Visualisering av optimaliserte digitale bilder av hudsykdommer

Mastergradsoppgave

av

Jan Martin Langeland

Veileder

Dhayalan Velauthapillai

Institutt for Informatikk

Universitetet i Bergen

Vår 2008

Forord

Arbeidet presentert i denne rapporten er en del av det avsluttende arbeid på mastergraden i informatikk ved Universitetet i Bergen.

Jeg vil rette en stor takk til min veileder Dhayalan Velauthapillai for hans gode støtte og verdifulle innspill, som har vært til stor hjelp under arbeidet.

Oppdragsgiver Balter Medical har gitt meg innsikt i et spennende fagområde som jeg tidligere ikke hadde noen erfaring med. Jeg vil takke Kristian P. Nielsen, Endre Sommersten, Børge Hamre og de andre i Balter Medical.

Jeg vil også takke mine medstudenter Svein Even Vikshåland og Øyvind Kvalsund for tips, hjelp og sosiale avbrekk.

Tilslutt vil jeg takke familien min for deres støtte.

Jan Martin Langeland

Bergen, Juni 2008

Innhold

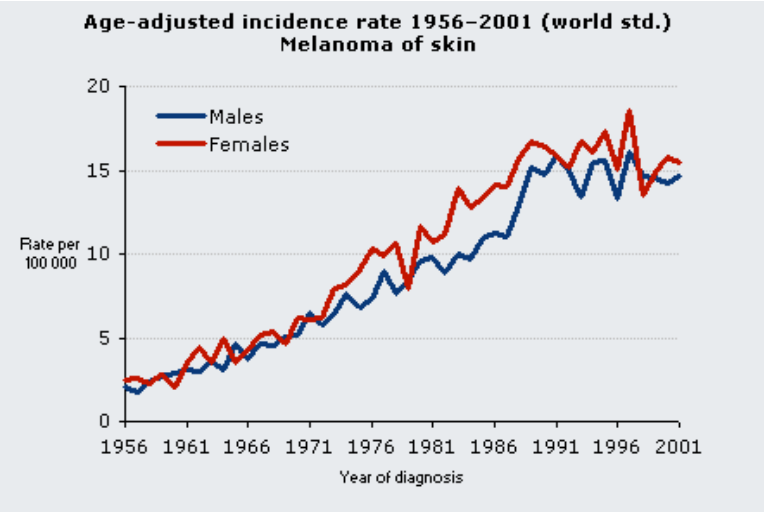
1 Introduksjon

Denne oppgaven ble gjort i tilknytning til, og i samarbeid med firmaet Balter Medical som utvikler apparater for optisk vevsdiagnostikk. Selskapet ble opprettet i Bergen i 2001, og er eid av investorer, entreprenører og ansatte i Norge og i USA. Det første apparatet, som fremdeles er under utvikling, retter seg mot å bistå dermatologer og leger i diagnostisering av føflekkreft. Det forventes at apparatet vil revolusjonere den omfattende og kostbare diagnostiseringsprosessen som eksisterer i dag.

1.1 Føflekkreft

Føflekkreft (malignt melanom) er en kreftform som dannes i hudens pigmentproduserende celler. Disse cellene, på fagspråk kalt melanocytter, produserer pigmentet melanin som gir huden brunfarge og beskytter den mot farlig sollys. Føflekker er et resultat av melanocytter som vokser sammen i klynger. Kreft oppstår når celler begynner å dele seg ukontrollert. Litt etter litt vil celler begynne å hope seg opp i organet der den ukontrollerte celledelingen startet. Melanomer oppstår vanligvis i eksisterende føflekker eller i form av nye føflekker, men noen ganger kan melanomer oppstå i øyne, lymfeknuter eller slimhinner.

Sjansen for å få føflekkreft øker med alderen. Kreftformen rammer flest personer over 50 år, men er samtidig den mest vanlige kreftformen blant kvinner fra 15-29 år og hos menn fra 30-54 år [1]. Overdreven soling med påfølgende solbrent hud er ofte en medvirkende årsak til føflekkreft. Man antar at for mye UV-stråling skader arvestoffet i cellene. Fra Figur 1.1 kan man se at siden 1956 har tilfellene av føflekkreft i Norge blitt ca. seksdoblet. Dette skyldes i stor grad endringer i folks vaner, men antakeligvis også bedre diagnoseteknikk og hyppigere prøvetagning [2]. Føflekkreft er den hurtigst voksende kreftformen, med en økning på 3-5% hvert år.



Figur 1.1: Tilfeller av hudkreft 1956-2001 [3].

1.2 Dagens diagnostiseringsprosess

Dagens diagnostiseringsprosess, se Figur 1.2, fra en person oppdager en skummel føflekk til en endelig diagnose blir gitt, er omfattende og kostbar. Når en pasient kommer til legen vil legen først undersøke lesjonen (uregelmessighet i huden) visuelt, for eksempel med et dermatoskop (forstørrelsesglass). Dersom den ser mistenkelig ut vil legen foreta en biopsi og skjære ut lesjonen. Den sendes så til et laboratorium for en vevs- eller celleprøve. Der blir lesjonen farget, lagt i voks, kuttet opp i tynne skiver, farget på nytt og tilslutt sendt til en patolog som vil se etter kreftceller i et mikroskop. Denne prosessen tar vanligvis 2 til 3 uker. Viser lesjonen seg å være positiv, vil mer av huden der føflekken satt blir fjernet for å være sikker på at alt kreftvev fjernes [4].

Figur 1.2: Dagens diagnostiseringsprosess. 1) Pasienten oppdager en skummel føflekk. 2) En lege utfører en visuell eksaminasjon. 3) Grunnet tvil vil føflekken skjæres bort i de fleste tilfeller. 4) En patolog mottar føflekken. 5) Patologen gir en diagnose.

En stor ulempe med dagens prosess er at det er vanskelig for en lege å gi en nøyaktig diagnose ved visuell eksaminasjon. Spesielt kan det være vanskelig å skille tidlige melanomer fra en normal føflekk (benign nevi). Legen kan bare inspisere overflaten, og har ingen mulighet til å finne indikasjoner på kreft som ligger lenger ned i huden. I de tilfeller hvor legen er usikker velges det normalt å skjære bort føflekken. Som et resultat av dette blir det funnet kreft i bare 1 av 40 av vevs- og celleprøvene som blir utført [4].

Tidlig diagnostisering av føflekkreft er svært viktig for å øke sjansene for overlevelse. For sen behandling øker faren for at kreften sprer seg til andre deler av kroppen. Det er knyttet store mørketall til hvor mange melanomer som blir oversett av leger. I et internasjonalt studie utført i 2002 overså legene melanomer i 25% av tilfellene [5]. Det er beregnet at kostnadene ved å overse et melanom, i verste fall, kan overstige $150000 [4].

For å få bukt med dagens problemer har Balter Medical utviklet en optisk skanner til diagnostisering av hudkreft. Det er forventet at skanneren vil fjerne mange av problemene ved dagens diagnostiseringsprosess. Å kunne stille diagnose gjennom en optisk skanner er et resultat av banebrytende fysikk, kombinert med raskere datamaskiner. På grunnlag av de kliniske prøvene utført til nå, har skanneren en sensitivitet på 99% og en spesifisitet på 94%. Dette betyr at den identifiserer 99% av alle skannede melanomer som melanomer, mens 94% av alle skannede ikke-melanomer blir identifisert som ikke-melanomer. Imidlertid vil det kreves mer testing før den endelige sensitivitet og spesifisitet kan fastsettes.

1.3 Ny diagnostiseringsprosess

Som nevnt i forrige avsnitt viser de foreløpige kliniske prøvene at skanneren kan identifisere melanomer med stor nøyaktighet. Det er likevel ikke gitt at skanneren alltid gir korrekt resultat. Den er derfor ment å være et hjelpemiddel til legen.

Fra Figur 1.3 kan vi se at når en pasient kommer til legen med en skummel føflekk, vil legen skanne lesjonen. Skanningen produserer digitale bilder, som i neste steg blir prosessert før skanneren kommer med et forslag til en diagnose. Skulle legen fremdeles være usikker på om det er kreft eller ikke, har han mulighet til å studere de prosesserte data nærmere. Disse data beskriver fysiologiske og morfologiske (formmessige) parametre i huden, som blodinnhold, blodoksygen nivå, tykkelsen på forskjellige hudlag, med mer. Samlet kan parameterne gi informasjon om lesjonen inneholder kreft eller ikke. For eksempel har melanomer høyere blodinnhold enn en normal føflekk, og kan derfor være en av flere indikasjoner på at det er kreft tilstede [6]. Prosessen som produserer fysiologi- og morfologidata vil bli beskrevet nærmere i kapittel 2.3.

Figur 1.3: Ny diagnostiseringsprosess. 1) Pasienten oppdager en skummel føflekk. 2) Legen skanner lesjonen og kan gi en diagnose øyeblikkelig.

1.3.1 Fordeler ved ny diagnostiseringsprosess

Både pasienter og helsevesen vil dra store fordeler av den nye diagnostiseringsprosessen. Den mest åpenbare er kanskje at diagnostiseringen blir mer nøyaktig. For pasienter innebærer dette at man i større grad unngår at føflekken må skjæres bort fordi legen er usikker, samt potensielle arr etter et slikt inngrep. I dagens situasjon gir også mange pasienter uttrykk for at tiden fra man er hos legen, til en diagnose blir stilt, kan være en påkjenning [7].

For helsevesenet betyr større nøyaktighet kraftig reduserte kostnader. Færre biopsier, hvor følgelig utførte vevs- og celleprøver på laboratorium vil reduseres, gjør at penger spares inn og kan brukes på andre områder. Imidlertid er ikke den største fordelen at kostnadene blir redusert. Med skanneren vil det bli enklere å oppdage tidlige melanomer, og få satt i gang behandling på et tidlig tidspunkt slik at liv vil bli spart.

1.4 Motivasjon

De siste 20 årene har det skjedd store fremskritt innenfor medisinsk visualisering. Raskere datamaskiner og teknologier som CT (Computed Tomography), PET (Positron Emission Tomography) og MRI (Magnetic Resonance Imaging) skal ha mye av æren for dette. Medisinsk visualisering er et tverrfaglig område som involverer problemstillinger fra medisin og informasjonsteknologi [8], i tillegg til fysikk.

Prototyper av skanneren har i lengre tid vært til klinisk utprøving hos en rekke sykehus. Hos Balter Medical forskes det fremdeles på hvordan man kan forbedre teknologien og diagnostiseringsprosessen. For å kunne analysere og validere data fra skanneren og den følgende diagnostiseringsprosessen er fysikerne avhenging av spesialtilpassede verktøy. Målet med denne masteroppgaven er av den grunn å utvikle et slikt verktøy. Verktøyet skal bestå av et grafisk brukergrensesnitt, hvor data kan visualiseres for raskere å kunne trekke konklusjoner fra mengden av dataene som blir produsert ved en skanning. Verktøyet skal også integreres mot en databaseløsning hvor data fra alle skanninger lagres.

1.5 Oppbygging av rapporten

Rapporten begynner i kapittel 2 med å se på skannerens oppbygning, med et innblikk i hvordan man kan fremstille en diagnose fra bildene som blir tatt ved en skanning. I kapittel 3 blir det sett på visualiseringsverktøy som fysikerne benytter seg av internt i bedriften for analyse og videreutvikling av skanneren, samt problemene med disse. Kapittel 4 tar for seg delproblemer, vurderinger og valg som har blitt tatt med hensyn til utviklingen av den nye løsningen, blant annet en velkjent metode for medisinsk visualisering. I kapittel 5 kan man lese om den nye løsningen og hvordan den har tatt form gjennom arbeidet. I tillegg vil det bli sett på en mulig fremtidig løsning, etter ønske om tilleggsfunksjonalitet fra oppdragsgivers side. Kapittel 6 tar for seg implementasjon av løsningen, med eksempler på hvordan den oppleves fra brukerens side. I kapittel 7 blir det foretatt en evaluering av løsningen og utviklingsmetoden. I tillegg blir det sett på spesielle utfordringer i arbeidet, og hvordan disse utfordringene ble løst. Bakerst i rapporten kan man lese en oppsummering av arbeidet og det videre arbeid.

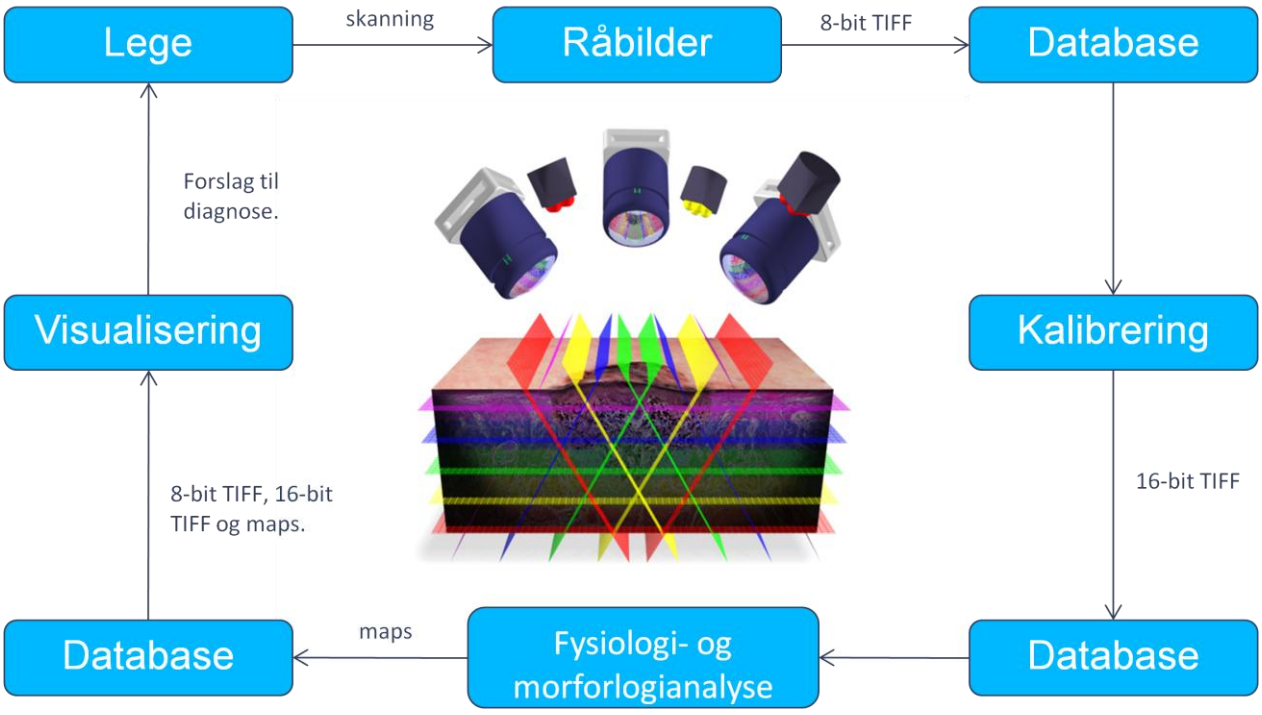
2 Oppbygging av skanneren – fra tagning til diagnose

I tillegg til CT, PET og MRI har nylig også OTR (Optical Transfer Retrieval) sett dagens lys. OTR er en ikke-kirurgisk metode som gjør det mulig å hente ut funksjonell informasjon fra vev, for eksempel blod- og oksygeninnhold. Basert på dette har Balter Medical utviklet sin egen metode kalt OTD (optical transfer diagnosis).

2.1 På innsiden av skanneren

OTD benytter seg av at friskt vev, godartede svulster og kreftsvulster har særegne optiske egenskaper, blant annet forskjellige grader av reflektans. En lesjon blir belyst med forskjellige bølgelengder, fra ultrafiolett (350nm) til nær infrarød (1000nm), som trenger ned til ulike dybder i huden. Dette gjør det mulig å hente ut informasjon fra forskjellige lag i huden som ellers er usynlig for det blotte øye [4].

En skanner består av et lukket system med 3 CCD (charged-couple device) kameraer plassert i 3 forskjellige observasjonsvinkler, samt tre LED-lamper (light-emitting diode) som sender ut lys på 10 forskjellige bølgelengder. Kameraene og lampene er rettet mot en glassplate. Ved en skanning (tagning) legges lesjonen mot glassplaten, og hvert kamera tar ett bilde for hver av bølgelengdene, til sammen 30 bilder.



Figur 2.1: Prosessoversikt - fra lege, til presentasjon av resultat.

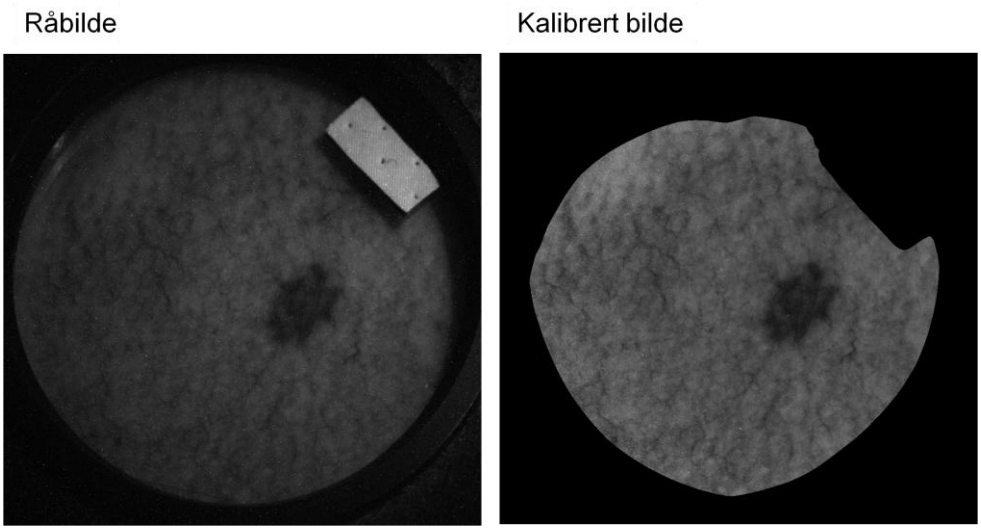
Bildene som blir tatt ved en skanning kalles råbilder, og er i svarthvitt(gråtonet) 8-bit enkanals TIFF format. For å få ut noe nyttig informasjon fra bildene må de legges inn på legens

datamaskin. Gjennom programvare på datamaskinen blir bildene prosessert. Dette innebærer en kalibrering og en fysiologi- og morfologianalyse som vil forklares nærmere i kapittel 2.2 og 2.3. Etter et par minutter vil programvaren komme med et forslag til diagnose, og gi legen mulighet for å inspisere lesjonen nærmere ved å se på visualiserte resultater fra fysiologi- og morfologianalysen. Fra Figur 2.1 kan man se at data fra hver delprosess lagres i en database. Per i dag gjøres dette i forsknings- og utviklingsøyemed, og det er derfor mulig at dette vil endre seg i fremtiden.

2.2 Kalibrering

Før man kan starte kalibreringsprosessen må man, enkelt forklart, i tillegg til råbildene ta et sett bilder av en kalibreringsstandard med en kjent BRDF (bi-retningsreflektans distribusjons funksjon). Disse kalles CS (calibration standard) bilder, og må tas en gang for dagen. Tilsvarende tas et sett med DC (dark current) bilder, som er bilder som er tatt i et helt mørkt miljø hvor lys ikke slipper til. Settene med CS og DC inneholder 300 bilder hver og brukes til å produsere et nytt sett med kalibreringsbilder. Dette settet inneholder ett kalibreringsbilde per bølgelengde for hvert kamera, totalt 30 bilder. Formålet med denne fremgangsmåten er å redusere støy, og å fjerne overeksponerte piksler og bakgrunn.

Råbildene og kalibreringsbildene blir så kjørt inn i en kalibreringsalgoritme. Bildene som blir produsert kalles kalibrerte bilder, og er i enkanals 16-bit TIFF format. I motsetning til et råbilde er intensiteten i et kalibrert bilde uavhengig av eksponeringstid og ujevn belysning. Således gir et kalibrert bilde objektiv informasjon om reflektansen til det avbildede hudområdet. I Figur 2.2 kan man forskjellen på et råbilde, og et kalibrert bilde.



Figur 2.2: Kalibreringsprosessen. Til venstre, et råbilde. Til høyre, det tilhørende kalibrerte bildet som kommer ut av kalibreringsprosessen. Per i dag måler bildene 1114x1105 piksler.

Parallelt med denne oppgaven har en medstudent skrevet masteroppgave rundt kalibreringsprosessen. Dersom man ønsker å vite mer om kalibreringsprosessen i detalj, kan man lese om dette i [9].

2.3 Fysiologi- og morfologianalyse

Som nevnt tidligere trenger forskjellige bølgelengder ned til ulike dybder i huden. Hvorav ultrafiolett lys trenger ned bare overflatisk, mens infrarødt lys går dypere ned i huden. Ved å se på et kalibrert bilde av en lesjon kan man kun trekke grove slutninger. I et bilde tatt med infrarødt lys vil områder med konsentrasjoner av melanin, som er pigmentet i huden, vanligvis fremstå som mørke områder [6]. Imidlertid har man ingen mulighet til å si noe om den nøyaktige mengden melanin som er tilstede. For å produsere informasjon som er av diagnostisk relevans, må de kalibrerte bildene gjennom en fysiologi- og morfologianalyse.

Det er mange fordeler ved å analysere den fysiologiske og morfologiske tilstanden i huden fremfor kun å studere bilder. Analysen gjør det mulig å hente ut målbare parametere som kan brukes til å trekke slutninger om tilstanden til huden. Parameterne, og hvorfor de har relevans for hudkreft er beskrevet kort nedenfor, men den nøyaktige fremgangsmåten, eller algoritme, for å stille en automatisk diagnose basert på disse parameterne er utelatt i denne rapporten da dette regnes som en forretningshemmelighet.

Morfologiske parametere beskriver den formmessige tilstanden i huden. For melanomer blir det naturlig å trekke frem asymmetri, diameter og grense. Melanomer er vanligvis store og asymmetriske, i motsetning til en godartet føflekk som er symmetrisk og har en diameter på under 5mm. I tillegg har melanomer ofte uregelmessig avgrensning mot friskt vev.

