, Prewitt, Roberts ja Sobel. Valinnan jälkeen ohjelma ajaa MATLAB:in reunantunnistusfunktion käyttäjän valitsemalla parametrilla.

Todettiin että kaikkien merkintöjen poistaminen kerralla on myös yleishyödyllinen ominaisuus, joten ohjelman päävalikkoon lisättiin vaihtoehto, jolla kaikki kuvaan tehdyt merkinnät voidaan poistaa. Ohjelmassa oli myös useita funktioita, jotka ottavat suoraan kirjoitettuja syötteitä käyttäjältä. Ongelmatilanteiden välttämiseksi lisättiin try-catch rakenteita näihin funktioihin. Esimerkkejä näistä funktioista olivat istunnon lataaminen käyttäjän määrittämästä tiedostosta, tai kuvan kontrastin kasvattaminen käyttäjän antamilla parametreilla. Käyttöliittymän kuvia näyttävän komponentin leveys-korkeus suhde muutettiin 16:9 suhteeksi. Käyttäjä voi halutessaan kuitenkin

mitoittaa uudelleen käyttöliittymän koon ja leveys-korkeus suhteen haluamakseen. Painikkeiden painaminen samalla kun kuvaan lisätään piirtoelementtejä pystyi aiheuttamaan ohjelmaan virhetilanteita. Tämän takia ohjelmaan lisättiin ominaisuus joka poistaa käyttöliittymän painikkeita käytöstä, samalla kun käyttäjä on lisäämässä piirtoelementtejä kuvaan.

Ohjelmaan lisättiin nappi, josta käyttäjä voi aloittaa piirtämään monikulmioita. Toteutettiin myös nappi, josta käyttäjä voi aloittaa piirtämän suorakaiteita. Luotiin tarkistus ohjelmaan ettei ohjelma hyväksyisi suorakaiteita joiden korkeus tai leveys ovat 0. Lisättiin nappi, josta käyttäjä voi vapaasti lisätä ja muokata merkintöjen luokkia ja värejä.

Merkintöjä muokkaavan napin toiminnallisuutta laajennettiin siten, että piirtoelementit pysyvät haluttuina piirtoelementteinä, esimerkiksi suorakulmio pysyy suorakulmiona vaikka sen yhtä kulmapistettä siirrettäisiin. Lisättiin ladatun kuvan nimi ja polku kuvan datarakenteeseen. Tallennettaessa kuvaan tehtyjä merkintöjä, ohjelma tallentaa myös valitun kuvan polun ja nimen. Käyttäjä voi valita usean merkintöjä sisältävän tiedoston ladattavaksi samaan aikaa. Valitut tiedostot avataan uusiksi kuviksi. Lisättiin pudotusvalikko josta käyttäjä voi valita värikarttoja mustavalkokuville. Toteutettuina vaihtoehtoina värikartalle olivat default, jet ja gray.
Ohjelma kirjoittaa näkyviin merkintöjen luokat merkinnän ensimmäisen kontrollipisteen luokse. Luotiin pudotusvalikko, jossa on lista kuvan piirtoelementtien luokista. Luotiin toiminnallisuus datapisteiden Excel-taulukkoon vientiin. Toteutettiin kohinanpoistofunktioiden valinta. Käyttäjä saa valita aukenevasta pudotusvalikosta wiener2()- ja medfilt2()-kohinanpoistofunktioiden väliltä.

Lähdettiin liikkeelle siitä ettei kuvassa saisi olla samalla luokalla kahta eri väriä. Kun luokan väriä muutetaan, muutetaan myös kaikkien samanluokkaisten piirtoelementtien väriä. Tähän sisältyi editLabel()-funktio ja värien pudotusvalikon muokkaaminen. Haluttiin myös että pudotusvalikosta luokkaa vaihdettaessa, muuttuisi värien pudotusvalikon arvo vastaamaan valitun luokan väriä. Tarkasteltuaan erinäköisiä kuvia, päädyttiin siihen lopputulokseen että ympyrä on varsin kätevä piirtoelementti. Tämän seurauksena toteutettiin ohjelmaan nappi, jonka avulla käyttäjä pystyy piirtämän ympyröitä kuvaan.

Ohjelman toteutuksen tarkka aikataulu on esitetty taulukoissa 3-5.

