VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Mikael Anton Kurki

Aortan 3D-geometrian tuottaminen lääke-tieteellisestä kuvadatasta

Tekniikka
2025

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Mikael Kurki
Opinnäytetyön nimi	Aortan 3D-geometrian tuottaminen lääketieteellisestä kuvadatasta
Vuosi	2025
Kieli	suomi
Sivumäärä	52 + 10 liitettä
Ohjaaja	Harri Lehtinen

Aortta on sydämestä lähtevä päävaltimo, joka kuljettaa verta koko elimistöön. Aortan seinämän repeäminen on hengen vaarallinen ja sen seuranta vaatii huomattavia kustannuksia. Kuitenkin aortan laajentumisen varhaisen havaitsemisen avulla voidaan pelastaa ihmishenkiä ja alentaa terveydenhuollon kustannuksia. Tehokkaalle aortan aneurysman riskin arviointityökaluille, joilla voidaan ennakoida ja ennaltaehkäistä vaaratilanteita, on tarvetta. Tässä työssä oli tavoitteena ohjelmoida esikäsittely MRI:n (magnetic resonance imaging) tuottamille kuville ja täten muodostaa laadukasta 3D-kuvamateriaalia aorttageometrian jatkoanalyyseihin. Ohjelman on tarkoitus kuvantaa aortan nousevaa osaa. Tämä kehittämishanke tehtiin osana CorFlux-projektia, jossa tavoitteena oli kehittää aortan aneurysman analyysityökaluohjelma.

Tässä opinnäytetyössä havainnollistetaan 3D-kuvan esikäsittely ja aorttageometrian määritys, jonka pohjana on ohjelmaan syötetyt potilastiedot ja potilaasta otetut Tietokonetomografia- tai Magneettikuvat. Työ ohjelmoitiin C#-ohjelmointikielellä Windows Forms alustalla, Visual Studio 2022 kehitysympäristöä käyttäen. Kuvien prosessointivaiheita ovat mm. tiedostojenkäsittely ja segmentointi.

Projektissa ohjelmasta tehtiin modulaarinen testiversio, joka oli valmistuneilta osiltaan toimiva, vaikkakin viimeistä moduulia, jossa varsinainen 3D-kuva olisi muodostettu, ei projektin aikataulurajoitteiden vuoksi ehditty tehdä. Testiversion ajaminen tapahtui manuaalisesti ja useassa vaiheessa, joten sen käyttö oli vielä kankeaa ja vaatisi merkittävää jatkokehitystä, eikä siten edusta mahdollista valmista kaupallista tuotetta.

Avainsanat kardiologia, sovelluskehitys, radiologia, kuvantaminen

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Mikael Kurki
Title	Aortan 3D-geometrian tuottaminen lääketieteellisestä kuvadatasta
Year	2025
Language	Finnish
Pages	52 + 10 Appendices
Name of Supervisor	Harri Lehtinen

The aorta is the main artery originating from the heart that carries blood throughout the body. A rupture of the aortic wall is life threatening, and its monitoring requires substantial resources. However, early detection of aortic enlargement can save lives and reduce healthcare costs. There is a need for effective risk-assessment tools for aortic aneurysms that can predict and prevent dangerous situations. The aim of this work was to program preprocessing for images produced by MRI (magnetic resonance imaging) and thereby generate high-quality 3D image material for further analyses of aortic geometry. The program is intended to image the ascending part of the aorta. This development project was carried out as a part of the CorFlux project, which aimed to develop a software tool for analyzing aortic aneurysms.

This thesis illustrates the preprocessing of 3D images and the determination of aortic geometry, the basis of which consists of patient data entered into the program and CT or MRI images taken of the patient. The work was programmed in the C# programming language on the Windows Forms platform, using the Visual Studio 2022 development environment. The image-processing steps include, among other things, file handling and segmentation.

In the project, a modular test version of the program was created. The program was functional in the parts that had been completed, although the final module—where the actual 3D image would have been generated—could not be completed due to project time constraints. Running the test version had to be done manually and, in several stages, so its use was still cumbersome and would require significant amount of further development and therefore does not represent a potentially finished commercial product.

Keywords cardiology, software development, radiology, imaging

SISÄLLYS

TIIVISTELMÄ

ABSTRACT

1 JOHDANTO 13

2 TARVEANALYYSI JA NYKYTILA 14

3 TIETOPERUSTA 16

3.1 Kardiologia – Aortan Aneurysma 16

3.2 Radiologia 17

3.3 Segmentointi 19

3.4 C#-ohjelmointi kieli 19

4 KEHITTÄMISPROSESSIN KUVAUS 22

4.1 Kehitysympäristö 22

4.2 Kehitysprosessi 24

5 KEHITTÄMISTOIMINNAN TULOKSEN KUVAUS 26

5.1 Tiedosto-operaatiot 28

5.2 Segmentointialueiden valinta 35

5.3 Perusgeometrian määritys 37

5.4 Segmentointigeometrian määritys 41

5.5 3D pinta mallin määritys 47

6 JOHTOPÄÄTÖKSET JA ARVIOINTI 48

LÄHTEET 50

LIITTEET 52

KUVIO- JA TAULUKKOLUETTELO

Kuva 1. Piirros kuva Aortasta https://upload.wikimedia.org/wikipedia/commons/a/a0/Aorta_Anatomy.jpg 16

Kuva 2. Anonymisoitu MRI kuva, Aortan yläosa ääriviivattu punaisella. 18

Kuva 3. CorFlux-Ohjelmiston Moduuli Rakenne. Moduulit 1–14 ovat osa tätä projektia. 26

Kuva 4. Form1 Moduuli 2 Graafinen ulkoasu. 28

Kuva 5. Form3 Moduuli 3 Graafinen ulkoasu. 29

Kuva 6. Form4 Moduuli 4 Graafinen ulkoasu. 31

Kuva 7. Form5 Moduuli 5 Graafinen ulkoasu. 33

Kuva 8. Form6 Moduuli 6 Graafinen ulkoasu. Tämä on myös Form7 Moduulin 7 ulkoasu. 36

Kuva 9. Form8 Moduuli 8 Graafinen ulkoasu. 41

Kuva 10. Form10 Moduuli 10 Graafinen ulkoasu. 42

Kuva 11. Kuvasarja esittää aortan poikkileikkauskuvan segmentoinnin etenemisen vaihe vaiheelta, vasemmalta oikealle. (1) Raaka MRI-kuva. (2) Kuvasta on rajattu pois kaikki alueet, joissa aortan nouseva osa ei esiinny. (3) Harmaasävyksi muunnettu binäärikuva, jossa aortta ja muut verisuonet näkyvät valkoisina mustaa taustaa vasten. (4) Eroosiolla käsitelty kuva, jossa ylimääräiset verisuonet on poistettu. (5) Dilaation tulos, jonka avulla osa eroosiossa hävinneestä rakenteesta palautetaan. (6) Lopullinen segmentointi, jossa aortan seinämät on korostettu valkoisena ja muu kuva-alue on musta. 46

Kuva 12. Aortan alueen 3D-esitys. Aortan tasot (4 musta viivaa), tasojen poikkileikkauspisteet (vihreät viivat) ja aortan kaari (musta viiva osoitettu punaisella nuolella). 47

Koodi 1. Koodiosio, jossa määritetään kovakoodatut syöte- ja ulostulotiedostot sekä muita ulostuloon liittyviä toimintoja. 27

Koodi 2. Koodiosio, jossa ulostulokansion sisällä olevat tiedostot poistetaan, jos se sisältää tiedostoja. 29/30

Koodi 3. Koodiosio, joka tarvittaessa luo uuden ulostulokansio, tiedostot kopioidaan ja niiden perään lisätään *.dcm-pääte. 30

Koodi 4. Tämä koodiosio lukee parametrejä *.ini asetustiedosta ja tallentaa ne merkkipohjaisiin ja numeerisiin sanakirjoihin. 31/32

Koodi 5. Tämä koodiosio järjestää, uudelleen nimeää ja tallentaa joukon DICOM-tiedostoja haluttuun kansioon. 32

Koodi 6. Tämä koodiosio listaa ja tallentaa ”start” ja ”end” tekstikentistä saadut luvut CSV-tiedostoon. 34

Koodi 7. Tämä osio käy läpi valitun indeksivälin DICOM-kuvat, kopioi jokaisen alkuperäisen DICOM-tiedoston uuteen kansioon ja tekee niistä kopiot, jotka ovat TIFF-muodossa. 34/35

Koodi 8. Tämä metodi avaa listasta valitun kuvan DICOM-kuvan IrfanView sovelluksessa. 35

Koodi 9. Metodi, joka tallentaa Keskiviivan koordinaatit CSV-tiedostoon. ... 37/38

Koodi 10. Metodi, jolla suoritetaan ympyräkaaren sovittaminen koordinaattipisteisiin PNS-menetelmällä. 38/39

Koodi 11. MovePoints-metodi 40

Koodi 12. Tämä koodiosio suorittaan eroosion syötekansion TIFF kuville. …….. 43

Koodi 13. Tämä Koodiosio tekee dilaation jokaiselle TIFF-kuvalle ja tallentaa tulokset erilliseen hakemistoon jatkokäsittelyä varten. …… 44

Koodi 14. Dilaatio metodi suorittaan dilaation syötekansion TIFF kuville. …….. 44

Koodi 15. hakee kaikki syötekansiossa olevat TIFF-kuvat, suorittaa niille Sobel-pohjaisen reunanilmaisun Emgu CV -kirjaston avulla ja binarisoi tuloksen Otsun automaattisella kynnysarvotuksella. …… 45/46

Taulukko 1 CorFlux-Ohjelmistossa käytetyt paketit. 25

LIITELUETTELO

LIITE 1. Koodiosio, jossa segmentointialue määritellään.

LIITE 2. Koodiosio, jossa viivauksen ulkopuolella oleva osa kuvasta poistetaan.

LIITE 3. Keskilinja säteen laskeminen ja määritys.

LIITE 4. divideArc-metodi

LIITE 5. Koodiosio, joka suorittaa intensiteettikynnytyksen.

LIITE 6. Foreach silmukka, jonka sisällä suoriutuu alueiden valinta (Area select) jokaiselle syötekansiossa olevalle kuvalle.

TERMIT JA SANASTO

Aortta

Aortta on kehon suurin valtimo, joka kuljettaa happea ja ravinteita sydämestä eri elimiin. Se lähtee vasemmasta kammiosta ja kulkee eri alueiden kautta kohti kehon alaosaa.

Aortan aneurysma

Aortan aneurysma tarkoittaa aortan seinämän heikentymistä ja laajentumista, mikä voi johtaa aortan repeämiseen. Aneurysma voi esiintyä missä tahansa aortan osassa, mutta yleisimmin se tapahtuu vatsan tai rinta-alueella.

CT / TT (Computed Tomography / Tietokonetomografia)

Tietokonetomografia (CT) on kuvantamismenetelmä, jossa röntgensäteet ja tietokonealgoritmit luovat kehon poikkileikkauksellisia kuvia. CT-kuvauksella voidaan tutkia elimistön sisäisiä rakenteita tarkasti.

MRI / MK (Magnetic Resonance Imaging / Magneettikuvaus)

Magneettikuvaus (MRI) käyttää voimakasta magneettikenttää ja radioaaltoja kehon sisäisten rakenteiden kuvaamiseen. Tämä menetelmä ei käytä ionisoivaa säteilyä kuten CT, ja se on erityisen tehokas pehmytkudosten kuvantamiseen.

DICOM (Digital Imaging and Communications in Medicine)

DICOM on standardi lääketieteellisille kuvantamiselle ja tiedonvaihdolle. Se määrittelee kuvatiedostojen formaatin ja kommunikoinnin terveydenhuollon laitteiden välillä, kuten röntgen- ja magneettikuvauslaitteiden.

TIFF (Tagged Image File Format)

TIFF on kuvaformaatti, jota käytetään erityisesti tarkkojen ja korkealaatuisten kuvien tallentamiseen. Se tukee väritiloja, kerroksia ja läpinäkyvyyksiä, ja se on usein käytössä lääketieteellisissä kuvantamissovelluksissa.

Segmentointi

Segmentointi on prosessi, jossa kuvasta eristetään tietty alue tai objekti, kuten kudos tai elin. Lääketieteellisessä kuvantamisessa segmentointi voi auttaa erottamaan sairaudet tai poikkeamat terveistä kudoksista.

Kynnystys (Thresholding)

Kynnystys on segmentointitekniikka, jossa kuvan pikselit jaetaan eri ryhmiin perustuen niiden intensiteettiin. Tämä auttaa erottamaan tietyt rakenteet tai alueet kuvasta.

Reunantunnistus (Edge Detection)

Reunantunnistus on kuvankäsittelytekniikka, joka etsii kuvan reunoja tai ääriviivoja. Tämä on tärkeää rakenteiden tunnistamisessa ja segmentoinnissa.

Morfologiset operaatiot

Morfologiset operaatiot ovat kuvankäsittelytekniikoita, joita käytetään rakenteiden muotojen analysointiin ja muokkaamiseen. Tällaisia operaatioita ovat esimerkiksi eroosio ja dilaatio.

Eroosio

Eroosio on morfologinen operaatio, joka pienentää kuvan rakenteita poistamalla yksittäisiä pikseleitä rakenteen reunoilta. Tämä voi auttaa poistamaan kohinaa tai pienentämään alueen kokoa.

Dilaatio

Dilaatio on morfologinen operaatio, jossa kuvan rakenteet laajenevat lisäämällä pikseleitä niiden reunoille. Tämä voi auttaa täyttämään reikiä tai liittämään osia yhteen.

Artefakti

Artefakti on kuvassa oleva epätoivottu tai virheellinen rakenne, joka ei vastaa todellisuutta. Artefakteja voi syntyä muun muassa laitteiston tai kuvankäsittelyprosessin virheiden seurauksena.

Keskiviiva (Centerline)

Keskiviiva tarkoittaa geometriassa keskeistä viivaa, joka kulkee objektin keskikohdan kautta. Kuvantamisessa se voi tarkoittaa esimerkiksi verisuonien tai elinten keskiosaa, joka voi olla tärkeä rakenteen analysoinnissa.

Kaarevuussäde (Radius of Curvature)

Kaarevuussäde on käyrän tai kaaren kaarevuuden mitta. Se määritellään kaarevuuden käänteislukuna ja se voi olla tärkeä geometristen rakenteiden, kuten verisuonten, analysoinnissa.

Pienimmän neliösumman menetelmä (PNS / Least Squares)

Pienimmän neliösumman menetelmä on matemaattinen tekniikka, jota käytetään parhaiten sovittamaan tietoja (esim. käyrää tai suoraa) minimaalisella virheellä. Se minimoi poikkeamien neliösumman tarkasteltavissa olevasta datasta.

Polygonimaski / monikulmiomaski

Polygonimaski on alue, joka on määritelty monikulmion (polygonin) muotoiseksi. Se voi olla käytössä esimerkiksi segmentoinnissa, jossa pyritään rajaamaan alueita kuvatiedoista.

CSV-tiedosto (Comma-Separated Values)

CSV on tiedostomuoto, jossa data on tallennettu taulukkomuodossa ja kentät erotetaan toisistaan pilkuilla. Tämä tiedostomuoto on yleinen tietojen tallentamiseen ja jakamiseen.

INI-tiedosto

INI on yksinkertainen asetustiedostomuoto, joka tallentaa ohjelmien asetuksia ja konfiguraatiotietoja. Tiedostossa on usein avain-arvo-pareja, jotka määrittävät ohjelman toiminnallisuuksia.

Windows Forms (WinForms)

Windows Forms on Microsoftin kehittämä käyttöliittymäkehys (UI framework) Windows-sovelluksille. Se mahdollistaa graafisten käyttöliittymien luomisen .NET-ympäristössä.

C#

C# on ohjelmointikieli, joka on osa Microsoftin .NET-kehitysalustaa. Sitä käytetään laajasti ohjelmistokehityksessä, erityisesti Windows-sovellusten ja verkkopalveluiden luomisessa.

Emgu CV

Emgu CV on .NET-kirjasto, joka tarjoaa rajapinnan OpenCV:n (Open Source Computer Vision Library) käyttöön C#-ohjelmointikielessä. Se mahdollistaa kuvankäsittelyn ja tietokonenäön sovelluksia.

fo-dicom

fo-dicom on .NET-kirjasto, joka tukee DICOM-standardin käyttöä lääketieteellisissä kuvantamisohjelmissa. Se mahdollistaa DICOM-kuvien lukemisen, käsittelyn ja tallentamisen.

