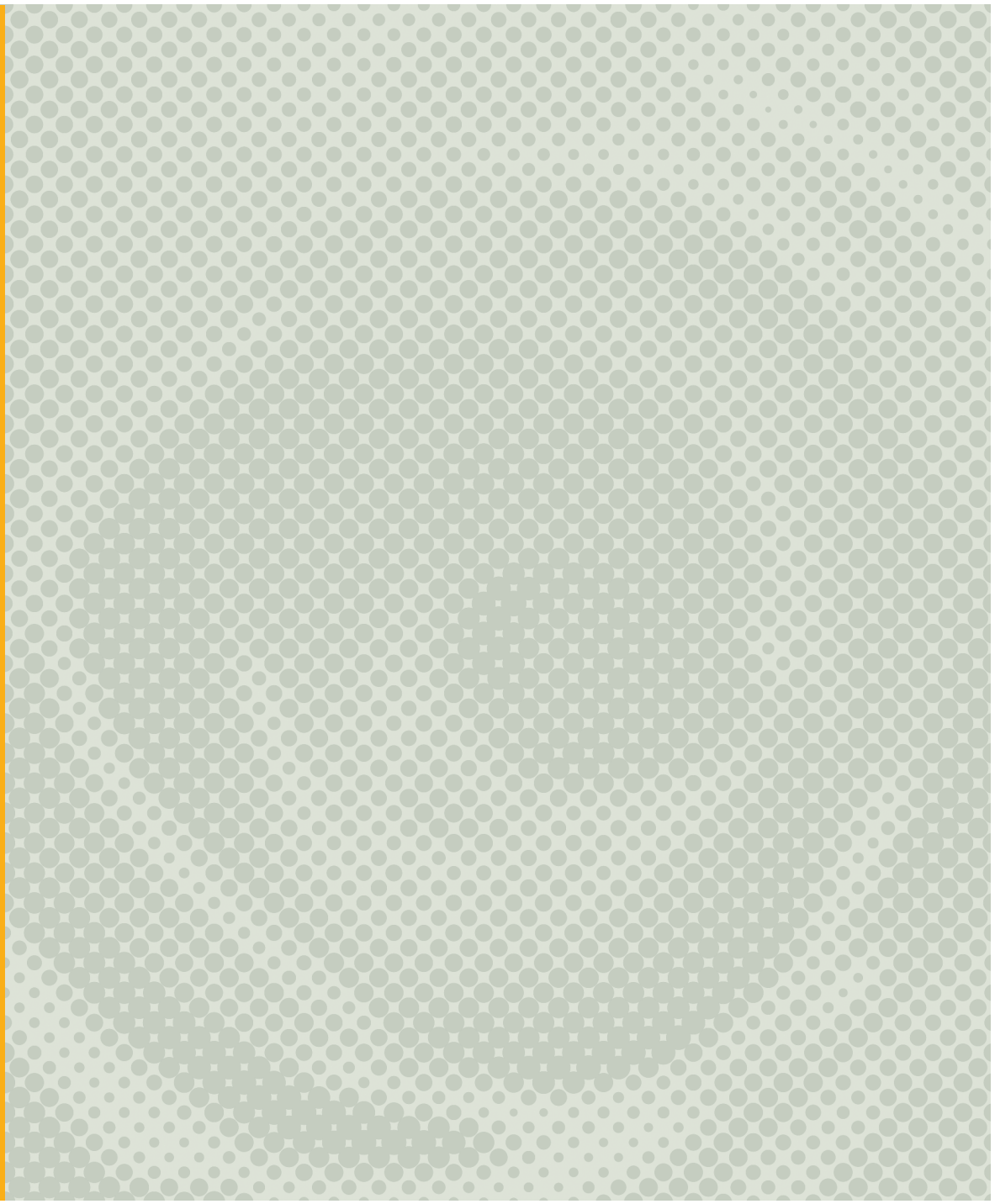
En verktøykasse for klassifisering av tilstanden til bygningar

A toolkit for classification of state of buildings

Kristian S. Førde

Universitetet for miljø- og biovitenskap
institutt for matematiske realfag og Teknologi
Masteroppgave 30 stp. Våren 2013



Forord

Denne masteroppgåva er avslutninga til mi utdanning innan Master i Teknologi, industriell økonomi med fordjuping innan energifysikk ved Universitetet for Miljø- og biovitskap, våren 2013. Saman med energifysikk i det ordinære studieløpet har eg og teke eit år ekstra for å få tid til å ta informatikkfaga biletebehandling og vidaregåande programmering. Desse informatikkfaga gav meirsmak til å utforska fleire bruksområde for ein vanleg datamaskin.

Etter samtale med rettleiar Knut Kvaal kom me fram til ideen om denne oppgåva som eg vart svært interessert i. Eg må samstundes få takke Knut Kvaal for særs god støtte, innspel og motivasjon.

Elles må eg og få takka Oliver Tomic for PCA-modulen og nyttige tips for programvareutviklinga, Thomas Thiis for engasjement og litteratur om frostskader og Ulf Indahl for rettleiing og tilrettelegging for LDA.

Vidare vil eg og takka venner og familie for støtte og motivasjon under arbeidet med oppgåva. Spesielt til Astrid Sandvik for å ta seg tid til å lese korrektur av tampen av skrivinga.

Ekstra hyggeleg er det å kunne skriva denne oppgåva på hovudmålet nynorsk i Språkåret 2013 og Ivar Aasen sitt 200 års jubileum.

Det har vore eit særs lærerikt halvår med eit bratt læringskurve for å knyte teori til praksis for fleire ulike programvaremiljø og -språk.

Kristian S. Førde, Ås, 14. mai 2013

Samandrag

Oppgåva tar sikte på å lage både ei verktøykasse for klassifisering av tilstanden til bygningar, i denne omgang på mursteinsbygningar. I tillegg skal verktøykassen brukast til å lage ein automatisk rutine for å identifisera frostskader på mursteinsveggar. Med informasjonsinnhenting, programvareutvikling og -testing har ein klart å utvikle ein verktøykasse. Hovudpunkta har vore å automatisere prosessen med å skilje ut murstein frå eit fargebilete, datainnsamling på identifisert murstein og analyse for å avdekkje eventuelle frostskader.

Ein god tilstandsrapport til slutt ser ut til å avhenge sterkt av 3 faktorar: eit godt eksponert bilete av mursteinsveggen, at mursteinane på bilete har tilstrekkelig storleik til å fange opp eventuelle skader og at programvaren klarar å skilje mursteinen frå fugen.

Abstract

This thesis aimed to make a toolkit for classification of the state of buildings, restricted to buildings with brick walls. An additionally task is to see if the developed toolkit is able to identify and recognize brick with freeze-thaw damages. With literature search, software development and testing has we been able to develop a toolkit. from color image of brick walls. With search for literature, software development and testing have I managed to develop a toolkit. The main task was to develop an automatic process to separate bricks from the wall out of a color image. Collect information from the bricks and at the end detect if the bricks have any frost damages.

A good finally report seems to depend on 3 factors. A well exposed image of a brick wall. The bricks on the image has sufficient size to capture any damages and the software is able to distinguish the brick from the wall.

Innhald

Alle figurar er laga av underteikna med mindre noko anna er angitt i figurteksten.

Figurliste

Tabelliste

Formelliste

1 Innleiing

Det er ynskjelig å lage ein prototype på programvare som tek for seg prosessen frå eit bilete av ein mursteinsvegg fram til ei tilstandsanalyse av kva grad vegg eller bygning innehar frostskader. Eit anna moment er at det vil vera interessant å sjå korleis veggen er samansatt basert på andre parameter som vil dukka opp i løpet av datainnsamling og køyring av ymse analyse.