Taulukko 3. Ohjelman toteutuksen tarkka aikataulu

Päivämäärä	Mitä tehtiin	Huomioita
17.4.2016	Luotiin yksinkertainen käyttöliittymä. Toteutettiin päävalikkoon vaihtoehto kuvien lataamiseen. Luotiin työkalupalkki kuville joka sisältää perus MATLAB-funktioita.	viewImage()-funtio kuvien näyttämiselle. Kuvan data handles.data{i} rakenteessa, jossa i on kuvan indeksi. Työkalupalkissa kuvan zoomaus- ja vieritystyökalu.
18.4.2016	Luotiin kuville pudotusvalikko ja painikkeet kuvien välillä vaihtamiseksi kuvan työkalupalkkiin.	Pudotusvalikko ja painikkeet kutsuvat viewImage()-funktiota sopivilla parametreilla.
20.4.2016	Toteutettiin kuvia sulkeva nappi työkalupalkkiin. Lisättiin vaihtoehto ladata usea kuva samaan aikaan.	Kuvien sulkeminen kutsuu viewImage()-funktiota ja näyttää seuraavana listassa olevan kuvan. Useaa kuvaa ladattaessa ladataan näkyviin viimeisenä ohjelmaan ladattu kuva.
21.4.2016	Toteutettiin painike pisteiden piirtämiselle. Lisättiin valintalaatikko, jolla janoja voidaan piirtää automaattisesti. Muutettiin viewImage()-funktio myös piirtämään kuvan janat kuvaa näytettäessä. Luotiin pudotusvalikko, josta käyttäjä pystyy valitsemaan kaikkien piirtoelementtien värin.	Pisteet rakenteessa handles.data{i}.M nx2 matriisissa, jossa n on pisteiden määrä. Janat rakenteessa handles.data{i}.L 2x2xn matriisissa, jossa n on janojen määrä. Värien pudotusvalikossa punainen, sininen, valkoinen ja vihreä. getColor()-funktio hakee pudotusvalikon valitun värin kuvaa ladattaessa.
22.4.2016	Järjestettiin ohjelman käyttöliittymän elementtejä. Toteutettiin painike janojen piirtämiseen pisteiden välille. Luotiin funktio, joka tarkastaa ettei kuvaan piirretä useita kopioita samasta janasta.	Janoja piirrettäessä käyttäjä valitsee kaksi pistettä joiden välille haluaa piirtää janan. Valittua pistettä korostetaan kasvattamalla sen kokoa. isDupplicate()-funktio ottaa sisäänsä janan ja palauttaa totuusarvon siitä onko samanlainen jana jo olemassa. getMarking()-funktio ottaa sisäänsä pisteen ja palautta olemassa olevista pisteistä riittävän lähellä olevan pisteen, jos sellainen on olemassa.
23.4.2016	Lisättiin ominaisuus kuvan zoomauksen pitämiseksi kuvaa päivitettäessä. Lisättiin tarkistus joka estää pisteiden piirtämisen liian lähelle toisiaan. Toteutettiin painike pisteiden siirtämiselle. Luotiin ominaisuus että kun käyttäjä yrittää piirtää pistettä toisen pisteen päälle, piirretään tähän pisteeseen jana viimeksi lisätystä pisteestä. (tapahtuu vain jos janojen automaattinen piirto on päällä) Toteutettiin painike pisteiden poistamiselle. Toteutettiin päävalikkoon vaihtoehto kuvan piirtoelementtien tallentamiseen .mat-tiedostoon. Toteutettiin päävalikkon vaihtoehto piirtoelementtien lataamiseen valittuun kuvaan.	imageView()-funktio ottaa sisäänsä nyt myös totuusarvon sille säilytetäänkö kuvan zoomausparametrit. Pistettä siirrettäessä muokataan myös pisteessä kiinni olevia janoja. updateElements()-funktio ottaa sisäänsä uuden ja vanhan pisteen ja päivittää ohjelman vanhan pisteen uudella. deleteElement()-funktio ottaa sisäänsä pisteen ja poistaa sen ohjelman datarakenteesta. Pistettä poistettaessa poistetaan myös pisteessä kiinni olevat janat. Piirtoelementtejä tallennettaessa tiedoston oletusnimi on kuvan nimi. Piirtoelementit tallennetaan tallentamalla tiedostoon handles.data{i}.M ja handles.data{i}.L rakenteet. Piirtoelementit ladataan kuvaan, jota ollaan lataushetkellä näyttämässä. Ohjelma korvaa kuvan piirtoelementit ladatuilla elementeillä.

Taulukko 4. Ohjelman toteutuksen tarkka aikataulu

Päivämäärä	Mitä tehtiin	Huomioita
23.4.2016	Toteutettiin päävalikkoon vaihtoehto istuntojen tallentamiseen. Toteutettiin päävalikkon vaihtoehto istuntojen lataamiselle.	Istuntoja tallennettaessa tallennetaan tiedostoon koko handles.data rakenne, sekä tieto kuvien määrästä, valitusta kuvasta ja valitusta väristä. Istuntoa ladatessa korvataan nykyinen istunto ladatulla istunnolla.
24.4.2016	Toteutettiin ohjelman päävalikkoon vaihtoehto kuvan kontrastin muokkaamiseksi. Toteutettiin päävalikkoon vaihtoehto kuvan palauttamiseksi varmuuskopiosta. Toteutettiin päävalikkoon vaihtoehto reunantunnistusfunktioiden ajamiseksi. Toteutettiin päävalikkoon vaihtoehto josta kaikki kuvan merkinnät voidaan poistaa. Lisättiin try-catch rakenteita funktioihin, jotka ottavat suoraan tietoa käyttäjältä. Uudelleenjärjestettiin käyttöliittymän komponentteja. Lisättiin funktiot, jotka poistavat ja palauttavat käyttöliittymän painikkeet käyttöön.	Kontrastin muokkaus tapahtuu ponnahdusikkunan avulla johon käyttäjä syöttää haluamansa parametrit. Jos kuvan kontrastia muokataan, luodaan handles.data{i}.backup muuttujaan kopio alkuperäisestä kuvasta ja korvataan nykyinen kuva uudella kuvalla. Reunantunnistusfunktioiden ajaminen tapahtuu ponnahdusikkunan avulla, josta käyttäjä valitsee pudotusvalikosta haluamansa funktion, vaihtoehtoina Canny, log, Prewitt, Roberts ja Sobel. Ohjelma ajaa MATLAB:in edge()-funktion saaduilla parametreilla. Käyttöliittymän painikkeet poistetaan käytöstä piirtoelementtien lisäämisen ajaksi.
30.4.2016	Lisättiin painike monikulmioiden piirtoa varten Lisättiin painike suorakaiteiden piirtoa varten. Uudistettiin ohjelman datarakennetta ja muokattiin funktioita yhteensopiviksi uuden datarakenteen kanssa. Lisättiin painike merkintöjen luokkien ja värien muokkaamiselle.	Monikulmiot suljetaan automaattisesti, jos käyttäjä jättää piirron kesken. Ohjelma ei hyväksy suorakaiteita joiden korkeus tai leveys on 0. Ohjelman datarakenne uudistettiin. handles.data{i}.markings pitää tiedot kuvan merkinnöistä. markings{j}.label = merkinnän luokka. markings{j}.controlpoints = merkinnän pisteet. markings{j}.color = merkinnän väri. markings{j}.type = merkinnän tyyppi.
1.5.2016	Poistettiin painike janojen piirtämiselle. Janojen automaattisen piirron valintalaatikko uudistettiin näyttämään kuvan janat eikä se enää piirrä mitään. Pisteitä poistava painike muokattiin poistamaan koko merkintä. Tehtiin pisteiden siirtäminen yhteensopivaksi uuden datarakenteen kanssa. Ladatessa kuvia, ohjelma tallentaa kuvan nimen ja polun myös muistiin. Tallennettaessa merkintöjä, tallennetaan kuvan nimi ja polku myös tiedostoon. Ladatessa merkintöjä, ei korvata kuvan merkintöjä vaan avataan ladattava merkintätiedosto sen data avulla uudeksi kuvaksi, jossa ladattavat merkinnät.	Janoja ei enää tallenneta muistiin vaan ne voidaan piirtää automaattisesti merkinnän tyypin mukaan. getMarking()-funktio palauttaa nyt myös indeksin siihen merkintään mistä löydetty lähin piste löytyi. Muokattaessa esimerkiksi suorakulmion pisteitä, suorakulmio pysyy suorakulmaisena. Kuvan nimi ja polku handles.data{i}.filename ja handles.data{i}.pathname muuttujissa. Käyttäjä voi valita useita merkintöjä sisältäviä tiedostoja valittavaksi samaa aikaa.