Magick.NET

Magick.NET on .NET-pohjainen kuvankäsittelykirjasto, joka perustuu ImageMagick-ohjelmistoon. Se tukee laajasti kuvamuotojen käsittelyä ja muokkaamista.

IrfanView

IrfanView on kevyt ja monipuolinen kuvankatseluohjelma, joka tukee monia kuvamuotoja ja tarjoaa peruskuvankäsittelytoimintoja. Sitä käytetään usein kuvien katseluun ja muokkaamiseen.

1 JOHDANTO

Tietokoneohjelmat ovat parantaneet ja tehostaneet monien eri alojen perinteisiä ja manuaalisia toimia ja prosesseja. Lääketieteen ja sairaanhoidon alat ovat vain yksi monista eri aloista, joissa jatkuvasti kehittyvän teknologian vaikutukset nähdään. Erityisesti ihmishenkien pelastaminen on jotain, missä uudet ohjelmat voivat auttaa tärkein tavoin.

CorFlux-projektin yksi päämääristä on kehittää tämänkaltainen ohjelmisto. Projektin tavoitteena oli luoda standardimalli aortan aneurysman (AA) repeämäriskin kliinisessä arvioinnissa. Tavoitteena oli myös kaupallistaminen ja valmistella projektin sijoittajille sopivaan liiketoimintamalliin. Projekti nähtiin tarpeelliseksi, koska aortan aneurysmien diagnosointi, hoito ja seuranta aiheuttavat merkittäviä kustannuksia terveydenhuoltojärjestelmille ja talouksille maailmanlaajuisesti. (Building a better future…, 2024)

Osa tätä kokonaisuutta on C#-ohjelmointikielellä ohjelmoitu ohjelma, jolla aortta voidaan kuvantaa 3D muodossa. Tässä työssä tavoitteena oli esikäsitellä MRI kuvantamisesta syntyvää raakadataa ja tuottaa 3D kuvaa jatkoanalyysia varten. Tämän opinnäytetyön on tarkoituksena kuvata kehitetyn kuvantamisohjelman rakenne, toiminnallisuus ja keskeiset tekniset ratkaisut.

2 TARVEANALYYSI JA NYKYTILA

Ohjelman on tarkoitus tarjota analyysityökalu, jonka avulla terveydenhuollon ammattilaiset voivat tehdä aortan aneurysmapotilaiden hoitoon liittyviä päätöksiä paljon tarkemmin kuin mitä nykyiset käytännöt sallivat (Building a better future…, 2024).

Projektin päämääränä on: 1) Aortan repeämisten varhainen havaitsemisen, jolloin ennakoivan väliintulon avulla voidaan pelastaa ihmishengistä jopa 30 %. 2) Mahdollisuus alentaa terveydenhuoltokustannuksia saavuttaen jopa 80 %:n säästöt. 3) Resurssien kohdentamisen optimointi vähentämällä kuvantamisjonoja ja minimoimalla viivästyksiä. Parantamalla työnkulun tehokkuutta parannamme sekä potilaiden että palveluntarjoajien yleistä terveydenhuoltokokemusta. (Building a better future…, 2024)

Aortan aneurysma, eli aortan valtimonpullistuman, ilmaantuvuus on jatkuvassa kasvussa kuvantamistutkimusten yleistyessä ja väestön ikääntyessä. Aorttaaneurysmat ovat 20 yleisimmän kuolinsyyn joukossa, jonka vuoksi jokainen diagnosoitu aneurysma johtaa säännöllisiin kuvantamisseurantoihin. Aneurysmasairauksien diagnosointi, hoito ja seuranta muodostavat merkittävän kustannuserän terveydenhuoltojärjestelmille ja kansantalouksille. Aorttojen aneurysmiin ei ole olemassa tehokasta riskinarviointityökalua. Aneurysmasta johtuvan repeämisriskin arvioinnissa ei tällä hetkellä huomioida potilaan aortan yksilöllisiä rakenteellisia- ja virtausominaisuuksia, vaan aortta diagnosoidaan laajentuneeksi, kun sen halkaisija ylittää 40 mm. Tämän vuoksi arviointi on tehotonta ja johtaa säännöllisiin kuvantamisseurantoihin potilaan loppuelämän ajaksi, vaikka noin 80 %:lla potilaista laajentuma pysyy muuttumattomana tai ei muutu merkittävästi seurantajakson aikana. Rinta-aortta aneurysmien vuosittaiset seurantakustannukset ovat jopa 1,23 miljardia euroa pelkästään Euroopan talousalueella. Tehokkaan arviointityökalun markkina-arvon on arvioitu olevan globaalisti miljardeja euroja. (CorFlux – UEFConnect, n.d.)

Laajentuneen aortan diagnosointi perustuu yhä lähinnä vain sen suurimman halkaisijan mittaamiseen ja arviointiin sen sijainnista (esim. joko nousevan aortan keskikohta eli tubulaarinen osa tai läpän lähellä oleva alue, eli Sinus Valsava). Kardiologit maailmanlaajuisesti ovat kuitenkin sitä mieltä, että tämä menetelmä on liian yksinkertainen eikä se korreloi kovinkaan hyvin aortan repeämien ja dissekaatioiden kanssa. Tämän vuoksi on tarvetta kehittää parempia menetelmiä repeämistodennäköisyyden arviointiin.

Tämä työ on osa Business Finlandin rahoittamaa CorFlux-projektia. Projekti lähti liikkeelle kaupallisen prototyypin tarpeesta aorttageometrian segmentointiin. Tämänhetkiset ohjelmat eivät sovellu tällaiseen työhön. Tavoitteena on siis kaupallinen käyttö, kun taas monia julkisesti käytettäviä, tutkimuksellisia segmentointiohjelmistoja ei voinut käyttää johtuen niissä ilmoitetuista kaupallisen käytön rajoituksista.

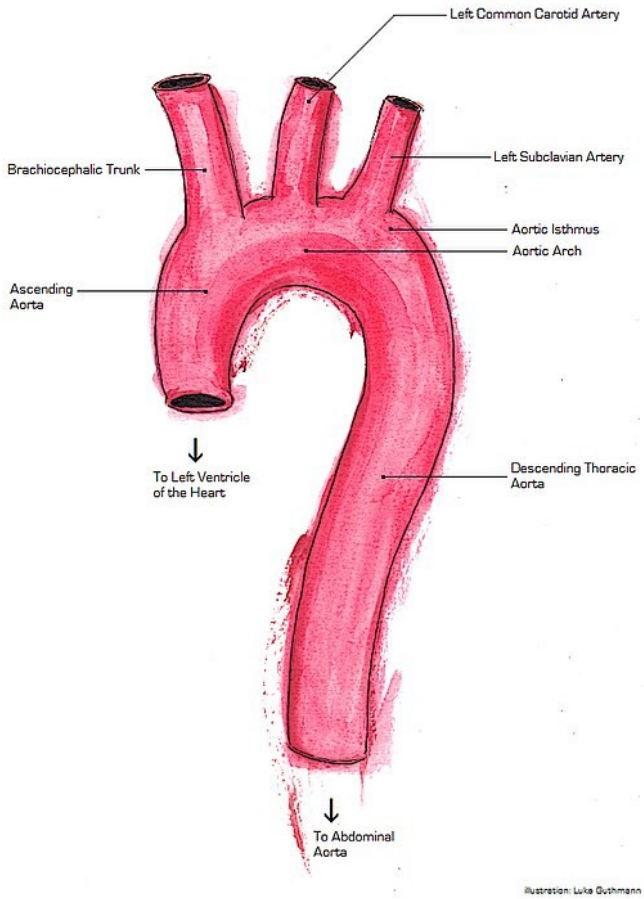
Osasyy siihen, miksi päätettiin luoda kokonaan uusi ohjelmisto eikä käyttää valmiiksi saatavilla olevia 3D-mallinnusohjelmia johtui artefakteista (artefacts), jotka ovat kuvissa esiintyviä vihreitä ja jotka eivät vastaa kuvassa esiintyvää anatomiaa. Niitä esiintyy lääketieteellisessä kuvantamisessa. Yleensä artefakteilla tarkoitetaan virheitä, joita tapahtuu potilaan kuvaamisvaiheessa, kun potilaasta otetaan esimerkiksi MRI tai CT kuvaa. Aortan kuvantaminen julkisesti saatavilla olevilla 3D-kuvantamisohjelmia käyttäen on johtanut artefakteihin, jotka kokonaisuudessa saavat 3D kuvannetun aortan ulkopinnan näyttämään karkealta/epätasaiselta, vaikka aortan pinta on todellisuudessa sileä. (Bibb, 2006)

3 TIETOPERUSTA

Projektin osalta tärkeimmät tietoperustat ovat kardiologia, radiologia, segmentointi ja C#-ohjelmointikieli.

3.1 Kardiologia – Aortan Aneurysma

Aortan Aneurysma selkokielellä tarkoittaa aortan laajentuma. Aortta on sydämestä alkava iso valtimo, josta muut valtimot haarautuvat (Aortan laajentuma, 2025). Aortta jaetaan rinta-aorttaan ja vatsa-aorttaan sekä toisaalta aortta jaetaan myös nousevaan osaan, kaareen ja laskevaan osaan (Kuva 1) (Airaksinen ym., 2016).



Kuva 1. Piirros kuva Aortasta https://upload.wikimedia.org/wikipedia/commons/a/a0/Aorta_Anatomy.jpg

Kyseessä oleva ohjelma keskittyy mallintamaan aortan nousevaa osaa.

Aortan kaaren ja nousevan osan läpimitat ovat noin 25–35 millimetriä potilaan sukupuolesta ja koosta riippuen. Aortan aneurysma johtaa repeämään 70 % potilaista. 55–60 millimetrin kokoisen laajentuman repeämisriski on aina merkittävä. Kuten monien muiden aorttasairauksien kanssa, aortan laajeneminen hoidetaan perinteisen kirurgisen leikkaushoidon avulla. Vaikka aneurysma on yleensä oireeton, leikkaus on aiheellinen, jos aortan läpimitta ylittää 55 millimetriä. (Airaksinen ym., 2016)

Tietokonetomografia ja magneettikuvaus ovat yleisimmin käytettyjä menetelmiä saada luotettavasti tietoa aortan komplikaatioista (Blanco Sequeiros ym., 2017).

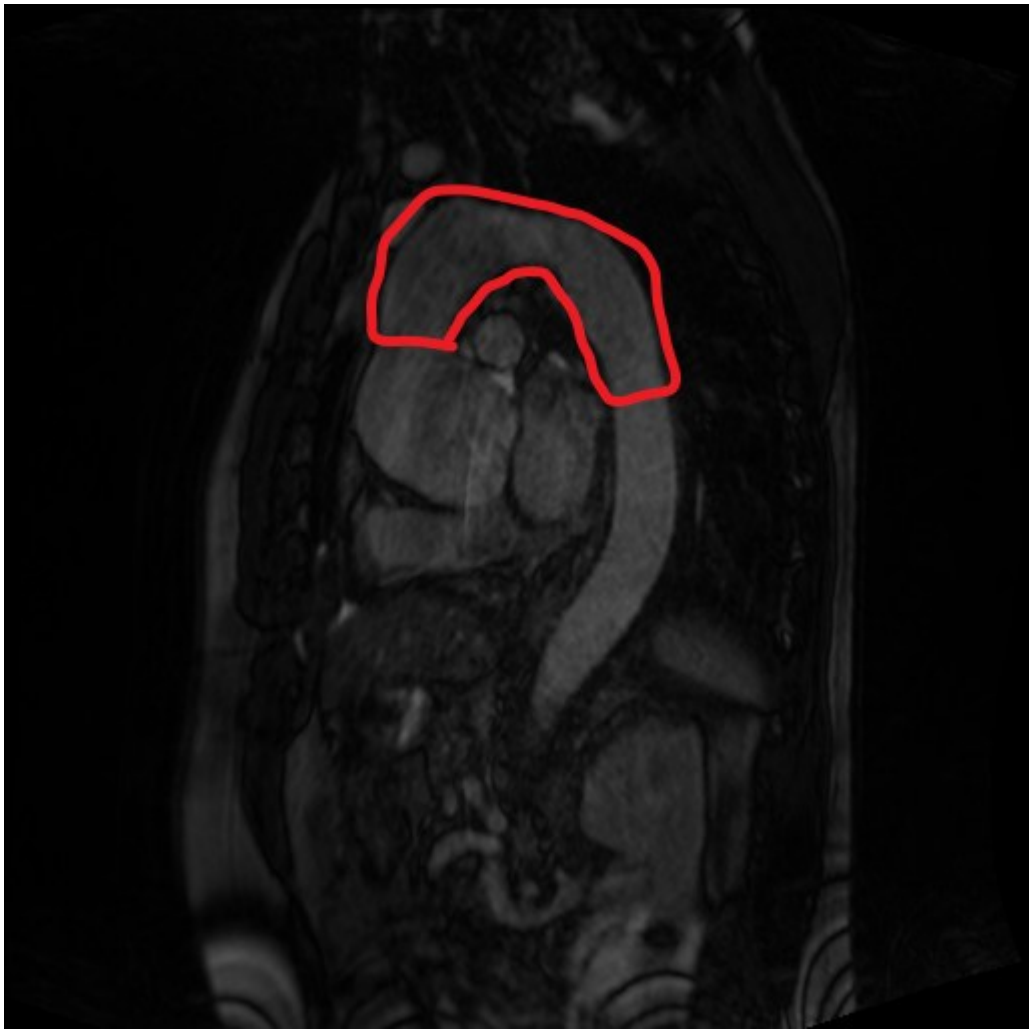
3.2 Radiologia

Kuvantamisohjelman on tarkoitus käyttää magneettikuvauksen ja tietokonetomografian kuvaformaatteja geometrian määrittämisessä ja 3D mallintamisessa.

Tietokonetomografiassa (TT) (computed tomography (CT)) käytetään röntgensäteilyä potilaan tutkimuksessa. Potilaan ympäri pyörivä röntgenputken lähettämä sädekeila lävistää potilaan ja kulkee synkronoidusti pyörivään anturiin. Kuva muodostus perustuu kehon läpäisseen säteilymäärän muutokseen. Lopputuloksena on poikkileikekuva. TT:n suurin etu on, että kuvaus tarjoaa yksityiskohtaista, samanaikaisesti sekä spesifistä, että sensitiivistä tietoa suuresta elinjoukosta samanaikaisesti. Toisaalta tutkittavan potilaan säderasitus on suurempi kuin tavanomaisessa röntgentutkimuksessa ja TT on kalliimpi kuin röntgentutkimus. (Blanco Sequeiros ym., 2017)

Aorttaa tutkittaessa Tietokonetomografia (TT) on ensisijainen menetelmä. TT on nopea, hyvin saatavilla ja kuvia voidaan tutkia halutuista suunnista. TT:n avulla saadut mittaustulokset ovat erittäin toistettavissa ja luotettavia ja säderasitus on paljon pienempi kuin nykyisillä laitteilla. (Airaksinen ym., 2016)

Magneettikuvaus (MK), magnetic resonance imaging (MRI) perustuu elimistön vety-ytimien eli protonien käyttäytymiseen ulkoisessa magneettikentässä (Kuva 2). Käytännössä vain herkkäliikkeiset kudoksien vesi ja rasvan protonit osallistuvat magneettikuvaussignaaliin. Kuvaussekvenssin aikana protonien spinien suuntaa pyritään muokkaamaan radiotaajuisilla poikkeuspulsseilla. Tätä kautta välittyy tietoa kudoksen kiinteästä makromolekyylirakenteesta ja sen muutoksista tautitiloissa. (Blanco Sequeiros ym., 2017)



Kuva 2. Anonymisoitu MRI kuva, Aortan yläosa ääriviivattu punaisella.

Sydämen ja verisuonten kuvaaminen ovat ehdottomasti eräitä magneettikuvauksen vahvuusalueita. Tietokonetomografiaan verrattuna MRI:lla on ylivertainen kudoserottelukyky, koska sen signaali riippuu vain kudoksen kemiallisesta koostumuksesta. Toisaalta MRI tutkimukset kestävät pitkään ja ne ovat kalliita. Lisäksi pitää varmistaa, että mitään magneettista materiaalia ei ole tuotu kuvauslaitteen läheisyyteen. (Blanco Sequeiros ym., 2017)

MRI on hyvä tutkimus aorttaa arvioitaessa, ja sitä käytetään erityisesti nuorilla potilailla seurantatutkimuksissa (Airaksinen ym., 2016).