I fysiologianalysen utledes parametere gjennom fastsatte modeller av lysabsorbering og spredning i kromoforer (fargebærere), for eksempel hemoglobin som gir blodet sin karakteristiske rødfarge [10]. Analysen gir informasjon om blodinnhold, blodoksygen nivå, keratininnhold, melanininnhold og distribusjon og tykkelse på forskjellige lag av huden. Utover at melanomer har høyt blodinnhold, har de skadde kapillærårer slik at det vil være områder i disse med lavt oksygennivå. Videre finner man ofte mindre keratin i melanomer. Keratin er et protein som det finnes mye av i hår og negler. I huden gir keratin fasthet, og man finner det i det helt øverst i huden. Melanin gir huden brunfarge og blir produsert av melanocytter som befinner seg i epidermis. Uregelmessige konsentrasjoner av melanin utenfor dette området kan være et tegn på kreft. Isolert sett er ikke tykkelse en direkte indikator på kreft. Godartede svulster kan også føre til at tykkelsen i hudlagene øker, men sett i sammenheng med de øvrige parameterne kan man trekke konklusjoner ved hjelp av tykkelsen [6].

Etter at de kalibrerte bildene har vært gjennom fysiologi- og morfologianalysen får man ut såkalte fysiologimaps (heretter kalt maps) som kan brukes til å stille en diagnose. Dette er

tekstfiler med flyttall i en matrise som representerer hver av de overnevnte parameterne fra et forhåndsdefinert, interessant område i en lesjon, kalt ”*region of interest*”.

3 Problembeskrivelse

Arbeidet med skanneren er på nåværende tidspunkt i en utviklingsfase. Det jobbes blant annet med forbedringer i skanneren, samt forskning for å forbedre diagnostiseringen. Når fysikere utvikler ny teknologi ligger det vanligvis mye tung teori bak. Med tung teori følger det også et behov for å kunne undersøke at teorien holder mål i praksis. En fysiker vil gjerne teste at beregninger som er gjort underveis er korrekte, eller at data som er prosessert av en algoritme kommer ut som de skal.

3.1 Dagens situasjon

Når det tas bilder med en skanner og disse prosesseres på en datamaskin, er det i forsknings- og utviklingsøyemed mange situasjoner hvor det kan være behov for å verifisere resultater. En svak lampe, eller et dårlig kamera i skanneren kan føre til at man får store mengder støy på CS, DC og råbildene. Dette kan forplante seg videre ved at resultater fra kalibreringsprosessen, og fysiologi- og morfologianalysen blir unøyaktig, eller i verste fall gale. I noen tilfeller er det mulig å lage automatiserte tester som kan gi beskjed, for eksempel hvis et kamera er dårlig eller en lampe begynner å bli svak. Inni skanneren finnes en hvit brikke, kalt *"relative calibration area"*, som brukes til å sammenligne en kombinasjon av kamera og lampe mot et forhåndsdefinert standardavvik. Dersom en lampe er dårlig, vil lys som reflekteres fra brikken falle utenfor standardavviket, og skanneren vil gi beskjed om at den må på service.

Ved en tagning er det potensielt mange ting som kan gå galt. En liten bevegelse som gjør at et av bildene kommer ut av fokus er nok til å ødelegge en hel tagning. Å skrive en automatisert tester som oppdager og sjekker dette kan være vanskelig. I stedet må man ty til manuell testing og validering av data. For fysikerne hos Balter Medical går en god del av forsknings- og utviklingsarbeidet ut på nettopp manuell validering av data. Man luker da ut data som kan generere gale resultater videre. Derfor må bilder fra forskjellige lesjoner studeres og sjekkes visuelt. I dag benytter de seg av en felles server hvor MATLAB, og egenutviklede MATLAB-skript blir kjørt for å vise bilder og maps. MATLAB inneholder funksjonalitet som gjør at man kan fargelegge bilder og maps, slik at man får frem interessante detaljer eller feil i bildene på en forholdsvis enkel måte.

Bilder og maps ligger lagret i filstruktur og er organisert i mapper etter hvilket nivå (level) de tilhører. Nivået angir om det er råbilder, kalibrerte bilder eller maps. I Eksempel 3.1 kan man se en oversikt over dagens filstruktur.

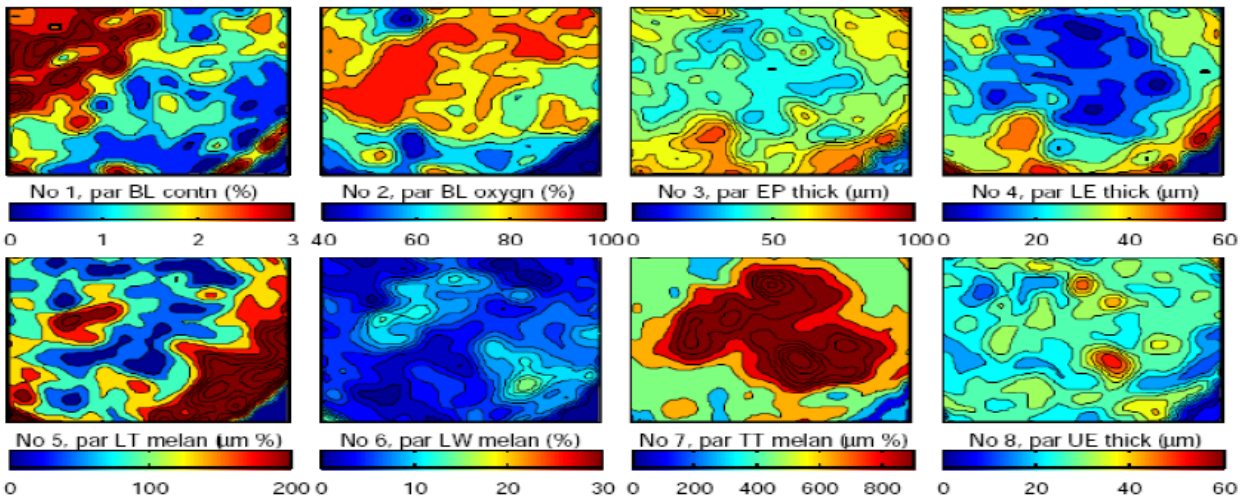
```
|-- level1
|   |-- hk
|   |   |-- 2005_12_01
|   |   |   |-- hk0034_0002_view1_123nm.tif
|   |   |   |-- hk0034_0002_view2_234nm.tif
…
|-- level2
…
|-- level3
|   |-- physiology
|   |   |-- OUT_hk0034_1
|   |   |   |-- par_BL.contn
|   |   |   |-- par_BL.oxygn
|   |   |   |-- par_EP.thick
|   |   |   |-- par_UP.melan
…
```

Eksempel 3.1: Dagens filstruktur. Level1 er råbilder, level2 er kalibrerte bilder, mens level3 er fysiologi- og morfologidata. Level2 har samme filstruktur som level1. I tillegg finnes egne mapper for DC/CS-bilder og kalibreringsbilder.

Råbilder og kalibrerte bilder, henholdsvis level1 og level2, er delt inn i mapper for sykehus, og hvilken dato de er tatt. I tillegg finnes det også egne mapper for kalibreringsbilder, CS- og DC-bilder. For å åpne et bilde gis det input til en spesifikk funksjon i et MATLAB-skript som parser filstrukturen og leter frem riktig bilde. Man benytter seg da av et saksnummer som inneholder forkortet sykehusnavn, pasientnummer, lesjonsnummer og tagning. Eksempelvis betyr ”hk0034_0002” at bildet er tatt ved Haukeland Sykehus, pasientnummeret er 0034, lesjonsnummeret er 00 og tagningsnummeret er 02.

I tillegg må man gi input om hvilket kamera bildet er tatt med, en ID som tilsvarer en bestemt bølgelengde, samt ønsket nivå. Uttrykt direkte i MATLAB-syntaks blir dette GetImage(saksNummer,kamera,bølgelengdeID,nivå).

Ved åpning av maps må man kalle inspect_parameters({'saksNummer'}). Man vil da laste inn alle maps for det aktuelle saksnummeret. Filene blir så visualisert og vist side om side gjennom innebygde MATLAB-funksjoner [11]. I gjennomgangen av fysiologi- og morfologianalysen i kapittel 2.3 ble det nevnt at ikke alle parameterne nødvendigvis er direkte indikatorer på kreft, men at man må se dem i sammenheng med de andre parametrene. Ved å ha visualiserte maps side om side kan man lettere trekke slutninger om det er kreft tilstede eller ikke, for eksempel hvis det er mye blod og høyt melanininnhold på samme posisjon, samt en generell ujevn fordeling av de ulike parameterne. I Figur 3.1 kan man se et eksempel på dagens visualisering av maps.



Figur 3.1: Utsnitt fra dagens visualisering av maps. Hvert rektangel tilsvarer en mapfil.

I andre sammenhenger kan det være interessant å sammenligne bilder og maps fra forskjellige tagninger av samme pasient. I blant tas det flere tagninger, for eksempel hvis bildene av en lesjon blir uklare fordi skanneren ikke ble holdt skikkelig inntil huden.

Utenom skriptene som er nevnt over, finnes en mengde andre skript som tar seg av ting som relasjoner mellom filer, og parsing av filstruktur. Ved siden av denne oppgaven har en annen student skrevet masteroppgave som gikk ut på å samle og strukturere alle data i en relasjonsdatabase [12].

3.2 Problemer med dagens situasjon

Det finnes en del ulemper med dagens løsning. I utviklingsarbeidet med skanneren har fysikerne ulike ansvarsområder. Hvor noen har behov for å analysere en full oversikt av alle maps fra en tagning, ønsker andre en enklere fremstilling. Dette har ført til at bestemte skript, slik de var i sin opprinnelige form, har blitt endret og tilpasset etter behov. Følgelig har man fått mange forskjellige versjoner, og det blir vanskelig å vite hvilket av skriptene man skal benytte seg av uten å kikke på koden.

Når fysikerne skal studere bilder åpner de vanligvis 5-6 bilder om gangen, til tross for at de kan ha behov for å åpne flere. Det er flere grunner til dette. I MATLAB vises hvert bilde i et separat vindu, og åpner man flere bilder blir det fort uoversiktlig og vanskelig å holde styr på hva som er i hvilket vindu. Visualiseringen av bildene i dag gjøres hovedsakelig ved å legge pseudofarge på bildene for å få frem detaljer i bildene. Bilder som blir fargelagt bruker betraktelig mye mer minne enn et som er i sin originale form. Med tanke på at flere brukere kjører MATLAB fra samme server, kan dette bli problematisk dersom flere brukere har mange bilder åpne samtidig. En mulig årsak til den økte minnebruken kan være at MATLAB konverterer originale 8-bits (eller 16-bits) bilder, til 32-bits per piksel ved fargelegging. Dette bildet vil da inneholde de originale

dataene. I tillegg blir opprettet en kopi av dette igjen som vises på skjerm. Dette er dog bare spekulasjoner, så den reelle årsaken kan være av en helt annen sort.

Bruken av MATLAB-skriptene baserer seg på input fra konsoll. Noen ganger kan kommandolinjebaserte verktøy være svært effektive, mens i andre tilfeller fungerer grafiske brukergrensesnitt bedre. Å spesifisere input og kalle GetImage(saksNummer,kamera,bølgelengdeID,nivå) for hvert bilde man vil åpne er upraktisk. Skal man åpne alle bildene fra en tagning kan man skrive kode (en løkke) i MATLAB som itererer over kamera og bølgelengdeID, men det kan diskuteres hvorvidt en slik løsning er enklere enn å åpne et og et. Åpnes alle bildene blir det som nevnt problematisk å holde styr på alle vinduene, og hvis det er behov for å se på et utdrag av bildene fra en tagning, ender man likevel opp med å måtte forandre koden og spesifisere kamera og bølgelengder fra gang til gang.

Visuell sammenligning av bilder fra forskjellige tagninger kan være vanskelig. For å forenkle denne prosessen, sammenlignes pikselverdiene fra en bestemt skannlinje (rad eller kolonne) i to eller flere bilder. Dette gjøres ved å legge pikselverdiene fra hvert bilde inn i en profilvisning (kurveplott) slik at det blir enklere å se forskjellene. I dag må man manuelt skrive kode hvor man spesifiserer hvilke bilder og rader som skal benyttes, samt hvordan det skal plottes. Ved sammenligning av mange bilder blir dette tungvint.

Prosessen med å åpne 5-6 bilder tar ca 30 sekunder, mens fargelegging tar noe lenger tid. For å kunne studere lesjonen i detalj, må en for hvert bilde zoome inn på det aktuelle området, noe som totalt tar 30 nye sekunder. Åpner man maps for et gitt saksnummer tar denne prosessen ca et minutt. I et enkelt tilfelle er dette kanskje ikke så mye, men i det lange løp blir dette mye verdifull tid som sløses bort.

3.3 Problemstilling

Frem til i dag har MATLAB blitt brukt i nesten alle sammenhenger hva gjelder analyse og visning av data. I forhold til arbeidet som utføres har det vist seg vanskelig å finne noen bedre verktøy, da disse må spesialtilpasses. MATLAB fungerer greit, men som nevnt over er det en del problemer og irritasjonsmomenter, især med hensyn til tidsbruk. I den sammenheng er det uttrykt et ønske om å lage en ny applikasjon, skreddersydd for visning av bilder og maps. Denne skal være et visualiseringsverktøy til bruk hovedsakelig i videre forskning, men skal også danne en basis for hva som kan være aktuell programvare hos en lege. For å spare lisenskostnader er det ønskelig at applikasjonen skal være åpen kildekode. Applikasjonen skal ha et grafisk brukergrensesnitt (GUI) og skal kjøres på Linux. Visualiserte data skal kunne vises opp mot hverandre slik at man får sett dem i en større sammenheng, og det skal være mulig å vise mange bilder og maps av gangen. Åpning og visning av disse skal være enkelt og hurtig slik at man i størst mulig grad slipper ventetid. Videre skal det være enkelt å sammenligne bestemte

områder i bilder og maps, for eksempel gjennom profilvisning. Det er også viktig at applikasjonen er utvidbar, slik at man kan legge til funksjonalitet i ettertid. Til slutt, for å gjøre det enklere å hente ut og vise data, er det et ønske om at applikasjonen skal knyttes opp mot databaseløsningen som blir utviklet av en medstudent [12].

Applikasjonen er ment å gjøre det enklere for fysikerne å analysere og validere data. Hverdagen til fysikerne blir mer effektiv, og de slipper å kaste bort verdifull tid på venting og tungvint behandling av data.

4 Problemanalyse

Punktene som er nevnt over er generelle ønsker fra oppdragsgiver. Mange ting har blitt diskutert underveis, for eksempel hvordan det grafiske brukergrensesnittet skal se ut, eller hvordan bilder og maps skal visualiseres i detalj. For å organisere arbeidet og identifisere de ulike oppgavene bedre, faller det naturlig å dele opp problemstillingen i mindre delproblemer.

4.1 Delproblemer

Det er viktig å først avklare problemer som har innflytelse på applikasjonen i sin helhet, for dermed å gjøre det enklere å ta gode beslutninger i forhold til mer konkrete delproblemer. I lys av at oppdragsgiver ønsket åpen kildekode var det betydningsfullt å diskutere om en slik løsning var hensiktsmessig. Et sentralt spørsmål var hvorvidt applikasjonen ville inneholde forretningssensitiv informasjon. I et slikt tilfelle ville det være fordelaktig å benytte seg av lukket kildekode for å unngå at denne informasjonen kom på avveie. Etter nøye vurdering ble det besluttet at dette ikke kom til å være et problem, da kildekoden ikke ville berøre denne type informasjon.

Som nevnt ble det parallelt med denne oppgaven utviklet en databaseløsning. Fordi det i en tidlig fase av arbeidet ble klart at databaseløsningen først ville bli implementert lenger frem i tid, ble det valgt å innføre støtte for å kunne åpne bilder og maps fra filsystem. Dette ble gjort med hensyn til at det på dette tidspunktet ikke var kjent hvorvidt bilder og maps ville bli lagret direkte i databasen, eller om man i stedet ville nøye seg med å lagre filstiene. Uavhengig av dette ville det også uten innføring av slik støtte vært svært vanskelig å få testet data i applikasjonen underveis.

Med bakgrunn i dette kan man dele opp problemstillingen i følgende delproblemer:

- Grafisk brukergrensesnitt
- Innlesning av bilder og maps fra filsystem
- Visualisering av maps
- Visualisering av bilder
- Profilvisning (kurveplott)
- Integrasjon med databaseløsning
- Høy ytelse
- Mulighet for utvidelse

Hvorav de fem første punktene kan sies å være funksjonelle krav, er de to sistnevnte ikke-funksjonelle krav. Ytelse og mulighet for utvidelse vil berøre alle deler av applikasjonen og må derfor tas spesielt hensyn til, da de vil ha innvirking på alle de andre delproblemene. Det er også verdt å nevne at utformingen av det grafiske brukergrensesnittet kan ha stor betydning for

hvordan visualisering og visning av data vil foregå, og omvendt. Inndelingen av delproblemene har også fungert som milepæler i form av delmål.

4.2 Grafisk brukergrensesnitt og visning av data

Det finnes flere bibliotek man kan benytte seg av for å lage et grafisk brukergrensesnitt. I sammenheng med at oppdragsgiver ønsket at systemet skulle være åpen kildekode, var det ønskelig å finne et alternativ som i størst mulig grad hadde støtte for nødvendig funksjonalitet. Av de mest interessante var wxWidgets, GTK+ og Qt. Valget her falt fort på Qt [13], først og fremst fordi det inneholdt det aller meste av påkrevd funksjonalitet, men også fordi Qt er et anerkjent og veldokumentert bibliotek. At jeg tidligere har brukt dette i sammenheng med studiene på Høgskolen i Bergen hadde også en viss innflytelse. Qt er bygget rundt et stort antall ferdige GUI komponenter, eller widgets, som man kan benytte seg av for å sette sammen et GUI. I denne oppgaven har Qt 4.3 blitt benyttet.

Qt har et tilleggsbibliotek, kalt Qwt (Qt Widgets for Technical applications) [14], som inneholder funksjonalitet for visning av forskjellige typer grafer og plott. Da en del av oppgaven gikk ut på å lage profilvisninger av bilder og maps, samt det faktum at Qt allerede var valgt, ble Qwt et enkelt valg. Ettersom bare en liten bit av funksjonaliteten i Qwt ville bli benyttet i oppgaven kunne det jo argumenteres med at det ville vært fordelaktig å lage denne delen selv, men tanken på å lage noe som allerede fungerer bra på nytt, var lite tiltalende.

I Qt 4.2 ble GVF (Graphics View Framework) introdusert. Dette er et innebygget rammeverk i Qt som inneholder funksjonalitet for å behandle et stort antall egendefinerte, grafiske elementer, og et lerret (view) som man kan vise frem disse elementene på. Rammeverket tilbyr også muligheter for brukerinteraksjon, for eksempel zooming eller klikking på elementer for å vise ekstra informasjon. Med litt tilpasning var det klart at dette ville by på mange muligheter med tanke på visualiseringen av bilder og maps. Den første tanken var å definere et bilde eller et map som et grafisk element, slik at man kan vise mange av disse side om side slik at de blir lettere å sammenligne. I tillegg, ved å ta i bruk interaksjon, kan man for eksempel la brukeren dobbelklikke på et gitt punkt i et bilde slik at dataene i kolonnen (hvor punktet befinner seg) blir vist i en profilvisning.

4.3 Visualisering av bilder og maps

I [15] er det nevnt at visualiseringer genereres for et spesielt formål. Videre er det beskrevet at ved visualisering i medisinske applikasjoner, så er dette formålet ofte et scenario hvor man bruker medisinske data til en diagnostisering. Slik er det også i denne oppgaven, hvor medisinske data som brukes til diagnostisering består av maps. I tillegg innebærer visualiseringen i denne oppgaven også bilder. Til forskjell fra maps er i større grad formålet med å visualisere disse å forenkle videreutviklingen av skanneren og teknologien som ligger bak.

Når data skal visualiseres kan man benytte seg av mange forskjellige fremgangsmåter. I noen tilfeller er en todimensjonal fremstilling best, mens i andre tilfeller fungerer tredimensjonale fremstillinger bedre. For bilder ville det naturlig nok bli vanskelig med noe annet enn en todimensjonal fremstilling, mens for maps ville det være mulig å lage en tredimensjonal fremstilling. Fordi en todimensjonal fremstilling hadde størst nytteverdi og var det enkleste å arbeide med, hadde oppdragsgiver et sterkt ønske om å også benytte en todimensjonal fremstilling for maps.

Siden visualisering ikke bare dreier seg om dimensjonalitet, er det viktig også å undersøke mer konkrete metoder og teknikker for hvordan man skal vise data. I den sammenheng må man også studere hvordan dataene er representert. Som nevnt inneholder en mapfil en matrise med flyttall, der hver fil representerer en parameter i huden. Hver for seg inneholder de ulike filene forskjellige spektre av tall, definert av hvilken type parameter det er. En mapfil for blodinnhold vil typisk inneholde tall fra 0 til 3, mens for keratininnhold vil disse tallene vanligvis ligge mellom 20 og 50. Disse spektrene er fastsatt gjennom målinger fra forskningsarbeidet hos Balter Medical og er sådan ikke tilfeldige. Et map kan derfor sees på som et lite bilde av huden som i stedet for å vise den fysiske overflaten, avbilder den underliggende tilstanden.

Da selve fremstillingen av data i MATLAB fungerte bra, ble det foreslått at pseudofarge på bilder og fargekart for maps (som vist i Figur 3.1) ville være gode måter å visualisere på. Disse fremstillingene baserer seg begge på bruk av en transferfunksjon, der et gitt tall i et map eller en pikselverdi i et bilde omdannes til en farge og avbildes etter et forhåndsbestemt fargeskjema. For å illustrere dette i et konkret, men meget forenklet eksempel, kan man ta utgangspunkt i et map for blodinnhold og angi farger:

- Grønt ved lite blod.
- Rødt ved mye blod.

Ved gjennomgang av matrisen vil tall nært 0 tegnes opp som en grønn piksel, mens tall nærmere 3 vil bli røde piksler. Etter gjennomgangen får man da et bilde der områder med mye blod vises som en samling røde piksler. Bruken av transferfunksjoner er mye brukt innen annen medisinsk visualisering, blant annet ved visualisering av volumetriske data fra CT og PET [16].

Matrisen i en mapfil måler per dags dato maks 101x101, men kan også være mindre. Med slike størrelser vil en grafisk representasjon av et map kunne oppleves som lite og kornete. For å unngå at resultatet ble enda mer kornete ved oppskalering, ble det valgt å benytte bilineær interpolering. Med denne metoden kan man, kort fortalt, finne verdien (fargen) til en ny piksel ved å beregne et vektet gjennomsnitt av de fire nærmeste kjente piksler. En mer detaljert fremstilling av dette kan leses i [17].