Taulukko 5. Ohjelman toteutuksen tarkka aikataulu

Päivämäärä	Mitä tehtiin	Huomioita
1.5.2016	Päävalikon kontrastia muuttava ominaisuus muutettiin automaattiseksi eikä se pyydä käyttäjältä parametreja. Toteutettiin pudotusvalikko, josta käyttäjä pystyy valitsemaan värikarttoja mustavalkokuville.	Kontrastin muokkaus tapahtuu nyt automaattisesti. Ohjelma laskee MATLAB:in stretchlim()-funktion avulla sopivat parametrit imadjust-funktiolle. Tieto kuvan värikartasta tallennetaan handles.data{i}.colormap muuttujaan. Vaihtoehtoina MATLAB:in värikarttojen parametreiksi ovat 'default', 'jet' ja 'gray'
7.5.2016	Uudelleennimettiin käyttöliittymän painikkeita paremmin kuvaamaan niiden uusia toiminnallisuuksia. imageView()-funktio kirjoittaa nyt näkyviin merkintöjen luokat, jos luokka on määritetty. Toteutettiin pudotusvalikko, jossa lista kuvan luokista. Siirrettiin värikartan pudotusvalikko päävalikosta aukeavaan ponnahdusikkunaan ja lisättiin päävalikkoon vaihtoehto värikartan muokkaamiselle.	Määrittämättömät luokat ovat arvoltaan 'null'. getLabelList()-funktio palauttaa lista kuvan sisältämistä erilaisista luokista. Pudotusvalikon sisältämän ponnahdusikkunan luova funktio muokattiin yleiseen muotoon, joka ottaa sisäänsä tiedon pudotusvalikon sisällöstä ja kehotetekstistä. Ponnahdusikkuna aukeaa nyt myös näytön keskelle. getLabel()-funktio palauttaa luokkien pudotusvalikosta valitun luokan.
8.5.2016	Toteutettiin päävalikkoon vaihtoehto merkintöjen tallentamiseen Excel-taulukkoon. Toteutettiin päävalikkoon vaihtoehto kuvan kohinanpoistolle. Kun luokan väriä muutetaan, muutetaan nyt myös kaikkien samannimisten luokkien värejä. Värien pudotusvalikko muokkaa nyt luokkien pudotusvalikon valittua luokaa vastaavien merkintöjen väriä.	Taulukko sisältää merkintöjä kahden sarakkeen vierekkäisinä matriiseina. Kohinanpoistofunktio avaa käyttäjälle ponnahdusikkunan, josta käyttäjä saa valita suodattimen wiener2()- ja medfilt2()-funktioiden väliltä. luokkien värien päivittämiseen sisältyy myös luokkien muokkauspainikkeen värin muokkaus vaihtoehto.
11.5.2016	Toteutettiin ohjelmaan painike ympyröiden piirtämistä varten.	Käyttäjä piirtää kolme pistettä, jonka jälkeen ohjelma laskee näiden kolmen pisteen kautta kulkevan ympyrän ja piirtää sen.

4.4 Testausvaihe

Testausvaihetta suoritettiin samaan aikaan ohjelman toteutusvaiheen kanssa. Suurinosa ohjelmointivirheistä korjattiin ohjelman toteutuksen yhteydessä. Erillisiä testaussessioita pidettiin varsin vähän, koska ohjelmointivirheillä on tapana nousta esille uusia ominaisuuksia lisätessä.

MATLAB:in tarjoama zoomausfunktio toimi hieman odottamattomalla tavalla. Esimerkiksi värinvaihdon yhteydessä, kun ohjelma piirtää kuvan uudestaan ja asettaa zoomauksen parametrit, käyttäjä ei tuntemattomasta syystä pystynyt suoraan loitontamaan kuvan zoomausta. Käyttäjä pystyi kuitenkin loitontamaan zoomausta, kunhan ensin vieritti kuvaa. Lisäksi käyttäjä pystyi normaalisti poistamaan zoomauksen hiiren kaksoisnapautuksella. Tämä vaikuttaisi olevat MATLAB:in zoomausfunktion ominaisuus.