Det kan vera fleire institusjonar og grupperingar som treng ei objektiv og enkel metode for å samla inn tilstandsanalyse av både eigne og/eller andre sine bygningar. Til dømes eit augeblikksbilete for å skaffe eit overblikk ved mistanke om frostskader før ei eventuell dyrare konsultasjon av fagfolk, kan vera greit.

Elles kan det vera svært nyttig å sjå korleis eit skadeomfang spreiar seg og utviklar seg over tid ved å foreta fleire målingar i ein serie over fleire år. Då er det viktig at dei innsamla data kan samanliknast mot kvarandre, spesielt sett opp mot den same veggen. Difor bør programvaren klare å plukke ut kvar einskilde murstein ut frå veggen og analysere den åleine, og moglegvis i samanheng med andre mursteinar på veggen. Det kan og tenkjast at forsikringsselskap eller vedlikehaldsentreprenørar vil ha raske rapportar før utbetring av mursteinsveggar.

Ved å ha tilgang til enkle metodar for å analysere ein mursteinsvegg kan ein fange opp frostskader i ein tideleg fase, før det vert store skader som følgje av øydelagde mursteinsvegger.

Ved å bruke fleire ulike metodar og verktøy for å kartlegge skadeomfang på mursteinsveggar, kan ein over tid samla eit godt bilete på tilstanden og utviklinga til slitasje på mursteinsbygningar. Då er det viktig å ha ei objektiv og enkel rutine for datainnsamling og prosessering. Vidare kan det nyttast som god dokumentasjon for å finne årsaker til skade og spreiing på murstein.

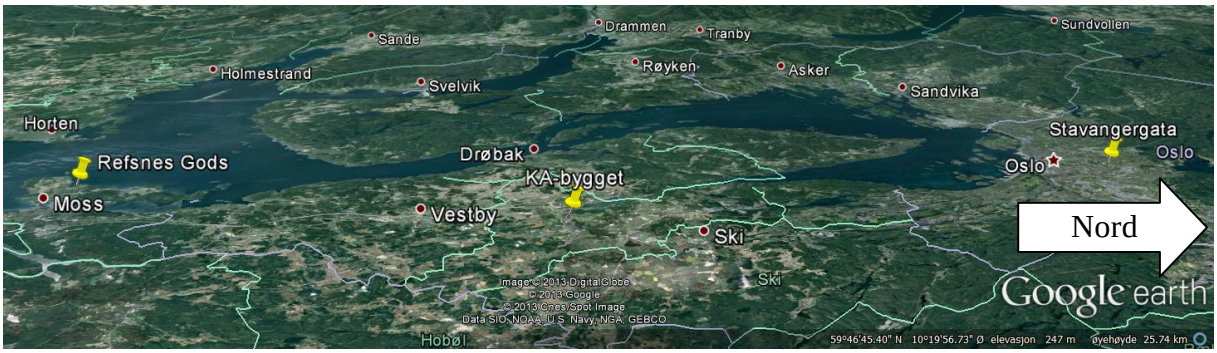
Etter punkta ovanfor vert det eit mål å sjå om det er mogleg å automatisere ein prosess som klarar å identifisera frostskader på ein mursteinsvegg ved å fotografere veggen. Då må kvar einskilde murstein identifiserast og analyserast før det kan setjast i ei større samanheng for heile veggen. For at dette skal vera mogleg er det viktig å leite etter eksisterande programvare, teknikkar, analysar og statistiske metodar som kan bidra til å oppnå ein tilstandsanalyse. Ved å setje saman og vidareutvikla programvaren og teknikkar kan ein oppnå ein enkel rutine og ei kraftfull verktøykasse for å identifisera murstein frå eit fargebilete. I tillegg vil verktøykassa kunne identifisera eventuelle frostskader på mursteinane. Deriblant vert det interessant å sjå om dette er oppnåeleg å automatisere til å få eit tilstrekkelig og pålitelig resultat av tilstanden på mursteinsveggen.

Resultata frå datainnsamlinga og analysen må til slutt vera samanliknbare for ulike kjelder, sett opp mot eit representativt variasjon i mursteinen si tilstand. Difor er det viktig å plukke ut murstein og lage ein skadeindeks som er upåverka av skalaforskjell med tanke på ulike kameratypar, synsvinklar og avstand til veggene.

Spørsmålet vert difor om det er mogleg å leggje til rette funksjonar og rutinar for å identifisera og synleggjere frostskader på murstein ut ifrå eit vanleg fargebilete. Tanken er då at holromma som oppstår av avskaling på frostskader vil gje tilstrekkelege skyggeeffektar og/eller fargeendring til å skilje seg ut på steinen. Programvaren skal på eit eller anna vis identifisera murstein og frostskaden frå kvarandre.

2 Teori

2.1 Bygningar for studie av frostskader og prototypetesting



Figur 2.1: Kart over lokaliseringa av dei 3 ulike datasetta som vert brukt i denne oppgåva. Alle punkta er på Austlandet og strekkjer seg frå Jeløy i sør med Refsnes Gods og Oslo i nord representert av blokker i Stavangergata. I midten er Kjemisk Analyse på Campus Ås og Ås kyrkje. Alle fire lokaliseringane er mursteinsbygningar. ***(Kart frå Google Earth)***

2.1.1 Stavangergata

I Stavangergata i Oslo står det 4 bustadblokker bygd i siste halvdel av 1960-tallet. Dei har 8-10 etasjar. Alle bygningane er mursteinsbygningar som er plassert oppå ein rygg i forhold til det nærliggande området. Dette førar til at bygningane står utsett til for vindkast som blant anna førar med seg regn [1]. Figur 2.2 visar korleis mursteinsveggene ser ut på ei av dei 4 blokkene i Stavangergata.

Blokkene i Stavangergata er interessante som testobjekt sidan det allereie har vore utført analyser og berekningar på effekten av vinddriven regn av Kyllingstad et. al i [1]. I tillegg til har Stavangergata synlege frostskader som denne oppgåva tek mål på seg å avdekkje automatisk. Det vert nytta bilete tatt på to ulike vegger (Figur 2.3) i Stavangergata teken på ulike dagar for å prøve ut programvaren på ulike forhold.

Figur 2.2: Sørligaste bygningen i Stavangergata med den synlege veggen vendt austover.

a) b)

Figur 2.3: Bilete av to vegger frå Stavangergata i Oslo. Fotograf for bilete a) Andreas Fløø, bilete b) Kristian S. Førde

2.1.2 Refsnes Gods

Refsnes Gods ligg på Jeløy i Oslofjorden, rett utanfor Moss. Godset er eit herskapelig gods bygd i 1767 med utviding av tårn i 1855 [2]. Godset har fleire interessante frostskader på murveggen som er lett synleg. Figur 2.4 visar Refsnes Gods som eit ærverdig gammalt bygg. For å halde på arkitektur- og bygningshistorie vil det vera interessant å sjå om programvaren klarar å kartlegge murveggen frå Refsnes.

Figur 2.4: Refsnes Gods på Jeløya i Oslofjorden. Foto: Knut Kvaal

a) b)

Figur 2.5: Mursteinsvegg frå Refsnes Gods med synlege frostskader og forstyrrande element i bilete i form av vasskran i bilete a). Bilete b) er eit nærmare bilete med tydlegare murstein. Foto: Knut Kvaal.