Med bakgrunn i dette ble det laget en enkel prototype som leste inn og visualiserte mapfiler fra filsystem. Denne prototypen gjorde det mulig å få en oversikt over hvor komplisert det ville være å visualisere maps, samt danne et grunnlag for GUIet. Sammen med innspill og tilbakemeldinger fra oppdragsgiver, gav dette et utgangspunkt for den videre utviklingen.

4.4 Integrasjon mot database

Som nevnt er hensikten med å integrere applikasjonen med den nye databaseløsningen å forenkle åpning av data. Integrasjonen gjelder altså ikke innsetting, men bare uthenting og visning av data. Bortsett fra at det måtte tas hensyn til hvordan databasen var implementert, ble det fra oppdragsgiver sin side gitt relativt frie tøyler i forhold hvordan denne integrasjonen skulle foregå.

Med dagens situasjon tatt i betraktning, ble det tatt utgangspunkt i en søkefunksjon som gjorde det mulig å søke etter bilder og maps basert på parametre som forkortet sykehusnavn, pasientnummer, tagningsdato, patologisk diagnose og type (råbilde, kalibrert bilde, kalibreringsbilde, DC/CS-bilde eller maps) m.m. Selv om tagningsdato og patologisk diagnose alltid har blitt registrert ved skanninger, har det ikke med dagens løsning vært mulig å åpne data basert på disse. For eksempel, ved kun å spesifisere forkortet sykehusnavn og patologisk diagnose kan man da finne bilder og maps fra alle melanomer som er tatt ved et gitt sykehus.

4.5 Utviklingsmetode

Det er vanskelig å knytte arbeidet med denne oppgaven direkte opp mot en kjent utviklingsmetode, men skal man trekke frem noe som nærliggende må det være en smidig metode, for eksempel Extreme Programming (XP) [18]. En smidig metode har kjerneverdier som å samarbeide nært med kunden og å ta i mot forandringer med åpne armer fremfor å følge en plan. Videre er det fokus på mindre dokumentasjon, og desto større fokus på leveranser av virkende programvare. En vanlig feiltolkning av dette er at man ikke produserer dokumentasjon i det hele tatt, mens det i realiteten betyr at man skal være tilpasningsdyktig og produsere dokumentasjon dersom det er behov for det. Selv om det i smidige metoder er leveranser av virkende programvare som er den prinsipielle metoden for å måle fremgang, har det blitt skrevet en enkel fremdriftsplan. Intensjonen med denne planen var ikke å følge de fastsatte datoene slavisk, men rett og slett å dele opp og strukturere arbeidet bedre, slik at en sammen med leveranser av programvare kan danne seg et bilde av hvordan en ligger an. Under arbeidet har applikasjonen kontinuerlig blitt vist frem til oppdragsgiver etter hvert som ny funksjonalitet ble lagt til. Det kan selvsagt diskuteres om dette alltid har kunnet kalles en leveranse. I noen tilfeller ville det nok være drøyt å bruke dette ordet, da endringer eller ny funksjonalitet har vært av liten art.

Enkelhet har vært en annen av de grunnleggende praksisene som har blitt benyttet i arbeidet. Dette går ut på at man skal vurdere det enkleste som kan virke, og designe systemet deretter. I dette inngår ting som ”du kommer ikke til å trenge det” (YAGNI) og ”en gang, og bare en gang” (Once, and only once) [18]. Dette går ut på at selv om man for eksempel vet at applikasjonen skal integreres med en database, så legges det ikke opp til det i koden før slik infrastruktur og funksjonalitet er helt nødvendig. Med ”en gang, og bare en gang” menes at man ikke skal ha duplikatkode. Hvis man oppdager slik kode skal den fjernes ved å plassere den i felles funksjoner eller baseklasser. Dette kan gjøres gjennom regelmessige refaktoreringer som forbedrer kodens interne struktur, uten å endre utvendig virkemåte.

I smidige metoder er akseptansetesting nært forbundet til brukerhistorier. En brukerhistorie regnes ikke som ferdig før den har blitt godkjent gjennom en akseptansetest. I dette arbeidet har det ikke blitt skrevet konkrete brukerhistorier, men praksisen med kontinuerlig å vise frem ny funksjonalitet har fungert som en form for akseptansetesting, der oppdragsgiver har verifisert at denne funksjonaliteten fungerte slik den var tiltenkt. En annen sentral praksis i smidige metoder, og da spesielt XP, er testdreven utvikling (TDD). TDD går ut på å først skrive enhetstester, for så å skrive koden som det testes på. I dette arbeidet har det vist seg vanskelig å benytte denne praksisen. For eksempel er det vanskelig å teste grafiske brukergrensesnitt på en god måte, eller skrive en enhetstest som sjekker at visualiserte data vises frem korrekt.

4.6 Programmeringsspråk

Å ta et nøye overveid valg av programmeringsspråk trenger ikke være så viktig hvis man skal lage et svært enkelt program hvor det ikke foreligger noen spesielle krav, for eksempel til ytelse. Det går da an å velge programmeringsspråk basert på personlige preferanser uten at det får store konsekvenser, men om systemet er av en viss størrelse kan det være uheldig å ta en avgjørelse på denne måten. I et slikt tilfelle vil det være naturlig å velge ut i fra attributtene til programmeringsspråket, og hvor gode disse attributtene er til å løse et bestemt problem. Fra en utvikler sitt synspunkt går det an å tenke på programmeringsspråk som et hvilket som helst verktøy, der det er viktig å finne det verktøyet som passer best til problemet eller oppgaven man skal løse. I denne oppgaven var det fokus på å benytte et objektorientert språk som har støtte for arv, interfaces og abstrakte klasser. Dette er attributter som kan være med på å gjøre videreutvikling enklere.

Qt er opprinnelig skrevet i C++, men er også tilgjengelig i Java. Der disse to er de mest brukte, finnes det også bindinger for en rekke andre programmeringsspråk, blant annet Python. I forhold til Qt var inntrykket at disse, med unntak av Python, var preget av dårligere dokumentasjon enn det som var tilgjengelig for C++ og Java. Da jeg fra tidligere ikke har noe erfaring med Python, samt at det var lite fristende å bruke tid på å lære seg dette fra bunnen av, gjorde at det ble valgt bort. Man stod da igjen med C++ og Java. Til forskjell fra C++ blir Java

kompilert til bytecode, og tolket gjennom en JVM (Java Virtual Machine) som kan kjøres på en datamaskin, uavhengig av operativsystem eller arkitektur. Grunnet mellomleddet med tolkning er Java noe tregere enn C++, som kompileres direkte til maskinkode. Til fordel har Java automatisk minnehåndtering, noe som kan forenkle arbeidet rent utviklingsmessig. Med bakgrunn i at det var ønskelig med høy ytelse, og fordi jeg tidligere hadde brukt C++ versjonen av Qt, ble det valgt å benytte C++.

4.7 Språk i systemet

Valget av språk kan deles inn i språk utad mot brukeren, og språk internt i systemet. For førstnevnte ville det være enklest å hardkode inn teksten som vises for brukeren, men med tanke på at applikasjonen skulle være utvidbar ville det være fordelaktig med internasjonalisering. I Qt finnes et oversettingsverktøy, kalt Qt Linguist [19], som forenkler arbeidet med oversetting. Dette krever at man i koden omslutter all tekst som vises for brukeren med en spesiell syntaks. Basert på koden kan man da, gjennom Qt Linguist, produsere en språkfil som kan oversettes til ønsket språk. Den oversatte språkfilen kompileres så inn med resten av applikasjonskoden. Til tross for at koden har blitt tilrettelagt for internasjonalisering ble det mot slutten nedprioritert å produsere en språkfil, grunnet knapt med tid.

Som nevnt har Balter Medical avdelinger i Norge og USA, med ansatte fra mange forskjellige land. Kommunikasjon mellom ansatte forgår for det meste på engelsk, og det var derfor ønskelig at engelsk skulle være det primære språket i applikasjonen. Med tanke på videreutvikling er det vanskelig å vite om dette vil foregå internt eller outsources til en tredjepart i Norge eller et annet sted i verden. For en utvikler er det nok tungvint å måtte bruke tid på å oversette variabel-, funksjons- og klassenavn. Kode skrevet på norsk kunne derfor føre til at videreutvikling og vedlikehold ville bli vanskeligere. Av den grunn ble det besluttet å kode på engelsk.

4.8 Navnekonvensjoner

Med hensyn til videreutvikling har det blitt lagt vekt på å bruke utfyllende funksjons-, klasse- og variabelnavn. Dette øker kodens leselighet og gjør den mer ryddig og oversiktlig. I dette arbeidet CamelCase blitt brukt som en standard for navngivning. Med CamelCase blir navn bestående av flere ord satt sammen til et samlet ”uttrykk”, hvor hvert ord begynner med stor forbokstav. For klassenavn og funksjons- og variabelnavn deler man henholdsvis inn i ”UpperCamelCase” og ”lowerCamelCase”.

4.9 Verktøy

Utviklingen i dette arbeidet har foregått på Ubuntu 7.10 [20]. At Linux ble valgt fremfor Windows var av den enkle grunn at oppdragsgiver benyttet Linux. I forhold til oppgaven var

dette ellers av mindre betydning, da Qt er et kryssplattform rammeverk som kan kompileres både for Linux, Windows og Mac.

I begynnelsen ble det prøvd en del forskjellige grafiske IDE. Med god erfaring med Eclipse [21] ble denne prøvd sammen med tilleggsmodulene CDT [22] og Qt-Eclipse [23]. Hvor Eclipse vanligvis er et IDE for Java, gjør CDT det mulig også å programmere i C++. Qt-Eclipse gir på sin side støtte for Qt spesifikke ting som håndtering av prosjektfiler og en WYSIWYG-editor for Qt-baserte GUI. Dessverre krasjet Eclipse flere ganger daglig med disse tilleggsmodulene, og ble håpløst å arbeide med. En mulig årsak til dette var at Qt-Eclipse bare var tilgjengelig som beta på dette tidspunktet. Det ble derfor byttet til KDevelop [24] som har innebygget støtte for Qt. Da denne hadde en tendens til å glemme innstillinger, ble det tilslutt byttet til QDevelop [25]. Til forskjell fra Eclipse og KDevelop er QDevelop et forholdsvis enkelt IDE, dedikert til utvikling av Qt-applikasjoner. Det viste seg veldig greit å arbeide med og har autofullføring av kode, fargekoding og innebygget støtte for debugging med GDB [26]. GDB er et feilsøkingsverktøy som blant annet lar en se på variabler og rekkefølge av funksjonskall under kjøring. Sammen med Valgrind [27] (som brukes til å finne minnelekkasjer) gjorde dette feilsøkingsprosessen mye enklere, og viste seg nyttig ved flere anledninger.

Med tanke på videreutvikling kan dokumentasjon være nyttig for å forenkle denne prosessen. Doxygen [28] er et verktøy for dokumentasjon av kildekode for språk som Java, C++, Fortran, PHP m.m. Ved å angi kommentarer i kildekoden med spesiell syntaks vil Doxygen automatisk produsere kildekode i form av HTML-sider, PDF eller Latex dokumenter.

For versjonskontroll og sikkerhetskopiering har det blitt benyttet Subversion [29]. Et slikt verktøy har kanskje størst potensial når man er flere som jobber på den samme kodebasen, men det har likevel vært nyttig for å holde orden på endringer da utviklingsarbeidet har foregått på to datamaskiner.

For kommunikasjon mellom objekter har Qt en innebygget funksjon kalt signaler og slotter. Når en endring skjer i et objekt, har andre objekter ofte behov for å vite om denne endringen slik at de kan utføre en passende handling. Man kan da koble et signal fra et objekt til en slot i et annet objekt. Ved en endring i førstnevnte objekt vil det da sendes ut et signal (event) til det andre, lyttende objektet, som deretter utfører en handling gjennom en gitt slot (funksjon). På denne måten kan objekter kommunisere uten at de har direkte kjennskap til hverandre. I praksis kan signaler og slotter beskrives som en litt avansert versjon av Observer-pattern [30]. I tillegg til et stort antall forhåndsdefinerte signaler og slotter, er det også mulig å definere egne. Disse programmeres ved bruk av Qt-spesifikk syntaks, som så kompileres til vanlig C++ kode gjennom en MOC (Meta-Object-Compiler) [31] .

5 Løsningsoppbygging

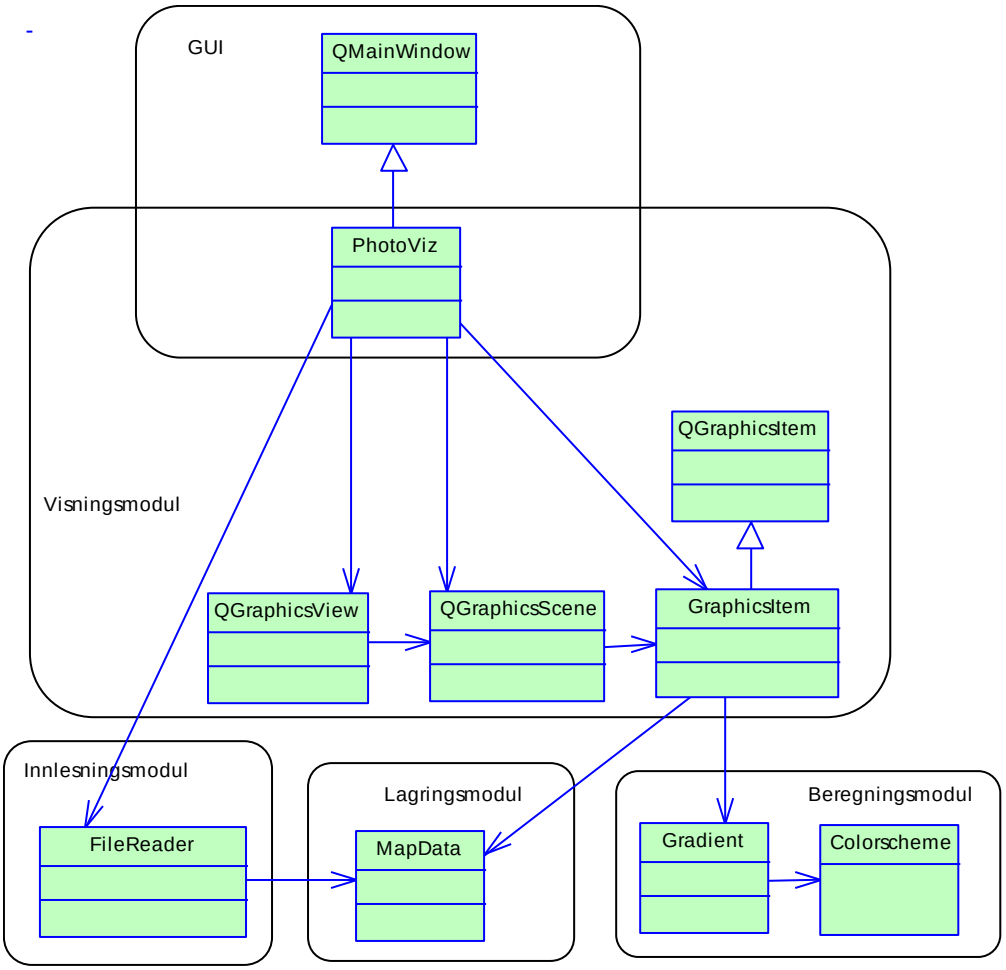
Utviklingen av applikasjonen har vært en kontinuerlig prosess hvor ny funksjonalitet har blitt lagt til litt etter litt, for så å bli forbedret gjennom refaktoreringer underveis. For å illustrere hvordan arbeidet har forløpt vil dette kapittelet ta for seg løsningen etter at første milepæl, visualisering av maps, var gjennomført. Videre vil den nye løsningen og endringer underveis gjennomgås, før det til slutt blir sett på en mulig fremtidig løsning.

5.1 Delløsning

Delløsningen tar for seg hvordan systemet var strukturert på et forholdsvis tidlig tidspunkt. Designet av systemet ble utviklet med hensyn til forskjellige moduler, som har ansvar for ulike deler. Hensikten med dette var å gjøre det enklere å modifisere og bytte ut de forskjellige delene på et senere tidspunkt. Dermed blir vedlikehold og videreutvikling blir enklere. Med hensyn til dette ble systemet delt inn i:

- GUI
- Visningsmodul
- Innlesningsmodul
- Lagringsmodul
- Beregningsmodul

I Figur 5.1 kan man se en oversikt over denne løsningen. Av hensyn til at flere av klassene inneholder et stort antall funksjoner og variabler er disse utelatt fra denne oversikten for å gjøre den mer ryddig.



Figur 5.1: Oversikt over delløsning.

I arbeidet med denne delløsningen ble det fokusert på å lage en visualiseringsløsning som egnet seg både for visning av maps og bilder. Da høres det kanskje rart ut at det kun er implementert støtte for visning av maps. Intensjonen bak dette var å forenkle systemet og skape en generell struktur for de forskjellige delene, med hovedfokus på den visualiseringsmessige delen. Ved å se bort i fra ting som ikke var nødvendig for det rent visualiseringsmessige, ble det lettere å konsentrere seg om å skape en struktur for visualisering av maps som videre kunne gjenbrukes på visualisering av bilder. Det er også verdt å merke seg at denne delløsningen, av grunnene nevnt over, ikke er integrert mot databasen.

5.1.1 Visnings- og lagringsmodul

Som konsept kan visningsmodulen betegnes på som en grafisk vegg hvor man kan henge opp mange grafiske bilder. Ved å gå nærmere veggen kan man studere de enkelte elementene i detalj, mens om man beveger seg bort fra veggen får man en enkel oversikt over alle elementene.

Innad består visningsmodulen av tre forskjellige klasser, hvorav klassen ”GraphicsItem” (heretter kalt visningsklassen) står for den grafiske representasjonen av dataene på skjerm. Denne arver fra en standardklasse for visning av grafiske elementer som kan vise grafikk blant annet gjennom Qt sitt interne bildeobjekt. Ett eller flere av objekter fra visningsklassen legges til i en scene. Scenen blir så vist på skjerm av et lerret (view) som kan zoomes inn og ut. Med dataene representert i lagringsmodulen som input, omformer visningsklassen disse til et bildeobjekt som så tegnes på skjerm. Dette gjør den ved å benytte seg av en beregningsmodul som står for beregning av farger, basert på et gitt fargeskjema. Dette vil forklares nærmere i 5.1.4. På dette tidspunktet var denne delen hardkodet inn, slik at man bare kunne benytte seg av ett fargeskjema.

Innad i ”MapData” lagres dataene i en matrise. Med hensyn til videre arbeid med visualisering av bilder, var tanken at lagringsmodulen og de tre klassene i visningsmodulen kunne benyttes videre til denne oppgaven. Alternativene her var å bruke ”MapData” klassen som den var, forandre den til å støtte templates slik at man kan benytte typer som parameter (heltall for bilder og flyttall for maps) eller å ta i bruk abstraksjon og lage en baseklasse som både ”MapData” og en ny klasse for bildedata arver fra. For sistnevnte fører dette til at visningsklassen kan benyttes både på bilder og maps uten at den må forholde seg til en bestemt implementasjon av de lagrede dataene.

5.1.2 GUI

Applikasjonen har fått navnet ”PhotoViz”. Dette er også navnet på klassen for hovedvinduet (heretter kalt applikasjonsklassen), som hovedsakelig består av menylinjer, en fildialog og en visningsmodul for å vise maps. I tillegg inneholder den en stor andel styringslogikk, og er sådan ikke i henhold til SRP (Single-Responsibility Principle) [32]. SRP omtales også som kohesjon, og er et designprinsipp som går ut på at en klasse ikke skal ha ansvar for mer enn en ting. Et ansvar er en grunn til forandring, og en klasse skal bare ha en grunn til å forandre seg. Slik det er nå kan en endring i visningsmodulen eller innlesningsklassen også føre til endringer i applikasjonsklassen. Dermed kan det bli vanskeligere å gjenbruke og vedlikeholde de forskjellige delene av systemet.

Styringslogikken i denne klassen innbefatter ting som input og kommunikasjon med brukeren, zooming av lerretet og kommunikasjon mellom de forskjellige delene av systemet. En del av denne kommunikasjonen innebærer bruk av egendefinerte signaler og slotter for å knytte sammen data i lagringsmodulen med visningsmodulen. Etter hvert som nye data blir lest inn blir det sendt ut signaler slik at lerretet oppdateres og viser de nye dataene automatisk. Fordelen med dette er at objekter i de forskjellige modulene kommuniserer uten å ha direkte kjennskap til hverandre, og man får dermed et løsere koblet system. Likevel, et problem i denne løsningen er at for mye funksjonalitet er plassert i applikasjonsklassen. I lys av dette bør

applikasjonsklassen derfor splittes opp i flere deler slik at man i større grad unngår at den har ansvar for flere ting.

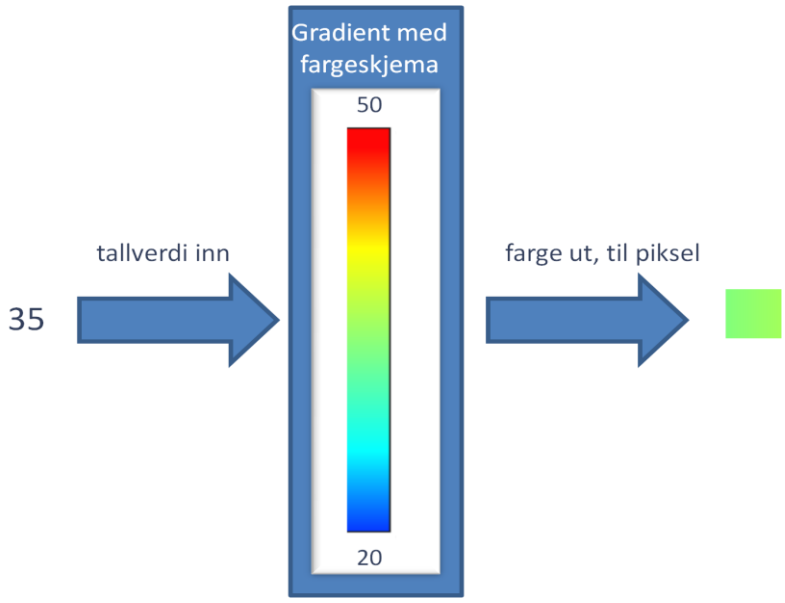
5.1.3 Innlesningsmodulen

Ved innlesning av mapfiler blir det fra fildialogen i applikasjonsklassen sendt filstier til et objekt av filinnlesningsklassen, som prosesserer disse og lager ”MapData” objekter av matrisene i filene. Dette innebærer også en oppskalering og rekonstruksjon av de innleste dataene ved bruk av bilineær interpolering. På den måten får man en størrelse som egner seg bedre for visning på skjerm. For bilder vil en slik oppskalering ikke være nødvendig av den enkle grunn at de er store nok til å vises i sin originale størrelse. Vel og merke er det ikke helt hensiktmessig å foreta interpolering av maps i innlesningsklassen, da dette bryter med SRP. I det videre arbeid bør denne delen derfor trekkes ut fra innlesningsklassen.