3.3 Segmentointi

Lääketieteellisessä kuvasegmentoinnissa erotellaan ja tunnistetaan tietyt alueet tai rakenteet, jotka ovat käsittelyn kannalta tärkeimpiä esimerkiksi 3D-kuva-aineistossa, kuten magneettikuvauksessa (MRI) tai tietokonetomografiassa (CT). Segmentoinnin päätavoitteena on tunnistaa anatomian alueet, joita tarvitaan tiettyä tutkimusta varten. Ei-triviaalien lääketieteellisten kuvien segmentointi voi olla yksi vaikeammista ja aikaa vievimmistä kuvien prosessoinnin tehtävistä. (Synopsys, 2025)

Tässä opinnäytetyössä segmentointi keskittyy erityisesti aortan nousevan osan tunnistamiseen ja erottamiseen MRI-poikkikuvista. Segmentoinnin tavoitteena on luoda tarkka ja luotettava malli, joka mahdollistaa aortan ja muiden verisuonten automaattisen erottamisen kuvasta, jotta niitä voidaan käyttää jatkokäsittelyyn ja analyysiin. Tämä tapahtuu muun muassa rajauksen, kynnystyksen, reunojen tunnistamisen sekä morfologisten operaatioiden kuten eroosion ja dilaation avulla.

3.4 C#-ohjelmointi kieli

C# on Microsoftin kehittämä ohjelmointikieli, joka esiteltiin ensikertaa vuonna 2000 ja toimii .NET sovelluskehyksessä (W3Schools, n.d.). C# on yksinkertainen, turvallinen, moderni, olio-ohjelmointiin perustuva, Internet-keskeinen ja suorituskykyinen kieli .NET-kehitykseen (Liberty, 2002), (Ogala, 2020).

C# soveltuu erityisen hyvin kehitysympäristöihin, joissa kaivataan nopeaa ohjelmiston kehitystä, helppokäyttöisyyttä, ja integraatiota Microsoftin ekosysteemin kanssa. Sen yksinkertaistettu syntaksi, automatisoitu muistin käsittely ja monipuoliset/kattavat kirjastot tekevät siitä erinomaisen valinnan yrityssovelluksiin, verkkosovellusten kehittämiseen ja työpöytäohjelmistojen kehitykseen. Saumaton integraatio Visuals studion ja .NET framework parantavat edelleen kehittäjän tuottavuutta, nopeuttaen kehityssyklejä ja helpottavat laaja-alaisten sovellusten ylläpitoa ja huoltoa. (Ponggawa, 2024)

.NET ympäristöön kuuluvat ohjelmointikielet ovat Visual Basic .NET, C#, Visual C++ ja Visual J#. .NET-sovelluskehys koostuu kolmesta pääkomponentista: Ohjelmointikielet ja työkalut, ajonaikainen ympäristö, perusluokkakirjasto. (Moghadampour, 2012)

.NET oli alun perin Windows-käyttöjärjestelmien kehitysympäristöksi luotu sovelluskehys, jonka ensimmäinen kaupallinen versio ilmestyi vuoden 2000 alkupuolella. .NET-sovelluskehyksellä voidaan kirjoittaa muun muassa Windows- ja websovelluksia, mobiililaitteisiin tarkoitettuja sovelluksia sekä järjestelmiä, jotka koostuvat keskenään Internetin kautta kommunikoivista toisiinsa liitetyistä palveluista. (Moghadampour, 2012)

Visual Studio .NET on Microsoftin Rapid application development-integroitu kehitysympäristö, jolla voidaan kehittää monenlaisia .NET-sovelluksia. VS.NETillä voidaan kirjoittaa konsoliohjelmien lisäksi seuraavanlaisia sovelluksia: Windows-sovelluksia, joissa käytetään Windowsin tarjoamia erilaisia graafisia komponentteja, kuten näppäimiä, tekstikenttiä, valikoita jne. Tärkein Microsoftin tarjoama ohjelmointityökalu on Visual Studio.NET, joka on saatavilla eri kokoonpanoissa. (Moghadampour, 2012)

Kuvantamisohjelma koodattiin C# kieltä käyttäen. Sovellus on pääosin Windows-sovellus (Windows Forms (WinForms)) projekti. WinForms on .NET-kirjasto, jota käytetään graafisten käyttöliittymien luomiseen kehitysympäristössä. Projekti

tehtiin Visual Studio 2022 kehitysympäristöä käyttäen. Projekti käytti myös muita ylimääräisiä NuGet-paketteja kuten esimerkiksi fo-dicom 5.1.2, jonka avulla ohjelman kuvanesittämistoiminnat pystyvät esittämään *.dcm formaatin MRI-kuvia. Useimmiten sairaaloiden työasemat käyttävät Windowsia, joten C# ja .NET tarjoavat hyvän yhteensopivuuden. Samalla Windows Forms on kevyt ja nopea tapa rakentaa graafisia käyttöliittymiä ilman monimutkaisia riippuvuuksia.

4 KEHITTÄMISPROSESSIN KUVAUS

Nykypäivänä on monia erilaisia ohjelmistonkehitysmalleja ja -metodeja, jotka soveltuvat erilaisiin projekteihin, näistä tunnetuimpia ovat esim. Agile ja vesiputousmalli. Ohjelmistotuotannon projekteissa lähes aina käytetään versionhallinta- ja projektinhallintapalveluita. CorFlux-projektissa päätavoitteena oli käyttää Jira- ja Confluence tuotteita projektin hallintaan ja BitBucket-alustaa version hallintaan.

CorFlux-projektin ohjelmistokehitys ei tässä vaiheessa tapahtunut minkään tunnetun ohjelmistotuotantoprosessimallin mukaan. Prosessissa edettiin projektin tarpeiden ja ryhmän kokemuksen mukaisesti. Projektiryhmän päätöksellä tämä ensimmäinen, ns. demoversio päätettiin toteuttaa vielä ilman Jira/Confluencea jolloin ohjelmisto jaettiin moduuleihin, jotka ohjelmoitiin yksi kerrallaan niiden toimivuuden testaamiseksi. Tähän vaikutti myös se, että Jira/Confluence-osaamista ei projektin alkaessa vielä ollut vaikka ko. kehitysympäristön koulutus projektiryhmälle aloitettiinkin myöhemmin.

4.1 Kehitysympäristö

Aortan repeämäriskiä laskevan CorFlux-ohjelmiston ensimmäinen prototyyppiversio toteutettiin Matlab-ohjelmistolla, jolla yleinen laskentaperiaatteen toimivuus tarkistettiin. Kaupallinen Matlab on lisenssikulujen takia kuitenkin varsin kallis, joten varsinaisen CorFlux-projektin alussa päätettiin projektiryhmän toimesta tehdä varsinainen toteutus C#-ohjelmointikielellä. Toiminnallinen toteutus pohjautui C#-versiossa Matlabilla luotuun 1. prototyyppiversioon. Tämän projektin kuvankäsittelyosion prosessit ja vaiheet olivat suurin piirtein jo määritelty Matlab-prototyypissä. (Kurki, henkilökohtainen tiedonanto, 2025)

CorFlux-projektin kuvantamisohjelma tehtiin Visual Studio 2022 kehitysympäristöä käyttäen. Käyttäen C#-ohjelmointikieltä ohjelmasta ohjelmoitiin käyttäen

Windows-sovellus WinForm-mallipohjaa, joka käyttää Windowsin tarjoamia erilaisia graafisia komponentteja. Ohjelman on tarkoitus kuvantaa aorttaa ja määrittää aortan geometriaa käyttäen kuvantamisessa saatuja MRI ja CT kuvia.

C# ei ollut ainoa mahdollinen vaihtoehto, mutta kaupallisen ohjelmiston kehityksessä kielivalinta on tehtävä huolellisesti. Tiedettiin, että tämän tason tuotetta ei voi toteuttaa esim. Matlabilla. Esimerkiksi Python on tuotantokielenä myös ongelmallinen tuotteen ylläpidettävyyden ja suojaamisen kannalta. Tavoitteena oli pitäytyä täysin binäärijakelussa, joten kaikki tulkkaavat sovellukset ovat tällöin epäsopivia. C++ harkittiin, mutta arvioimme sen olevan liian matalan tason kieli ja tarjoavan rajallisesti tukea kuvankäsittelyyn ja käyttöliittymien toteutukseen. Lääketieteen alueella tehtävissä ohjelmoinneissa C# tarjoaa suuremman määrän valmiita ohjelmistokirjastoja erilaisten lääketieteellisesti laskennallisten tapausten koodaukseen. (Karjalainen, henkilökohtainen tiedonanto, 2025)

Kaikki ohjelman syötteenä olevat kuvat käyttävät DICOM (Digital Imaging and Communications in Medicine) formaattia. DICOM on lääketieteellisen kuvantamisen kansainvälinen standardi määritellen lääketieteellisten kuvien formaatit. DICOM-järjestelmää käytetään lähes kaikissa radiologian, kardiologian kuvantamisen ja sädehoidon laitteissa (röntgen, TT, MRI, ultraääni jne.). (About DICOM Overview, n.d.)

Tässä työssä esitetyn toteutuksen rajoittavana puolena on se, että kaikki aiempien moduulien tuottama uusi data tulee välittää seuraavalle moduulille ja tämä päätettiin tehdä hakemistokohtaisesti kuvaformaatti- (*.dcm, *.tif) ja tekstiformaattitiedostoina (*.txt, *.csv). Toisaalta tämä mahdollistaa suoremman datan oikeellisuuden tarkastamisen ennen kuin seuraava moduuli lukee sen ja alkaa käyttämään sitä. Myöhemmässä, kaupallisessa vaiheessa, prototyyppivaiheen tiedostosiirto olisi tarkoitus muuttaa yhden isomman tietorakenteen hallinnaksi ilman datan aukikirjoittamista.

4.2 Kehitysprosessi

Kehityksen alkuvaiheissa tehtiin päätös jakaa ohjelman eri prosessit omiin moduuleihinsa, joilla jokaisella on omat käyttöliittymäikkunansa (WinForms), sekä syöte- ja ulostulokansionsa, joiden sisältö määräytyy aiempien moduulien ajon tuloksena, lukuun ottamatta ensimmäisiä login- ja määrittelymoduuleja. Tällä tavoin virheet voitiin eristää pienempiin kokonaisuuksiin, mikä helpottaa niiden ennaltaehkäisyä ja korjaamista. Näin virheiden etsiminen ei edellytä koko ohjelman läpikäymistä.

Ohjelmiston kehityksen ja toiminnan kannalta alkuvaiheen C#-ohjelmointi päätettiin toteuttaa ns. Microsoft form-periaatteella. Tämä mahdollistaa sen, että jokainen tiedoston hallinnan, kliinisten potilasparametrien syötön sekä segmentoinnin osaprosessien toteutus voidaan toteuttaa modulaarisesti sekä suorittaa itsenäisesti. Tällöin virheiden hallinta ja korjaukset voidaan suorittaa myös moduulikohtaisesti.

Alkuperäinen suunnitelma oli, että DICOM formaatti olisi ollut ainoa kuva fomaatti, jota ohjelma olisi käyttänyt, mutta tämä ei ollut mahdollista. Huolimatta siitä, että ohjelmaan oli liitetty paketti (package), jonka kautta DICOM kuvien käsittely olisi ollut mahdollista, ohjelmisto kaatui toistuvasti, kun sillä käsiteltiin DICOM kuvia. Tästä syystä tehtiin päätös, että MRi DICOM kuvat muutettaisiin niiden DICOM (*.dcm) formaatista, tiff (*.tif) formaattiin.

Ohjelmistoa varten osaksi sitä prosessia, jonka ohjelma ajaa, liitettiin IrfanView-kuvankäsittelyohjelma, joka voitaisiin avata CorFlux ohjelman kautta, kun halutiin tarkastella DICOM kuvia. Se pystyy myös näyttämään laajan valikoiman kuvia eri formaateissa kuten DICOM ja tiff. (What is IrfanView?, 1996–2025)

Aluksi ohjelmaan koodattiin tekstilaatikot, jotka voitiin joko täyttää manuaalisesti, ”kopioi”- ja ”liitä”-toiminoilla taikka käyttämällä, ”Avaa”- ja ”Tallenna nimellä”-toimintoja, joiden avaamiseen oli ohjelmoitu omat painikkeet. Projektin aikataulusta

johtuen täytyi kovakoodata ne tiedostot, joita ohjelman moduulit käyttäisivät prosesseissaan ja mihin tiedostoihin näiden prosessien tulokset sijoittuisivat.

Kuvantamisohjelma käytti useaa eri paketteja (Taulukko 1), jotta ohjelmaan saatiin tarvittavat toimivuudet. Useampi näistä lisättiin projektin aikana, kun kävi ilmi, että niille oli tarvetta.

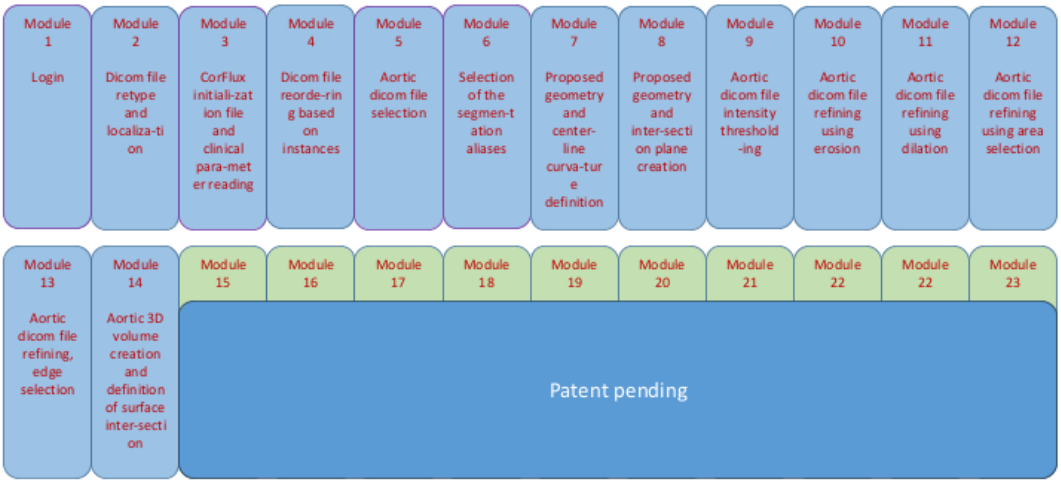
Taulukko 1 CorFlux-Ohjelmistossa käytetyt paketit.

Kategoria	**Paketti**
Kuvankäsittely ja Tietokonenäkö	AForge, AForge.Imaging, Emgu.CV, Emgu.CV.Bitmap, Emgu.CV.runtime.windows, Magick.NET
Data Prosessointi	CsvHelper, MathNet.Numerics
Visualisointi	WinForms.DataVisualization
Grafiikka ja UI integraatio	System.Drawing.Common, WindowsAPICodePack-Shell
Lääketieteellinen kuvantaminen	fo-dicom.

5 KEHITTÄMISTOIMINNAN TULOKSEN KUVAUS

CorFlux projektin kuvantamisohjelmasta valmistui 13 moduulia. Moduulin 14 ohjelmointi jäi kesken, kun projektin ohjelmistokehitys osuus päättyi toukokuussa 2024. Kaikkien valmiiksi saatujen moduulien (moduulit 1–13) osalta saavutettiin erinomainen toimintavarmuus.

CorFlux-ohjelmisto koostuu noin 23 moduulista (Kuva 3). Kuvien käsittelyyn ja segmentointiin liittyvät moduulit ovat esillä, repeämäanalyysin moduulit ovat peitettyinä salassapidon vuoksi, joka johtuu meneillään olevasta patentointiprosessista.



Kuva 3. CorFlux-Ohjelmiston Moduuli Rakenne. Moduulit 1–14 ovat osa tätä projektia

Jokainen moduuli muodostettiin omaksi erilliseksi Form-osiokseen. Moduuli 1 eli Form2-osio sisältää ohjelmaan sisäänkirjautumiseen tarkoitetun käyttöliittymän, joka pyytää käyttäjältä tunnusta ja salasanaa. Tässä tapauksessa on kovakoodattu tietty tunnus ja salasana, jotka annettaessa omiin tekstilaatikoihinsa käyttäjä pääsee sisään. Sisäänkirjautuminen on vain muodollisuus.