2.1.3 Kjemisk Analyse (KA-bygget), Campus Ås

Figur 2.6: Sørlege fasaden av Kjemisk Analyse bygger på Campus Ås der bilete til høgre vert nytta for identifisering av murstein.

Kjemisk Analyse på Campus Ås har ikkje særlig med synlege frostskader på den måten som denne oppgåva i utgangspunktet skal sjå på. Men bygningen er likevel interessant på grunn av den store variasjonen i farge og mønster på sjølve veggen. Dette gir gode moglegheiter til å teste programvaren på ein vanskelig mursteinsvegg. Det vert difor ein større test for å sjå kor godt programvaren klarar å skilje murstein frå fuge og anna støy under vanskelege forhold.

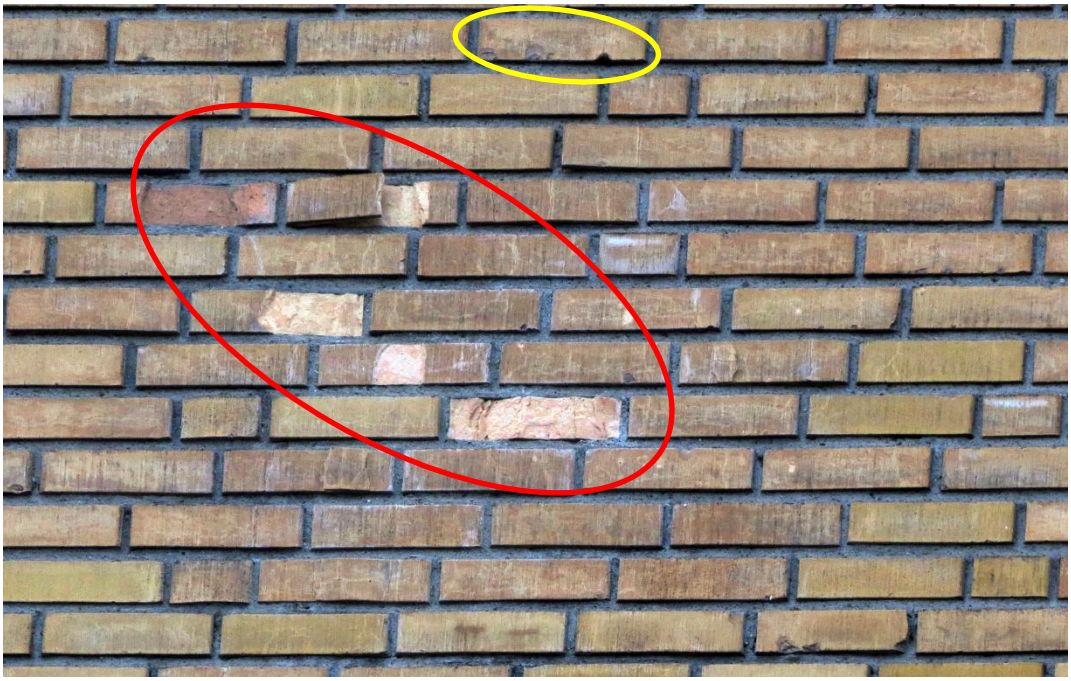
Samstundes kan ein nytte datainnsamlinga til å leite etter andre samanhengar internt på veggen.

2.2 Frostskade på murstein

I fuktige og kjølige klima som til dømes Noreg har, vil det vera gode forhold for frostskader på porøst materiale. Ved skiftande temperaturar rundt frysepunktet 0°C vil vatn fryse til is og tines tilbake til flytende vatn. Vatn har som kjent den eigenskapen at volumet utvidar seg når det går frå væskeform til fastform, is. Dette kan føre til ei rekkje små og store problem der regn kjem ned i sprekker og holrom på porøse materiale. Ved tilfrysning vil det oppstå krefter i form av spenningar på grunn av volumendringa til vatnet og det faste materialet rundt. Volumutvidinga til det fryste vatnet førar til at delar av mursteinen sprekk [3].

Ein vesentlig grunn til at vatnet kjem inn i steinen er vinddriven regn som treff mursteinsveggane og fyller dei porøse holromma med vatn[1]. Klimaet med omsyn på vær og vind i nærmiljøet har altså mykje å seie for forvitringa av mursteinane.

Figur 2.7 er ein del av ein vegg i Stavangergata, Oslo som synar svært godt fleire ulike stadium innanfor forvitring av mursteinsveggar. Markert med gul sirkel er det teikn til små biter av steinen som har flakka av på grunn av forvitring, medan det raude området er tydelig avskaling frå store deler av fasaden til steinen.



Figur 2.7: Døme på frostskader i ulike stadium frå Stavangergata, Oslo. Raud ring visar heilt avflakka stein, medan gul ring viser gryande frostskadar med avskala biter.

Faren ved store frostskader på veggen er at dei kan trenge seg eit stykke inn i veggen ved danning av nye groper og holrom. Skadane kan trengje tvers gjennom veggen og det vil oppstå lekkasje som kan føre til råteskader og muggsoppdanning på innsida av bygningen. Råteskader fører vidare til dårleg inneklima og rotning av trevirke i bygningen [4].

Frostskader utan behandling og vedlikehald kan dermed i siste instans føre til ubebolige bustadar grunna svikt i konstruksjon og dårlege buforhold. Samstundes er det og viktig å følgje med på korleis fasaden endrar seg, på til dømes verna bygningar og spesielle fasadar. I fleire samanhenger vil det og vera nyttig å følgje spreiinga av frostskader for å finne årsaker og andre uoppdaga ulemper som kan oppstå.

Renovering og skadeopprettning etter hussopp og andre råteskader kan verte svært kostnadskrevjande [5] og vanskelig dersom skadeomfanget vert oppdaga for seint. Det kan difor vera ein billeg forsikring å følgje med på korleis frostskader på vegger utviklar seg for å finne ut kva tid ein bør utføre vedlikehaldet.

2.3 Bilete teori

Digitale bilete kan i hovudsak delast inn i to forskjellige kategoriar. Ein er figurar som vert forklart ut ifrå matematiske formlar og geometriske figurar. Desse vert kalla vektorisert bilete og kan skalerast opp og ned i storleik utan større problem. I tillegg tek desse bilete lite plass

då det som oftast bare er få matematiske likningar som skal til for å representere dei. Problemet er derimot at desse bileta egnar seg dårleg til å framstille og prosjektere bilete slik som auga ser det. Derfor er det mest brukte metoden for å lagre og vise bilete å representere bilete som ei matrise, der kvart punkt i matrisa er ein piksel og verdien til denne pikselen representerer ein intensitet i bilete.

2.3.1 Fargemodellar

Matrisene som representerer eit bilete vil bare innehalde ulike talverdiar og er difor avhengige av ein modell for at pikselverdiane skal kunne representere ein farge og på den måten gi ein meining til bilete. Desse ulike fargemodellane kan i neste omgang nyttast til å trekke ut forskjellig informasjon sidan modellane opptrer ulikt og dermed vil spenne ut sitt tilgjenglige intervall på ulike måtar. Dette kan så utnyttast til å finne den, eller dei, modellane som skapar størst skilnad innanfor det fargespekteret ein er ute etter. Nokre fargemodellar vil kanskje gjere seg gjeldande i sjølve oppfatninga av fargetone, medan andre tek meir omsyn til gradering innan lysstyrke.