En antakelse som ble gjort i sammenheng med innlesning er at formatet på mapfilene ikke vil endre seg i fremtiden. Selv om det er lite sannsynlig at en slik endring vil skje, er det viktig å nevne fordi dette kan føre til at innlesningsklassen brekker sammen. I verste fall kan dette få applikasjonen til å krasje. Av den grunn ble det lagt inn feilsjekker for å unngå dette, men noen garanti for at alle tenkelige situasjoner dekkes kan ikke gis. Da detaljene for hvordan filer skulle lagres i databasen ennå ikke var kjent, ble innlesningen delt opp i mindre steg. Meningen med dette var å kunne gjenbruke alle, eller deler av stegene i integrasjonen mot databasen på et senere tidspunkt.

5.1.4 Beregningsmodulen

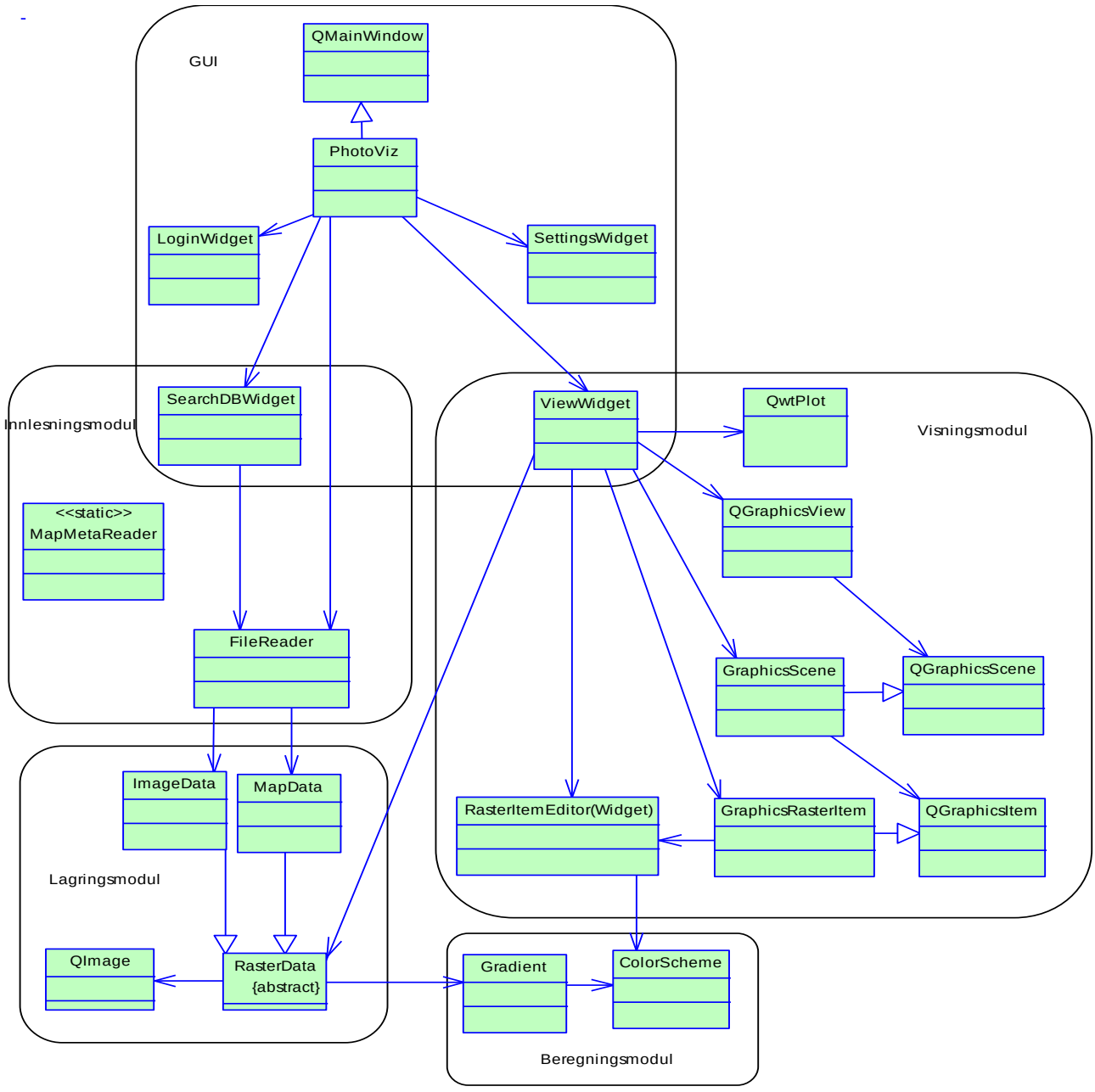
Et digitalt bilde og et map kan sammenlignes på den måten at de begge kan betegnes som en matrise, der hvert tall i matrisen indikerer fargen til en piksel. I digitale bilder er disse tallene innenfor et forhåndsdefinert spekter, bestemt av fargedybden. For eksempel, i et 16-bits enkanals bilde går tallene fra 0 til 65535. Til forskjell fra dette inneholder de ulike mapfilene som nevnt forskjellige spektre av tall, definert av hvilken parameter en fil representerer. Man må derfor for maps, som nevnt ovenfor, først gjøre en omdanning av tallene i matrisen slik at det er mulig å representere mapet grafisk gjennom et bildeobjekt. Til denne omformingen ble det laget en klasse ”Gradient” som kan inneholde et forhåndsdefinert spekter av tall. Denne benytter seg av et fargeskjema, ”Colorscheme”, som kan inneholde en eller flere farger. Basert på spekteret av tall og fargeskjemaet blir det da foretatt en lineærinterpolasjon, slik at en for hver tallverdi inn, kan beregne en farge ut. Et eksempel på dette vises i Figur 5.2.



Figur 5.2: Forenklet oversikt over fargeleggingsmodulen.

5.2 Ny løsning

Fra delløsningen har systemet i den nye løsningen gjennomgått en del endringer. I tillegg er systemet utvidet gjennom flere nye elementer. Av de forskjellige modulene består bare beregningsmodulen som den var i delløsningen. Av nye elementer kan man blant annet trekke frem visning av bilder, integrasjon mot database, profilvisning og ny funksjonalitet for brukerinput mot visningsmodulen. Som vi kan se i Figur 5.3 er noen av klassene inkludert i flere moduler, og vil derfor omtales flere steder i dette delkapittelet.



Figur 5.3: Oversikt over ny løsning.

5.2.1 GUI og innlesningsmodul

Fra delløsningen er GUIet utvidet en god del, og består nå av totalt fire klasser. På generell basis representerer hver klasse et vindu, som sammen med klassene i visningsmodulen behandler all input fra bruker. I tillegg til dette finnes en GUI klasse, kalt ”RasterItemEditor”, plassert i visningsmodulen. Denne består av en liten dialog som lar brukeren bytte grenseverdier og fargeskjema for gradienten. På den måten kan visningen av et bilde eller map forandres slik at man i et bilde kan få frem detaljer som ellers er vanskelig å få øye på. Så hvorfor er denne klassen plassert i visningsmodulen? Som forklart i 5.2.4 håndteres det meste av brukerinputen i visningsmodulen direkte av de grafiske klassene (visnings, lerret og sceneklassen). Selv om det ville vært mulig å håndtere også bytting av grenseverdier og fargeskjema gjennom de grafiske klassene, hadde man måttet lage funksjonalitet for dette fra bunnen av. ”RasterItemEditor” er derfor et unntak der det ble valgt å benytte et mer eller mindre ferdig GUI-element fremfor å lage noe som fungerer bra på nytt.

For å la brukeren endre og håndtere applikasjonsinnstillinger ble det laget en klasse, kalt ”SettingsWidget”. I utgangspunktet var tanken å gjøre det mulig å endre en stor andel innstillinger hva angikk både det visualiserings- og ytelsesmessige. Visualiseringsinnstillingene er ting som standardinnstillinger for gradient og fargeskjema til bilder, og ikke minst maps. De ulike mapfilene inneholder som nevnt forskjellige spektre av tall, og ved sammenligning av to maps av samme parametertype er det viktig at disse tegnes opp med like innstillinger. Sammen med ytelsesinnstillingene er disse lagret i en fil som leses etter behov. Dessverre ble det grunnet dårlig tid mot slutten bare tid til kun å gjøre de ytelsesmessige innstillingene tilgjengelig i brukergrensesnittet.

I motsetning til i delløsningen hvor bare ett hardkodet fargeskjema var tilgjengelig, kan brukeren i den nye løsningen velge mellom mange fargeskjema. Disse er lagret som en rekke av RGB verdier i egne filer, en for hvert fargeskjema. Grunnet fargeskjemaenes enkle struktur ble det valgt å lagre dataene i tekstfiler fremfor å benytte seg av et metaspråk som XML. I Eksempel 5.1 kan man se et eksempel på en fargeskjemafil.

```
127     0       0
255     0       0
255     127     0
255     255     0
127     255     127
0       255     255
0       127     255
0       0       255
```

Eksempel 5.1: Fargeskjemafil med åtte farger.

Nye fargeskjema kan opprettes ved å definere RGB verdier i en ny tekstfil som lagres i en mappe for fargeskjema, ett nivå under den kjørbare applikasjonsfilen. Fargeskjemaene leses inn gjennom funksjoner i klassen ”MapMetaReader”, hvor de så lagres i en statisk tabell. Denne fremgangsmåten ble brukt for å unngå å lagre alle fargeskjema for hvert eneste bilde eller map som vises. I tillegg inneholder ”MapMetaReader” en rekke mindre hjelpefunksjoner som blant annet tar seg av innlesning av standard gradientverdier, måleenhet og visningstekst for de forskjellige maps. Disse hjelpeklassene benyttes av flere steder av de andre klassene i systemet og var årsaken til at ”MapMetaReader” ble laget som en statisk klasse.

Brukergrensesnittet mot databasen består av innloggingsvindu, ”LoginWidget”, og et søkeskjema representert ved databaseklassen ”SearchDBWidget”. Sistnevnte håndterer åpning av filer fra databasen, søk og presenterer eventuelle treff tilbake til brukeren. En liten raritet i denne løsningen er at databaseklassen klassen benytter seg av filinnlesningsklassen ved åpning av bilder og maps fra databasen. Dette er av den enkle grunn at når en fil hentes fra databasen, så følger en rekke operasjoner som er likt for åpning av filer fra database og filsystem. Ved å benytte funksjonaliteten i filinnlesningsklassen unngår man dermed duplikatkode.

5.2.2 Lagringsmodulen

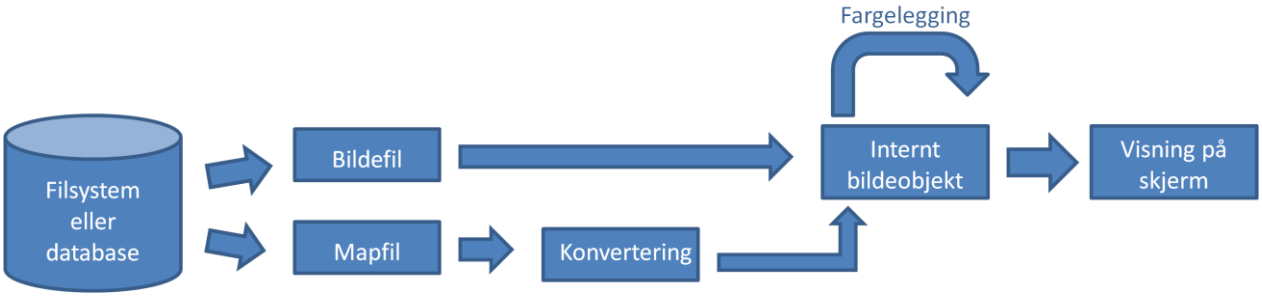
I delløsningen ble det nevnt at bruk av abstraksjon, eller et interface, var et alternativ ved visning av bilder. I denne løsningen har dette blitt gjort ved å lage en abstrakt klasse, kalt ”RasterData”, som fungerer som et interface. ”MapData” og en ny klasse for bildedata, ”ImageData” arver fra denne og til lagring av data benytter de seg begge av Qt sitt interne bildeobjekt, ”QImage”. I Qt kan ”QImage” (heretter kalt bildeobjekt) brukes til å lagre alle støttede typer bilder. Med tanke på systemet er fordelen med dette er at visningsklassen blir helt uavhengig av lagringsmodulen, og kun trenger å forholde seg til visning av bildetyper som er støttet av Qt. Dette inkluderer TIFF, GIF, BMP, JPEG og PNG for å nevne noen. Om Balter Medical senere skulle forandre bildene fra TIFF til et annet bildeformat vil dette trolig ikke få noen konsekvenser for applikasjonen.

Til tross for at ”MapData” i sin opprinnelige form også kunne blitt benyttet for lagring av bildedata ville dette medført økt kompleksitet. Grunnen til dette er at et bilde, fra øyeblikket det leses inn, representeres gjennom et bildeobjekt slik at det kan vises direkte på skjerm. Ved å benytte ”MapData” også for bilder, må et bilde først leses inn som et bildeobjekt, konverteres piksel for piksel til et ”MapData” objekt, for så å konverteres tilbake til et bildeobjekt før visning på skjerm. Med andre ord svært tungvint, spesielt med tanke på ytelse.

En annen endring fra delløsningen er at beregningsmodulen er flyttet fra visningsklassen, til klassene i lagringsmodulen. Der visningsklassen tidligere også hadde ansvar for å bygge opp de visualiserte dataene gjennom fargelegging, har den nå kun ansvar for å vise disse på skjerm. Det kan selvsagt diskuteres om det er bedre at ansvaret for oppbygging av visualiserte data flyttes

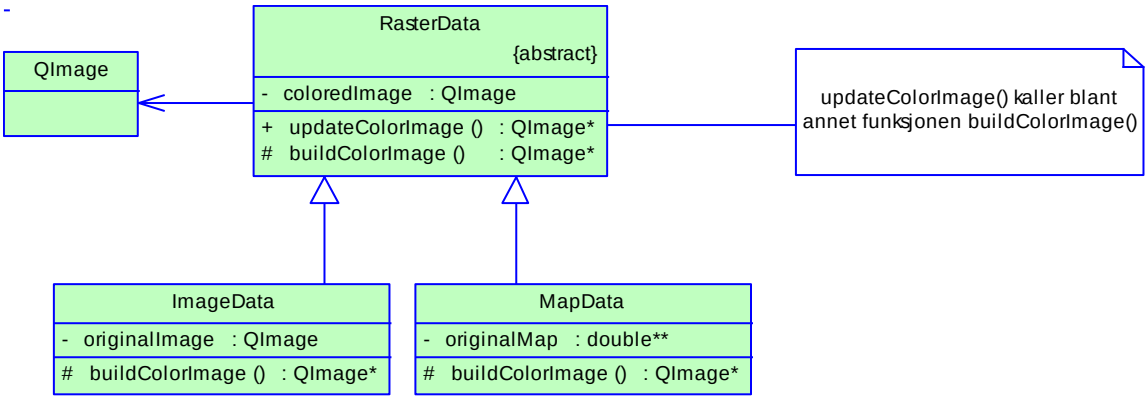
over på klassene som strengt tatt bare burde lagre data. Man kunne trukket denne delen ut i en egen klasse, men dette i betraktning føltes det som mer kompliserende enn forenklende da fargeleggingen er en mindre operasjon.

I motsetning til bilder som kan fargelegges direkte, må maps igjennom en tilleggsoperasjon hvor matrisen konverteres til et bildeobjekt før fargelegging. En del av denne konverteringen inkluderer bilineær interpolering av dataene. Denne biten ble flyttet fra filinnlesningsklassen til ”MapData” for å la filinnlesningsklassen kun ha ansvar for å håndtere innlesning. I tillegg, ved å plassere denne funksjonaliteten i ”MapData”, kan man enklere skalere dataene som man vil. Dette skaper større frihet i visningen av maps, slik at man ikke må forholde seg til en fast størrelse som bestemmes ved innlesning. I Figur 5.4 kan man se en forenkelt oversikt over prosessen fra innlesning til visning på skjerm.



Figur 5.4: Forenklet oversikt av innlesning av filer til visning på skjerm.

Når en bruker forandrer fargeskjema eller gradientverdier er det viktig at den nye fargeleggingen skjer med hensyn på de originale dataene, og ikke på allerede visualiserte data. Klassene i lagringsmodulen består derfor av to representasjoner av dataene, en som fargelegges og vises, og en annen som inneholder de originale dataene. Klassen ”ImageData” inneholder sådan to bildeobjekt, mens ”MapData” inneholder et bildeobjekt for visning, samt en matrise som de originale dataene lagres i. Ettersom maps inneholder flyttall, egner ikke bildeobjekt seg til lagring av de originale dataene. En kan selvsagt argumentere med at disse kan konverteres ved innlesning, men dette vil i noen tilfeller føre til tap av presisjon. Som et resultat av dette forløper fargeleggingen noe forskjellig i de to klassene. For bilder må man iterere gjennom orignalverdiene i et bildeobjekt, mens for maps skjer denne itereringen over en matrise. Denne delen er derfor implementert som template method pattern [18], hvor skjelettet til fargeleggingsrutinen er definert i baseklassen. Videre inneholder denne rutinen flere steg, der et av stegene er implementert forskjellig i ”ImageData” og ”MapData”. Delene av fargeleggingsrutinen som er forskjellig blir med andre ord definert i de respektive underklassene. I Figur 5.5 kan man se en oversikt av dette.



Figur 5.5: Template method pattern.

5.2.3 Visningsmodulen

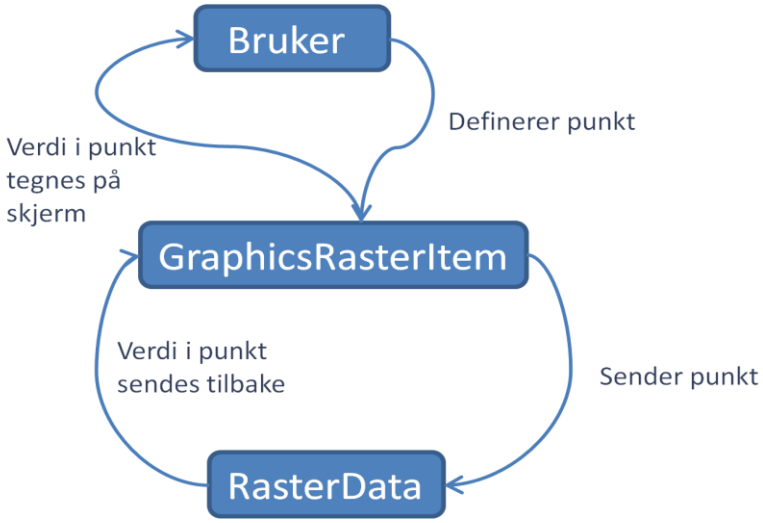
Nytt i denne løsningen er at hele visningsmodulen er flyttet ut fra applikasjonsklassen og inn i en ny klasse, kalt ”ViewWidget”. Denne fungerer som en ramme rundt de andre klassene i visningsmodulen. At det ble valgt å flytte visningsmodulen er det flere grunner til. Først og fremst blir en stor andel logikk trukket ut fra applikasjonsklassen, slik at den ikke blir for stor og uoversiktlig. I tillegg blir det mulig å ha flere visningsvinduer åpne på en gang, med mange bilder eller maps i hvert vindu. For eksempel, kan man ha kalibrerte bilder fra en lesjon i et vindu, og maps fra samme lesjonen i et annet. Som en tilføyelse til visningsmodulen inneholder ”ViewWidget” også funksjonalitet for profilvisning. Dette skaper også en del ekstra styringslogikk som går på kommunikasjonen mellom input fra bruker, grafiske elementer og profilvisningen.

5.2.4 Styringslogikk og kommunikasjon mellom klasser

Tidligere begrenset brukerinput seg til åpning av maps gjennom en fildialog, og zooming av lerretet. Utenom de nye GUI elementene kan brukeren i denne løsningen benytte seg av punktsammenligning og profilvisning i analysen av bilder og maps. Punktsammenligning håndteres direkte av visningsklassen og lar brukeren definere et punkt i et map, hvor så alle maps i lerretet viser sin verdi i dette punktet. Dette gjør det ikke bare enklere å sammenligne to maps fra samme lesjon, men kan også brukes for å trekke konklusjoner fra et sett med maps fra en lesjon. Hvis det, for henholdsvis mapsene blodinnhold, blodoksygennivå og keratin, i samme punkt er høyt blodinnhold, lavt blodoksygennivå og lite keratin kan dette være et tegn på kreft.

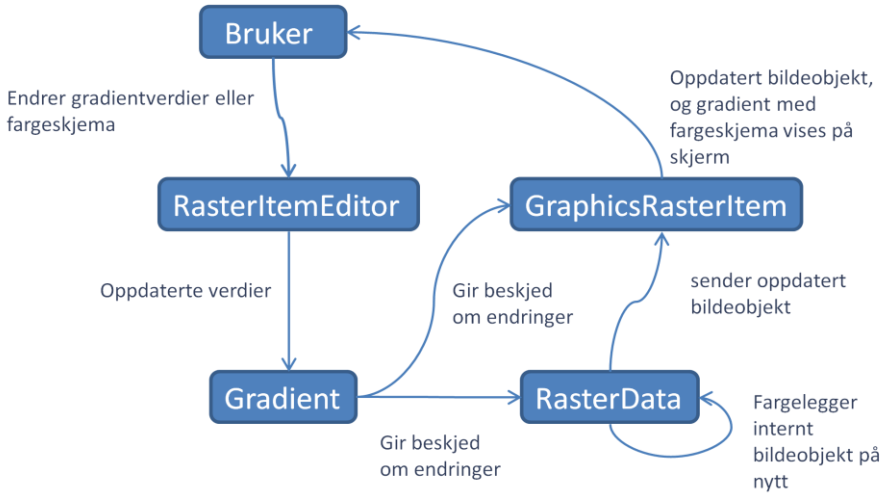
Når brukeren definerer et punkt i et bilde eller map skjer kommunikasjonen videre i systemet utelukkende gjennom signaler og slotter. Posisjonen til punktet registreres av et visningsobjekt (objekt av visningsklassen), som så videresender denne posisjonen til alle andre visningsobjekt i lerretet. Herfra sender visningsobjektene posisjonen til sitt respektive ”RasterData” objekt. Verdien ved dette punktet hentes så ut, og sendes tilbake til visningsobjektet hvor den tegnes på skjerm. En oversikt over dette kan sees i Figur 5.6. For å gjøre det mulig å la de forskjellige

visningsobjektene kommunisere med signaler og slotter var det i visningsmodulen nødvendig å lage en underklasse av Qt sin standard sceneklasse. Det kan dog nevnes at omfanget av ny funksjonalitet som er lagt til underklassen er av svært liten art, men som sådan var nødvendig.