Moduuleissa 3–7 pitää täyttää tekstikentät joko kirjoittamalla tai ”browse”-painikkeen avulla, joka avaa tiedostodialogin, täytetään. Yleensä sellainen löytyy painikkeelle luodun metodin sisältä (Koodi 1).

```
CommonOpenFileDialog dialog = new CommonOpenFileDialog();
dialog.InitialDirectory = "C:\\Users";
dialog.IsFolderPicker = true;
if (dialog.ShowDialog() == CommonFileDialogResult.Ok)
{
    MessageBox.Show("You selected: " + dialog.FileName);
    SourceTextBox.Text = dialog.FileName;
}
```

Moduulista 8 eteenpäin käytettiin kovakoodattuja syöte- ja ulostulotiedostoja ja -kansioita, jotka moduuli etsii siitä kansiosta, jossa sovelluksen *.exe-tiedosto sijaitsee. Rivit 1–3, koostuvat siitä, kun ohjelma omasta kansiostaan etsii, syöte- ja ulostulokansiot. Rivit 4–5 yhdistävät syöte- ja ulostulokansioiden polun ja itse kansion nimen. Riveillä 6–17 tarkistetaan, onko ulostulokansiota olemassa ja jos ei, niin sellainen luodaan. Tämän lisäksi alla olevassa esimerkissä on vielä silmukka, joka tarkistaa, onko tiedostossa vanhoja kuvia joltain aiemmalta käyttökerralta. Jos vanhoja kuvia on ulostulokansiossa, ne poistetaan ennen kuin viimeisimmän prosessin kuvat lisätään. (Koodi 1)

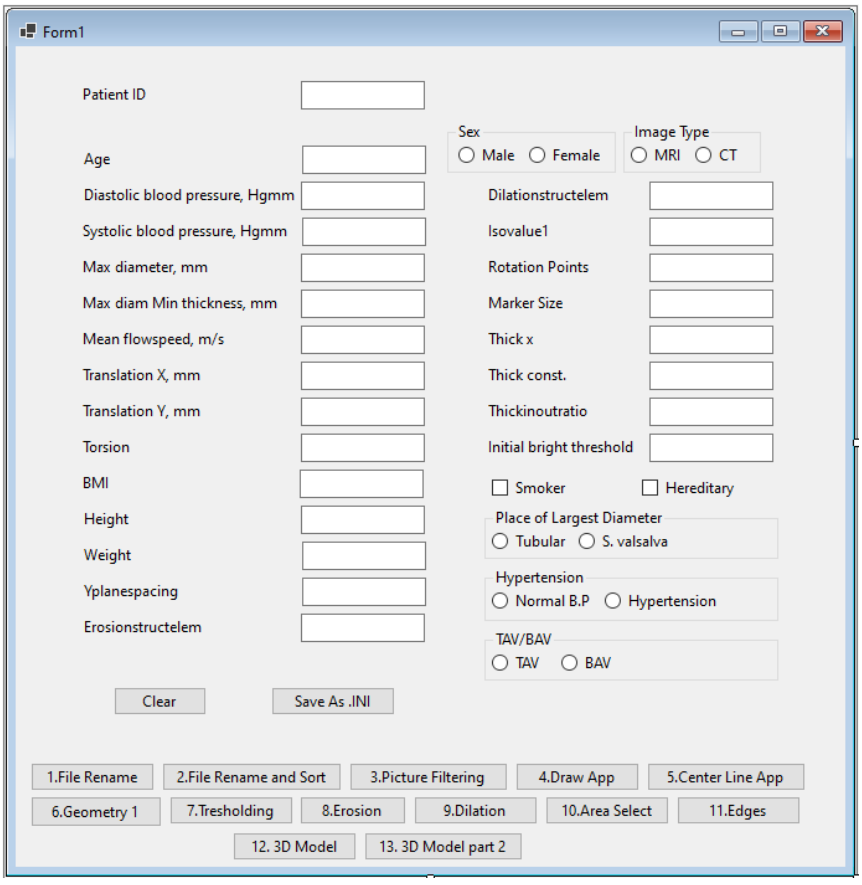
```
// Open the folder St3 and St4 for the *.tif file thresholding
// Get the current working directory
string currentFolder = Directory.GetCurrentDirectory();
// Open the folder for *.tif file thresholding
string inputFolderName = "St3";
string outputFolderName = "St4";
// Construct absolute paths
string inputFolderPath = Path.Combine(currentFolder, inputFolderName);
string outputFolderPath = Path.Combine(currentFolder,
outputFolderName);
// Create the output folder if it doesn't exist
if (!Directory.Exists(outputFolderPath))
{
    Directory.CreateDirectory(outputFolderPath);
}
else
{
    // Delete old files from the output folder
    string[] oldFiles = Directory.GetFiles(outputFolderPath, "*.dcm");
    foreach (string file in oldFiles)
    {
        File.Delete(file);
    }
}
```

Koodi 1. Koodiosio, jossa määritetään kovakoodatut syöte- ja ulostulotiedostot sekä muita ulostuloon liittyviä toimintoja.

Ohjelmisto myös käyttää moduuleissaan *.csv-tiedostoja ja käyttäjän sovelluksella luomia *.ini-tiedostoja, joista tiedot tarvittaessa haetaan komennolla `string[] lines = File.ReadAllLines("CorFlux.ini");`

5.1 Tiedosto-operaatiot

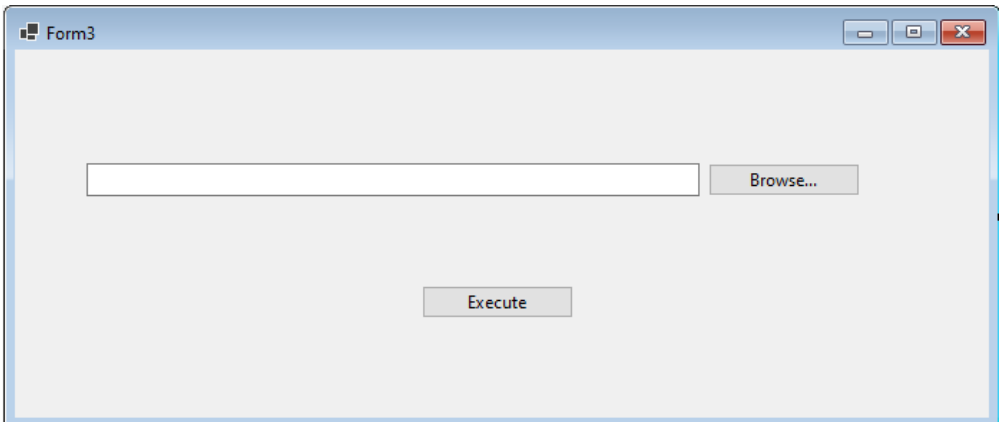
Ohjelma avautuu sisäänkirjautumisen jälkeen Moduuliin 2 eli Form1-osioon. Tässä Form-osiossa on näkymä, johon täytetään potilastiedot. Annettavat tiedot ovat mm. ikä, verenpaine, aortan halkaisija, pituus, paino, seinämän paksuus, sukupuoli ja valitaan kuvatyyppi MRI:n ja TT:n välillä. Nämä tiedot tallennetaan *.ini-tiedostoon. Suurinta osaa tässä kohtaa annetuista tiedoista ei käytetä muissa moduuleissa. (Kuva 4)



Kuva 4. Form1 Moduulin 2 Graafinen ulkoasu.

Kun ”Save As .INI”-painiketta painetaan, tallentuvat kaikki täytetyt kohdat ja valinnat *.ini-tiedostoon `WriteLine`-metodin avulla. Metodi kirjoittaa dokumenttiin avaimen ja sen perään käyttäjän määrittämät arvot. `CheckBox` antaa binäärisen arvon 0 tai 1 riippuen siitä, onko kyseinen laatikko valittu. `RadioButton` ovat ryhmitelty pareittain omiin `GroupBox`-metodeihinsa ja antavat myös binäärisen arvon 0 tai 1 avaimien perään riippuen siitä, kumpi pari on valittu.

Moduuli 3, Form3: MRI Kuvatiedostojen uudelleen nimeäminen. Tyypillisesti yhdestä potilaasta on 64 MRI poikkileikekuvaa, jotka ovat yleensä mielivaltaisessa järjestyksessä ko. hakemistossa. Tarkoitus on antaa jokaiselle DICOM-kuvalle, joka on valitun kansion sisällä, *.dcm pääte. Kuvatiedostot sisältävät pikselidataa ja metadata-alueen, jossa on potilaan kuvantamisessa käytetyt menetelmäparametrit (CT- ja MRi-kuvaus). Nämä uudelleen nimetyt kuvat kopioidaan uuteen ulostulokansioon, joka toimii seuraavien moduulien syötekansiona.



Kuva 5. Form3 Moduulin 3 Graafinen ulkoasu

Kuvien uudelleen nimeäminen tapahtuu seuraavasti: kun *.dcm kuvien lähde on valittu ja ”Execute”-painiketta on painettu (Kuva 5), ohjelma ensin tarkistaa, onko ulostulokansio olemassa, ja onko siellä tiedostoja, tarvittaessa luoden uuden kansion ja poistaa vanhat kuvat kansiosta, jos sellainen on olemassa (Koodi 2).

```
// Define destination folder
string destinationFolder = "St1";
// Remove files from the destination folder if it exists and is not
empty
```

```
if (Directory.Exists(destinationFolder))
{
    string[] filesInDestination =
Directory.GetFiles(destinationFolder);
    if (filesInDestination.Length > 0)
    {
        foreach (var file in filesInDestination)
        {
            File.Delete(file);
        }
    }
}
```

Koodi 2. Koodiosio jossa ulostulokansion sisällä olevat tiedostot poistetaan, jos se sisältää tiedostoja.

Seuraavassa osiossa, jos ulostulokansio ei ole, luodaan uusi kansio. Lopulta lisätään jokaisen kuvatiedoston perään *.dcm-pääte (Koodi 3).

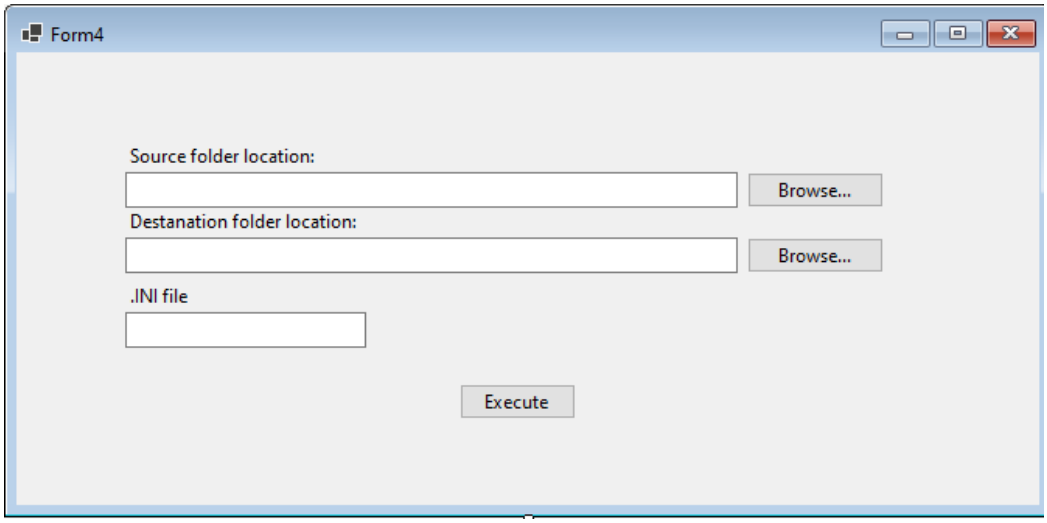
```
Directory.CreateDirectory(destinationFolder);
// Get a list of files in the source folder
string[] fileList = Directory.GetFiles(sourceFolder);
// Loop through each file and copy to the destination folder with the
new extension
foreach (var filePath in fileList)
{
    FileInfo fileInfo = new FileInfo(filePath);

    if (fileInfo.Name.Length > 4 && fileInfo.Extension != ".dcm")
    {
        string fileName =
Path.GetFileNameWithoutExtension(fileInfo.Name);
        string destinationFilePath = Path.Combine(destinationFolder,
fileName + ".dcm");
        File.Copy(filePath, destinationFilePath);
    }
}
```

Koodi 3. Koodiosio, joka tarvittaessa luo uuden ulostulokansio, tiedostot kopioidaan ja niiden perään lisätään *.dcm-pääte.

Moduuli 4, Form4: Tässä vaiheessa uudelleennimetyt kuvat ovat epäjärjestyksessä. Kaikki kuvat käydään läpi siten, että katsotaan kunkin *.dcm-tiedoston metadataosa ja haetaan kuvien instanssinumerot (`instanceNumber`), joiden mukaan prototyyppi lajittelee kuvat oikeaan kuvantamisjärjestykseen ja uudelleennimeää tiedostot instanssinumeroinnin ja Moduulissa 2 annetun potilaan ID:n mukaisesti.

Tämän lisäksi nimeen liitetään kuvatyyppi (CT tai MRI). Aluksi täytetään 3 teksti-laatikkoa. Niihin laitetaan kansio, jossa syötteenä olevat kuvat sijaitsevat, lajiteltujen kuvien ulostulohakemisto ja *.ini-tiedoston nimi, joka haetaan prototyypin omasta hakemistosta, jossa sen *.exe-tiedosto sijaitsee.



Kuva 6. Form4 Moduuli 4 Graafinen ulkoasu.

Form4 on ensimmäinen, joka käyttää DICOM-kuvatiedostojen käsittelyä varten ladattua ”fo-dicom”-kirjastopakettia. Kun syöte- ja ulostulotiedostot sekä *.ini-tiedosto on määritetty ja moduulin prosessi käynnistetty painamalla ”Execute” (Kuva 6), etsii se ensin *.ini-tiedoston ja hakee sieltä potilaan ID:n (Koodi 4).

```
// Open the file for reading
StreamReader reader = new StreamReader(INITextBox.Text);
// Check if the file was opened successfully
if (reader == null)
{
    MessageBox.Show("Unable to open the file.");
    return;
}
// Initialize variables to store parameters
Dictionary<string, string> charParams = new Dictionary<string,
string>();
Dictionary<string, double> numParams = new Dictionary<string,
double>();

// Read lines from the file
while (!reader.EndOfStream)
{
    // Read the current line
    string line = reader.ReadLine();
```

```
    if (line.Contains("="))
    {
        // Split the line into key and value
        string[] keyValue = line.Split('=');
        string key = keyValue[0].Trim();
        string value = keyValue[1].Trim();
        // Check if the line starts with 'CorFluxChar' or 'CFAParam'
        if (line.Contains("CorFluxChar"))
        {
            charParams[key] = value;
        }
        else if (line.Contains("CFAParam"))
        {
            numParams[key] = double.Parse(value);
        }
    }
}
```

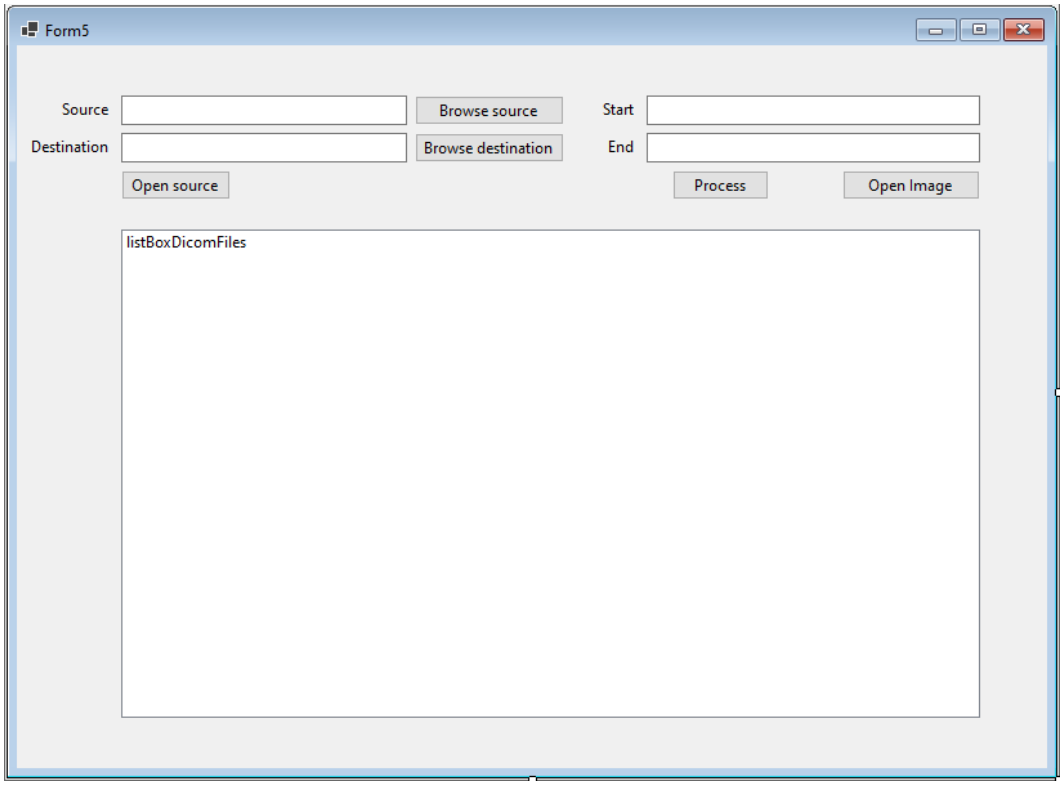
Koodi 4. Tämä koodiosio lukee parametrejä *.ini asetustiedosta ja tallentaa ne merkkipohjaisiin ja numeerisiin sanakirjoihin.