Binært bilete

Eit binært bilete vert representert med bare to pikselverdiar. Verdiane er enten 0 eller 1 for å representere bakgrunn og forgrunn i bilete. Pikselverdiane vil representere svart eller kvit der ein ofte vel svart som forgrunnsfarge og kvit som bakgrunnsfarge sidan dette samsvarar godt med utskrift på kvite ark [6]. Figur 2.13 visar døme på korleis eit fargebilete a) kan bli sjåande ut etter ei konvertering til eit binært bilete b).

Gråtone

I eit gråtonebilete vil pikselverdiane angi ein gradering av gråtonar mellom svart og kvit. Svart vil då vera den minste verdien, medan kvit er mest intensitet med høgast verdi. Det vanlege er å nytta seg av eit 8-bit per piksel, der maksverdien kan vera 2^8=256. Heile intervallet innanfor 8-bit vert då frå 0 og opp til og med 255 sidan 0 er eit teljande siffer. Dermed er det plass til 256 ulike nyansar av gråfarge mellom svart og kvit [6].

RGB, Raud, Grøn og Blå

Den mest vanglege måten å representere eit fargebilete i datamaskinen er med tre fargekanalar raud, grøn og blå. Kvar fargekanal er ei matrise med gråtoneskala, men målt på forskjellige intensitetar. Forskjellen mellom pikselverdien i dei tre ulike matrisene vil difor utgjere fargekombinasjonen. Dersom alle tre kanalane har same pikselverdi vil fargen vera ein gråfargeintensitet mellom svart og kvit.
RGB-modellen er ein kube der intensiteten spriker ut i raud, grøn og blå retning, der desse til saman utgjer ein farge. Men flytting rundt på kuben kan gje forskjellige utslag og det kan difor vera vanskelig å endre fargar i RGB-fargerommet.

Oftast vert det nytta 8-bit per fargekanal for å representere fargerommet i RGB. Difor vert det mogleg å representere heile $2^{8*3}=2^{24}=16\ 777\ 216$ ulike fargar. Dette vil i dei fleste tilfelle kunne gje eit tilsynelatande representasjon av alle moglege fargar.

sRGB

For å kunne konvertere RGB-fargerommet til andre fargerom er det i mange samanhengar viktig å ha det same referansegrunnlaget. Difor har Hewlett-Packard og Microsoft laga ein standard for korleis fargane i RGB skal sjåast ut. Denne standarden er kalla *sRGB* og tek omsyn til korleis RGB-fargane skal framstillast på tvers av ulike visnings- og lagringsmedium. Dette er naudsynt ettersom at RGB i utgangspunktet kan, og vil, ha litt ulike justeringar i fargen basert på fabrikkanten sin implementering. Det viktigast sRGB gjer er å angi kvar ulike lysstyrkar ligg på gråtoneskalaen i RGB-fargerommet [6].

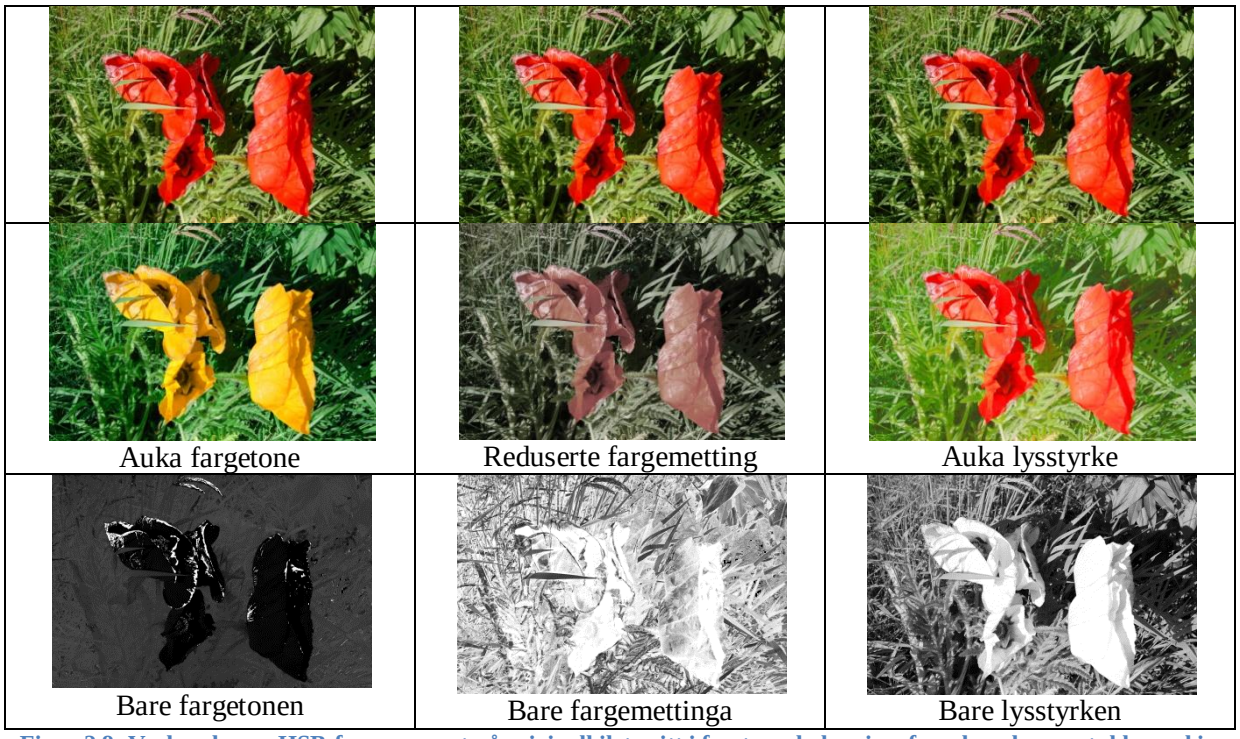
HSB, Fargetone(Hue), -metting(Saturation) og lysstyrke(Brightness)

Ein annan måte å representere fargar på, er med HSB-fargerommet som og er tre matriser med ulike verdiar. Forskjellen er at matrisene representerer ulike delar av fargene på ein annan måte enn RGB.

Hue står for fargetone, intensiteten her vil gi utslag på kva farge som skal nyttast i kvar einskilde piksel. Det er difor veldig lett å skifte ut heile fargen i eit bilete ved å endre intensiteten i fargetone. Figur 2.8 visar blant anna kor enkelt ein kan skifte farge på den raude blomen til gul blom. Einaste som er gjort er å auke fargetonen Hue. Av same figur ser ein og tydleg at grøn og raud tilhøyrer forskjellige endar av skalaen til fargetonar.

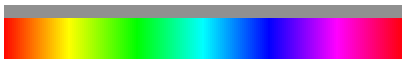
Saturation er fargemettinga til bilete og forklarar kor mykje det skal nyttast av ein farge. Der fargemettinga er 0 vil det bare vera eit gråtonebilete att, medan full fargemetting vil gje skarpe fargar. Figur 2.8 visar eit døme på korleis ein kan nytta ein reduksjon i fargemettinga til å framstille bilete som litt tristare, tamt og fargelaust.

Brightness er lysstyrken til eit bilete og gir same utslag som å skru av eller på ei lampe som metafor. Figur 2.8 visar at lysstyrken aleine gir ei svært god representasjon av gråtonebilete.



Figur 2.8: Verknaden av HSB-fargerommet på originalbilete gitt i første rad, der ei av fargekanalane er tukla med i andre rad. Siste rad visar orginalbilete med bare informasjon frå den gitte fargekanalen.

Forholdet mellom dei tre ulike kanalane kan best symboliserast som ein sylinder. Omkrinsen av sirkelen i enden symboliserer ulike fargetonar ein kan velje mellom som vist i Figur 2.9. Der fargetonen vil vera vinkelen mellom aksen mot raud og vald farge.