Figur 5.6: Kommunikasjonen mellom visningsmodul og lagrede data ved punktsammenligning.

Ved profilvisning er denne kommunikasjon relativt lik. Punktet definerer da en linje i bildet, enten horisontal eller vertikal, som sendes ned og hentes ut fra et objekt i lagringsmodulen. Forskjellen her er at dataene i stedet sendes tilbake til profilvisningsvinduet, representert ved klassen ”QwtPlot”. I Figur 5.7 under vises kommunikasjonen mellom klassene ved endring av gradientverdier eller fargeskjema.



Figur 5.7: Kommunikasjonen mellom visningsmodul, fargeleggingsmodul og lagrede data ved endring av gradientverdier eller fargeskjema.

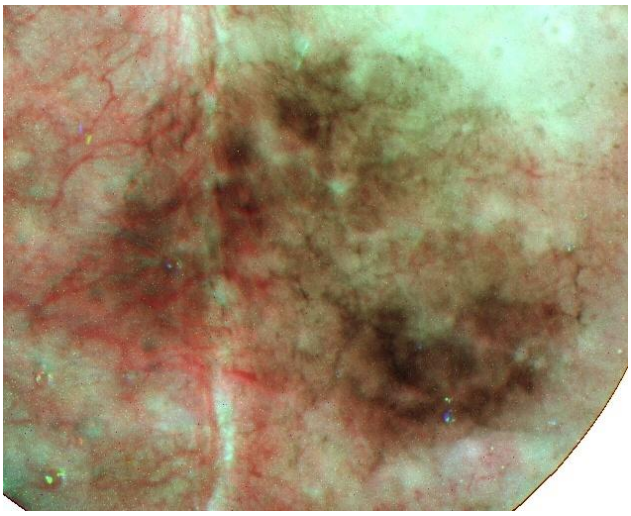
Så langt har bare systemets oppførsel basert brukerinput gjennom klassene i visningsmodulen vært omtalt. I tillegg til dette kommer brukerinput fra de forskjellige GUI elementene. Dette

håndteres stort sett i de respektive GUI klassene, ofte gjennom bruk av signaler og slotter. I de tilfellene hvor signaler og slotter ikke egner seg for direkte kommunikasjon på tvers av klasser, blir dette håndtert av applikasjonsklassen. Ved åpning av bilder fra database eller filsystem har for eksempel ikke innlesningsklassene mulighet til å vite hvilket visningsvindu bildene skal åpnes i. Kommunikasjonen må derfor gå igjennom applikasjonsklassen som bestemmer hvilket vindu de innleste dataene skal vises i.

5.3 Fremtidig løsning

Med hensyn til videreutvikling er applikasjonen forsøkt designet slik at det skal være enkelt å utvide systemet i ettertid. Dette delkapittelet vil ta for seg hvorvidt dette har vært vellykket. I tillegg vil det bli skissert hvordan systemet kan struktureres med bakgrunn i en ny type visning av data som det ble fremmet ønske om underveis i arbeidet.

Ønsket, som var RGB-vektet visning av bilder, baserer seg på at man i en tagning gjør en vekting av bildene som er tatt med bølgelengder nærliggende rødt, grønt og blått lys. Dette gjøres ved å ta dataene i tre råbilder, og legge de inn i henholdsvis rød, grønn og blå kanal i et nytt RGB-bilde. Egentlig er ikke RGB-vekting et godt ord, for vektingen gjelder en generell kombinasjon av tre bilder med tre bestemte bølgelengder. Bildene kan for eksempel også være tatt med oransje, gult og gulgrønt lys. Der et rødt, grønt og blått vektet bilde vil vise en svært nær en naturlig fargegjengivelse, vil et oransje, gul og gulgrønt bilde fremheve blodårer som i svarthvitt vil være nærmest usynlig. Et eksempel på et slikt bilde er vist i Figur 5.8 hvor man tydelig kan se blodårer i venstre del av bildet. Andre kombinasjoner vil igjen kunne fremheve andre ting.

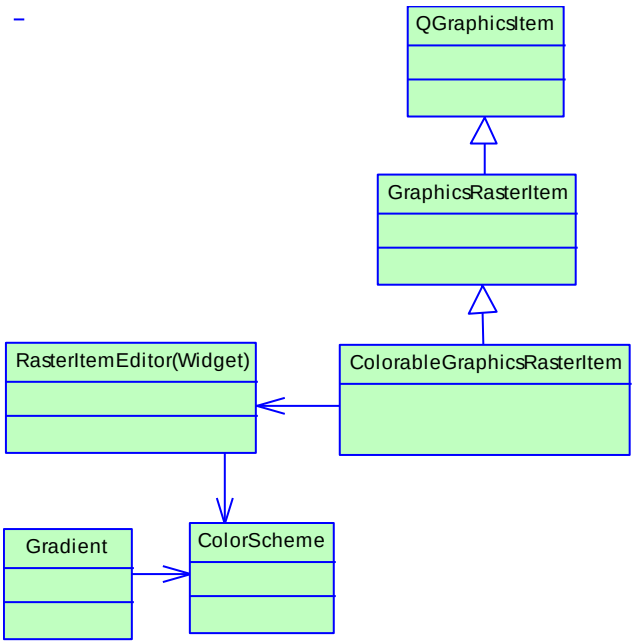


Figur 5.8: Et vektet bilde som fremhever blodårer.

Det som er verdt å merke seg ved et slikt bilde er at fargene ikke beregnes med en gradient og et fargeskjema. Slik som systemet er bygget opp nå må en vekting skje etter at tre bilder er lest

inn, hvor så en fargelegging med gradient og fargeskjema om ønskelig kan skje i etterkant av dette igjen.

At det er mulig å bruke beregningsmodulen som den er også for vektede bilder er vel og bra, men det viser likevel en svakhet i designet. Hva om systemet utvides med en ny type data ikke har behov for beregning av farger? Fra Figur 5.3 kan man se at det er en sterk kobling mellom beregningsmodulen og visningsmodulen. Av den grunn er visningsklassen låst til å benytte den bestemte dialogklassen. Samtidig henger dialogklassen nært sammen med visningen av fargelagte bilder og maps, og en avhengighet mellom disse klassene vil i så måte være naturlig. En mulig løsning på dette kan derfor være å trekke kommunikasjonen med dialogen ut av visningsklassen og ned i en underklasse som arver fra den nåværende visningsklassen. Et eksempel på dette kan sees i Figur 5.9.

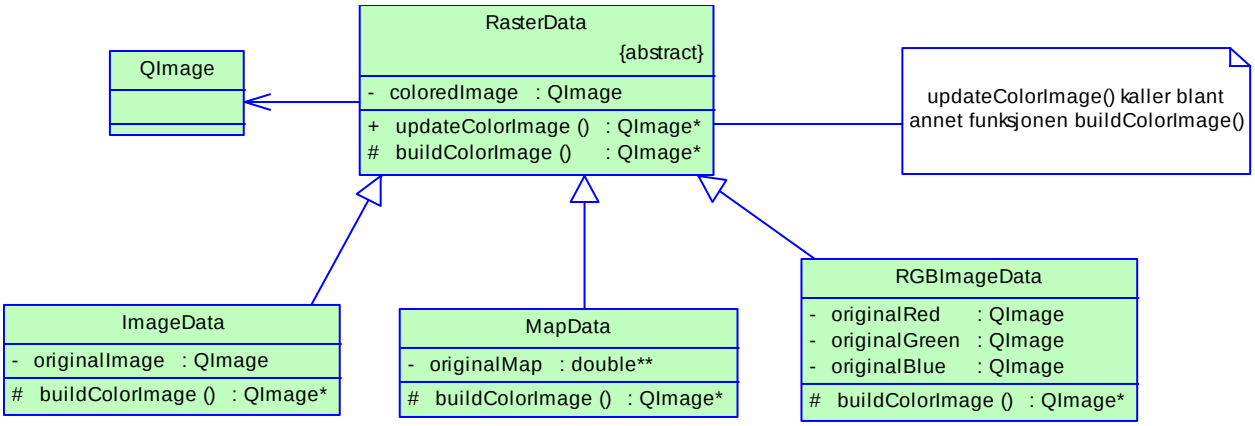


Figur 5.9: Forslag til å skille visning av fargelagte data, og data som skal fargelegges.

Riktignok er problemstillingen over hypotetisk, og det vil være vanskelig å ta høyde for alle mulige nye typer data som systemet kan utvides med. Gjør man det kan man i verste fall risikere å ende opp med et system som er unødvendig komplekst.

I forhold til vektede bilder var en tanke at ”Gradient” klassen kunne abstraheres ut slik at denne og en ny fargeberegner som foretar vekting kunne benyttet seg av et felles interface. Problemet er at det er vanskelig å finne fellestrekk ved de to fargeberegningene som kan trekkes ut i et slikt interface. Det eneste som er felles er at vekting og fargeberegning med gradient begge produserer farger basert på input. Men i dette tilfellet er det input som er helt forskjellig. En slik løsning vil derfor trolig ikke egne seg her.

En mulig løsning på dette blir i stedet å opprette en ny underklasse av ”RasterData”, hvor så vektingen blir en rutine i denne nye klassen. Som nevnt i forrige delkapittel ble det benyttet template method pattern for fargelegging av visingsbildet i de forskjellige klassene i lagringsmodulen. Vektingen kan derfor i den nye klassen implementeres som et eget steg i denne rutinen. En ulempe med dette er at denne klassen dermed får ekstra beregningslogikk utover det som finnes i ”Gradient” klassen. Et designforslag til en slik løsning vises i Figur 5.10. Som vi kan se inneholder den nye klassen tre bilder som brukes til å produsere et vektet bilde.

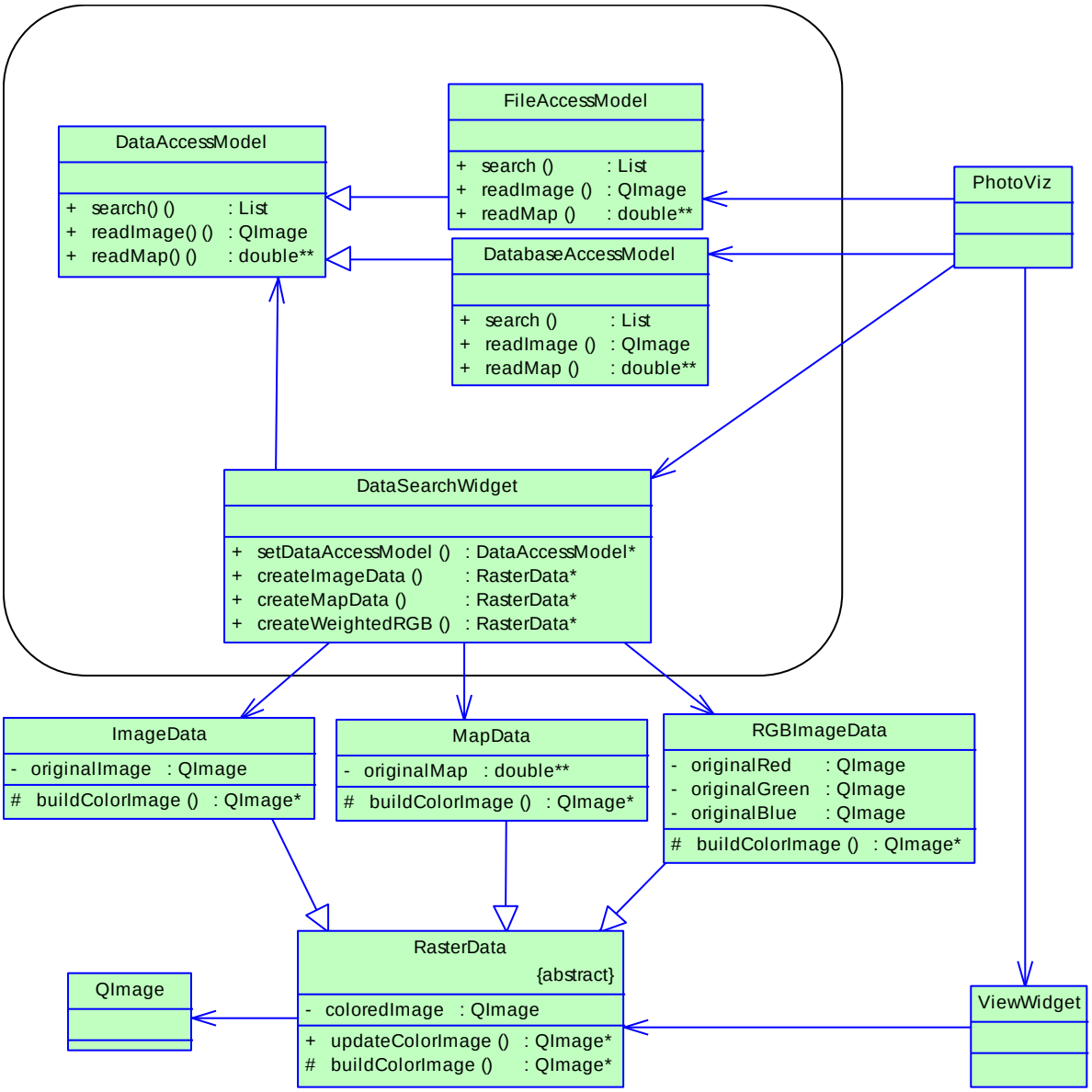


Figur 5.10: Lagringsmodulen - Designforslag til fremtidig implementasjon av vektede bilder.

Videre, for å foreta en vekting må det defineres hvor stor andel (prosent) hvert av de tre bildene skal utgjøre i det vektede bildet. Dette kan for eksempel legges til i visningsinnstillingene som leses fra fil. På sikt bør disse innstillingene også gjøres tilgjengelig for brukeren gjennom å utvide GUI klassen for innstillingsvinduet.

Et annet valg som må tas her er hvilke data som skal returneres ved punktsammenligning og profilvisning. Sannsynligvis vil det spille liten rolle hva man velger, ettersom det ikke vil være spesielt meningsfylt å bruke disse verktøyene på et vektet bilde.

I tillegg må det legges til kode ved innlesning fra filsystem og database, samt valgmuligheter i GUI slik at en bruker kan velge å åpne en RGB-vektet visning av 3 bilder. Det er mange måter å legge til slik funksjonalitet på, men for å gjøre denne delen enklere å utvide i fremtiden er en mulig løsning å lage et felles grensesnitt mot dataene i filsystem og database. Denne løsningen, som vist i Figur 5.11, er noe omfattende og vil ta litt tid å implementere. Som en enklere løsning, men som ikke er like utvidbar, kunne man lagt til nye funksjoner for oppretting av vektede bilder i innlesningsklassen. Utover dette vil innlesningsmodulen bestå slik den er i dag. Videre må søkeskjemaklassen og fildialogen utvides med nye funksjoner, slik at brukeren kan velge å åpne en vektet visning av 3 bilder. Hvorvidt man bør velge en slik løsning eller den som er beskrevet under kan blant annet være avhengig av om hvorvidt det er sannsynlig at applikasjonen utvides ytterligere.



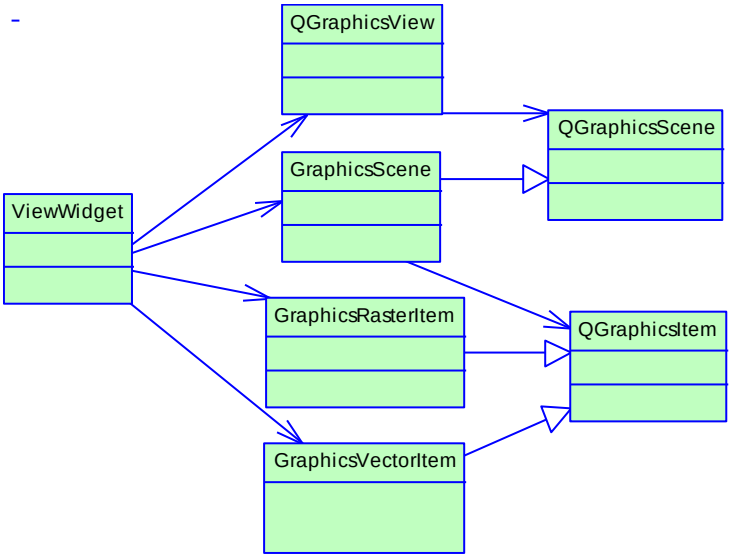
Figur 5.11: Designforslag til fremtidig implementasjon av innlesningsmodulen. De mest essensielle metodene er skissert her. Andre deler av systemet forblir i sin nåværende form.

Som vist i figuren kan en slik løsning bygges opp rundt en abstrakt klasse, ”DataAccessModel”. Denne fungerer som et interface mot innlesningen av data. Denne finnes det to implementasjoner av, en for filsystem og en fra database. Det som ikke er med i figuren er at dataaksessmodellen for filsystem vil trenge en filsystemparser, mens databaseaksessmodellen vil trenge en database tilkobling. I tillegg vil funksjonene i ”DataSearchWidget” (søkeskjemaklassen) for å opprette de forskjellige objekter av klassene lagringsmodulen trenge input i form av et QImage eller en double** tabell.

I denne løsningen trenger ”DataSearchWidget” (søkeskjemaklassen) bare å forholde seg til å vise en ”DataAccessModel”, uten at den selv er klar over hvilken type som faktisk brukes. Dette styres av applikasjonsklassen, om brukeren velger å bruke en dataaksessmodell for filsystem eller database. Det kunne også være aktuelt og trekke opprettingen av lagringsobjekter ut fra søkeskjemaklassen, og inn i egne klasser med factory-logikk. Dermed får søkeskjemaklassen bare ansvar for å vise søkeskjema og behandle input. Det har dog ikke blitt valgt å gjøre dette i figuren over fordi det kanskje vil være å overkomplisere innlesningsmodulen.

Felles for begge implementasjonene av dataaksessmodellen er at de tar imot input (for eksempel saksnummer) gjennom søkeskjemaklassen. Søket resulterer som før i en liste med treff som vises i GUIet, hvor så bilder eller maps kan markeres og åpnes for visning. For RGB-vektet visning vil dette naturligvis kreve at minimum 3 bilder er markert. I kapittel 5.2.1 ble det nevnt at databaseklassen benyttet seg av funksjoner i filinnlesningsklassen. I denne løsningen vil det være mer naturlig å plassere fellesfunksjoner i baseklassen for å unngå duplikatkode.

Tilslutt vil kan det kort nevnes hvordan eventuelle nye grafiske objekter kan legges til i visningsmodulen. Grunnen til dette er at det for eksempel kan bli aktuelt å vise andre ting enn bilder, hvor den nåværende visningsklassen ikke vil være egnet. I Figur 5.12 kan man se et eksempel på dette.



Figur 5.12: Visningsmodulen - fremtidig utvidelse av grafiske visningsklasser.

Figuren viser en ny tenkt klasse for vektorgrafikk. Klassen arver fra Qt sin standard visningsklasse, og man får dermed en god del funksjonalitet gratis. Det som selvsagt må legges til er hvordan man skal tegne vektorgrafikken, samt et par funksjoner som definerer omrisset til et grafisk element ved visning. Når dette er gjort kan objekter av klassen legges til i nåværende

sceneklasse og vises på skjerm. På denne måten kan man i fremtiden enkelt legge til nye måter å vise data på, uten å måtte legge til mer enn en klasse i visningsmodulen.

For å summere opp kapittelet har designet i den nye løsningen definitivt sine svakheter. Samtidig er også deler av systemet godt egnet for utvidelse. Hvor komplisert det vil bli å legge til nye datatyper og visninger av disse, vil i stor grad være avhengig av hva som skal legges til.

6 Implementasjon

Dette kapittelet vil ta for seg en kort gjennomgang av systemets implementasjon og tekniske oppbygging. I tillegg vil det vises eksempler fra den ferdige applikasjonen.

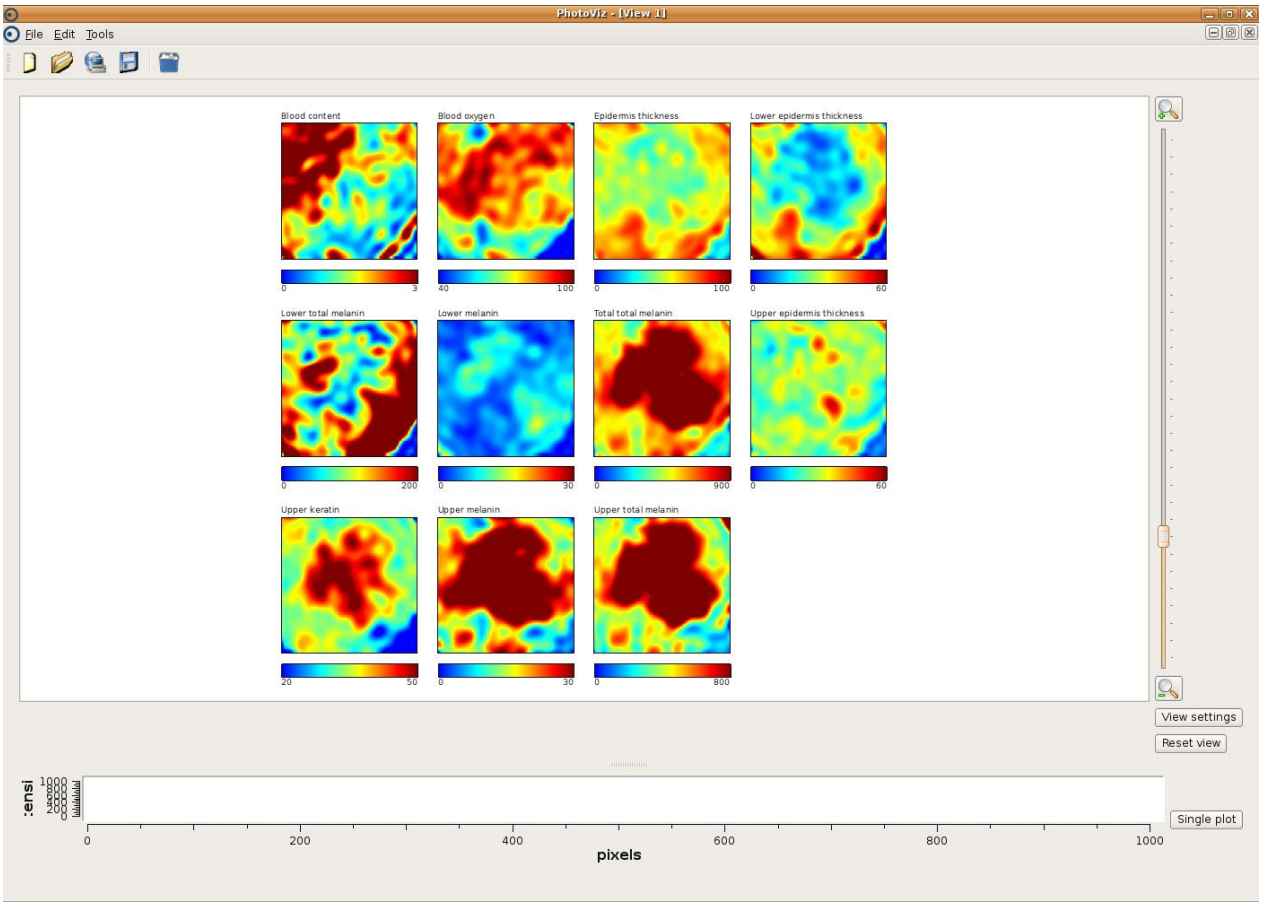
De forskjellige bølgelengder som bildene er tatt med er som nevnt en forretningshemmelighet. Bølgelengdene som vises i dette kapittelet er derfor ikke reelle.