Tämän jälkeen prosessi siirtyy lajittelemaan syötekansiossa olevia DICOM-kuvia. Tässä vaiheessa prosessi käyttää ”fo-dicom”-kirjastoa selvittämään DICOM-kuvien metadatasta niiden instanssinumerot. Potilaan ID:tä käytetään kuvatiedostojen nimeämiseen. (Koodi 5)

```
// Get a list of DICOM files in the input folder
string[] dicomFiles = Directory.GetFiles(inputFolderPath, "*.dcm");
// Loop through each DICOM file
foreach (string dicomFile in dicomFiles)
{
    // Read DICOM file using the dicom library
    DicomFile dicomFileObject = DicomFile.Open(dicomFile);
    // Get instance number (assuming there is a tag for instance
number)
    int instanceNumber =
dicomFileObject.Dataset.GetSingleValue<int>(DicomTag.InstanceNumber);
    // Create the new filename
    string newFilename =
$"Pat_{patientID}_Typ_{imageType}_inst_{instanceNumber:D4}.dcm";
    // Copy the DICOM file to the output folder with the new filename
    dicomFileObject.Save(Path.Combine(outputFolderPath, newFilename));
}
MessageBox.Show("DICOM files organized and stored in St2.");
```

Koodi 5. Tämä koodiosio järjestää, uudelleen nimeää ja tallentaa joukon DICOM-tiedostoja haluttuun kansioon.

Moduuli 5, Form5: Tässä osassa poistetaan kaikki ne ylimääräiset kuvat, joissa ei näy aorttaa. Kuvat talletetaan sekä *.dcm- että *.tif-formaatissa. Tämä siksi, että käytetyt ohjelmakirjastot myöhemmissä Form-osioissa eivät tue dcm-kuvien muokkausta ja tulevat piirtotoiminnot toimivat vain standardikuvaformaateille (*.tif, *.gif, *.png, jne.). "Open Image"-näppäin käynnistää ohjelman IrfanView I_view64.exe, joka avaa *.tif-tiedostot. Form5 luo myös CSV-tiedoston, jossa on valittujen leikekuvien ensimmäinen ja viimeinen instanssinumero. Valinta tapahtuu siten, että avataan dialogi ja siellä omiin tekstilaatikoihinsa annetaan ensimmäisen ja viimeisen leikekuvan numero (Kuva 7).



Kuva 7. Form5 Moduuli 5 Graafinen ulkoasu.

Kuvien valikointi alkaa siitä, että käyttäjä hakee IrfanViewn avulla aorttaa esittävien kuvien alku- ja loppuindeksit, jotka hän syöttää ”start”- ja ”end”-tekstikenttiin (Kuva 7). Tämän jälkeen ohjelma luo yksinkertaisen yksirivisen listan, joka sisältää nämä luvut. Luvut tallennetaan CSV-tiedostoon "S4d_dcmstartstop.csv" riippuvuuskirjaston ”CsvHelper” avulla (Koodi 6).

```
int a = int.Parse(textBoxStart.Text);
int b = int.Parse(textBoxEnd.Text);
// Store the values in a matrix
List<List<int>> M = new List<List<int>>
{
    new List<int> { a, b }
};
// Write the matrix to a CSV file
using (var writer = new StreamWriter("S4d_dcmstartstop.csv"))
using (var csv = new CsvWriter(writer,
System.Globalization.CultureInfo.InvariantCulture))
{
    foreach (var record in M)
    {
        foreach (var field in record)
        {
            csv.WriteField(field);
            csv.NextRecord();
        }
        csv.NextRecord();
    }
}
```

Koodi 6. Tämä koodiosio listaa ja tallentaa ”start” ja ”end” tekstikentistä saadut luvut CSV-tiedostoon.

Tämän jälkeen käyttäen ”fo-dicom”-kirjastoa prosessi kopioi ulostulokansioon vain arvojen a ja b välillä olevat DICOM-kuvat. Kuvista tehdään kopiot, jotka ovat tiff (*.tif) kuvaformaatissa. DICOM-kuvat muutetaan käyttämällä ”Magick.NET-Q16-AnyCPU”-riippuvuuskirjastoa. (Koodi 7)

```
// Iterate over the specified range of DICOM files
for (int i = a; i <= b; i++)
{
    // Check if the file index is within the valid range
    if (i < dicomFiles.Length)
    {
        // Construct the paths for the current input and output images
        string currentImagePath = dicomFiles[i];
        string outputImageNameWithoutExtension =
Path.GetFileNameWithoutExtension(currentImagePath);
        string outputImagePathDCM = Path.Combine(outputFolderPath,
Path.GetFileName(currentImagePath));
        string outputImagePathTIF = Path.Combine(outputFolderPath,
outputImageNameWithoutExtension + ".tif");
        // Copy the original DICOM image to the output folder
        File.Copy(currentImagePath, outputImagePathDCM);
        using (MagickImage image = new MagickImage(currentImagePath))
        {
            image.Format = MagickFormat.Tiff;
            image.Write(outputImagePathTIF);
        }
```

```
    }
    else
    {
        MessageBox.Show("Image index out of range.");
    }
}
```

Koodi 7. Tämä osio käy läpi valitun indeksivälin DICOM-kuvat, kopioi jokaisen alkuperäisen DICOM-tiedoston uuteen kansioon ja tekee niistä kopiot jotka ovat TIFF-muodossa.

”Open Image”-painike näyttää kuvan, jos käyttäjä on sellaisen listakentästä valinnut. Painike avaa kuvan ”IrfanView”-ohjelmassa (Koodi 8).

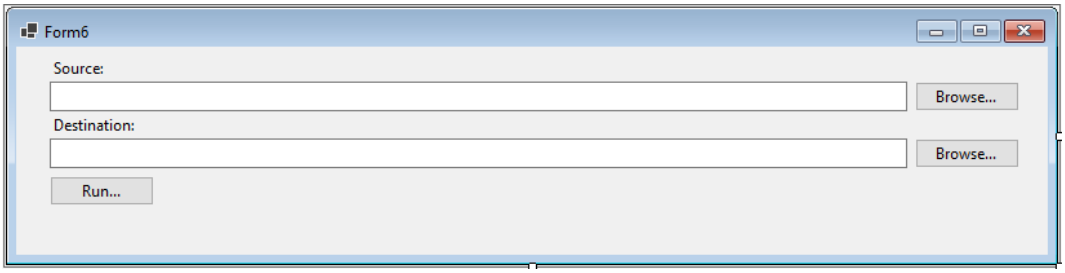
```
private void buttonOpenImage_Click(object sender, EventArgs e)
{
    // Open the selected image using IrfanView
    if (listBoxDicomFiles.SelectedIndex >= 0)
    {
        string selectedImagePath =
dicomFiles[listBoxDicomFiles.SelectedIndex];
        System.Diagnostics.Process.Start("C:\\Program
Files\\IrfanView\\i_view64.exe", $"\"{selectedImagePath}\"");
    }
}
```

Koodi 8. Tämä metodi avaa listasta valitun kuvan DICOM-kuvan IrfanView sovelluksessa.

5.2 Segmentointialueiden valinta

Moduuli 6, Form6: Tässä Form6-osiossa valitaan edellisessä osiossa valituista kuvista 3 kuvaa, jotka edustavat ns. segmentointialiaksia eli kuvia, jotka ohjaavat segmentointiprosessia. Tässä tapauksessa valituista alku-, keski-, ja loppukuvista rajataan alue, joka sisältää aortan poikkileikkeen. Kun prosessi ajetaan, avautuu `PictureBox`, jossa haluttu aorttaa sisältävä alue rajataan monikulmiolla. pictureBoxissa näkyvästä kuvasta valitaan pisteet aortan ympäriltä ja pisteiden välille tulevat janat rajaavat alueen. Käyttäjä asettaa pisteet kaikissa kuvissa aortta-alueen välittömään läheisyyteen. Keskikuvan osalta piirto poikkeaa siten, että kuvassa on luotava aortan alueen ulko- ja sisäkaarteen puolelle yhtä monta pistettä (tyypillisesti

5–8 per ulko- ja sisäkaarre). Näiden kolmen kuvan alias numerot lisätään luotavaan CSV tiedostoon. Alueiden valinnan jälkeen kaikille valituille *.tif -kuville suoritetaan aluevalinta siten, että alku-, keski- ja loppukuvien välisten alueiden koot interpoloidaan eli monikulmion ala muuttuu lineaarisesti segmentointialiasten välisissä leikkeissä. Näin leikekuviin jää ainoastaan monikulmion sisältämä aortan alue ja samalla varmistetaan, että aortan kuva-alan muuttuessa myös sitä ympäröivän monikulmion ala muuttuu vastaavasti. Keskikuvan monikulmion pisteiden koordinaatit tallennetaan omaan CSV-tiedostoon; tätä käytetään seuraavassa vaiheessa.



Kuva 8. Form6 Moduuli 6 Graafinen ulkoasu. Tämä on myös Form7 Moduulin 7 ulkoasu.

Prosessi käynnistää ensin IrfanViewn, joka näyttää syötetiedoston sisällä olevat aorttakuvat. Tämän jälkeen avautuu Windowsin tiedostodialogi, jonka avulla valitaan haluttu ensimmäinen segmentointialias, joka on .tif-kuva halutusta aortan MRI leikkeestä. Kun tämä kuva on valittu, avautuu `PictureBox`-kuvalaatikko. Tälle kuvalle luodaan liite CSV-tiedostoon. Tähän kuvalaatikkoon rajataan pistemonikulmiona tarvittava segmentointialue. Tälle monikulmiorajaukselle on oma Form metodi. Pisteiden koordinaateille luodaan oma `polygonCoords` niminen lista. Tapahtumakäsittelijä `form.MouseClick` määrittää, että koordinaatit määritetään hiiren klikkauksella ja `form.Paint` luo pisteet ja janat pisteiden välille, kun hiirellä painetaan `PictureBox` kuvaa. Lopuksi kaikki polygon-koordinaatit tallennetaan CSV-tiedostoon. Tämä sama prosessi tapahtuu kaksi kertaa tässä ohjelmassa, sekä alku- että keskikuville. (Liite 1)

Seuraava tarkastelu liittyy Liitteessä 2 esitettyyn koodiin. Seuraavaksi tapahtuu varsinainen segmentointiprosessi. DICOM-lähdetiedostoista haetaan kahden viitekuvan metatieto, tarkalleen ottaen kuvan `instanceNumber`. Nämä tallennetaan tiedostoon "S4_aliasnumbers.csv”. Seuraavaksi kaikki kuvat käydään läpi ja valitaan käytettävä, polygonin määrittämä maskialue. Polygonista luodaan maski ja sovelletaan sitä hakemiston kuviin. Ulostulokansioon talletetaan vain kuvien maskialueet. (Liite 2)

5.3 Perusgeometrian määritys

Moduuli 7, Form7: Form7 on graafisesti samannäköinen kuin Form6. Tässä formissa käytetään toisen segmentointialiaksen tallennettujen pisteiden koordinaatteja laskemaan aortan keskilinjalla kulkevan kaaren alustava säde. Se lasketaan piirrettyjen ulko- ja sisäkaarteen muodostamien pisteparien etäisyyden puoliväleistä. Tallennetaan etäisyyksien keskikohdat, eli arvioidun keskilinjan koordinaatit uuteen CSV-tiedostoon. Keskilinjan muodostaman kaarevuussäteen arvolla on suuri vaikutus aortan repeämäindeksiin (Kurki, henkilökohtainen tiedonanto, 2025).

Seuraava tarkastelu liittyy Liitteessä 3 esitettyyn koodiin. Käyttäjän on ensin valittava kuva. Kun kuva on valittu, ohjelma lataa kuva koordinaatit "S4_polygon1.csv"-tiedostosta. Tämä kuva muutetaan harmaasävyseksi ja haetaan `blobCounter`-olion avulla suurin yhtenäinen alue. Kun blob on haettu, niin luodaan keskilinja. Keskilinjan koordinaatit lasketaan pareittain polygonin vastakkaisien reunojen koordinaateista. Lopulta uusi kuva tallennetaan nimellä "CenterlineImage.tif" jossa keskiviivan koordinaattipisteet esiintyvät vihreinä. (Liite 3)

Keskiviiva talletetaan "S4b_aarcent.csv"-tiedostoon `SaveCenterlineCoordinates` -metodin avulla, joka sisältää ulostulon tallennuksen ja kirjoittaa tarvittavat koordinaatit tiedostoon (Koodi 9).

```
static void SaveCenterlineCoordinates(string inputFolderPath,
List<System.Drawing.PointF> centerlineCoords)
{
```

```
    string outputFilePath = Path.Combine(inputFolderPath,
"S4b_aarcent.csv");
    using (StreamWriter writer = new StreamWriter(outputFilePath))
    {
        foreach (var point in centerlineCoords)
        {
            writer.WriteLine(point.X + "," + point.Y);
        }
    }
    MessageBox.Show("Centerline coordinates saved to: " +
outputFilePath);
}
```

Koodi 9. Metodi joka tallentaa Keskiviivan koordinaatit CSV-tiedostoon.

Moduuli 8, Form8: Tämä form tuottaa edellisessä formissa luotujen keskilinjan koordinaattien kautta ympyräkaarisovituksen pienimmän neliösumman menetelmää (PNS-menetelmä) hyödyntämällä. PNS-menetelmässä tehdään sovitus niin, että alkuperäisten ja tavoitepisteiden välistä etäisyyden neliötä (eli toista potenssia) minimoidaan. Oletuksena on, että kuvannettavan nousevan aortan osalla sen kaarevuus on likimain vakio. Tämän jälkeen pisteet sekä kaari piirretään XY-kaavioon.

Kun ”Load”-painiketta on painettu, ohjelma avaa *.ini-tiedoston sekä tiedostot "S4_polygon1.csv" ja "S4b_aarcent.csv". Jos nämä tiedostot löytyvät, alkaa prosessi muodostamaan graafista kuvaajaa koordinaatistoon (Kuva 9), joka suoritetaan omassa metodissa. Keskilinjan ja aortan rajaavat pisteet saadaan suoraan edellä mainituista tiedostoista.

`FitCircleUsingLeastSquares`-metodi pyrkii sovittamaan PNS-menetelmällä ympyränkaaren niihin koordinaattipisteisiin, jotka saadaan muuttujasta `centerlineCoords`, joka sisältää tiedostosta "S4b_aarcent.csv" luetun sisällön. Tämän tuloksena alla oleva metodi luo uudet, sovitetun ympyrän kaarella olevat vihreät pisteet. (Koodi 10)

```
private List<PointF> FitCircleUsingLeastSquares(List<PointF>
centerlineCoords)
{
    // Extract x and y coordinates
    var x = new List<float>();
    var y = new List<float>();
```

```
    foreach (var point in centerlineCoords)
    {
        x.Add(point.X);
        y.Add(point.Y);
    }

    // Arc fit to the x-y points using least squares method
    // Construct the matrices A and B
    var A = new List<List<float>>();
    var B = new List<float>();

    for (int i = 0; i < x.Count; i++)
    {
        A.Add(new List<float> { -2 * x[i], -2 * y[i], 1 });
        B.Add(x[i] * x[i] + y[i] * y[i]);
    }

    // Solve for X = [h k C]
    var X = SolveLeastSquares(A, B);

    // Compute the circle parameters
    var h = X[0] * (-1);
    var k = X[1] * (-1);
    var r = (float)Math.Sqrt(X[2] + h * h + k * k);

    // Set the class-level variables r,h and k
    this.r = r;
    this.h = h;
    this.k = k;

    // Print the circle parameters
    Console.WriteLine($"Center: ({h}, {k}), Radius: {r}");

    // Example: generate some dummy circle points
    List<PointF> circlePoints = new List<PointF>();
    for (double theta = 0; theta < 2 * Math.PI; theta += 0.1)
    {
        float xr = (float)(r * Math.Cos(theta) + h);
        float yr = (float)(r * Math.Sin(theta) + k);
        circlePoints.Add(new PointF(xr, yr));
    }

    return circlePoints;
}
```

Koodi 10. Metodi jolla suoritetaan ympyräkaaren sovittaminen koordinaattipisteisiin PNS-menetelmällä.