Ohjelman datarakenteen muuttamisen seurauksena kaikki luodut ominaisuudet testattiin uudestaan. Uusia ongelmia, virhetilanteita ja epätoiminnallisuuksia ei löytynyt. Ohjelman painikkeiden painaminen samalla kun ohjelmaan lisätään piirtoelementtejä saa ohjelman virheelliseen tilaan. Tämän takia ohjelmaan lisättiin piirtoelementtien lisäämiseen, muokkaukseen ja poistamiseen ominaisuus, joka estää käyttäjää painamasta käyttöliittymän painikkeita. Löydettiin ja korjattiin myös outo ohjelmavirhe, jolla ei ollut ohjelman käytön kannalta mitään merkitystä. Ohjelma tulosti varoitustekstin konsoliin, jos käyttäjä sulki kuvalistan viimeisen kuvan, eikä kuva ollut viimeinen auki oleva kuva. Ohjelmavirhe korjattiin, mutta sen aiheutumisen syy ei kuitenkaan noussut esille.

Osassa yliopiston tietokoneista oli vanhempi MATLAB versio, kuin millä ohjelma on luotu ja tämä saattaa tämä aiheuttaa yhteensopivuusvirheitä. Tästä johtuen testattiin ohjelman yhteensopivuutta vanhemman MATLAB-version kanssa. Käytiin kaikki ohjelaman perusominaisuudet läpi. Yhtäkään yhteensopivuusongelmaa ei löytynyt ohjelman ja vanhemman MATLAB-verion kanssa. Tutkittiin ohjelman funktioiden optimointia. Huomattiin etteivät ohjelman piirtoelementtejä poistava ja siirtävä ominaisuus ole täysin optimoituja. Kun ohjelmalla muokattiin tai poistettiin piirtoelementtejä elementtien tallennusrakenteesta, ohjelma piirsi uudestaan koko kuvan. Tästä syntyi hyvin suurien kuvien kanssa suoritustehollisia ongelmia. Ideaalisesti ohjelma vain piirtäisi uudestaan muokatut piirtoelementit. Tämä vaatisi uuden data-rakenteen, johon ohjelma tallentaisi piirrettyjen piirtoelementtien kahvoja. Tämän ominaisuuden toteutus ja integrointi ohjelmaan todettiin liian työlääksi ja resursseja kuluttavaksi työksi.

Tutkittiin onko tarvetta piirtoelementtien kahvojen tallentamiseen. Testattiin piirtoelementtien muokkaamista suhteellisen suurikokoisessa kuvassa, kooltaan 4890x3260 kuvapistettä. Huomattiin suoraan viivettä käyttäjän syötteen ja ohjelman palautteen välillä. Laskettiin syntynyt viive ja verrattiin sitä pienempi-kokoisen kuvan piirtoelementtien muokkauksesta syntyneeseen viiveeseen. 4890x3260 kuvapisteen kokoiselle kuvalla viive vaihteli välillä 130-140 millisekuntia ja 1390x799 kuvapisteen kokoiselle kuvalle viive vaihteli välillä 50-55 millisekuntia. Henkilökohtaisten kokemuksien perusteella ihminen alkaa tiedostamaan viivettä, jos viive on suuruudeltaan yli 80 millisekuntia. Pelkkien piirtoelementtien kahvojen muokkauksella viive saataisiin kaiken-kokoisille kuville hyvin todennäköisesti alle 10 millisekuntiin, eli huomattavasti alle rajan, jossa käyttäjä alkaa tiedostamaan viivettä. 130-140 millisekunnin viive ei kuitenkaan ole niin suuri että se suoraan häiritsisi pisteiden muokkaamista. Testattiin vielä lisäksi viivettä 8K kuvalle, 7680x4320 pikseliä, ja saatu viive vaihteli välillä 170-180 millisekuntia. Ohjelman loppukäyttäjä ei todennäköisesti käsittele valtavankokoisia kuvia ohjelmalla, ja suurista kuvista syntyvä viive ei ole niin suuri, että se haittaa suoraan ohjelman käyttämistä. Tämän seurauksena todettiin että ohjelman piirtoelementtejä muokkaavat ja poistavat ominaisuudet ovat riittävän hyvin optimoitu.

Huomattiin että ohjelman try-catch rakenteiden epäonnistuessa ohjelma tekei paljon turhaa työtä muutamassa tilanteessa. Muokattiin näiden rakenteiden sisältöä niin ettei ohjelma tee turhaa työtä kyseisissä tilanteissa. Korjattiin lukuisia ohjelmointivirheitä liittyen värien pudotusvalikon ja luokkien pudotusvalikon päivittymiseen. Nyt käyttäjä ei enää pysty lisäämään samannimisiä erivärisiä luokkia vaan kaikki samanluokkaiset piirtoelementit ovat aina nyt samanvärisiä. Korjattiin pitkään ohjelmassa ollut huomaamaton ohjelmointivirhe, joka sai kaikki piirtoelementit päivittymään pisteitä siirtäessä aivan kuin ne olisivat olleen suorakulmioita, jos piirtoelementin datapisteillä oli samoja x- tai y-koordinaatteja.

Ohjelman testauksen tarkka aikataulu (ei sisällä testaussessioita, joissa mitään ei löytynyt) on esitetty taulukossa 6.