Figur 2.9: Fargetonar ein kan velje i fargerommet HSB

Konvertere frå RGB til HSB

Det er fullt mulig å konvertere frå RGB-fargerommet til HSB ved å køyre dei ulike matrisene som fargerommet består av gjennom ulike formlar som går igjennom pikselverdiane R,G,B for høvesvis pikselverdien i raud, grøn og blå fargekanal. Framgangsmåten er henta frå Burger & Burge [6].

$$C_{high} = \max(R, G, B)$$
$$C_{low} = \min(R, G, B)$$
$$C_{rng} = C_{high} - C_{low} \qquad \text{Formel 2.1}$$
$$C_{max} = den\ største\ moglege\ intensiteten, ofte\ 255$$

Formel 2.1: Startverdiane i konvertering frå RGB-fargerommet til HSB, der R,G,B er pikselverdien for dei ulike fargekanalane i RGB.

Ein byrjar først å rekne ut fargemettinga:

$$S_{HSB} = \begin{cases} \frac{C_{rng}}{C_{high}}, C_{high} > 0 \\ 0, C_{high} \leq 0 \end{cases}$$ Formel 2.2

Formel 2.2:Utrekning av fargemettinga i konvertering frå RGB til HSB. Der C_high høgste intensitet i pikslen, , C_low er minste verdi i pikselen medan C_rng er differansen mellom desse.

Etterpå vert lysstyrken utrekna baser på dei same tala:

$$B_{HSB} = \frac{C_{high}}{C_{\max}}$$ Formel 2.3

Formel 2.3: Utrekning av lysstyrken i konvertering frå RGB til HSB. C_max er den største oppnåelige intensiteten i fargerommet (Oftest 255) medan C_high er den høgaste pikselverdien av dei tre ulike fargekanalane RGB

For å rekne ut sjølve fargetonen krev det litt fleire ledd for å få alt på plass. Først av alt må dei tre fargekanalane normaliserast:

$$R' = \frac{C_{high} - R}{C_{rng}}$$
$$G' = \frac{C_{high} - G}{C_{rng}}$$
$$B' = \frac{C_{high} - B}{C_{rng}}$$ Formel 2.4

Formel 2.4: Normaliserer pikselverdien i RGB

$$H' = \begin{cases} B' - G', & R = C_{high} \\ R' - B' + 2, & G = C_{high} \\ G' - R' + 4, & B = C_{high} \end{cases}$$ Formel 2.5

Formel 2.5: Skiljer dei ulike fargetonane frå kvarandre i konvertering frå RGB til HSB

Sida verdiane i H' nå ligger mellom [-1,5] kan det vera greit å normalisere denne til å vera mellom [0,1]:

$$H_{HSB} = \frac{1}{6} \cdot \begin{cases} (H' + 6), & H' < 0 \\ H', & H' \geq 0 \end{cases}$$ Formel 2.6

Formel 2.6: Fargetonen H i HSB rommet på normalisert form

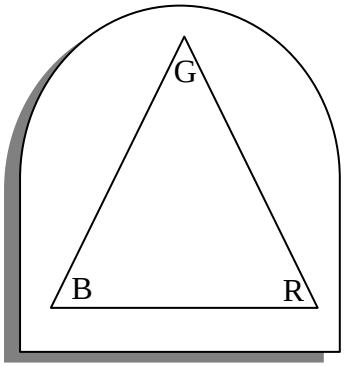
Etter at fargetonen er på normalisert form er det og enkelt å endre denne til å representere til dømes ein sirkel ved å multiplisere den opp med 360 grader. Elles kan det nemnast at ytterpunkta 0 og 1 inneheld den raude fargen som ein kan sjå i Figur 2.9.

CIELab

Dei føregåande fargemodellane har alle til felles at dei er prisgitt mediet for framsyning for korleis fargene vil sjå ut. Difor finnes det standardiserte fargemodellar for å sikre at alle som nyttar det har same utgangspunkt og same fargar. Dei mest brukte standardane kjem frå Den internasjonale kommisjon for belysning(CIE). Etter mykje kontrollert testing av mennesket sitt syn på fargar har dei kome fram til ei standardisert fargemodell basert på fargelæra frå fysikken.

Fargemodellen har då kome fram til ein representasjon som ser ut som ein hestesko der fargane ligg rundt ytterkanten til hesteskoen basert på bølgjelengda til fargeskalaen som regnbogen. For å avgjere lysstyrke har og CIE ei eige akse for dette. Lysstyrken går normalt på denne hesteskoen der låg lysstyrke vil gje eit lite areal på hesteskoen, og dermed færre nyansar innan fargerommet. Ved auka lysstyrke vil og radiusen frå midtpunktet av hesteskoen og ut til randsonen verte større med tilhøyrande auka areal av hesteskoen. Dermed får ein og auka distansane mellom fargane og gjeve rom for fleire nyansar innan fargespekteret.

Til dømes ligg RGB kanalane som ein trekant innanfor denne modellen. Raud ligg nede i ytste hjørna med høgast bølgjelengde, grøn er i toppen av hesteskoen, medan blå ligg nesten nederst i motsett ende av raud, ei særs enkel framstilling kan ein sjå i Figur 2.10. Hesteskoen i figuren representerer bogen der ulike bølgjelengder spenner seg rundt frå venstre hjørna, opp til toppen og ned igjen til høgre hjørne med aukande bøljelengde. Rekkefølgja til fargane er svært lik Figur 2.9 der raudfargen til venstre er ved punktet R i hesteskoen. Dei påfølgjande fargane legg seg rundt til grønt i toppen, blå i venstre hjørne og indigo i midten mellom B og R i Figur 2.10.

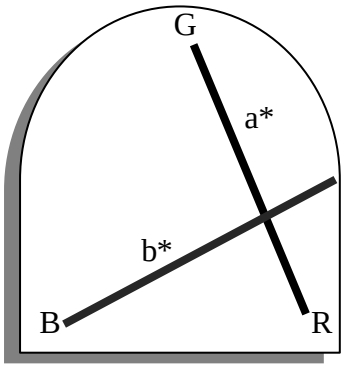


Figur 2.10: Enkel skisse over tverrsnittet i spennet til fargerommet CIE. Hesteskoen representerer fargerommet der bølgjelengdane til dei ulikefargane spenner seg frå låg i venstre hjørna(B) og går rundt til midtpunktet i toppen med høgaste bølgjelengder i høgre hjørne ved R. Trekanten inni er spennet til RGB-kanalane i forhold til CIE-fargerommet.

CIE-fargeromma, spenner ut alle moglege fargar som det menneskelige auge kan oppfatte. Det er difor nært umogleg å ta inn heile fargemodellen direkte inn i den digitale verden. Likevel har dei fleste datamaskinar i dag stor nok kapasitet til å servere mange nok fargar til at det vert oppfatta som om mest alle fargar kan nyttast.