Metodi, jota tarvitaan tässä yhteydessä, on `divideArc`-metodi, joka pyrkii jakamaan ympyrän kaaren tarvittavan moneen osaan. Tämä jako määrittää myöhemmin piirrettävien tasojen välimatkat toisistaan. Käyttämällä `centerlineCoords`-muuttujaa, kaaren pituus ja välimatka lasketaan ja siten määritellään ja luodaan

tasot ja pinnat. Nämä tasot tallennetaan tiedostoon ”S4b_recplanes.csv” myöhemmin tulevissa moduuleissa käytettäväksi. (Liite 4)

`MovePoints`-metodi määrittää uudet pisteet niin, että ne sijaitsevat aortan sovitetulla kaarella. Tämä suoritetaan määrittämällä ensin ympyrän kaari PNS-menetelmällä. Kun tiedetään alkuperäisten, sovituksessa käytettyjen pisteiden paikka, voidaan uudet, ympyrän kaarella sijaitsevat pisteet luoda laskemalla sovituspisteiden etäisyys ympyrän kaaresta. Tämän jälkeen uudet ympyrän kaarella sijaitsevat pisteet luodaan yllä mainitun lähimmän etäisyyden ilmoittamaan kohtaan. (Koodi 11)

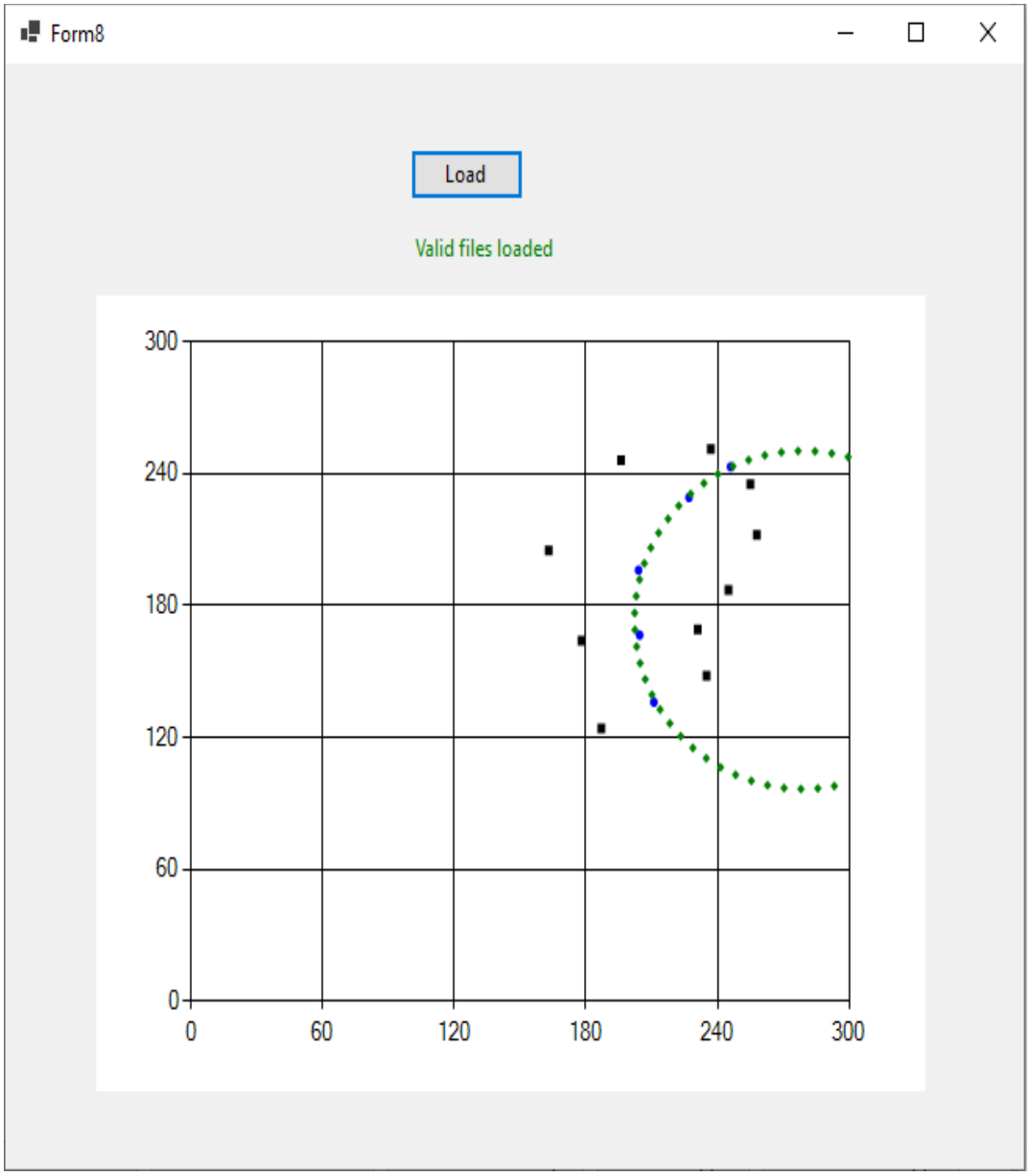
```
private List<PointF> MovePoints(List<PointF> centerlineCoords)
{
    // Extract x and y coordinates
    var x = new List<float>();
    var y = new List<float>();
    List<PointF> movedPoints = new List<PointF>();
    foreach (var point in centerlineCoords)
    {
        x.Add(point.X);
        y.Add(point.Y);
    }

    // Construct the matrices A and B
    var A = new List<List<float>>();
    var B = new List<float>();

    for (int i = 0; i < x.Count; i++)
    {
        A.Add(new List<float> { -2 * x[i], -2 * y[i], 1 });
        B.Add(x[i] * x[i] + y[i] * y[i]);
    }

    // Solve for X = [h k C]
    var X = SolveLeastSquares(A, B);
    // Compute the circle parameters
    var h = X[0] * (-1);
    var k = X[1] * (-1);
    var r = (float)Math.Sqrt(X[2] + h * h + k * k);
    foreach (var point in centerlineCoords)
    {
        var angle = Math.Atan2(point.Y - k, point.X - h);
        float newX = (float)(r * Math.Cos(angle) + h);
        float newY = (float)(r * Math.Sin(angle) + k);
        angles.Add(angle); // Store angle in the angles list
    }
    return movedPoints;
}
```

Koodi 11. MovePoints-metodi



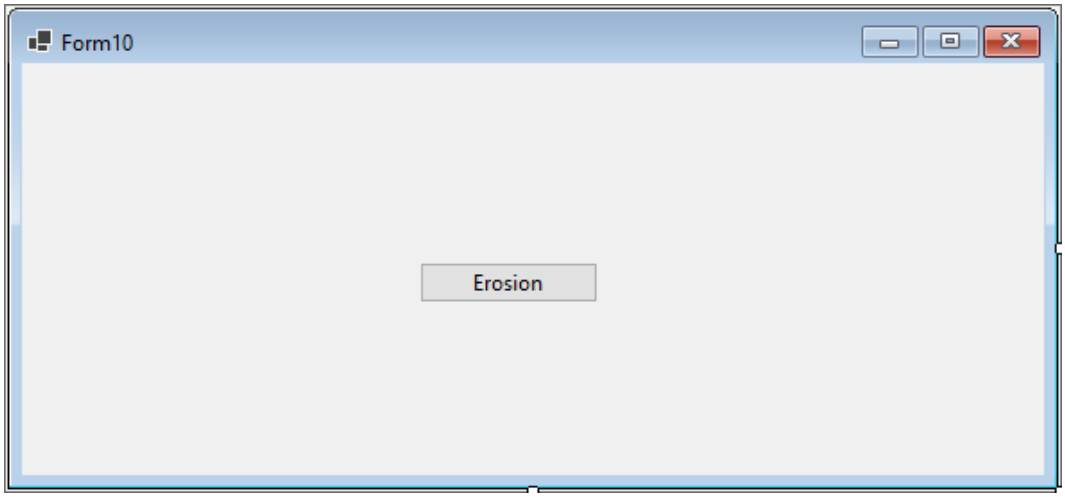
Kuva 9. Form8 Moduuli 8 Graafinen ulkoasu.

5.4 Segmentointigeometrian määritys

Moduuli 9, Form9: Form9:ssä suoritetaan intensiteettikynnystys. Tavoitteena on se, että kaikissa leikkeissä, joissa ovat vain aortan poikkileikkausalueet jäljellä, korkeamman intensiteetin alueet eli ne osat, joissa on verta (näkyvät vaaleampina MRI ja CT kuvissa) jätetään kuvaan niin, että kaikki alueet, joiden intensiteetti on alle kynnysarvon, jätetään mustaksi ja sen yläpuolella olevat valkeiksi eli suoritetaan ns. binäärikynnystys. Mikäli kuva on RGB-kuva, tehdään ennen kynnystystä

harmaasävymuunnos Gray Value = (color.R * 0.3 + color.G * 0.59 + color.B * 0.11). Eri värikanavien painokertoimet perustuvat 1953 NTSC:n säätämään standardiin, jolla värikuva muutetaan harmaasävykuvaksi ja joka on edelleen käytössä esim. moderneissa televisioissa (ITU-R BT.601–7, 2011). Muokatut leikkeet siirretään omaan ulostulokansioon, nämä vaiheet esitellään liitteessä 5. Mikäli halutaan käyttää intensiteettikynnystyksessä menetelmää, jossa annetaan manuaalisesti kynnystysraja, tällöin arvo syötetään *.ini-tiedostossa InitialBrightTreshold-parametrina, mutta tässä tapauksessa on kovakoodattu threshold/kynnysarvo. (Liite 5)

Moduuli 10, Form10: Form10 (Kuva 10) suorittaa eroosion, jolla poistetaan erilliset pienet, aortta-alueen ulkopuoliset valkeat "pikseliroskat". Rintakehän alueella on paljon myös pieniä verisuonia, jotka näkyvät MRI-kuvissa valkoisena, jolloin kuvaan kertyy ylimääräisiä valkoisia alueita. Eroosiomenetelmällä on tarkoitus poistaa kaikki ylimääräiset pienet verisuonet, eli ohjelma käy kuvan läpi eroosiorakenne-elementtien avulla.



Kuva 10. Form10 Moduuli 10 Graafinen ulkoasu.

Elementillä käydään läpi kaikki kuvan 512x512 pikseliä. Eroosiorakenne-elementti määrittää ”pikselinaapuruston”, jonka alueella eroosiota tarkastellaan. Uudessa eroosiolla käsitellyssä kuvassa keskimmäinen pikseli saa pienimmän arvon, eli binäärikynnystetyssä mustavalkokuvassa arvon 0 mikäli yksikin pikseli on musta (Chen & Haralick, n.d.). Mikäli koko alue on valkoinen, keskipikseli saa maksimi-

intensitettiarvon. Tämän seurauksena pienet, eroosioelementin kokoluokkaa olevat valkoiset alueet poistuvat mutta samalla myös pääkohteen eli aortan poikkileikkauspinta-ala pienenee keinotekoisesti. Tämä palautetaan seuraavassa Form11:ssa.

Eroosion rakenne-elementti haetaan potilaan *.ini-tiedostosta (`erosionStructElem`), jota prosessi käyttää. Eroosio suoritetaan jokaiselle syötekansion kuvalle. Ensin määritellään neliöllinen rakenne-elementti matriisiksi muuttujaan `se`. Seuraavaksi prosessi lukee kuvan, jonka jälkeen suoritetaan itse eroosio-operaatio kaikille kuville. Lopulta kuvat tallennetaan taas uuteen kansioon. (Koodi 12)

```
// Read the structuring element size (in pixels = millimeters) for
erosion from CorFlux.ini
int erosionStructElem = (int)numParams["CFAParam_erosionstructelem"];
// Start erosion process for each file
foreach (string filePath in Directory.GetFiles(inputFolderPath,
"*.tif"))
{
    // Step 1: Specify the structuring element for erosion
    Emgu.CV.Matrix<float> se = new
Emgu.CV.Matrix<float>(erosionStructElem, erosionStructElem);
    se.SetValue(1);

    // Step 2: Read the image
    Emgu.CV.Mat originalImage = Emgu.CV.CvInvoke.Imread(filePath);

    // Step 3: Erosion
    Emgu.CV.Mat erodedImage = new Emgu.CV.Mat();
    Emgu.CV.CvInvoke.Erode(originalImage, erodedImage, se, new
System.Drawing.Point(-1, -1), 1, Emgu.CV.CvEnum.BorderType.Default, new
Emgu.CV.Structure.MCvScalar());

    // Step 4: Save the processed image
    string filename = Path.GetFileNameWithoutExtension(filePath);
    string outputFilePath = Path.Combine(outputFolderPath, filename +
"_processed.tif");
    erodedImage.Save(outputFilePath);
}
```

Koodi 12. Tämä koodiosio suorittaan eroosion syötekansion TIFF kuville.

Moduuli 11, Form11: Eroosion jälkeen seuraavassa moduulissa tehdään dilaatio, joka on käänteinen eroosiolle. `dilationStructElem` eli dilaatioelementin pinta-ala on tyypillisesti sama kuin `erosionStructElem`. Dilaatioprosessi käy läpi kuva-alu-

een ja keskipikselin intensiteettiarvoksi annetaan uudessa kuvassa maksimi-intensiteetin arvo. Kun elementtien koko valitaan oikein, eroosio poistaa pienet ja tarpeettomat valkoiset alueet, jotka dilaatiossa ne eivät enää palaudu, vaan sitä vastoin laajemmat alueet (kuten aortan poikkileikkaus) palaavat kooltaan alkuperäiseen pinta-alaan. (Koodi 13)

```
int dilationStructElem = (int)numParams["CFAParam_dilationstructelem"];
// Iterate through each TIFF file
string[] tiffFiles = Directory.GetFiles(inputFolderPath, "*.tif");
foreach (string tiffFile in tiffFiles)
{
    // Read the TIFF image
    Image<Bgr, byte> originalImage = new Image<Bgr, byte>(tiffFile);
    // Perform dilation
    Image<Bgr, byte> dilatedImage = Dilation(originalImage,
dilationStructElem);
    // Save the processed image to the output folder
    string filename = Path.GetFileNameWithoutExtension(tiffFile);
    string outputTiffPath = Path.Combine(outputFolderPath, filename +
"_processed.tif");
    dilatedImage.Save(outputTiffPath);
}
```

Koodi 13. Tämä Koodiosio tekee dilaation jokaiselle TIFF-kuvalle ja tallentaa tulokset erilliseen hakemistoon jatkokäsittelyä varten.

Dilaatioprosessi haetaan omasta metodistaan, joka näkyy alla olevassa koodinosiossa. Dilaatio toimii samalla tavalla kuin eroosio, mutta sen vaikutus käsiteltävään kuvaan on päinvastainen. (Koodi 14)

```
private Image<Bgr, byte> Dilation(Image<Bgr, byte> image, int
structElemSize)
{
    Mat element =
CvInvoke.GetStructuringElement(ElementShape.Rectangle, new
Size(structElemSize, structElemSize), new Point(-1, -1));
    Image<Bgr, byte> dilatedImage = new Image<Bgr, byte>(image.Size);
    CvInvoke.Dilate(image, dilatedImage, element, new Point(-1, -1), 1,
BorderType.Default, new MCvScalar());
    return dilatedImage;
}
```

Koodi 14. Dilaatio metodi suorittaan dilaation syötekansion TIFF kuville.

Moduuli 12, Form12: Eroosion ja dilaation jälkeen on mahdollista, että kuva-alueella on vielä ylimääräisiä jäännösalueita. Tämä moduuli valitsee kaikista kuvista

(leikkeistä) kuvan suurimman valkoisen pinta-alan ja muuttaa kaiken muun mustaksi. Suurin valkoinen pinta-ala on tyypillisesti aortan alue.