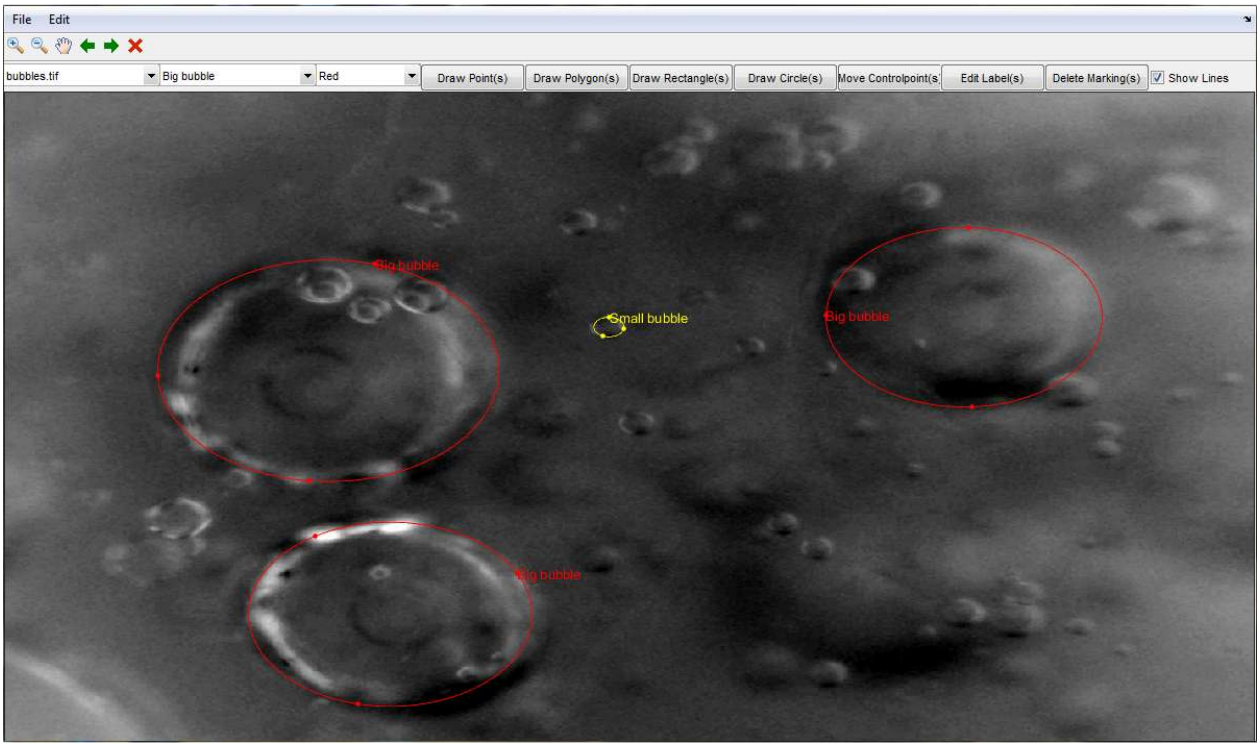
Taulukko 6. Testauksen tarkka aikataulu

Päivämäärä	Mitä löydettiin	Tila ja huomioita
23.4.2016	Käyttäjä ei pysty sulkemaan kuva jos se on viimeinen auki oleva kuva	Korjattu
24.4.2016	Ponnahdusikkunat aukeavat minne sattuu.	Korjattu (aukeavat näytön keskelle)
24.4.2016	Virhetiloja jos käyttäjä lisää piirtoelementtejä käyttöliittymän painikkeiden alle.	Korjattu (painikkeet poissa käytöstä kun piirtoelementtejä lisätään)
24.4.2016	Ohjelma tulostaa varoitustekstin konsoliin, kun käyttäjä sulkee ohjelman viimeisen kuvan.	Korjattu (aiheutumisen syy ei tiedossa, mutta korjattu)
25.4.2016	Ohjelman piirtoelementtejä poistavat ja muokkaavat funktiot lataavat kuvan uudestaan useaan kertaan muokkauksen aikana.	Korjattu (Kuva näytetään uudestaan vasta muokkauksen lopuksi)
8.5.2016	Luokkien pudotusvalikon arvo palautuu viimeksi valikkoon lisättyyn arvoon jos käyttäjä lataa saman kuvan uudestaan.	Korjattu
8.5.2016	Reunantunnistusfunktio ei toimi mustavalkokuvilla, koska se ensin muuttaa kuvan mustavalkokuvaksi.	Korjattu
8.5.2016	Tietyissä epäselvissä tilanteissa kontrastin palautus ei onnistu.	Korjattu
8.5.2016	try-catch rakenteiden rakenne ohjelmassa hyvin epäoptimaalissa muodossa. Tehdään paljon turhaa työtä rakenteen epäonnistuessa.	Korjattu
10.5.2016	Luokkien ja värien pudotusvalikot päivittyvät miten sattuu.	Korjattu (päivittyvät nyt toivotulla tavalla)
11.5.2016	Piirtoelementit päivittyvät ikään kuin ne olisivat suorakulmioita, jos niiden pisteillä on samoja x- tai y-koordinaatteja.	Korjattu
10.5.2016	Ympyrön piirto epäonnistuu, jos pisteillä on samoja x- tai y-koordinaatteja.	Korjattu