Ei av standardane som har kome frå CIE er CIELab der dei trefargekanalane er L*a*b*. L* er for lysstyrken og a* og b* står for to ulike fargeretningar. Både a* og b* tar innover seg

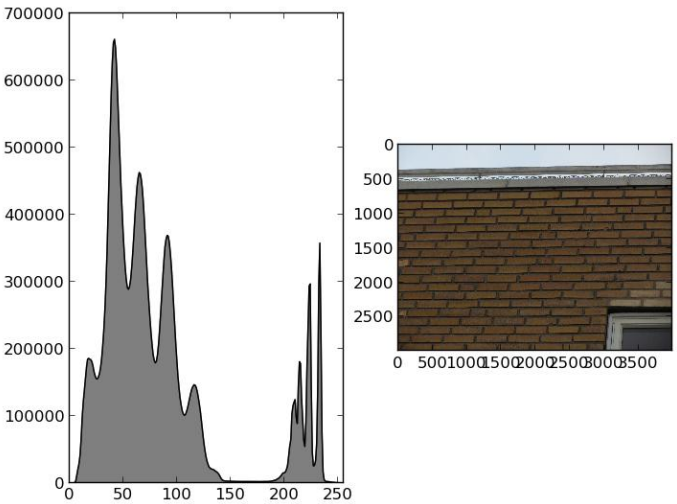
fargetonar og fargemettinga for høvesvis grøn-raud skalaen for a* og blå-gul skala for b*. Dette fargerommet oppsto etter eit ynskje om å ha ein fargeskala som representerer fargene lineært med tanke på menneskelig oppfatting av fargerommet [6] [7]. Den enkle skalaen innanfor hesteskoen er representert i Figur 2.11.



Figur 2.11: Enkel skisse på korleis Lab spenner seg ut i CIE fargerommet der L* går normalt på planet medan a* og b* kryssar kvar sin akse innan tverrsnittet.

2.3.2 Histogram

Histogrammet av eit bilete forteljar korleis mengda av pikselverdiar fordelar seg ut over tilgjengelig intervall. Intervallet er ofte representert i 8-bit og har då 256 ulike punkter. Visualiseringa av histogrammet gjev eit godt innblikk i distribusjonen og tettleiken til pikselverdiane i eit bilete. Ein kan difor nytta histogrammet til å sjå om bilete har klart å utnytte heile spekteret av intensitetar og om det er enkelte område av pikselverdiar som skiljer seg meir ut enn andre. Til dømes vert det enkelt å sjå på gråtonehistogrammet at eit bilete er mørkt dersom det er mykje låge intensitetsverdiar. Eller motsatt at det er lyst med mykje høge intensitetsverdiar. Eit histogram har mange ulike formar og kan på denne måten vera nyttig til klassifisering av ulike kategoriar i eit bilete [8].



Figur 2.12: Døme på eit histogram som utnyttar heile spekteret ved å ha både lyse og mørke partiar. Samstundes er det ein overvekt av dei mørke fargane med låg intensitetsverdiar på x-aksen. Y-aksen visar kor mange pikslar som innehar kvar pikselverdi.

Figur 2.12 viser eit histogram av ein mursteinsvegg med ei kvit region på toppen. Frå histogrammet kan ein lese at fleirtalet av pikselverdiane i bilete har ei litt mørk framtoning sida dei fleste pikselverdiane er mellom 0 og 150. Desse verdiane vil då representere sjølve mursteinsveggen. Det lyse partiet med intervall frå ca 190 og opp til 240 vil vera dei lyse partia som i hovudsak vil vera den kvite regionen på toppen og ramma til vindauget i venstre hjørna.

2.4 Tekstur på bilete

Alle bilete er ei prosjektering av røynda. På biletet er det vanskeleg å kjenne på overflater for å finne ut om dei er ru eller om det er glatt. I staden for kan ein sjå på pikselverdiar til å bestemme mykje av dei same eigenskapane. Det er fleire typar teksturar som vil dukka opp på eit bilete som til dømes blank, kornete eller systematisk.

2.5 Representasjon av bilete

Eit bilete tatt med dagens digitalkamera, og eventuelt scannere, vert lagra som eit rasterbilete. Det vil seie at bilete vert lagra som ein rektangulær matrise der kvar celle i matrisa forklarar kva farge bilete har i akkurat denne cella. Desse cellene vert kalla ein piksel i eit bilete. Det er fleire metodar å lagre eit slikt bilete på med varierande grad av reduksjon av filstorleik på harddisken. Innan komprimering opererer ein med komprimering utan tap av informasjon eller komprimering med tap av informasjon. Ved komprimering med tap vil pikselverdiane kunne verte endra ut ifrå originalverdien.

Kvar pikselverdi kan hentast ut frå bilete ved å velje koordinatane til pikselverdiane. Alle pikselverdiar kan hentast ut frå matrisa som representerar bilete ved å angi x og y verdiar innanfor høgda og bredda til bilete. For eit bilete I vil ein dermed skrive I(x,y) for å få fram pikselverdi i akkurat denne cella.

2.5.1 JPEG

Bileteformatet jpeg er det mest brukte formatet for lagring av bilete i dag. Formatet vert nytta av så å seie alle typar digitale kamera og programvare. Dette er fordi jpeg-formatet er laga for å kunne komprimere bilete til å ta mykje mindre plass på harddisken. På grunn av denne kraftige reduseringa av storleiken på fila er det mykje nytta i konsumentmarkedet for å kunne ha plass til fleire bilete på til dømes minnekort i kamera og på harddisken. Diverre er komprimeringa i jpeg med tap av informasjon.

Problemet er at i denne prosessen vert det mista informasjon frå bilete som kunne vore nytta i til dømes analyser. Grunnen til at JPEG er eit destruktivt format er måten det bearbeidar bileta på før og etter lagring. For at formatet skal ta mindre plass nyttar formatet seg av teknikkar frå bl.a. psykologien til å redusere i områda der mennesket ikkje oppfattar endringar så godt. Saman med å redusere på område der menneskeauga er minst følsamt, nyttar formatet

cosinus-likningar til å forklare bilete bolkvis. Dermed vil mykje av kvaliteten til bilete verta avgjort av kor mange likningar ein vil nytta til å forklare bilete. Men akkurat på grunn av at bilete vert gjort om til fleire mattematiske likningar vil bilete meir eller mindre alltid miste litt informasjon for kvar gong bilete vert lagra, då desse likningane må lagast på nytt [6]. Det er ved store endringar i pikselverdien at informasjonen vil verte svekka. Desse kantane i bilete vil ofte vera der ein går frå murstein til fuge. Ved å ha ei høg komprimering vil bilete vera meir glatta ut og forskjellen mellom desse to områda vil vera litt vanskelegare å oppdaga [6].

Ein bør difor ikkje nytta jpeg-formatet til å mellomlagra bilete ein nyttar i prosesseringa av bileta, då bilete ikkje vil vera det same mellom to arbeidsoppgåver. Men så lenge utgangspunktet er av høg kvalitet og komprimeringa lita vil det vera mogleg å nytte jpeg-bilete til å avdekkje interessante område frå bilete.

2.5.2 TIFF

Tiff formatet er eit superformat for lagring og deling av bilete ettersom at det er eit veldig fleksibelt format. Formatet kan nyttast på fleire ulike måtar med tanke på komprimering og utforming. Den store fleksibiliteten til formatet førar samstundes til at det ikkje er naudsynt at all programvare støttar alle variantar og funksjonar av tiff. Den største styrken i denne samanhengen er uansett at bilete lagra i Tiff-formatet kan lagrast utan tap av informasjon sidan det godtek fleire ikkje destruktive komprimeringar [6].

Stabel av bilete / biletestack

Ved fleire høve vil bileta omhandle det same eller dei bør av andre årsaker vera samla. Dette kan då løysast ved å lagre alle bileta med same dimensjon i ein stabel som eigentleg vert ei liste av bilete. Dette vert kalla ein biletestack.
Tiff formatet har støtte for å lagre bilete på denne måten slik at til dømes alle objekt som kjem frå i eit stort bilete kan samlast saman i ei liste.