6.1 Åpning av filer

Når applikasjonen er startet kan man i menylinjen enten velge å åpne filer fra filsystem, eller i fra database. Ved åpning fra filsystem må brukeren, ved hjelp av en fildialog, selv lete og finne frem til bildene som skal åpnes. Filene som vises i dialogen er filtrert etter gyldige bilde og map filformat. Til forskjell kan man ved åpning av filer fra database søke seg frem til filene man ønsker, basert på bestemte parametere. Dette vil omtales nærmere i kapittel 6.6. Mens filer leses inn får brukeren tilbakemelding på fremgangen, vist i et lite dialogvindu med en fremdriftssøyle.

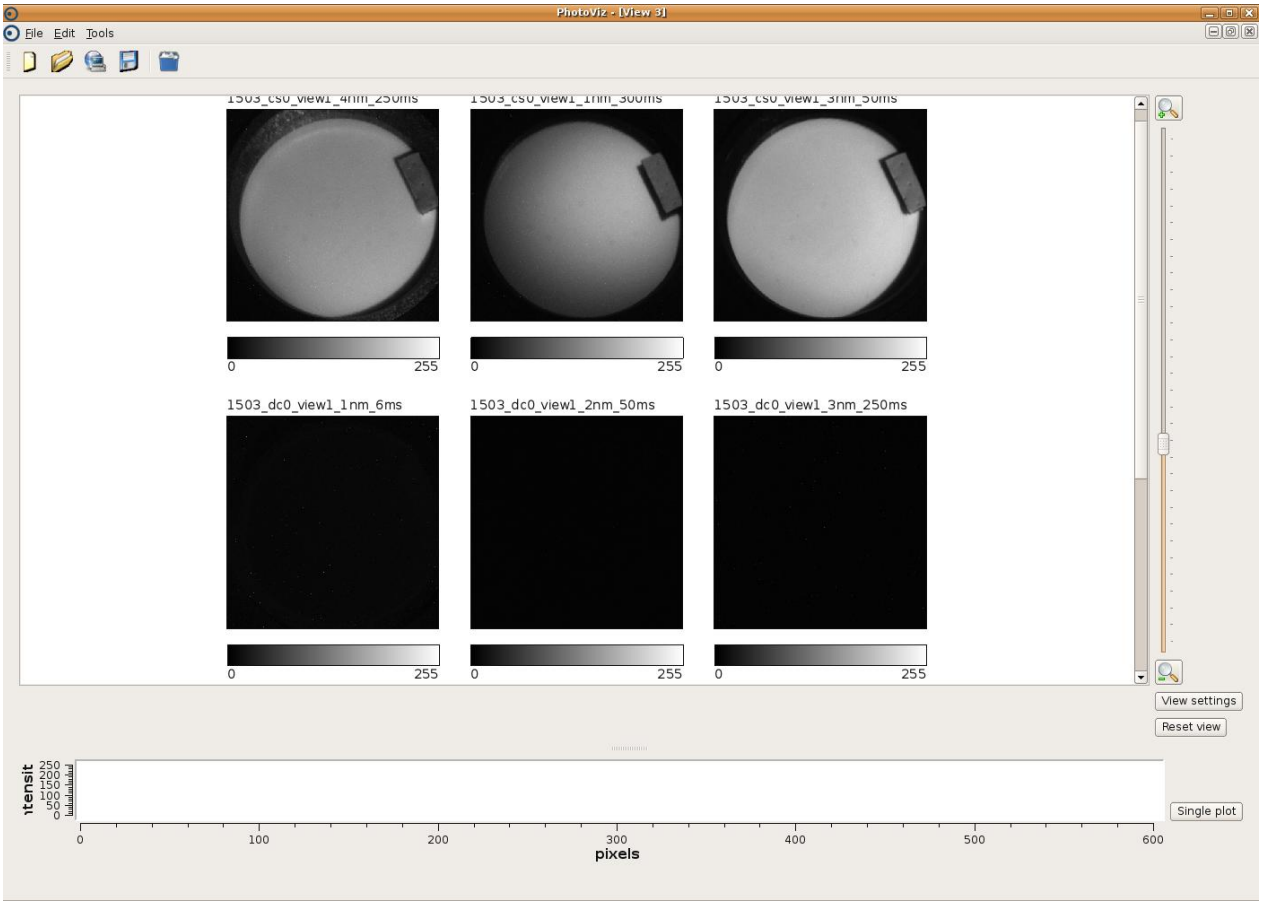
6.2 Visning av bilder og maps

Fra Figur 6.1 kan vi se det grafiske brukergrensesnittet i PhotoViz. Innleste maps tegnes opp på lerretet i visningsvinduet sammen med en grafisk representasjon av gradienten, dens fargeskjema og spekter (grenseverdier). Hvert map har også en overskrift som viser hvilken parameter den representerer. Til høyre kan brukeren benytte seg av en slider og zoome inn og ut til ønsket detaljnivå. Nederst i figuren vises en tom profilvisning. Denne er justerbar i forhold til lerretet og vil omtales nærmere i kapittel 6.4.



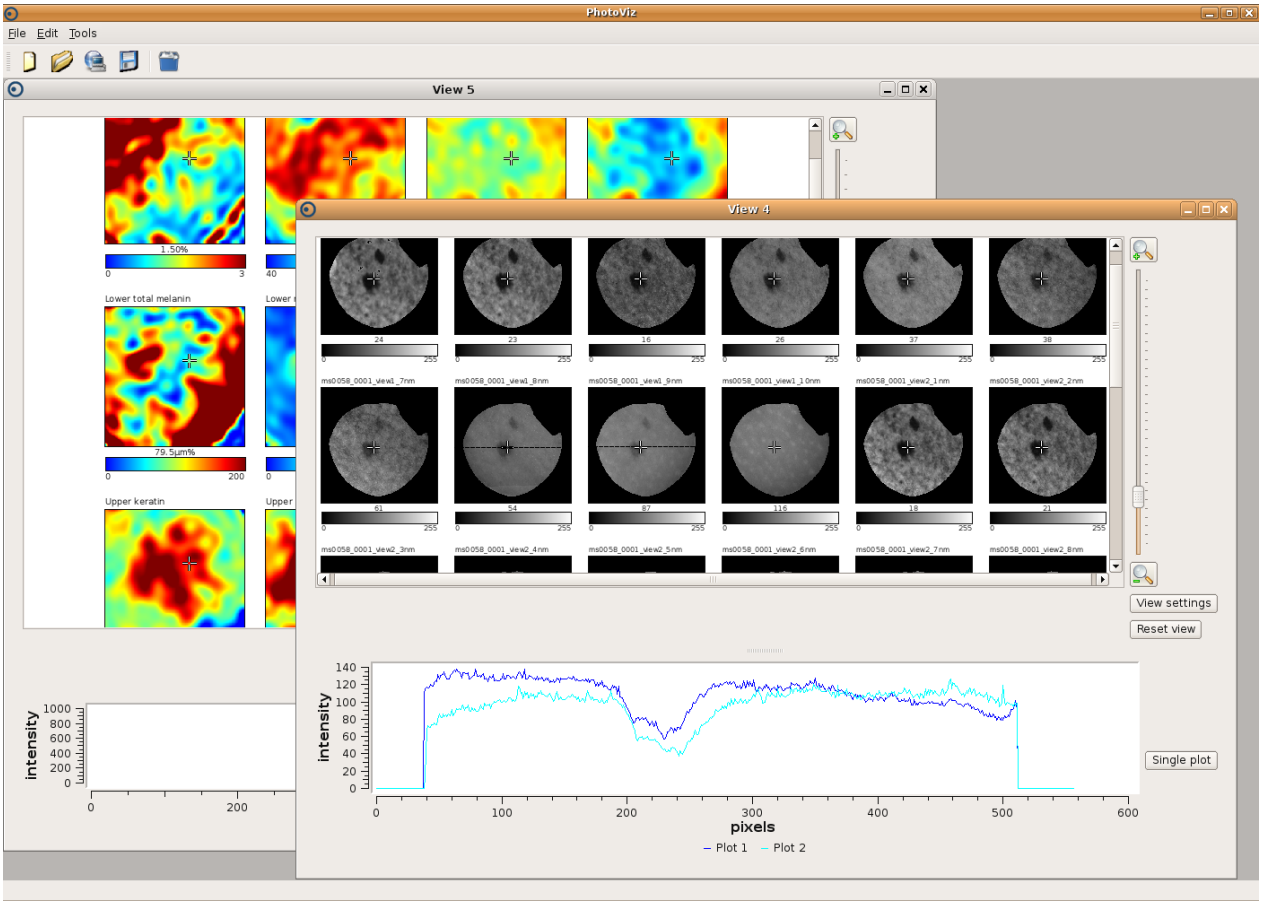
Figur 6.1: PhotoViz - visning av alle maps i en lesjon.

Visning av bilder fortoner seg omtrent helt likt som visningen for maps. Som en liten forskjell har et bilde en overskrift som består av saksnummer, kameranummer og bølgelengde, altså det samme som filnavnet. Fra Figur 6.2 kan vi se tre CS bilder og tre DC bilder. CS bildene er som nevnt tatt mot en helt hvit flate som reflekterer 99% av lys. DC bildene er tatt i et helt mørkt miljø og skal egentlig være helt svarte. Støy kan likevel skimtes som helt hvite piksler.



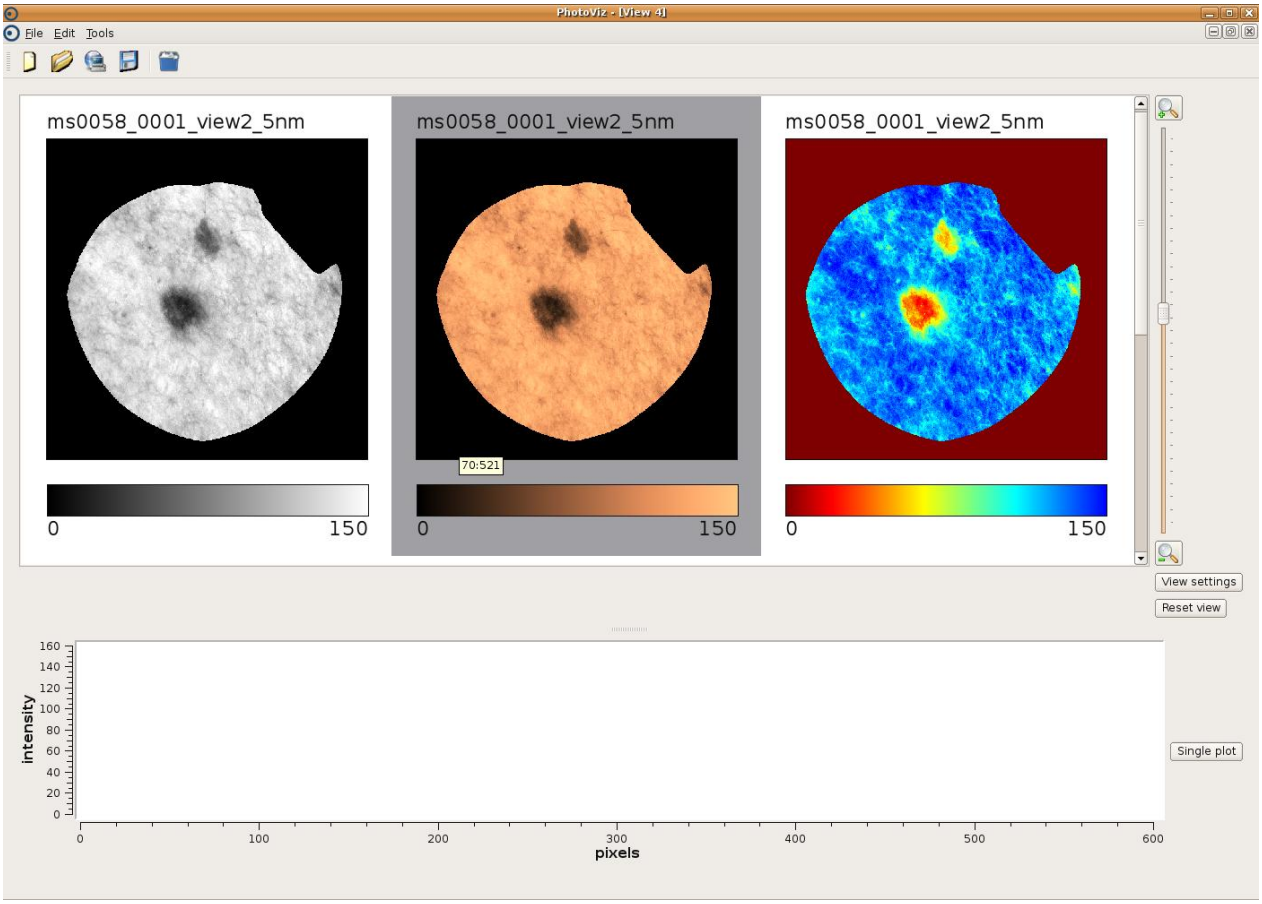
Figur 6.2: PhotoViz - visning av CS og DC bilder.

Figur 6.3 viser hvordan man kan benytte seg av flere visningsvinduer, her henholdsvis et med maps og et med kalibrerte bilder. Verdt å nevne er at visningsvinduene hele tiden befinner seg inni hovedvinduet. Det er ikke satt noen begrensning på hvor mange vinduer som kan være åpne samtidig, men det er viktig å bemerke at mange vinduer med mange bilder kan redusere ytelsen på maskinen drastisk.



Figur 6.3: PhotoViz - visning med multiple vinduer.

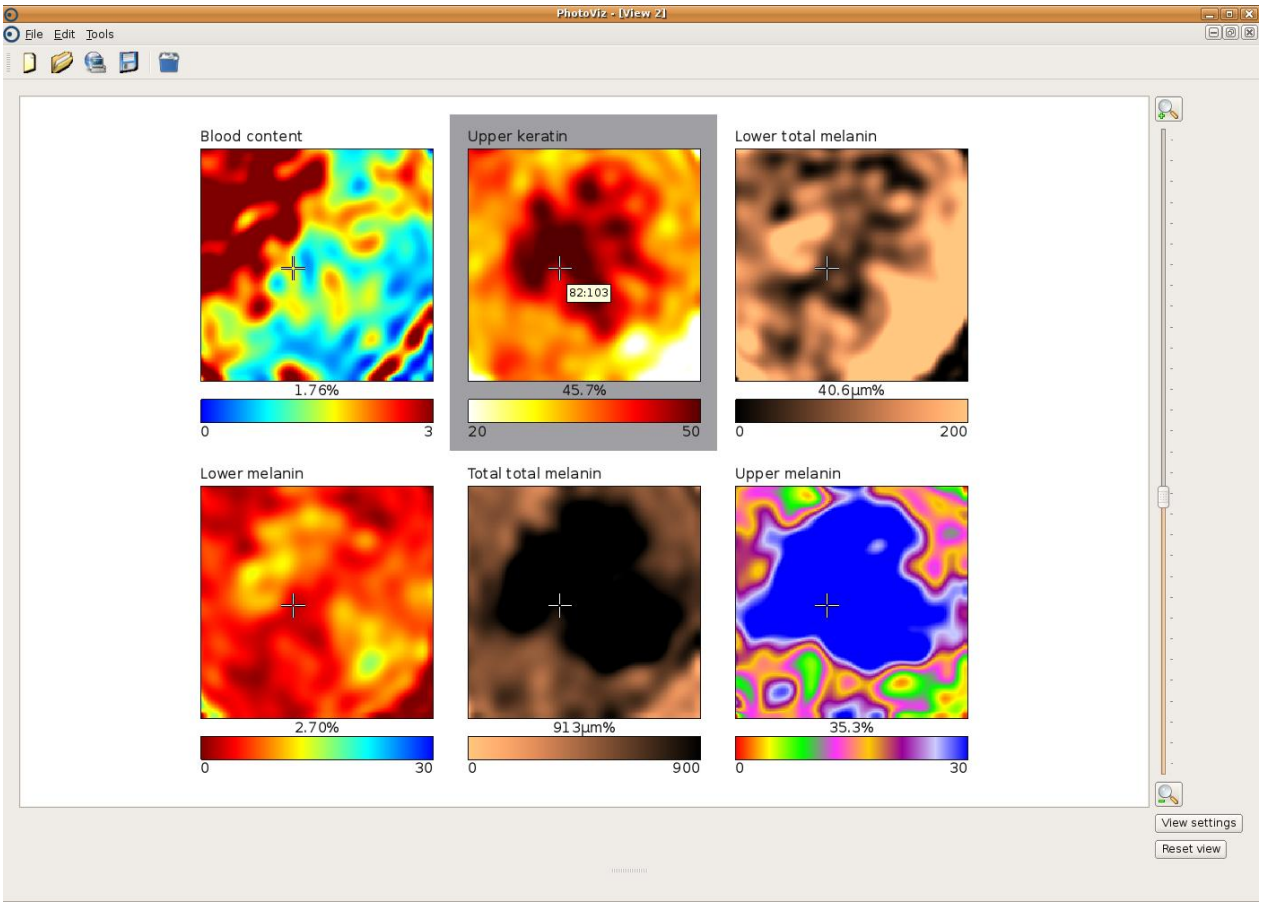
I Figur 6.4 kan man se det samme kalibrerte bilde, vist med tre forskjellige fargeskjema. Fordi bilder blir tatt i svarthvitt kan det være vanskelig å få øye på støy eller interessante detaljer med det blotte øyet. En endring av farger og gradientverdier kan derfor gjøre det enklere å analysere et bilde. Lengst til venstre i figuren vises bildet i sin opprinnelige form. Det midterste bildet er fargelagt med et fargeskjema som skaper en nær virkelig visning av føflekken, mens det i bildet til høyre blir brukt et fargeskjema som fremhever variasjoner og støy i bildet.



Figur 6.4: PhotoViz - visning av samme kalibrerte bilde med tre forskjellige fargeskjema.

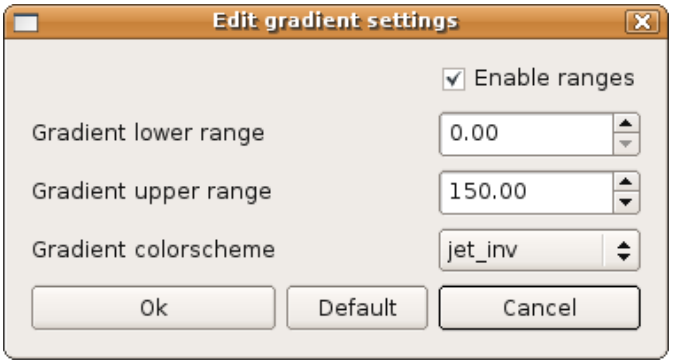
Tilsvarende kan man også endre fargeskjema for maps slik som vist i Figur 6.5. Hvilke fargeskjema som egner seg best til de forskjellige maps og bildetyper er det nok delte meninger om. Til nå har det blitt lagt til ca 25 fargeskjema, og nye kan opprettes ved å legge til en tekstfil med RGB verdier i en forhåndsdefinert mappe for fargeskjemafiler.

Dersom man ønsker å lagre visningen i lerretet som et bilde, kan dette gjøres ved å velge lagringsknappen i verktøylinjen. Dette vil åpne en fildialog hvor man kan velge bildeformat, filnavn og mappe som bildet skal lagres i.



Figur 6.5: PhotoViz - maps med noen utvalgte fargeskjema.

For å endre fargeskjema eller gradientverdier må brukeren høyreklikke på et bilde eller map. En dialog med nedtrekkmeny for fargeskjema og inputfelt for gradientverdier vil da vises. Denne dialogen kan sees i Figur 6.6. Når man har mange bilder åpne i et visningsvindu kan denne fremgangsmåten bli noe tungvint. Dialogvinduet derfor gjort tilgjengelig også ved å klikke på en knapp (View settings) i nedre høye hjørne. Forskjellen er at man da endrer innstillingene for alle elementer i visningsvinduet, slik at man slipper å endre et og et.