Ensin haetaan kaikki kuvat syötekansiosta, jotka sitten luetaan harmaasävyisinä Emgu:n `Mat`-toiminnon avulla. Seuraavassa askeleessa käytetään `CvInvoke.Threshold`-metodia, jonka tarkoitus on muuttaa harmaasävyisetkuvat binäärikuviksi. Tämän jälkeen prosessi etsii binäärikuvista kaikki erilliset alueet eli reunaviivat `contours`. Seuraavaksi luodaan tyhjä maski, johon reunaviivat piirretään sen mukaan, mikä segmentointialiaksen instanssinumero kuvalle saadaan. Koodiosio valitsee vain suurimman kontuurin tai kaksi suurinta kontuuria sen mukaan, onko kyseessä viimeisen segementointialiaksen instanssinumero. Syynä tähän menettelyyn on se, että MRI-kuvauksessa käytetään ns. vinosagittaalisuuntausta, jossa aortan alueet eri leikkeissä ovat joko yksi- tai kaksiosaisia. Vain valitut maskatut kohdat kopioidaan uuteen kuvaan ja nämä kuvat tallennetaan omaan ulostulokansioon. (Liite 6)

Moduuli 13, Form13: Tässä vaiheessa jokaisessa leikkeessä on vain 2 valkoista aluetta. "Edges" prosessi ottaa yhtenäisen valkoisen alueen pois jättäen reunat mikä on myös aortan seinämä. Reunat erotetaan kaksivaiheisesti 1) ensin Sobelin menetelmällä (Sobel Edge Detector, 2003), joka etsii intensiteettimuutokset ja sen jälkeen 2) Otsun menetelmällä (Chen & Haralick, n.d.), joka muodostaa reunaviivan. Sobel-algoritmi käy 3x3 -pikselialueen avulla (kts. eroosio) kuvien intensiteettimuutokset läpi sekä X- että Y-suunnissa. Kun intensiteettimuutos on suurempi kuin luokitteluarvo 127, uuteen kuvaan talletetaan ko. pikseli. Otsun menetelmässä kuva-alue luokitellaan pikseleiden intensiteetin perusteella taustaksi C0 ja etualaksi C1 eli tässä tapauksessa binäärikynnystyksen 0/255 intensiteetin luokitteluarvo on keskiarvo eli (0 + 255)/2 = 127. (Koodi 15)

```
// Start the loop procedure for extracting the edges from the *.tif
images from folder St5b
string[] tifFiles = Directory.GetFiles(inputFolderPath, "*.tif");
foreach (string tifFile in tifFiles)
{
    // Load the image
    Mat image = CvInvoke.Imread(tifFile, ImreadModes.Grayscale);
```

```
    // Perform edge detection using Sobel (to mimic MATLAB's 'edge'
function)
    Mat edges = new Mat();
    CvInvoke.Sobel(image, edges, DepthType.Cv16S, 1, 1);

    // Convert to 8-bit image
    Mat absEdges = new Mat();
    CvInvoke.ConvertScaleAbs(edges, absEdges, 1, 0);

    // Apply threshold to get binary edge image
    Mat edgeImage = new Mat();
    CvInvoke.Threshold(absEdges, edgeImage, 0, 255,
ThresholdType.Binary | ThresholdType.Otsu);

    // Save the result to the output folder
    string fileName = Path.GetFileName(tifFile);
    string outputFilePath = Path.Combine(outputFolderPath, fileName);
    edgeImage.Save(outputFilePath);
}
```

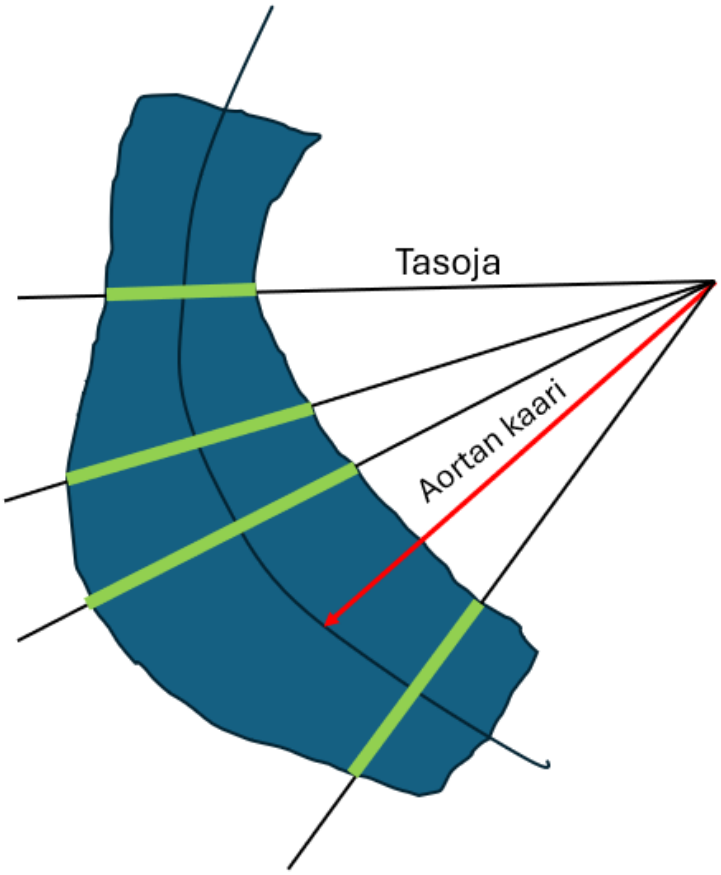
Koodi 15. hakee kaikki syötekansiossa olevat TIFF-kuvat, suorittaa niille Sobel-pohjaisen reunanilmaisun Emgu CV -kirjaston avulla ja binarisoi tuloksen Otsun automaattisella kynnysarvotuksella.

Kuva 11. Kuvasarja esittää aortan poikkileikkauskuvan segmentoinnin etenemisen vaihe vaiheelta, vasemmalta oikealle. (1) Raaka MRI-kuva. (2) Kuvasta on rajattu pois kaikki alueet, joissa aortan nouseva osa ei esiinny. (3) Harmaasävyksi muunnettu binäärikuva, jossa aortta ja muut verisuonet näkyvät valkoisina mustaa taustaa vasten. (4) Eroosiolla käsitelty kuva, jossa ylimääräiset verisuonet on poistettu. (5) Dilaation tulos, jonka avulla osa eroosiossa hävinneestä rakenteesta palautetaan. (6) Lopullinen segmentointi, jossa aortan seinämät on korostettu valkoisena ja muu kuva-alue on musta.

5.5 3D pinta mallin määritys

Tavoitteena tulevissa moduuleissa oli muodostaa aortan lopullinen 3D-muoto. Päädyimme valittuun menettelyyn siksi, että aortan kaarella on merkittävä vaikutus sen repeämätodennäköisyyteen. Tähän vaiheeseen ei kuitenkaan projektin loppumisen takia valitettavasti päästy.

Tämä vaihe olisi saavutettu niin, että kootaan ensin kaikki aortan *.tif-poikkileikkeet päällekkäin (kuvatasossa). Tämä jälkeen luodaan aiemmin määritetyn aortan kaaren alkupisteestä tasoja. Päällekkäisten leikkeiden ja tasojen poikkileikkauspisteet muodostavat lopullisen aortan ympyrämäiset poikkileikkaukset, joiden suuntaa ja suuruutta aortan kaari ohjaa (Kuva 12).



Kuva 11. Aortan alueen 3D-esitys. Aortan tasot (4 musta viivaa), tasojen poikkileikkauspisteet (vihreät viivat) ja aortan kaari (musta viiva osoitettu punaisella nuolella).

6 JOHTOPÄÄTOKSET JA ARVIOINTI

CorFlux-projektin tavoite oli kehittää analyysityökalu aortan aneurysman repeämisriskin kliiniseen arviointiin. Tämän opinnäytetyön osalta päämääränä oli luoda ohjelma, joka muokkaa MRI:n tuottamista raakakuvista 3D-kuvan jatkoanalyysia varten. Ikävä kyllä aika loppui, ennen kuin 3D kuvantamisvaiheen ohjelmointi oli valmis. Viimeisin valmiiksi saatu vaihe oli Moduuli 13, jossa segmentointiin aortan seinämät poikkileikekuvista esittämällä ne reunaviivoina. Valmistuneiden 13 moduulin kohdalla saavutettiin hyvä toimintavarmuus; vain yhtä moduulia (moduuli 14) ei ehditty ohjelmoimaan.

Projektin hallintatyökaluja käyttämällä ja huolellisemmalla suunnittelulla tämä projektityöosuus olisi todennäköisesti saatu valmiiksi. Päätös edetä ilman projektinhallintatyökaluja todennäköisesti vaikutti kokonaisuuteen. Epäselvä suunnittelu ja rakenteiden puute myös todennäköisesti vaikeuttivat työn etenemistä. Toisaalta projektiryhmän kokemus tällaisista ohjelmointiprojekteista olivat valitettavasti puutteelliset, mikä olisi mitätöinyt hallintatyökalujen käytöstä saadut hyödyt.

Vaikka tässä työssä kuvattu testiversio täyttää perustoiminnalliset vaatimukset, on se silti erittäin kömpelö ja kankea. Yksi keskeisistä tekijöistä ohjelman jäykkyydessä on sen vaiheittainen suoritusmalli. Käyttäjän on edettävä prosessin eri vaiheissa manuaalisesti yksi vaihe kerrallaan, mikä hidastaa työnkulun etenemistä ja tekee testauksesta työlästä. Tämä rakenne oli alun perin tarkoituksellinen ja testausvaiheen kannalta tarkoituksenmukainen, koska sen avulla voitiin tarkkailla datan oikeellisuutta ja eristää mahdollisia virheitä vaihekohtaisesti.

Siitä huolimatta, että C#-ohjelmointikieli on todennäköisesti se, joka soveltuu parhaiten täyttämään tämä projektin tarpeet ja vaatimukset, Windows Forms mallipohja ei välttämättä ole paras kehitysalusta tällaisen ohjelmiston rakentamiseen. Jos tätä kuvantamisohjelmaa kehitetään eteenpäin, .NET MAUI formaatti (tai jokin

samankaltainen) on todennäköisesti parempi vaihtoehto tämän ohjelma ohjelmointiin ja kehittämiseen, joka mahdollistaa ketterämmän toiminnan eri ohjelmaosioiden ja alustojen välillä.

Suunnitelman mukaisesti tämän testiversion olisi tarkoitus olla osa erillistä laskenta ohjelmaa, joka suorittaa analyysin aortan 3D-datasta. Myöhemmässä hypoteettisessa, kaupallisessa vaiheessa prototyyppivaiheen tiedostosiirto olisi muutettu isomman tietorakenteen hallinnaksi ilman käyttäjältä vaadittavaa toistuvaa toimintojen ja tiedon syöttämistä ja datan toistuvia välitallennuksia.

LÄHTEET

Building a better future for patients with aortic aneurysm. (2024). Noudettu 28.5.2025 osoitteesta https://corfluxanalysis.com/corflux/

Aortan laajentuma (aneurysma). (2025). Terveyskirjasto. Noudettu 31.5.2025 osoitteesta https://www.terveyskirjasto.fi/dlk00008

Airaksinen, J., Aalto-Setälä, K., Hartikainen, J., Junttila, J., Laine, M., Lommi, J., Raatikainen, P. & Saraste, A. (toim.). (n.d.). Kardiologia. Noudettu 31.5.2025.

Blanco Sequeiros, R., Koskinen, S., Aro, H., Lundbom, N., Vanninen, R. & Tervonen, O. (toim.). (2017). Kliininen radiologia (1. painos). Noudettu 31.5.2025.

W3Schools. (n.d.). C# Tutorial (C Sharp). Noudettu 31.5.2025 osoitteesta https://www.w3schools.com/cs/index.php

Liberty, J. (2002). Programming C# (2. painos). Noudettu 31.5.2025, sivu 9.

Moghadampour, G. (2012). C#-ohjelmointi. Noudettu 1.6.2025, sivut 12–13, 15–16.

CorFlux – UEFConnect. (n.d.). Noudettu 16.10.2025 osoitteesta https://uefconnect.uef.fi/corflux/

Veny Vita Ponggawa, Utari B. Santoso, Gita A. Talib, March A. Lamia, Ariel R.A Manuputty, Muhammad F. Yusuf. Comparative analysis of C, C++, C# and Java programming languages (Volume 5, No. 12). (2020). GSJ Global Scientific Journals. Ogala. Noudettu 16.10.2025 osoitteesta https://jurnal-syntaxadmiration.com/index.php/jurnal/article/view/1926/1961

Ogala, Justin Onyarin & Ojie, Deborah V. Comparative analysis of C, C++, C# and Java Programing Languages (Volume 8, Issue 5), (2024). GSJ Global scientific journals.

Karjalainen, P. (2025). Henkilökohtainen tiedonanto

About DICOM Overview. (n.d.). About DICOM Overview. Noudettu 3.11.2025 osoitteesta https://www.dicomstandard.org/about

What is IrfanView? (1996–2025). Noudettu 4.11.2025 osoitteesta https://www.irfanview.com/main_what_is_engl.htm

Bibb, R. (2006). Medical Modeling: The Application of Advanced Design and Development Techniques in Medicine. Woodhead Publishing. Noudettu 4.11.2025, sivut 14–15, 24–25.

RECOMMENDATION ITU-R BT.601-7 – Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios. (2011). Noudettu 4.11.2025 osoitteesta https://www.itu.int/dms_pubrec/itu-r/rec/bt/r-rec-bt.601-7-201103-i!!pdf-e.pdf

Kurki, M. (2025). Henkilökohtainen tiedonanto.

Sobel Edge Detector. (2003). Noudettu 12.11.2025 osoitteesta https://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm

Chen, — & Haralick, R. (n.d.). Recursive Erosion Dilation Opening and Closing Transforms. Noudettu 16.11.2025 osoitteesta https://www.haralick.org/journals/recursive_morphology.pdf

Synopsys. (2025). What Is Medical Image Segmentation and How Does It Work? Noudettu 16.11.2025 osoitteesta https://www.synopsys.com/glossary/what-is-medical-image-segmentation.html

Finto. (n.d.). YSO - Yleinen suomalainen ontologia. Noudettu 27.6.2023 osoitteesta https://finto.fi/yso/fi/

LIITTEET

LIITE 1 Koodiosio, jossa segmentointialue määritellään.

```
// Open IrfanView to display the second image
string currentImagePath2 = Path.Combine(inputFolderPath2,
Path.GetFileName(dicomFiles2[i]));
dicomInfoList2.Add(currentImagePath2);
System.Diagnostics.Process.Start("C:\\Program
Files\\IrfanView\\i_view64.exe", $"\"{currentImagePath2}\"");

// Prompt user to select the second reference DICOM file
OpenFileDialog openFileDialog1 = new OpenFileDialog();
openFileDialog1.Filter = "DICOM Files|*.tif";
openFileDialog1.Title = "Select Second Reference DICOM File";

if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
    refFilename2 = Path.GetFileName(openFileDialog1.FileName);
    refPath2 = Path.GetDirectoryName(openFileDialog1.FileName);

    // Load the reference DICOM image
    string refImagePath2 = Path.Combine(refPath2, refFilename2);
    refInfo2 = refImagePath2; // You can replace this with actual DICOM
information retrieval in C#
    PictureBox pictureBox = new PictureBox();
    pictureBox.SizeMode = PictureBoxSizeMode.AutoSize;
    pictureBox.Image = System.Drawing.Image.FromFile(refImagePath2); //
Load the DICOM image into the PictureBox
    PictureBox bgPictureBox = new PictureBox();
    File.WriteAllText("S4_refimg1.txt", refFilename2);

    // Prompt user to draw a polygon around the desired area for the
first selection
    MessageBox.Show("Draw a polygon around the area of interest for the
second selection");
    // Use a suitable graphics library or method to draw the polygon

    // Create a new form to host the drawing
    Form form = new Form
    {
        Size = new Size(512, 512),
        BackgroundImage = pictureBox.Image
    };
    bgPictureBox.Size = form.Size;
    bgPictureBox.Image = new Bitmap(form.Width, form.Height);

    // Create a list to store the polygon coordinates
    polygonCoords2 = new List<Point>();

    // Add a MouseClick event handler to the form
    form.MouseClick += (s2, e2) =>
    {
        // Add the clicked point to the polygonCoords1 list
        polygonCoords2.Add(e2.Location);
        // Redraw the form
        form.Invalidate();
    };
```

```
    form.Paint += (s2, e2) =>
    {
        // Create a graphics object from the form's handle
        Graphics g = e2.Graphics;
        // Set pen properties (color, width, etc.)
        Pen pen = new Pen(Color.Red, 2);

        // Draw the polygon using the polygonCoords1 array
        if (polygonCoords2.Count > 1)
        {
            g.DrawPolygon(pen, polygonCoords2.ToArray());
        }
        // Clean up resources
        pen.Dispose();
    };

    form.ShowDialog();

    File.WriteAllLines("S4_polygon2 .csv", polygonCoords2.Select(p =>
$"{p.X},{p.Y}"));

}
else
{
    Console.WriteLine("No reference file selected. Exiting.");
    return;
}
```