5 MERKITSEMISTYÖKALU

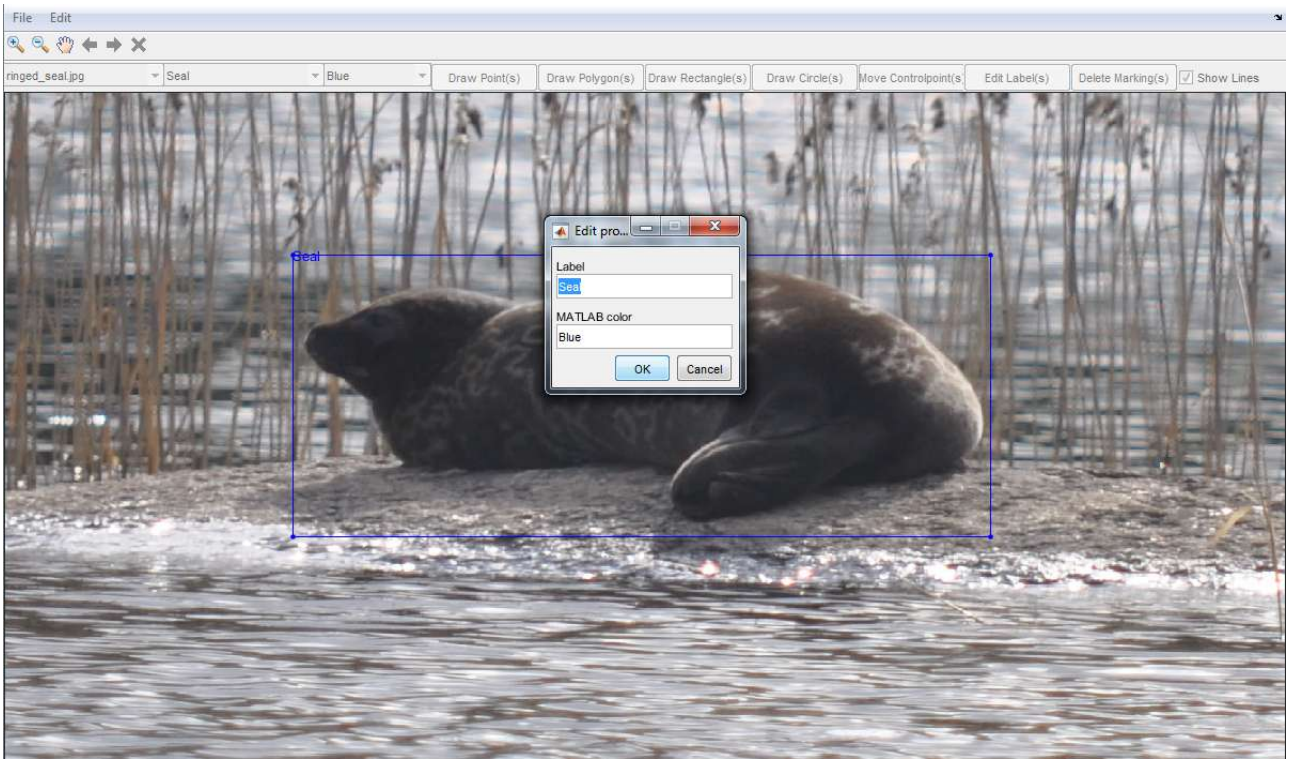
Lopullinen työ täyttää asiakasvaatimukset. Käyttäjä pystyy avaamaan kuvia, segmentoimaan ja luokittelemaan kuvista objekteja ja tallentamaan merkintänsä tiedostoon. Kuitenkin paljon parannusmahdollisuuksia löytyy. Esimerkiksi datapisteiden tallentaminen xml taulukkoon voisi tallentaa ympyrän pisteiden sijasta ympyrän keskipisteen ja säteen. Ohjelma voisi sisältää mahdollisuuden ajaa Canny reunantunnistusfunktio käyttäjän antamien parametrien mukaan. Ohjelman painikkeita voisi siirtää työkalupalkkiin ja niille voisi luoda niiden toimintoja vastaavat ikonit. Ohjelman voisi sisältää enemmän piirtoelementtejä, kuten vapaan käyrän.

Ohjelma toimii sen käyttötarkoituksen mukaisesti. Kuvasta 3 nähdään , että ympyrä soveltuu hyvin kuplien segmentoimiseen. Verrattuna monikulmioon, kupla pystytään helposti segmentoimaan kolmen pisteen avulla piirtämällä näiden pisteiden kautta kulkeva ympyrä.