2.5.3 EXIF

EXIF er sjølve formatet som vert nytta i lagring av bilete og er eit samleformat for å lagre biletet saman med tilleggsinformasjon om bilete. Tilleggsinformasjon omhandlar alt rundt bilete som informasjon om til dømes kameraprodusent, dato og om bilete vert tatt i landskapsmodus (bredde > høgde) eller portrettmodus (høgde > bredde). Fleire av dagens kamera kan i tillegg leggje til GPS-koordinatar, medan det finnes fleire metodar for å leggje til koordinatar i ettertid.

2.6 Teknikar for behandling av bilete

I prosessering av bilete er det fleire teknikkar som vert nytta for å få fram effektar og å skilje ut ulike område i bilete. Det er naudsynt å nytta teknikkane med omhug og justera bruka for å få fram interessante effektar og regionar i bilete.

2.6.1 Terskling (threshold)

Terskling av eit bilete vil seie å sortere ut områder basert på ein pikselverdi. Bilete vert konvertert frå ein gradering av gråtoner til eit svart/kvit bilete. For å avgjere kva som vert svarte piksler og kvite piksler vert det nytta ein pikselverdi som eit vippepunkt. Fargene vert svarte dersom pikselverdien i bilete er mindre enn den førehandsbestemte verdien og kvit dersom pikselverdien er høgare [6].

Formel 2.7 visar terksling med eit enkelt vippepunkt som forklart over:

$$p = \begin{cases} a_0, & a \leq a_{th} \\ a_1, & a \geq a_{th} \end{cases} \qquad \text{Formel 2.7}$$

Formel 2.7: Terskling med eit tersklingspunkt ath der a er pikselverdien ein ser på, a0 er nedre fargeverdi og a1 er øvre fargeverdi

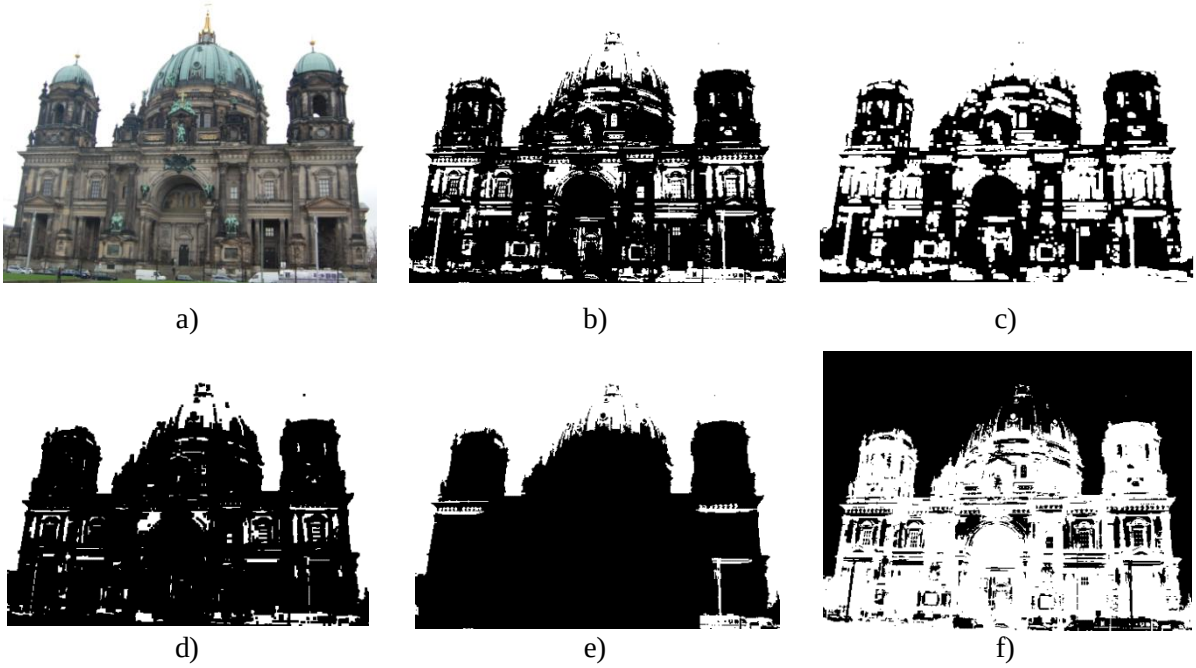
Elles kan det vera eit bestemt område ein er ute etter. Då kan ein seie at innanfor eit intervall vert det ein farge som svart, medan den nedre og øvre delen vert til dømes kvit. Formel 2.8 forklarar korleis dette vert gjort reint matematisk.

$$p = \begin{cases} a_0, & a \leq a_{th0} \\ a_1, & a_{th0} < a < a_{th1} \\ a_0, & a \geq a_{th1} \end{cases} \qquad \text{Formel 2.8}$$

Formel 2.8: Terskling med både øvre og nedre verdi, der a er pikslen me ser på, a0 er minste, a1 er høgste pikselverdi og ath er høvesvis nedre og øvre terskelverdi

2.6.2 Morfologi

Morfologi er ulike teknikkar som vert nytta på i hovudsak binære bilete. Operasjonane vert utført på bilete for å gjere regionar og detaljar i forgrunnen tydlegare. Dette vert gjort ved blant anna å fjerne støy eller få markert regionane skikkelig.



a) b) c)

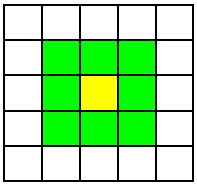
d) e) f)

Figur 2.13: Bilete av Berlinerdomen, a) original fargebilete, b) binært framstilling, c) erosjon på binært bilete, d) utvidning av bilete, e) er nytta fyll holrom og f) er invertert binær bilete.

Invertere(invert)

Skifter om på bilete der forgrunnsfargen svart vert til bakgrunnsfargen kvit medan den tidlegare kvite fargen nå vert svart. Dette gjere ein for at det skal verte lettare å sjå kva som er objekt og kva som er bakgrunn dersom fargane har vore omvendt på førehand. Døme på invertering av binært bilete kan ein sjå av Berlinerdomen i Figur 2.13f) .

Utvide(dilate)



Figur 2.14: Illustrasjon på naboskap frå ein sentrert piksel markt i gul. Naboskapet for den sentrerte pikselen er grøn og resten av bilete er kvite piksler.

Ved utviding i eit bilete ser ein på naboskapet til piksler som illustrert i Figur 2.14. Når pikselen, markert med gult i figuren, skal sjekkast for verdi, undersøkjar ein naboskapet rundt i eit 3 x 3 område, markert med grønt i figuren. Den pikselverdien i naboskapet som har høgast pikselverdi, vert den nye verdien for pikselen i midten. Effekten av dette er at objekta i bilete aukar i omfang, altså utvidar grensene sine. Det er tydelig i forskjellen mellom det

binære bilete Figur 2.13b) og utvidninga Figur 2.13d) at objektet utvidar sitt areal og grense mot bakgrunnen.

Erosjon(erode)

Erosjon på eit binært bilete er det motsette av utviding. Ein ser på naboskapet i ei 3 x 3 matrise rundt midten. Pikselen i midten av matrisa nyttar den verdien som er lågast i naboskapet på 3x3. Dermed får ein den effekten at areal og grenser av objekta krympe. Ved små partiklar i bilete kan desse forsvinne heilt frå bilete. Erosjon kan difor vera nyttig til å fjerne små partiklar i bilete. Figur 2.13c) er eit døme på erosjon frå eit binært bilete der det kvite bakgrunnsfeltet vert utvida, medan objektet i svart krympar.