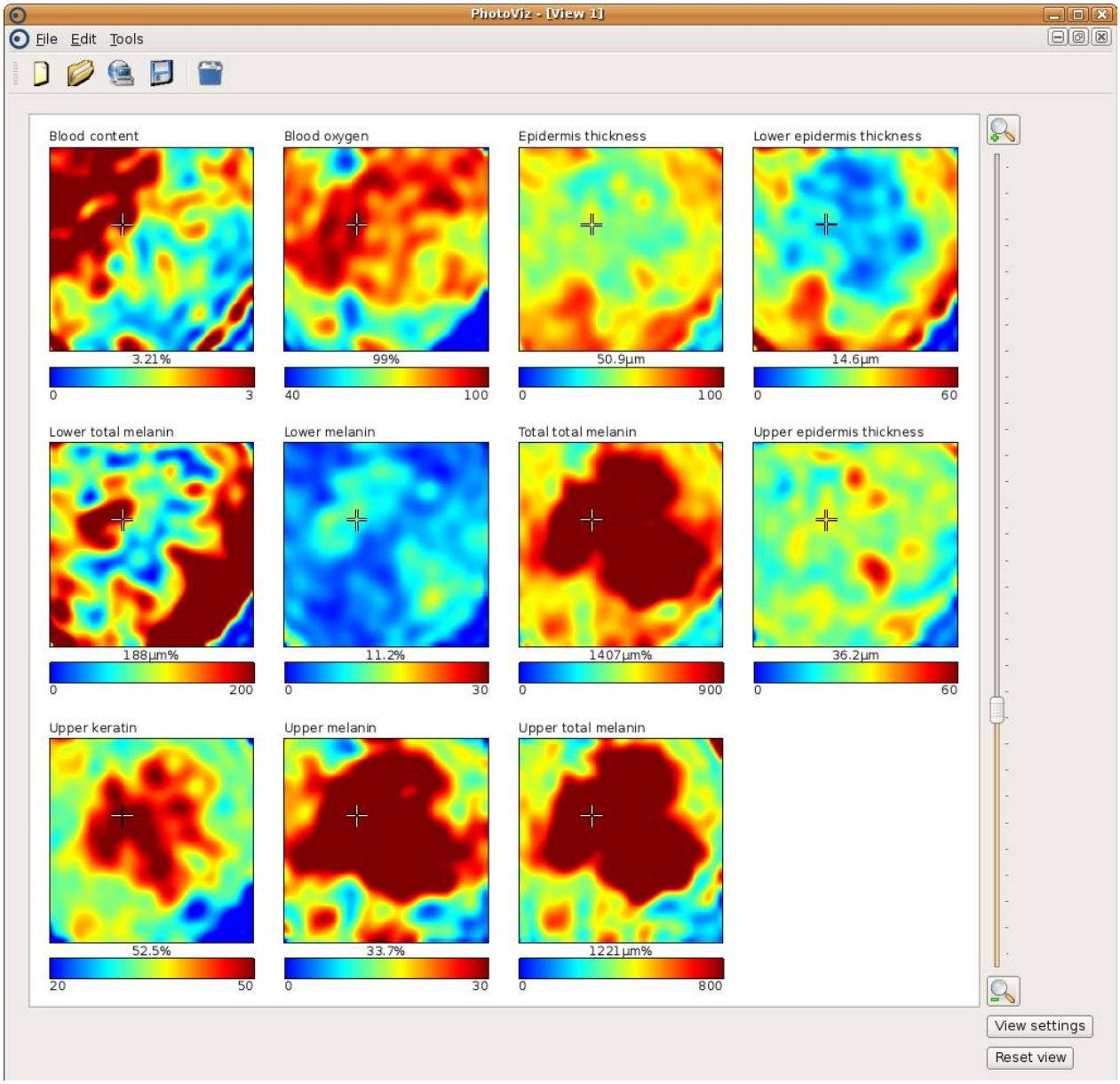


Figur 6.6: PhotoViz - Dialog for endring av gradient og fargeskjema

6.3 Punktsammenligning

For visning av bilder er ikke punktsammenligning et like bra verktøy som det er med maps. På bilder vil punktsammenligning bare vise hvorvidt to bilder er forskjellig på samme posisjon i bildene. For maps gjør punktsammenligning det mulig å trekke en mer nøyaktig slutning av tilstanden i huden. Dette er dog i kombinasjon med en visuell analyse av mapsene. Som vi kan se i Figur 6.5 viser et trådkors i hvert map posisjonen til punktet som sammenlignes. Verdien i dette punktet kan for hvert map sees mellom gradienten og mapet i seg selv. Punktsammenligningen utføres med et enkelt museklikk i et map, hvor så punktet og dets verdi automatisk oppdateres for alle andre maps. For mapet som brukes som utgangspunkt for punktsammenligningen markeres bakgrunnen i grått, og den nøyaktige posisjonen til punktet blir synlig i form av et teksttips.

I Figur 6.7 kan vi se at lesjonen inneholder forholdsvis mye blod, samt at den har et høyst varierende blodoksygennivå. Av tykkelsen på epidermis kan man se at det er spesielt to områder som skiller seg ut. Melanomer har den egenskapen at de vokser i ulike faser. Disse to områdene er typiske ”knuter” som holder på å spre seg ved å vokse nedover i huden. At resten av mapsene generelt har en noe ujevn fordeling er med på å forsterke mistanken om at dette er et melanom, noe som tidligere har blitt bekreftet gjennom en vevsprøve hos en patolog.

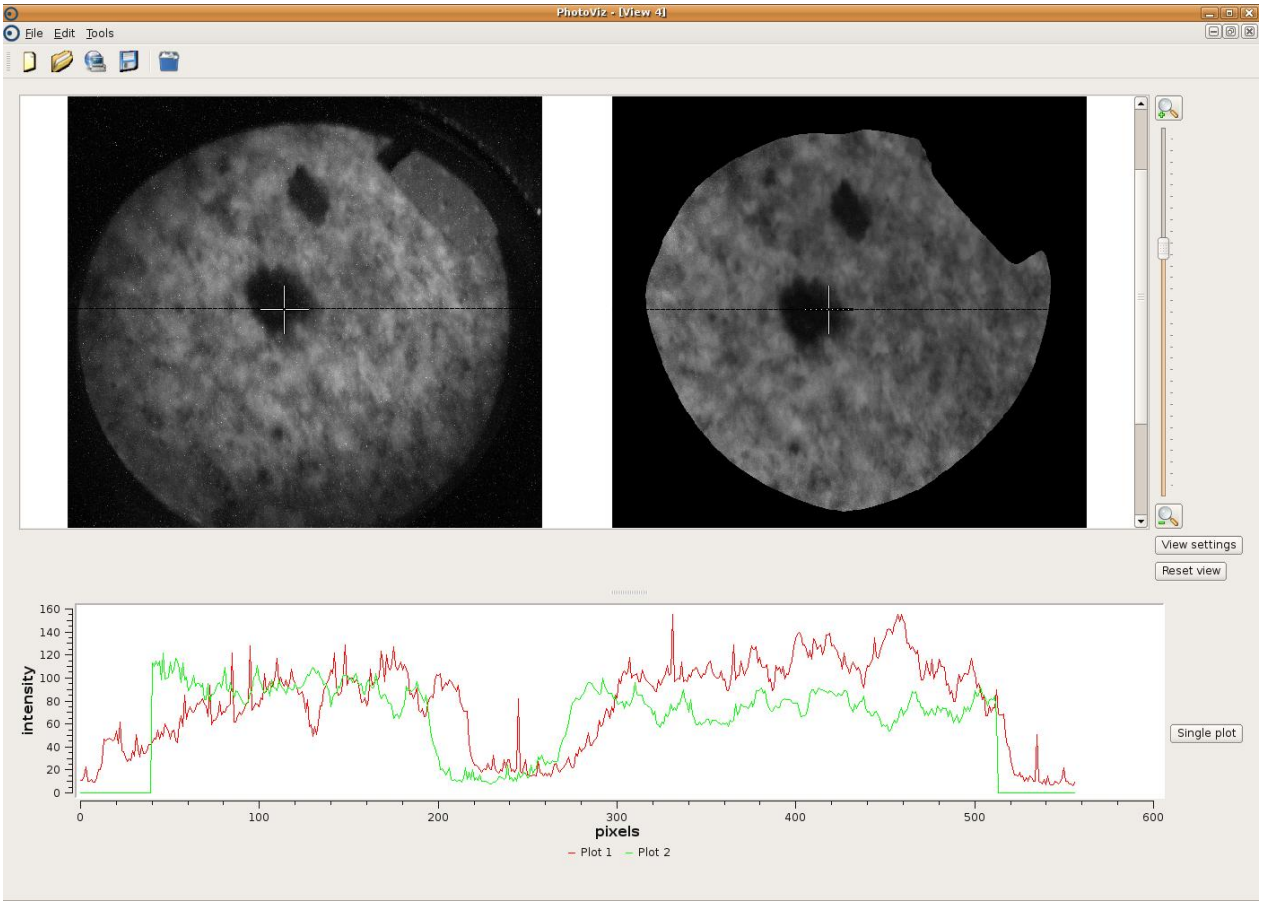


Figur 6.7: PhotoViz - punktsammenligning brukt på maps.

6.4 Profilvisning

Profilvisningen er først og fremst et verktøy for å validere bilder, men kan også brukes til å sammenligne flere bilder. Støy eller andre rariteter i bildet kommer tydelig frem i en profilvisning. I tillegg kan den benyttes til å evaluere om lukkertiden er som den skal. I Figur 6.8 kan man se en profilvisning fra et råbilde, og det tilsvarende kalibrerte bildet. Av profilvisningen kan man se at det kalibrerte bildet (grønn kurve) har pikselverdier som går fra 0 til 160. Optimalt sett burde disse verdiene gått fra 0 til 255, som er det maksimale spekteret av verdier (fargenyanser) et 8-bits bilde kan inneholde. Det ville derfor i dette tilfellet vært foredelaktig å øke lukkertiden. Videre kommer støy i råbildet (rød kurve) til syne som plutselige topper. I det kalibrerte bildet er denne støyen filtrert bort.

Skannlinjen som pikselverdiene hentes ut fra er representert ved en stiplet linje i selve bildet. En kurve blir lagt til i profilvisningen ved å dobbelklikke i et bilde eller map. Aksene i profilvisningen blir automatisk justert etter de høyeste verdiene i dataene. Det er forøvrig ingen begrensning på hvor mange kurver som kan legges til, men knappen ”Single plot” i nedre høyre hjørnet kan benyttes for å vise bare en og en kurve. Ved å klikke på musehjulet, eller begge knappene samtidig, kan man snu skannlinjen til å være vertikal fremfor horisontal.

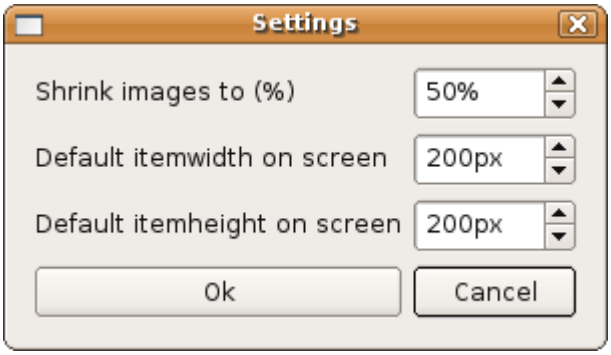


Figur 6.8: PhotoViz - profilvisning.

6.5 Innstillinger

For å endre applikasjonsinnstillinger som går på ytelse ble det laget en dialog som man kan se i Figur 6.9. Dette er tilgjengelig gjennom menylinjen i hovedvinduet. En visning av svært mange bilder samtidig vil kreve store mengder minne. Det ble derfor lagt til en innstiling som komprimerer bildene ved innlesning slik at man minnebruken kan begrenses. Dette gjelder kun bilder ettersom maps er liten av størrelse selv om de skaleres opp fra original størrelse før visning. Applikasjonens minnebruk var en stund et stort problem og vil omtales i kapittel 7.3.2.

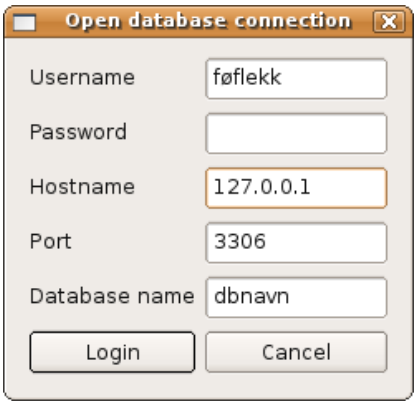
De to nederste innstillingene bestemmer standardstørrelsen for et bilde eller map ved visning på skjerm. Dette innebærer dog ikke komprimering, men kan heller sees på som hvor langt vekke et tiltenkt kamera står fra lerretet når dataene er lest inn.



Figur 6.9: PhotoViz - dialog for innstillinger

6.6 Søk mot database

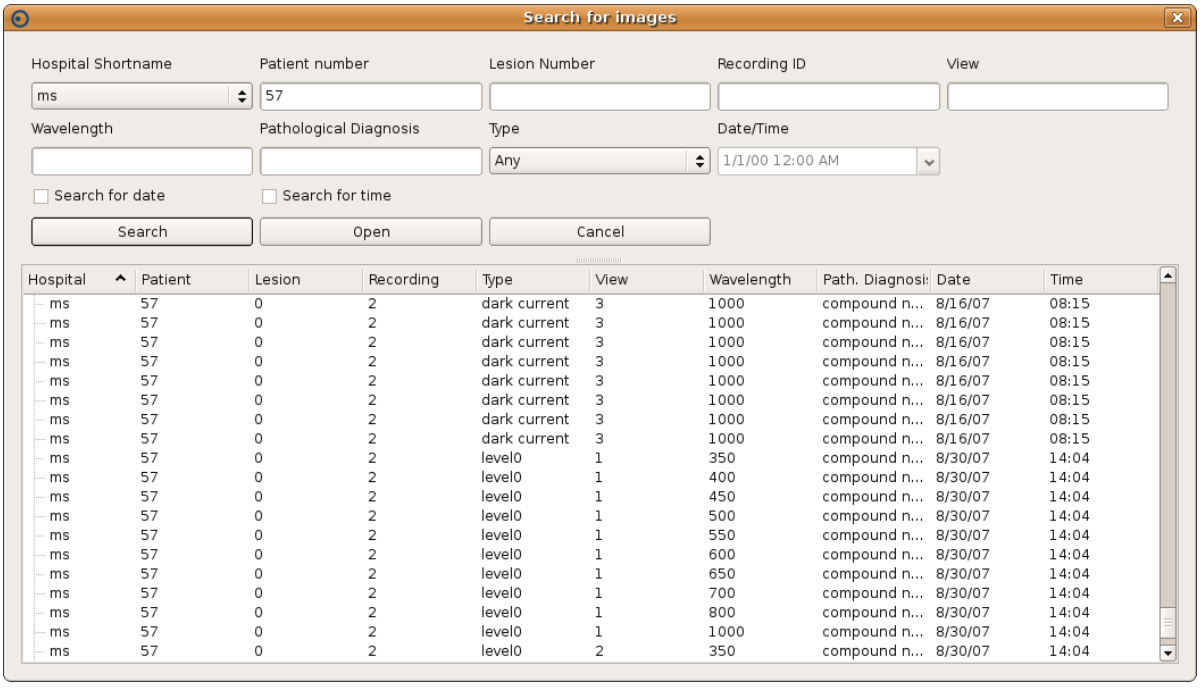
For å søke etter bilder og maps i databasen må man først logge seg på databasen ved å fylle inn nødvendige data som vist i Figur 6.10. For å gjøre denne delen mer brukervennlig blir alle feltene utenom passord lagret i en fil og lest inn ved åpning av dialogvinduet. Dermed må ikke brukeren fylle inn alle felter hver eneste gang man skal logge seg på databasen. I tillegg forblir man innlogget så lenge applikasjonen holdes åpen.



Figur 6.10: PhotoViz - dialog for innlogging til database.

Dersom inputen er gyldig vil søkevinduet automatisk vises. Som vist i Figur 6.11 inneholder dette en rekke felt som kan benyttes for å finne frem til ønsket lesjon. Forkortet sykehusnavn og type kan velges gjennom to nedtrekksbokser, mens tid og dato kan velges fra en minikalender. Av forskjellige typer kan brukeren velge mellom råbilder, kalibrerte bilder, DC bilder, CS bilder og maps. For å gjøre søket mer brukervennlig, ble uspesifiserte søkeparametere automatisk gjort til jokertegn (wildcard). Formålet med dette var å unngå at en bruker skulle være avhengig av å oppgi alle parametere, i motsetning til dagens situasjon hvor brukeren må spesifisere et fullstendig saksnummer.

Ved et søk vil bilder og maps som svarer til inputen vises som en linje. Det er dog viktig å nevne at bildene ikke lastes inn på dette tidspunktet. En slik fremgangsmåte ville ha vært svært treg og krevd store mengder minne. For å åpne bilder markeres linjene man ønsker, og trykker åpne. Som ved innlesning fra fil vil fremgangen vises i et lite dialogvindu med en fremdriftssøyle. Hvor fort bildene lastes inn vil være avhengig av nettverket man sitter på. Kjøres applikasjonen på samme maskin som databaseserveren vil dette naturligvis være det hurtigste. I et slikt tilfelle vil innlesning av bilder fra database være marginalt raskere enn innlesning fra filsystem.



Figur 6.11: PhotoViz - brukergrensesnitt for søk mot database.

7 Evaluering

Dette kapittelet vil ta for seg en evaluering av systemet og utviklingsprosessen, samt en gjennomgang av en rekke utfordringer underveis i arbeidet og hvordan de ble løst.

7.1 Generelt om applikasjonen

Utvikling av GUI er noe jeg tidligere har liten erfaring med, og må sies å ha vært tidkrevende og vanskelig. Det grafiske grensesnittet i Qt er hva man kan kalle standardisert, ved at det har et utseende som er likt andre applikasjoner og lett å kjenne igjen for brukeren. Det mest utfordrende har nok vært å utforme løsningene for punktsammenligning og profilvisning der brukerinputen skjer direkte mot de grafiske elementene. Dette, i tillegg til å knytte sammen alle de forskjellige delene av GUIet, har tatt mye tid og har vært en prosess hvor mye har måttet læres underveis i utviklingsarbeidet.

Sammenlignet med visning av bilder og maps i dagens MATLAB baserte løsning, innfører ikke visningen i den nye løsningen så mye nytt. Rent funksjonelt kan bilder og maps fargelegges i begge løsningene, og man kan generere profilvisninger. Den store forskjellen ligger i måten bilder og maps blir presentert på, samt at den nye løsningen samler funksjonaliteten i MATLAB skriptene i en og samme applikasjon. Til forskjell fra dagens løsning, kan man med den nye løsningen vise mange bilder og maps side om side slik at analyse og sammenligning blir enklere. Videre kan en bruker i den nye løsningen benytte punktsammenligning i analysen av bilder og spesielt maps, noe som tidligere ikke var mulig.

Underveis i arbeidet ble det nevnt fra oppdragsgivers side at det også blir produsert noen andre filer, kalt daps. Dataene i disse filene har samme format som i mapfilene, og kan derfor visualiseres på samme måte. Daps blir som maps produsert i fysiologi- og morfologianalysen, men beskriver i stedet usikkerheten knyttet til dataene i en bestemt mapfil. En slik usikkerhet kan for eksempel være at det er pluss eller minus 5% blod i et punkt i et map. Ved å åpne disse side om side med maps kan man med punktsammenligning vise den nøyaktige usikkerheten knyttet til et bestemt punkt i et map, en oppgave som tidligere har måttet gjøres manuelt.

Ettersom oppdragsgiver i nærmeste fremtid vil gå over til en databaseløsning for lagring av data har integrasjon mot denne vært av høy prioritet. Som forklart i kapittel 7.3.3 gikk ikke integrasjonen helt smertefritt. Men sammenlignet med dagens metode for åpning av bilder og maps, gir søkefunksjonen i den nye løsningen en enklere åpning og visning av data. Ved å spesifisere hvor nøyaktig søket skal være kan brukeren kan lettere finne frem til ønskede data, i motsetning til tidligere hvor man måtte oppgi et fullstendig saksnummer.

Et annet viktig moment i problemstillingen var ytelse. Dette kan en lese mer om i kapittel 7.3.2.

7.2 Testing

Det er ikke til å legge skjul på at testing av applikasjonen har vært problematisk. For hvordan tester man for eksempel at et map vises korrekt? I dette prosjektet har testingen foregått mer eller mindre manuelt, med bruk av debuggingsverktøy, utskrift til konsoll og visuell verifisering. En slik fremgangsmåte er langt i fra optimal, og krever at det blir satt av mye ekstra tid til testing. For å bøte litt på denne problematikken ble applikasjonen plassert på en server hvor brukere hadde anledning til å teste den ut. Det ble også arrangert uformelle møter med ansatte hvor applikasjonen ble gjennomgått. Hensikten med dette var å oppdage bugs underveis i utviklingen, samt å la flere brukere komme med forslag til forbedringer av applikasjonen. Av disse var det spesielt møtene som ga verdifulle tilbakemeldinger.

For å teste at bilder og maps vises korrekt har utvalgte tagninger blitt visuelt sammenlignet med visningen i dagens MATLAB baserte system, og i den nye løsningen. At manuell testing kan være risikabelt viste seg mot slutten av arbeidet. Først da ble det ved en tilfeldighet oppdaget en feil som gjorde at beregningen av farger var noe unøyaktig. Riktignok var ikke dette en feil som hadde store konsekvenser. Den var med andre ord svært vanskelig å få øye på rent visuelt. Samtidig viser den hvor lett det er å ikke oppdage feil når testingen skjer manuelt. Feilen eksisterte i en del av koden som ble skrevet i starten av arbeidet og gikk altså nesten usett forbi i hele utviklingsarbeidet.

Testing mot databasen har som det med det meste andre skjedd manuelt. Et problem med denne delen var at data i databasen kun besto av bilder og ikke maps. Løsningen her ble naturlig nok å laste opp noen maps, som så ble åpnet i applikasjonen. Disse ble så visuelt sjekket mot tilsvarende data lastet fra filsystem.

Som nevnt i kapittel 4.5 ble det valgt å ikke benytte TDD. Tatt i betraktning at TDD har nær sammenheng med refaktorering, medvirket dette valget til at det i blant oppsto feil etter refaktoreringer som var vanskelige å lokalisere. Der noen av disse var rene grafiske ting som ville vært vanskelig å skrive enhetstester for, gjaldt andre blant annet kommunikasjon mellom systemdeler, og som nevnt beregning av farger. For noen av disse feilene gikk mye tid med på bare å lokalisere feilen. Nå er det ikke sikkert at enhetstester alltid ville gjort det enklere å lokalisere feilene, men at det ville hjulpet i noen tilfeller er rimelig å anta. Det er derfor mulig, men vanskelig å svare på, om det kunne vært fordelaktig å benytte TDD for de delene av systemet som ikke gjaldt det rent grafiske.

7.3 Utfordringer

7.3.1 Manglende TIFF-støtte

Bildene som blir produsert gjennom skanning og kalibrering er som nevnt 8 og 16-bits enkanals TIFF-bilder. Av de forskjellige bildene er det bare kalibrerte bilder som er 16-bit. I Qt håndteres TIFF støtte av en tilleggsmodul, kalt TIFFIO, som igjen benytter seg av et bildebibliotek med navnet LibTIFF [33]. Mens LibTIFF originalt støtter TIFF bilder med 8 og 16-bits per kanal, ble det et stykke ut i arbeidet oppdaget at TIFFIO inneholdt en bug som gjorde at 16-bits bilder ikke var støttet. Slike bilder ble i stedet automatisk nedkonvertert til 8 bits per kanal ved innlesning. Rent visuelt skaper ikke dette noen nevneverdige problemer, da det er svært vanskelig å se forskjell på et 8 og 16-bits bilde. Problemet var at dataene fra et kalibrert bilde vist i profilvisningen ble unøyaktige. Der et 16-bits bilde kan inneholde 65536 forskjellige fargenyanser, ble dette som en følge av feilen krympet til bare 256 forskjellige. Fra oppdragsgivers side ble dette heldigvis ikke regnet som et stort problem ettersom data fra nedkonverterte bilder fremdeles var gode nok til bruk i profilvisningen. En oppretting av dette problemet ble derfor nedprioritert av hensyn til innføring av databasesøk og oppdragsgivers ønsker for det videre arbeid. En løsning på dette problemet ville vært å benytte det originale LibTIFF direkte, uavhengig av Qt sin tilleggsmodul TIFFIO. Dette har blitt gjort i [9] hvor en slik fremgangsmåte var nødvendig for å produsere kalibrerte bilder.