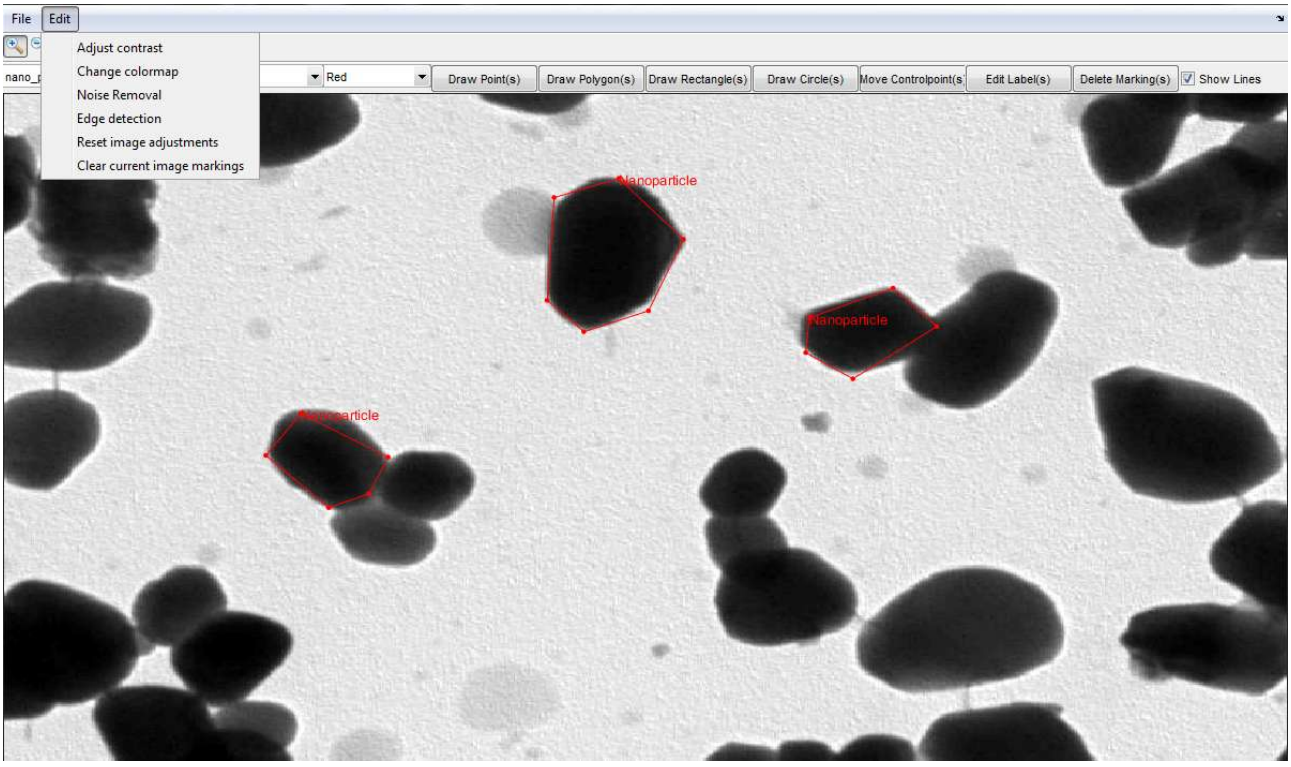


Kuva 3: Ohjelmalla segmentoituja kuplia

Kuvassa 4 hylje pystytään varsin vavattomasti rajaamaan kuvasta suorakaiteella. Kuvassa 5 nanopartikkeleja on segmentoitu kuvasta vapailla monikulmioilla

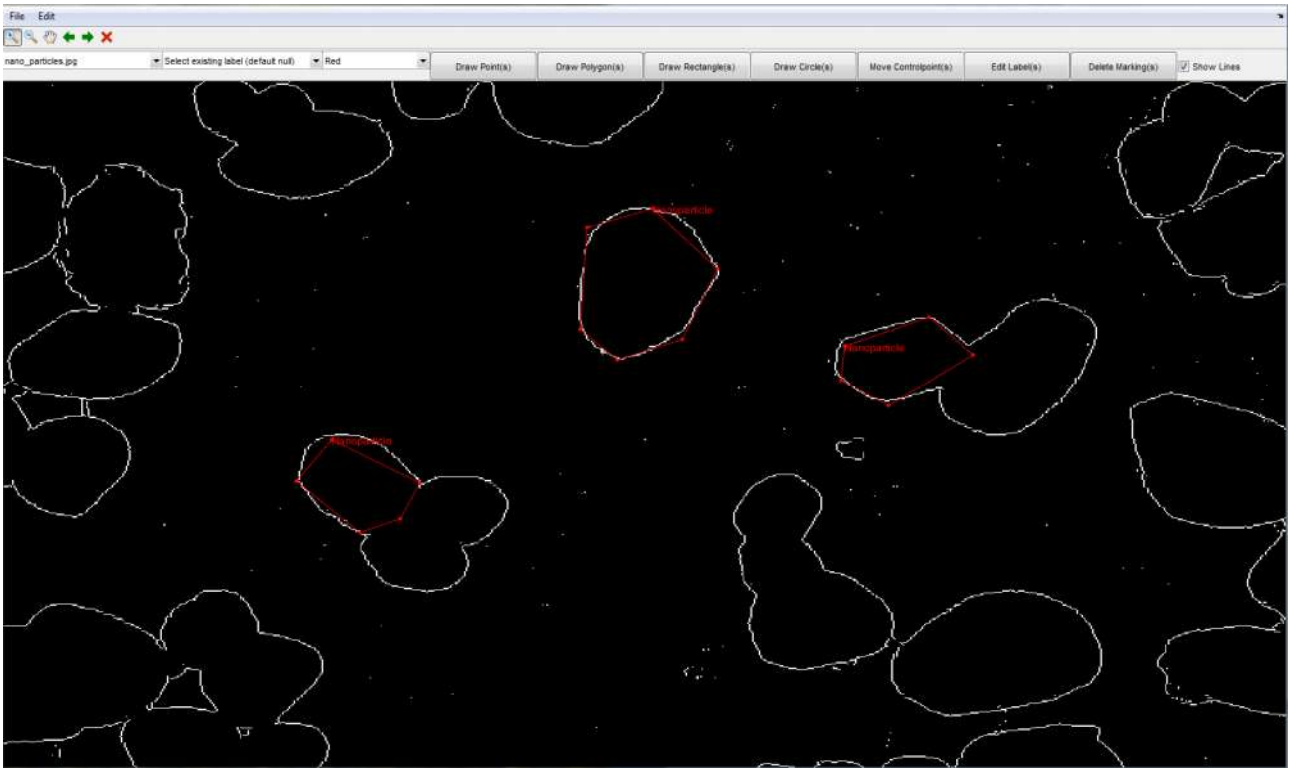


Kuva 4: Ohjelmalla rajattu hylje



Kuva 5: Ohjelmalla segmentoituja nanopartikkeleja

Kuvassa 6 on nanopartikkeleille ajettu ohjelman reunantunnistusfunktio Sobel-parametrilla. Reunojen tunnistaminen vaikuttaisi onnistuneen varsin hyvin.



Kuva 6: Ohjelmalla segmentoituja nanopartikkeleja

6 JOHTOPÄÄTÖKSET

Ohjelma tuotettiin hyvin pitkälti vesiputousmallin mukaisesti vaiheissa, vaikka useaa vaihetta suoritettiin samanaikaisesti. Ohjelmistotuotannon vesiputousmalli toimi halutulla tavalla. Se asetti selvät vaiheet ohjelman tuottamiselle. Ohjelman koodi tuotettiin päivittäisissä vaiheissa, joissa pyrittiin saamaan muutama ohjelman toiminnallinen kokonaisuus valmiiksi kerrallaan. Ohjelma oli aina päivien välillä toimivassa kunnossa ja tästä syntyi selvä tunne ohjelman tuottamisen edistymiselle. Ohjelmaan lisättyjen kokonaisuuksien testausta suoritettiin myös aina päivän lopuksi.