LIITE 2 Koodiosio, jossa viivauksen ulkopuolella oleva osa kuvasta poistetaan.

```
// Load the DICOM files corresponding to refFilename1 and refFilename2
var dicomFile1 = DicomFile.Open(Path.Combine(inputFolderPath,
Path.ChangeExtension(refFilename1, ".dcm")));
var dicomFile2 = DicomFile.Open(Path.Combine(inputFolderPath,
Path.ChangeExtension(refFilename2, ".dcm")));
// Retrieve the DICOM datasets
var dataset1 = dicomFile1.Dataset;
var dataset2 = dicomFile2.Dataset;
// Retrieve the instance numbers
instanceNumber1 = dataset1.GetValue<int>(DicomTag.InstanceNumber, 0);
instanceNumber2 = dataset2.GetValue<int>(DicomTag.InstanceNumber, 0);
// Store the instance numbers in a matrix
int[] instanceNumbers = { instanceNumber1, instanceNumber2 };
// Save the instance numbers to 'S4_aliasnumbers.csv' in the current
working directory
File.WriteAllLines("S4_aliasnumbers.csv", instanceNumbers.Select(x =>
x.ToString()));
// Sort the files by name
Array.Sort(dicomFiles);
// Open the first DICOM file
var firstDicomFile = DicomFile.Open(dicomFiles[0]);
// Retrieve the DICOM dataset
var firstDataset = firstDicomFile.Dataset;
// Retrieve the instance number
var firstInstanceNumber =
firstDataset.GetValue<int>(DicomTag.InstanceNumber, 0);
```

```
// Get a list of all DICOM files in the input folder
string[] fileList = Directory.GetFiles(inputFolderPath, "*.tif");
// Load the DICOM metadata of the second reference image outside the
loop
var info2 = dataset2;
// Load the DICOM information from the reference image excluding
'BodyPartExamined'
var refDicomInfo = dataset1;
refDicomInfo.Remove(DicomTag.BodyPartExamined);
MessageBox.Show(instanceNumber2.ToString());

for (int i = firstInstanceNumber; i < fileList.Length +
firstInstanceNumber; i++)
{
    // Load TIFF image
    using (Bitmap img = new Bitmap(fileList[i - firstInstanceNumber]))
    {
        Point[] polygon;
        if (i <= instanceNumber2)
        {
            polygon = polygonCoords1.ToArray();
        }
        else
        {
            polygon = polygonCoords2.ToArray();
        }
        // Apply the drawn polygon as a mask to the color image
        using (var mask = new Bitmap(img.Width, img.Height))
        {
            using (Graphics g = Graphics.FromImage(mask))
            {
                g.FillPolygon(Brushes.White, polygon);
            }

            using (var maskedImg = new Bitmap(img.Width, img.Height))
            {
                for (int x = 0; x < img.Width; x++)
                {
                    for (int y = 0; y < img.Height; y++)
                    {
                        if (mask.GetPixel(x, y).R > 0)
                        {
                            maskedImg.SetPixel(x, y, img.GetPixel(x,
y));
                        }
                    }
                }

                string filename =
Path.GetFileNameWithoutExtension(fileList[i - firstInstanceNumber]);
                string outputFilename = Path.Combine(outputFolderPath,
filename + ".tif");
                maskedImg.Save(outputFilename);
            }
        }
    }
}
```

LIITE 3 Keskilinja säteen laskeminen ja määritys.

```
OpenFileDialog openFileDialog1 = new OpenFileDialog();
openFileDialog1.Filter = "DICOM Files|*.tif";
openFileDialog1.Title = "Select First Reference DICOM File";

if (openFileDialog1.ShowDialog() == DialogResult.OK)
{

    string inputFolderPath = SrcTextBox.Text;
    string polygonFilePath = Path.Combine(inputFolderPath,
"S4_polygon1.csv");
    string refImgFilePath =
Path.GetFileName(openFileDialog1.FileName);
    string refDirName =
Path.GetDirectoryName(openFileDialog1.FileName);
    string refImagePath = Path.Combine(refDirName, refImgFilePath);
    // Load polygon coordinates
    var polygonCoords = File.ReadAllLines(polygonFilePath)
        .Select(line => line.Split(',').Select(double.Parse).ToArray())
        .ToArray();
    // Load the image
    var image = AForge.Imaging.Image.FromFile(refImagePath);
    // Convert to grayscale
    var grayImage =
Grayscale.CommonAlgorithms.BT709.Apply(image).Clone() as Bitmap;
    // Create a BlobCounter and configure it
    var blobCounter = new BlobCounter();
    blobCounter.ProcessImage(grayImage);
    var blobs = blobCounter.GetObjectsInformation();
    // Find the largest blob
    var largestBlob = blobs.OrderByDescending(b => b.Rectangle.Width *
b.Rectangle.Height).FirstOrDefault();

    if (largestBlob != null)
    {
        // Get the centerline points
        var edgePoints = blobCounter.GetBlobsEdgePoints(largestBlob);

        // Calculate centerline points
        List<System.Drawing.PointF> centerlineCoords = new
List<System.Drawing.PointF>();
        int numPoints = polygonCoords.Length;

        // Ensure there are at least two points to form pairs
        if (numPoints < 2)
        {
            // Handle the case where there are not enough points
            MessageBox.Show("There are less than 2 points in the
Image.");
            return;
        }
        // Calculate midpoints between pairs of points
        for (int i = 0; i < numPoints / 2; i++)
        {
            int index1 = (numPoints / 2 - i + numPoints - 1) %
numPoints; // Handling wrap-around
            int index2 = (numPoints / 2 + i) % numPoints; // Handling
wrap-around
```

```
            System.Drawing.PointF point1 = new
System.Drawing.PointF((float)polygonCoords[index1][0],
(float)polygonCoords[index1][1]);
            System.Drawing.PointF point2 = new
System.Drawing.PointF((float)polygonCoords[index2][0],
(float)polygonCoords[index2][1]);
            MessageBox.Show($"Point 1: ({point1.X}, {point1.Y})\nPoint
2: ({point2.X}, {point2.Y})");
            // Calculate the midpoint between the current pair of
points
            float midX = (point1.X + point2.X) / 2.0f;
            float midY = (point1.Y + point2.Y) / 2.0f;
            centerlineCoords.Add(new System.Drawing.PointF(midX,
midY));
        }

        // Save centerline coordinates to file
        SaveCenterlineCoordinates(inputFolderPath, centerlineCoords);

        // Convert to grayscale
        // Ensure that the image is not in indexed pixel format
        if (grayImage.PixelFormat ==
System.Drawing.Imaging.PixelFormat.Format8bppIndexed)
        {
            // Convert the image to a format that supports direct pixel
manipulation
            grayImage = AForge.Imaging.Image.Clone(grayImage,
System.Drawing.Imaging.PixelFormat.Format24bppRgb);
        }
        // Draw the centerline on the image
        foreach (var point in centerlineCoords)
        {
            grayImage.SetPixel((int)point.X, (int)point.Y,
System.Drawing.Color.Green);
        }
        // Save the image with the centerline drawn
        string outputImagePath = Path.Combine(inputFolderPath,
"CenterlineImage.tif");
        grayImage.Save(outputImagePath);
        Console.WriteLine("Centerline image saved to: " +
outputImagePath);
    }
    else
    {
        Console.WriteLine("No blob found in the image.");
    }
}
```

LIITE 4 divideArc-metodi

```
void divideArc()
{
    int numPoints = centerlineCoords.Count;

    // Extract x, y, and z coordinates from corrMiddlePoints
    double[,] correctedCoords = new double[numPoints, 2];
    for (int i = 0; i < numPoints; i++)
    {
        correctedCoords[i, 0] = centerlineCoords[i].X;
        correctedCoords[i, 1] = centerlineCoords[i].Y;
    }

    double TotalAngle = 0;
    for (int i = 0; i < angles.Count - 1; i++)
    {
        TotalAngle += Math.Abs(Math.Abs(angles[i + 1]) -
Math.Abs(angles[i]));
    }
    double TotalArcLength = TotalAngle * r;
    double desiredDist = numParams["CFAParam_yplanespacing"];
    int numPlanes = (int)Math.Round(TotalArcLength / desiredDist);

    // Calculate the start angle
    // Initialize a list to store the planes
    List<double[,]> planes = new List<double[,]>();
    // Set the height of the rectangular planes
    double plane_height = 100;
    // Compute the angles at which to place planes
    double startangle = Math.Atan2(centerlineCoords[0].Y - k,
centerlineCoords[0].X - h);
    double[] planeAngles = new double[numPlanes];
    for (int i = 0; i < numPlanes; i++)
    {
        planeAngles[i] = startangle - (i * desiredDist / r);
    }

    // Compute the coordinates of points on the arc for each angle
    double[,] arcPoints = new double[2, numPlanes];
    for (int i = 0; i < numPlanes; i++)
    {
        double angle = planeAngles[i];
        double x_point = h + r * Math.Cos(angle);
        double y_point = k + r * Math.Sin(angle);
        arcPoints[0, i] = x_point;
        arcPoints[1, i] = y_point;
    }

    // Create a StringBuilder to construct the message
    StringBuilder message = new StringBuilder();

    // Append arcPoints to the message
    message.AppendLine("Arc Points:");
    for (int i = 0; i < numPlanes; i++)
    {
        message.AppendLine($"Arc Point {i + 1}: ({arcPoints[0, i]},
{arcPoints[1, i]})");
    }
```

```
    // Display the arcPoints message
    MessageBox.Show(message.ToString(), "Arc Points Debug
Information");

    // Loop to add planes
    for (int j = 0; j < numPlanes; j++)
    {
        double x_point = arcPoints[0, j];
        double y_point = arcPoints[1, j];
        // Create rectangular plane at z=100
        double[] rectangle_x = { h, x_point, x_point, h };
        double[] rectangle_y = { k, y_point, y_point, k };
        double[] rectangle_z = { 0, 0, plane_height, plane_height };

        // Store the rectangular plane in the list
        double[,] plane = new double[3, 4];
        for (int i = 0; i < 4; i++)
        {
            plane[0, i] = rectangle_x[i];
            plane[1, i] = rectangle_y[i];
            plane[2, i] = rectangle_z[i];
        }
        planes.Add(plane);

        // Append individual plane coordinates to the message
        message.AppendLine($"Plane {j + 1} coordinates: ({x_point},
{y_point})");
        // Append added plane to the message
        message.AppendLine($"Added plane {j + 1}:");
        for (int i = 0; i < 3; i++)
        {
            for (int k = 0; k < 4; k++)
            {
                message.Append($"{plane[i, k]}, ");
            }
            message.AppendLine("");
        }
    }
    // Display the message box
    MessageBox.Show(message.ToString(), "Debug Information");

    // Save planes to a CSV file
    using (StreamWriter writer = new StreamWriter("S4b_recplanes.csv"))
    {
        foreach (var plane in planes)
        {
            for (int i = 0; i < 3; i++)
            {
                for (int j = 0; j < 4; j++)
                {
                    writer.Write(plane[i, j]);
                    writer.Write(";");
                }
                writer.WriteLine();
            }
        }
    }
}
```

LIITE 5 Koodiosio, joka suorittaa intensiteettikynnytyksen.

```
// Define threshold value
double threshold = 30.0000;
// Get a list of TIFF files in the input folder
string[] fileList = Directory.GetFiles(inputFolderPath, "*.tif");
// Loop through each file and apply intensity thresholding
foreach (string filePath in fileList)
{
    // Load the image
    Bitmap originalImage = new Bitmap(filePath);
    // Calculate average intensity
    double totalIntensity = 0;
    for (int y = 0; y < originalImage.Height; y++)
    {
        for (int x = 0; x < originalImage.Width; x++)
        {
            Color pixelColor = originalImage.GetPixel(x, y);
            int intensity = pixelColor.R; // Assuming grayscale image,
use R channel
            totalIntensity += intensity;
        }
    }
    double averageIntensity = totalIntensity / (originalImage.Width *
originalImage.Height);
    // Display average intensity
    message.AppendLine($"Average Intensity of
{Path.GetFileName(filePath)}: {averageIntensity}");

    Bitmap thresholdedImage = new Bitmap(originalImage.Width,
originalImage.Height);
    // Inside the loop where you load the image
    PixelFormat format = originalImage.PixelFormat;
    if (format != PixelFormat.Format8bppIndexed)
    {
        // Convert the image to grayscale
        originalImage = MakeGrayscale3(originalImage);
    }

    for (int y = 0; y < originalImage.Height; y++)
    {
        for (int x = 0; x < originalImage.Width; x++)
        {
            Color pixelColor = originalImage.GetPixel(x, y);
            int intensity = pixelColor.R; // Assuming grayscale image,
use R channel
            // Apply intensity thresholding
            if (intensity >= threshold)
            {
                thresholdedImage.SetPixel(x, y, Color.White);
            }
            else
            {
                thresholdedImage.SetPixel(x, y, Color.Black);
            }
        }
    }

    // Create the output file path
    string filename = Path.GetFileNameWithoutExtension(filePath);
```

```
    string outputFilePath = Path.Combine(outputFolderPath, filename +
"_thresh.tif");

    // Save the thresholded image
    thresholdedImage.Save(outputFilePath, ImageFormat.Tiff);

    // Console.WriteLine("Processed image: " + filePath);
}

// Display the average intensity for all images
MessageBox.Show(message.ToString(), "Average Intensity");

listBoxTreshold1.Items.Add(message);

// Function to convert an image to grayscale
static Bitmap MakeGrayscale3(Bitmap original)
{
    // Create a blank grayscale image of the same size
    Bitmap newBitmap = new Bitmap(original.Width, original.Height);

    // Convert each pixel to grayscale
    for (int y = 0; y < original.Height; y++)
    {
        for (int x = 0; x < original.Width; x++)
        {
            Color color = original.GetPixel(x, y);
            int grayValue = (int)(color.R * 0.3 + color.G * 0.59 +
color.B * 0.11);
            Color newColor = Color.FromArgb(grayValue, grayValue,
grayValue);
            newBitmap.SetPixel(x, y, newColor);
        }
    }

    return newBitmap;
}
```

LIITE 6 Foreach silmukka, jonka sisällä suoriutuu alueiden valinta (Area select) jokaiselle syötekansiossa olevalle kuvalle.

```
// Loop through each .tif file in the St5 folder
foreach (string filePath in Directory.GetFiles(inputFolderPath,
"*.tif"))
{
    // Load the image using EmguCV
    Mat image = CvInvoke.Imread(filePath, ImreadModes.Grayscale);

    // Apply threshold to extract white areas
    Mat thresholdedImage = new Mat();
    CvInvoke.Threshold(image, thresholdedImage, 200, 255,
ThresholdType.Binary);

    // Find contours
    VectorOfVectorOfPoint contours = new VectorOfVectorOfPoint();
    CvInvoke.FindContours(thresholdedImage, contours, null,
RetrType.List, ChainApproxMethod.ChainApproxSimple);

    // Create a mask to keep the largest white area
    Mat mask = new Mat(image.Size, DepthType.Cv8U, 1);
```

```
    mask.SetTo(new MCvScalar(0));

    // Extract the instance number from the first file in the folder
    // string firstFilePath = filesInSt5.FirstOrDefault();
    int instanceNumberVar = GetInstanceNumberFromFileName(filePath);

    if (instanceNumberVar <= x - instanceNumber)
    {
        // Find the index of the largest contour
        double maxArea = 0;
        int maxAreaContourIndex = -1;
        for (int i = 0; i < contours.Size; i++)
        {
            double area = CvInvoke.ContourArea(contours[i]);
            if (area > maxArea)
            {
                maxArea = area;
                maxAreaContourIndex = i;
            }
        }

        // Draw the largest contour onto the mask
        if (maxAreaContourIndex != -1)
        {
            CvInvoke.DrawContours(mask, contours, maxAreaContourIndex,
new MCvScalar(255), -1);
        }
    }
    else
    {
        // Sort contours by area in descending order
        List<Point[]> sortedContours = new List<Point[]>();
        for (int i = 0; i < contours.Size; i++)
        {
            sortedContours.Add(contours[i].ToArray());
        }
        sortedContours = sortedContours.OrderByDescending(c =>
CvInvoke.ContourArea(new VectorOfPoint(c))).ToList();

        // Draw the two largest contours onto the mask
        for (int i = 0; i < Math.Min(2, sortedContours.Count); i++)
        {
            CvInvoke.DrawContours(mask, new VectorOfVectorOfPoint(new[]
{ new VectorOfPoint(sortedContours[i]) }), -1, new MCvScalar(255), -1);
        }
    }

    // Apply the mask to the original image
    Mat result = new Mat();
    image.CopyTo(result, mask);

    // Save the result
    string fileName = Path.GetFileNameWithoutExtension(filePath);
    string outputPath = Path.Combine(outputFolderPath, fileName +
"_processed.tif");
    CvInvoke.Imwrite(outputPath, result);
}
```