7.3.2 Minnebruk og ytelse

Håndtering av applikasjonens minnebruk har vært den kanskje største utfordringen i utviklingsarbeidet. I avsnittet over ble det nevnt at et 16-bits enkanals bilde blir nedkonvertert til 8-bits per kanal, men dette er ikke hele sannheten. Alle TIFF bilder, enten de er 8 eller 16 bits, blir automatisk konvertert til et 8-bits firekanals (ARGB32) Qt bildeobjekt ved innlasting. For kalibrerte bilder innebærer denne konverteringen først en nedskalering til 8-bit. Ved innlasting av bilder medfører dette at den ene kanalen i det opprinnelige bildet dupliseres og lagres totalt 4 ganger i minnet. Mens en bildefil lagret på disk er på ca 1,2MB, vil et innlest bilde i applikasjonen benytte 8-9MB minne, omtrentlig åtte ganger mer enn originalen på disk. Dette er ikke bare på grunn av at bildet blir automatisk konvertert, men som sagt også fordi et bilde som lastes inn lagres i to eksemplarer internt i applikasjonen (en original, samt et som fargelegges og vises). Hvis en laster inn alle råbilder fra en enkelt tagning (30 bilder) vil dette kreve i underkant av 300MB minne. Dersom applikasjonen kjøres fra felles server, og andre brukere gjør det samme samtidig er det ganske opplagt minnebruken vil komme opp i størrelser som definitivt kan by på problemer. Hvorfor denne automatiske konverteringen skjer, om det er TIFFIO, ufullstendige headere i bildefiler eller andre ting har det ikke lykkes i å finne ut av.

En kjapp og enkel løsning som ville halvert minnebruken var å kun benytte seg av ett bildeobjekt internt i applikasjonen. Men som en stor ulempe ville dette krevd at man måtte lese inn

originalbildet fra fil hver gang man endret fargeskjema eller gradientverdier. Årsaken til dette er at disse operasjonene må gjøres med hensyn til de originale dataene, slik at man ikke får gale visninger og ender opp med å fargelegge et bilde basert på data som allerede er fargelagt. Det samme ville forøvrig også gjelde hver gang man benyttet seg av profilvisningen etter at et bilde var fargelagt. En slik innlesning for hver gang brukeren skulle gjort endringer ville derfor ført til at ytelsen ved fargelegging og profilvisning hadde gått betraktelig ned. Denne fremgangsmåten ble derfor regnet som for dårlig.

I stedet ble det valgt å gjøre en konvertering tilbake, til et 8-bits enkanals indeksert bildeobjekt. Et indeksert format benytter seg av en fargetabell som defineres på forhånd, slik at man kan vise alle slags farger og ikke er bundet til vanlige 8-bits farger [34]. Denne konverteringen går noe tregt sammenlignet med konvertering til andre Qt spesifikke bildeformat, men er likevel mye raskere enn selve innlesningen av et bilde. Mens det optimale hadde vært å slippe en tilbakekonvertering i det hele tatt, er denne fremgangsmåten tross alt mye bedre enn den enkle løsningen nevnt over. Bortsett fra noe overhead vil en visning av alle råbilder i en tagning nå benytte ca 75MB minne. Et minus med denne fremgangsmåten var at filinnlesningsklassen ble mer kompleks.

En annen utfordring i utviklingsarbeidet har vært å oppnå god ytelse slik at ikke brukergrensesnittet og operasjoner oppfattes som tregt av brukeren. I kapittel 3.2 ble det nevnt at operasjoner utført med dagens MATLAB baserte grensesnitt gikk tregt. I dette arbeidet har det ytelsesmessige målet vært å lage en løsning som er markant raskere enn dagens, men noen kvantifiserbare mål har ikke blitt satt da jeg tidligere ikke har erfaring med bildebehandling. Mens innlesning, fargelegging og zooming av lerretet er de mest krevende operasjonene, har det i stor grad bare vært ytelsen ved fargelegging som har kunnet påvirkes. Interpoleringen av maps kan også nevnes, men denne operasjonen tar mye kortere tid enn fargelegging av et bilde. Grunnen til dette er at maps er forholdsvis liten i størrelse, selv etter en oppskalering. Andre ting som zooming av lerret går på Qt spesifikk implementasjon og var derfor vanskelig å forbedre. Disse operasjonene kan likevel ikke betegnes som spesielt trege, og vil utføres mer eller mindre i sanntid dersom man holder seg til å vise ca 30 bilder.

I starten av arbeidet var fargelegging av et bilde kun marginalt raskere enn MATLAB, og tok 5-6 sekunder. På dette tidspunktet hadde en forbedring av ytelsen lav prioritet, til tross for at det var klart at operasjonen med beregning av farger var lite optimalisert. Men gjennom refaktoreringer underveis i arbeidet har fargeleggingen gradvis blitt raskere og raskere. Mot slutten innebar en av disse refaktoreringene innføring av bit-operasjoner på bildedataene for å heve ytelsen enda et hakk. Fargelegging av et bilde tar nå rundt 200 millisekunder, avhengig av maskinen applikasjonen kjøres på. Likeså tar hele prosessen med å lese inn og vise frem alle

råbilder fra en tagning 7-8 sekunder. Selv om dette er en stor forbedring fra hva det var i utgangspunktet, så er det langt fra usannsynlig at ytelsen kan forbedres ytterligere.

Alt i alt kunne man kanskje tro at minnebruk og ytelse nå var på et tilfredsstillende nivå, noe det faktisk i de fleste tilfeller vil være. Likevel, det ble valgt å forbedre dette ytterligere, spesielt med fokus på applikasjonens minnebruk. I ekstreme tilfeller kan det være nyttig å åpne et helt sett med CS eller DC bilder for å validere dem kjapt. Disse settene inneholder som nevnt 300 bilder. I slike tilfeller, og dersom flere brukere kjører applikasjonen fra samme server, kan minnebruket bli svært høyt. Derfor ble det som nevnt valgt å innføre brukerstyrte innstillinger som lar en komprimere bildene før visning. Et bilde komprimert til 50 prosent av originalstørrelsen reduserer minnebruken med en faktor på 4, og er tilsvarende 4 ganger raskere å fargelegge. Nå har helt klart en komprimering også sine ulemper ved at bildedata fjernes. For eksempel kan piksler med støy reduseres og bli vanskeligere å få øye på. Men for en generell validering, hvor det for eksempel sjekkes at bilder ikke er ute av fokus, vil bilder komprimert 50 prosent være tilstrekkelig. Når alt kommer til alt blir det tilslutt likevel opp til den enkelte bruker å bedømme om, eller hvor mye bildene skal komprimeres.

7.3.3 Mangelfull databasestøtte

I den nye databaseløsningen er bilder og maps implementert som binærdata (BLOB) i en MySQL database [35]. Da Qt har innebygget MySQL-støtte forløp integrasjonen mot databasen forholdsvis enkelt, men med ett ankepunkt. For å forhindre at applikasjonen måtte rekompileres ved endringer i databasen ble det, med bidrag fra medstudenten som utviklet databaseløsningen, skrevet SQL kode til bruk i lagrede prosedyrer. En lagret prosedyre kan tenkes på som en liten funksjon, eller program, som kjøres fra databaseserveren. Ved å benytte dette flyttes logikk fra applikasjonskoden til databaseserveren slik at SQL kode skilles fra applikasjonskoden, samtidig som det åpnes for at andre applikasjoner kan bruke de samme lagrede prosedyrene. Dessverre hadde Qt noe manglende støtte for bruk av lagrede prosedyrer mot MySQL databaser slik at det ikke lot seg gjøre å hente ut bilder fra databasen på denne måten. Dette gjorde at det likevel ble benyttet hardkodete SQL-setninger i applikasjonskoden, noe som ikke er optimalt og som kan by på problemer dersom det skjer strukturelle endringer i databasen.

Et problem som oppsto som en følge av dette igjen, var at det ikke var mulig å benytte seg av innebygd Qt funksjonalitet som beskyttet mot SQL-injection. Gjennom denne metoden kan en inntrenger få tak i fortrolig informasjon eller slette data fra databasen ved å skrive inn SQL-kode i inputfeltene i søkeskjemaet. For å forhindre dette ble det derfor laget regulære uttrykk som brukes til å sjekke at input fra søkeskjemaet ikke inneholder potensielt farlige tegn.

7.4 Vurdering av utviklingsmetode

Utviklingsmetoden som har blitt benyttet baserer seg som nevnt på en smidig metode. For å sikre god kommunikasjon er ”kunderepresentant i teamet” en sentral praksis i XP. Mesteparten av dette utviklingsarbeidet har foregått i oppdragsgivers lokaler, og har sådan vært en viktig faktor for å opprettholde god kommunikasjon. De gangene det har vært uklarheter eller spørsmål, har oppdragsgiver bortimot alltid vært tilgjengelig. Dette har vært veldig nyttig, ikke bare hva gjelder ting rundt utviklingen av applikasjonen, men også for å få en raskere forståelse av dagens system og teknologien som benyttes i skanneren.

Utviklingen av applikasjonen har forløpt med hensyn til de forskjellige delmålene, hvor arbeid med et delmål har startet når et annet er ferdig. Man kan derfor si at applikasjonen er bygget på et inkrementelt design. Dette innebærer at de forskjellige modulene så langt det har vært mulig har blitt utviklet en etter en, for så å integreres når de er ferdig. Dette betyr dog ikke at modulene ikke har gått igjennom endringer underveis. Gjennom faste refaktoreringer har designet gradvis blitt forbedret. Derfor vil jeg kalle utviklingsmetoden iterativ, til tross for at det ikke har blitt gjort noe konkret iterasjonsplanlegging utover prioriteringer og fremdriftsplanen som ble laget.

I forhold til enkelhet har det i arbeidet blitt forsøkt å lage det enkleste som kan virke. Som nevnt over har systemet blitt utvidet litt etter litt og underveis har det vært fokus på å ikke legge til mer funksjonalitet enn det som strengt tatt har vært nødvendig. En ting som er verdt å nevne er at det har blitt benyttet mye signaler og slotter. Dette er jo forsåvidt et meget nyttig redskap, men med tanke på videreutvikling kan dette i noen tilfeller være et kompliserende ledd. Problemet med signaler og slotter er at det kan være tungvint å sette seg inn i hvordan de forskjellige systemdelene kommuniserer. Med det menes at det kan være lett å miste oversikten over hvilke objekter som sender signaler, hvem som mottar signalene, og i hvilke tilfeller det skjer. Mens man for vanlige funksjonskall kan følge en typisk sti med kall, er man med signaler og slotter alltid avhengig av først å skaffe seg en oversikt over hvilke signaler som finnes i et objekt, og hvilke objekt som lytter til dette signalet. Vel og merke er ikke kodebasen i dette arbeidet veldig stor, så å skaffe seg en oversikt burde la seg gjøre uten nevneverdige problemer. Steder hvor det har vært naturlig med funksjonskall har bruk av signaler og slotter blitt unngått. Likevel, det viktig å bemerke at det også finnes baksider med signaler og slotter. Alt i alt, om designet i denne løsningen er enkelt blir til slutt opp til andre å bedømme.

8 Oppsummering og veien videre

8.1 Oppsummering

Denne oppgaven ble gjort i samarbeid med firmaet Balter Medical som har utviklet en ny metode for diagnostisering av hudkreft. De siste femti årene har det vært en dramatisk økning i antall tilfeller av føflekkreft. Per i dag er dette den hurtigst voksende kreftformen i Norge. For å hindre spredning og øke sjansene for overlevelse er det svært viktig med tidlig diagnostisering av føflekkreft. Dagens metode for diagnostisering er preget av å være kostbar, tidkrevende og ikke minst unøyaktig. For å få bukt med dette problemet har Balter Medical de siste årene arbeidet med å utvikle en ny ikke-kirurgisk teknologi for hurtig og presis diagnostisering av hudkreft. Diagnostiseringen gjøres ved å ta en mengde bilder med en optisk skanner. Gjennom en prosessering av bildene kan man trekke ut informasjon om tilstanden i huden. Teknologien er relativt ny og ennå ikke ferdig utprøvd. En stor del arbeidet hos Balter Medical går derfor med til videreutvikling og forbedring av skanneren og diagnostiseringen.

I denne masteroppgaven blitt laget et visualiseringsverktøy til validering og analyse av bildedata for digitale bilder av hudkreft. Systemet er hovedsakelig ment til bruk i videreutviklingen av diagnostiseringsteknologien, men skal også danne en basis for hva som kan være aktuell programvare hos leger. I den nye løsningen samles all funksjonalitet på et og samme sted, noe som skaper en enklere og mer effektiv hverdag for fysikerne. Dagens løsning har bestått av separate verktøy, spredt utover en rekke MATLAB-skript som har vært trege og tungvinte å benytte.

Applikasjonen består av en rekke hjelpeverktøy som forenkler validering, sammenligning og analyse av bilder og maps. Deriblant profilvisning, søk etter data og visning av mange bilder og maps samtidig. I tillegg er det verdt å nevne punktsammenligningen som kan gjøre det enklere å stille en nøyaktig diagnose i fra maps. Det har vært av høy prioritet å skape et brukervennlig og hurtig GUI, slik at man med enkelhet kan laste inn og vise data uten lang ventetid. Det har også blitt lagt vekt på å lage et system som er lett å utvide. Hvor jeg har lykkes noen steder, er det andre plasser ikke like bra og burde vært gjort annerledes fra starten av. For sistnevnte er det skissert forslag til forbedringer som er beskrevet i kapittel 5.3.

Underveis har jeg støtt på mange utfordringer som har vært med på å gjøre arbeidet til en lærerik prosess, fylt av mye nytt. Å ha fått muligheten til å arbeide med ny og spennende teknologi i et nytt fagfelt må sies å ha vært svært givende.

8.2 Veien videre

I kapittel 5.3 ble det skissert en løsning for en ønsket RGB-vektet visning av bilder. Dersom det hadde vært mer tid til rådighet ville nok dette blitt implementert. Dette inkluderer også en modifisering av innlesningsmodulen som vil gjøre det enklere å legge til nye typer data i fremtiden. Hvilke typer data dette kan være er vanskelig å forutse, og vil bli ren gjetning. Kanskje blir det bare bilder og maps også i fremtiden, hvor så det blir mer aktuelt å lage nye alternative visninger av disse.

En annen ting som bør forbedres er bruken av SQL-kode sammen med applikasjonskoden. Denne bør som nevnt skilles ut, for eksempel ved bruk av lagrede prosedyrer. For å få dette til vil man trolig være avhengig av oppdateringer av Qt. Videre bør man gjøre det mulig å endre visningsinnstillingene gjennom GUIet. Forskjellige brukere kan ha ulike ønsker og behov i forhold til standardvisningen av bilder og maps. Det vil derfor være en fordel om disse innstillingene kan personliggjøres. Innstillingene leses per i dag inn fra en fil, og kan bare endres ved å editere denne filen.

Som nevnt var applikasjonen ment å danne en basis for programvare som kan brukes av leger. Mens denne løsningen er konsentrert rundt visning av data, kan det i fremtiden kanskje være aktuelt å samle alt fra innhenting av data fra skanner, kalibrering, fysiologi- og morfologianalyse og visning av data i av en og samme applikasjon. Dette vil riktig nok kreve en hel del arbeid, og vil derfor ligge et stykke frem i tid. Imidlertid er det ved siden av denne oppgaven laget to andre programmer, et for opplasting av bilder til database og et for kalibrering av bilder. Qt har også blitt benyttet til å utvikle disse programmene. En integrasjon av disse programmene med løsningen i denne oppgaven bør derfor kunne forløpe mer eller mindre smertefritt.

9 Bibliografi

[1]. **Kreftregisteret.** Cancer in Norway 1957-2001 trends. [Internett] 26 April 2004. [Sitert: 13 Mars 2008.] http://www.kreftregisteret.no/forekomst_og_overlevelse_2001/download/tables/cancer_in_norway_1957-2001_trends.xls.

[2]. **Kreftforeningen.** Føflekkreft - Hudkreft / Malignt Melanom. [Internett] 25 Mai 2005. [Sitert: 13 Mars 2008.] http://www.kreftforeningen.no/portal/page?_pageid=35,3018&_dad=portal&_schema=PORTAL&navigation1_parentItemId=2448&navigation2_parentItemId=2448&navigation2_selectedItemId=2017&_piref35_3023_35_3018_3018.sectionId=288.

[3]. **Kreftregisteret.** Forekomst og overlevelse. [Internett] 19 April 2004. [Sitert: 14 Mars 2008.] http://www.kreftregisteret.no/frame.htm?forekomst_og_overlevelse_2001/melanoma/incidence.htm.

[4]. **Balter Medical Inc.** *Internal documents.*

[5]. **Scott W Menzies, et al.** *Time to diagnosis of melanoma: same trend in different continents.* Tuebingen, Tyskland : Department of Dermatology, University of Tuebingen, 2007.

[6]. **Nielsen, Kristian P.** Personal comment.

[7]. **Kreftforeningen.** *Brosjyre: Føflekkreft - og annen hudkreft.* Oslo : Kreftforeningen, Desember 2004. ISSN-1502-7295.

[8]. **Feng Dong, et. al.** IGI Global - User Centered Design for Medical Visualization. [Internett] Mai 2008. [Sitert: 25 Mai 2008.] http://www.igi-global.com/reference/details.asp?ID=7530&v=preface.

[9]. **Vikshåland, Svein Even.** *Optimalisering av bildebehandling for diagnose av hudsykdommer.* Bergen : Universitetet i Bergen, 2008.

[10]. **MD, David L. Swanson.** Morphological and physiological optical imaging of the skin: discrimination between melanoma and nevi. *ISDIS.* 2008, Vol. 1, No.3.

[11]. **MathWorks.** MATLAB documentation - colormap. [Internett] [Sitert: 10 April 2008.] http://www.mathworks.com/access/helpdesk/help/techdoc/index.html?/access/helpdesk/help/techdoc/ref/colormap.html.

[12]. **Kvalsund, Øyvind.** *Database for bildediagnostisk programvare for hudkreft.* Bergen : Universitetet i Bergen, 2008.

[13]. **Trolltech.** The Qt Class Libraries. [Internett] [Sitert: 22 April 2008.] http://trolltech.com/products/qt/features/qtlibrary.

[14]. **Uwe Rathmann, Josef Wilgen.** Qwt - Qt for technical applications. [Internett] [Sitert: 22 April 2008.] http://qwt.sourceforge.net/.

[15]. **Bernhard Preim, et. al.** *NPR, Focussing and Emphasis in Medical Visualizations.* Magdeburg/Bremen : Otto-von-Guericke-University of Magdeburg.

[16]. **Universität Paderborn.** Visualization of medical data. [Internett] 18 Desember 2007. [Sitert: 1 Mai 2008.] http://www.cs.uni-paderborn.de/fachgebiete/ag-domik/forschung/visualization-of-medical-data.html.

[17]. **Wikipedia.** Bilinear interpolation. [Internett] [Sitert: 2 Mai 2008.] http://en.wikipedia.org/wiki/Bilinear_interpolation.

[18]. **Martin, Robert C.** *Agile software development.* New Jersey : Prentice Hall, 2003. ISBN 0-13-597444-5.

[19]. **Trolltech ASA.** Qt Linguist. [Internett] [Sitert: 29 April 2008.] http://trolltech.com/products/qt/features/internationalization.

[20]. **Ubuntu.** About Ubuntu. [Internett] [Sitert: 29 April 2008.] http://www.ubuntu.com/.

[21]. **Eclipse Foundation.** About the Eclipse Foundation. [Internett] [Sitert: 29 April 2008.] http://www.eclipse.org/org/#about.

[22]. **Eclipse Foundation.** Eclipse C/C++ Development Tooling - CDT. [Internett] [Sitert: 29 April 2008.] http://www.eclipse.org/cdt/.

[23]. **Trolltech ASA.** A Sneak Peek at the Qt Eclipse™ Integration. [Internett] [Sitert: 29 April 2008.] http://trolltech.com/company/tt/eclipse-integration.

[24]. **KDevelop.** KDevelop - an integrated development enviroment. [Internett] [Sitert: 29 April 2008.] http://www.kdevelop.org/.

[25]. **Biord, Jean-Luc.** QDevelop : Free cross-platform Development Environment for Qt4. [Internett] [Sitert: 29 April 2008.] http://qdevelop.free.fr.

[26]. **The GNU Project.** GDB: The GNU Project Debugger. [Internett] [Sitert: 29 April 2008.] http://www.gnu.org/software/gdb/.

[27]. **Valgrind.** About Valgrind. [Internett] [Sitert: 29 April 2008.] http://valgrind.org/info/about.html.

[28]. **Heesch, Dimitri van.** Doxygen. [Internett] [Sitert: 30 April 2008.] http://www.stack.nl/~dimitri/doxygen/.

[29]. **Subversion.** tigris.org - Subversion. [Internett] [Sitert: 29 April 2008.] http://subversion.tigris.org/.

[30]. **CreativeCommons.** Observer pattern. [Internett] [Sitert: 03 Mai 2008.] http://sourcemaking.com/design_patterns/observer.

[31]. **Trolltech ASA.** Using the Meta-Object Compiler (moc). [Internett] [Sitert: 03 Mai 2008.] http://doc.trolltech.com/4.3/moc.html.

[32]. **Martin, Robert C.** Agile software development, Chapter 9. Upper Saddle River, New Jersey : Prentice Hall.

[33]. **LibTIFF.** [Internett] 19 Februar 2004. [Sitert: 09 Mai 2008.] http://www.libtiff.org/libtiff.html.

[34]. **Wikipedia.** 8-bit color. [Internett] [Sitert: 11 Mai 2008.] http://en.wikipedia.org/wiki/8-bit_color.

[35]. **MySQL.** MySQL :: The world's most popular open source database. [Internett] MySQL AB, Sun Microsystems. [Sitert: 28 April 2008.] http://mysql.com/.