Ohjelman tuottamisen prosessi on hyvin toimiva, mutta sitä ei välttämättä toteuteta aina ideaalilla tavalla. Esimerkiksi ohjelmalle olisi pitänyt hakea tarkemmat toiminnalliset asiakasvaatimukset. Nyt projektin ohjelman ensimmäinen versio oli varsin kaukana siitä mitä asiakas halusi. Tästä ei kuitenkaan seurannut massiivisia lisäkustannuksia, koska ohjelman ranka, eli käyttöliittymä, pysyi hyvin pitkälti samana. Lisäksi tarpeettomilla luoduilla toiminnallisuuksilla oli paljon sovelluskohteita lopullisessa ohjelmassa. Esimerkiksi janojen piirto-funktio automatisoitiin hyvin pitkälti kun saatiin selville ettei janoista tarvitse erikseen pitää kirjaa. Lisäksi janojen piirron funktiota käytettiin rankana funktiolle, jolla kuvan merkintöjen pisteitä voi siirtää. Suurin kompastuskivi oli todennäköisesti ohjelman datarakenteen muuttaminen. Tämä olisi suuremmassa projektissa voinut aiheuttaa massiivisia lisäkustannuksia. Kuitenkin tässä pienessä projektissa datarakenteen muuttamisesta selvittiin alle tunnin lisätyöllä.

LÄHTEET

1. Zhelezniakov, A., Eerola, T., Koivuniemi, M., Auttila, M., Levänen, R., Niemi, M., & Kälviäinen, H. (2015, December). Segmentation of Saimaa ringed seals for identification purposes. In *International Symposium on Visual Computing* (pp. 227-236). Springer International Publishing.
2. Zafari, S., Eerola, T., Sampo, J., Kälviäinen, H., & Haario, H. (2015). Segmentation of Overlapping Elliptical Objects in Silhouette Images. *IEEE Transactions on Image Processing*, *24*(12), 5942-5952.
3. Royce, W. W. (1970, August). Managing the development of large software systems. In *proceedings of IEEE WESCON* (Vol. 26, No. 8, pp. 328-338).
4. MathWorks MATLAB GUIDE, http://se.mathworks.com/discovery/matlab-gui.html , viitattu 27.6.2016
5. Mäkinen, E. (2014). Tietojenkäsittelytieteellisiä tutkielmia Kevät 2014. *Tampereen Yliopisto, Informaatiotieteiden yksikkö*
6. Opintojakso BM40A0500 Johdatus laskennalliseen älykkyyteen, Lappeenrannan teknillinen yliopisto, Teknillistaloudellinen tiedekunta, Laskennallisen tekniikan koulutusohjelma, 2016, https://noppa.lut.fi/noppa/opintojakso/bm40a0500/etusivu
7. Sonka, M., Hlavac, V., & Boyle, R. (2014). *Image processing, analysis, and machine vision*. Cengage Learning.
8. von Gioi, R. G., Jakubowicz, J., Morel, J. M., & Randall, G. (2012). LSD: a line segment detector. *Image Processing On Line*, *2*, 35-55.
9. Marr, D., & Hildreth, E. (1980). Theory of edge detection. *Proceedings of the Royal Society of London B: Biological Sciences*, *207*(1167), 187-217.
10. Lindeberg, T. (1998). Edge detection and ridge detection with automatic scale selection. *International Journal of Computer Vision*, *30*(2), 117-156.
11. Sobel, I., Duda, R., Hart, P., & Wiley, J. Sobel-Feldman Operator.
12. Prewitt, J. M. (1970). Object enhancement and extraction. *Picture processing and Psychopictorics*, *10*(1), 15-19.
13. Davis, L. S. (1975). A survey of edge detection techniques. *Computer graphics and image processing*, *4*(3), 248-270.
14. Canny, J. (1986). A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6), 679-698.

15. Kotkar, V. A., & Gharde, S. S. (2013). Review of various image contrast enhancement techniques. *International Journal of Innovative Research in Science, Engineering and Technology*, *2*(7).

16. Davies, E. R. (2004). *Machine vision: theory, algorithms, practicalities*. Elsevier.

17. Jain, A. K. (1989). *Fundamentals of digital image processing*. Prentice-Hall.

18. Farooque, M. A., & Rohankar, J. S. (2013). Survey on various noises and techniques for denoising the color image. *International Journal of Application or Innovation in Engineering & Management (IJAIEM)*, *2*(11), 217-221.

19. Simoncelli, E. P., & Adelson, E. H. (1996, September). Noise removal via Bayesian wavelet coring. In *Image Processing, 1996. Proceedings., International Conference on* (Vol. 1, pp. 379-382). IEEE.

20. Opintojakso CT60A4001 Ohjelmistotuotanto, Lappeenrannan teknillinen yliopisto, Teknistaloudellinen tiedekunta, Tietotekniikan koulutusohjelma, lukuvuosi 2015, https://noppa.lut.fi/noppa/opintojakso/ct60a4001

21. Russell, B. C., Torralba, A., Murphy, K. P., & Freeman, W. T. (2008). LabelMe: a database and web-based tool for image annotation. *International journal of computer vision*, *77*(1-3), 157-173.

22. Szoter websivut
http://www.szoter.com/#intro , viitattu 27.6.2016

23. Tzutalin, Labellmg, *Github*
https://github.com/tzutalin/labelImg , viitattu 27.6.2016

24. Strokina, N., Matas, J., Eerola, T., Lensu, L., & Kälviäinen, H. (2016). Detection of bubbles as concentric circular arrangements. *Machine Vision and Applications*, *27*(3), 387-396.

25. Hienonen, P. (2014). Automatic traffic sign inventory-and condition analysis. *Diplomityö, Lappeenrannan teknillinen yliopisto*