Fjern støy(Remove outliers)

Ein annan måte å redusere for små partiklar i maskebiletet, er å fjerne støy. Teknikken leitar etter regionar i bilete som er mindre ein gitt storleik. Dersom regionen er større enn den gitte storleiken får den vera forgrunn som den allereie er, medan for små regionar vert omgjort til bakgrunn og dermed forsvinn frå biletet. På denne måten kan ein ganske effektivt fjerna uavhengige regionar som eigentleg ikkje skal vera til stades.

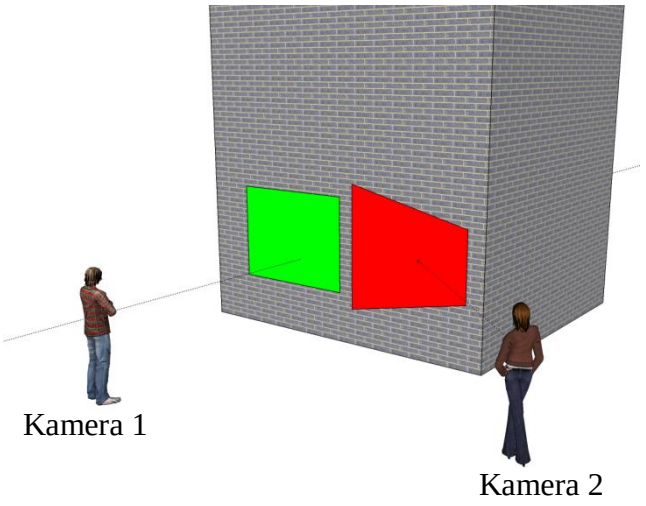
Fyll hol(fill holes)

Fylle holrom vil seie at ein går gjennom bilete og registrerer holrom til objekta, dvs bakgrunnsfarge som ikkje heng saman med yttarste kantane av hovudbilete. Etter å ha identifisert holromma, startar funksjonen å fylle igjen holrom med forgrunnsfargen slik at regionane vert eit heilt stykke utan holrom. Etter denne operasjonen er ein dermed sikker på at ingen blobar har interne holrom, og alle bakgrunnspikslar har fritt leide ut til kantane av bilete. Figur 2.13e) er eit døme når ein stenger igjen alle holrom inne i regionane, medan område som har fritt leide til kanten framleis får vera bakgrunnsfargen(kvit).

2.6.3 Fordreie perspektiv(Warp perspective)

Å fordreie perspektivet i eit bilete vil seie å endre synsvinkelen til bilete ved å prøve å dreie bilete til eit nytt synspunkt. For å få til dette trengs det ein 3 x 3 transformasjonsmatrise som er utrekna basert på endringane til dei fire hjørnepunkta i biletet. Framgangsmåte og algoritmen er henta frå Solem [9].

Ved relativt små endringar, som endra synsvinkel, fungerer denne metoden bra til å korrigere for projeksjonen av bilete. Figur 2.15 er ein illustrasjon på korleis perspektivet endrar seg etter kvar bilete vert tatt. Kameraet tar eit rektangulært bilete og vil dermed prosjektere det ein ser gjennom linsa til eit rektangel. Bilete vil for kamera 1 og 2 verte det høvesvis grønt og raudt område som dekkar prosjekteringsflata for bilete til dei to kamera.

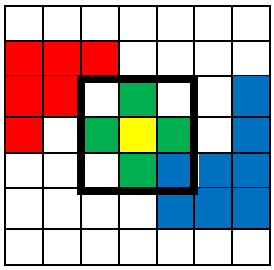


Figur 2.15: Perspektiv i forhold til synsvinkel der kamera 1 får tatt bilete av muren normalt på, representert av det grøne område. Kamera 2 får derimot ei forskyving på synsvinkelen og får dermed eit forskyve bilete av muren, representert av det raude området.

2.6.4 Blob

Blobar er regionar i eit bilete som er skilt ut som eigne objekt. På eit maskebilete vil ein blob vera områda som står for seg sjølv og i denne samanhengen vil det verta kvar einskilde murstein i bilete. Kvar blob vil inneha ei rekkje eigenskapar som kan samlast inn. Desse blobane vert difor svært viktige for å halde reie på kvar einskilde murstein frå kvarandre. Eigenskapane vil vera knytte til kvar einskilde blob og vera målingar som omkrins, areal, bredde/høgde forhold til fargesamansetning, grenseinndeling i bilete. Det kan hentast inn data på mange forskjellig vis og det er i hovudsak tre ting dei kan og vil handla om; form, pikselintensitet og plassering på originalbilete.

For å avgjere grensene til blobane er det to måtar å leite etter samanhengande regionar. Ein kan enten sjå på eit naboskap med 4 nabopikslar for å sjekke om pikslane heng saman i same region. Ved eit 4-kobla naboskap har ein bare naboar i fire retningar, opp, ned, høgre eller venstre. Den grøne regionen i Figur 2.16 vil bare ha naboskap med den blå regionen ved 4-koblet naboskap. Dersom det er 8-koblet naboskap er det 8 ulike nabopiksler, dvs. alle innafor det markerte rektangelet i Figur 2.16 er i naboskap med den gule pikselen. For den grøne regionen betyr det at den er i naboskap med den raude region, sidan grensene møtes på diagonalen.



Figur 2.16: Fargeskilte regionar som heng saman i ein region ved 8-kobla naboskap(markert med svart rektangel), medan det er 2 regionar ved 4- kobla(markert med grøne piksler) naboskap sett ut ifrå midtpikselen i gult.

2.7 Analyse og datainnsamling på bilete

2.7.1 Partikkelanalyse(Particle analyzer)

I ei partikkelanalyse leitar ein etter regionar i eit binært bilete basert på forgrunnsfargen. Bilete som vert nytta er eit maskebilete som er eit binært bilete, klargjort for å identifisere blobar. Det første som skjer er at alle regionar i forgrunnen vert identifisert og gitt kvart sitt nummer som ein region.

Det er fleire metodar for å finne desse regionane, men det handlar om å traversere gjennom bilete til ein treffer ein forgrunnspiksel. Då går ein gjennom alle naboskap til pikselen for å sjekke om forgrunnen heng saman med andre forgrunnspikslar. Når heile regionen er identifisert, traverserer ein vidare til neste region vert oppdaga. Dersom forgrunnspiksel vert oppdaga settes det eit nummer på pikselen som høyrar til regionen. Dermed hoppar ein over allereie sjekka pikslar sidan ein leitar etter forgrunnspiksel (=1), medan regionane bør vera merka frå 2 og oppover [6].

Då regionane er identifisert byrjar ein å finne grensene til regionen ved å gå rundt kanten av regionen. Etter dette byrjar analysen å rekne ut eigenskapar som areal og omkrins på regionane. I tillegg vil den rekne ut gjennomsnitt, median, standardavvik, maksimum og minimum av pikselintensitetn til regionen. Under køyringa vil ikkje dette ha nokon effekt sidan pikselintensiteten er forgrunnspikselen 1 heile vegen. Derimot kan ein skifte underlagsgrunnlag til fargekanalane og køyre statistiske målingar på pikselintensitetane for kvar fargekanal for kvar blob. Då vil ein få samanliknbare pikselintensitetar for alle fargekanalane inkludert gråtoneskalaen.

2.7.2 Angle Measure Technique(AMT)

AMT lagar ei ny måleeining av ein 1-dimensjonal måleserie. Denne nye måleeininga er basert på skala og gir ny informasjon om måleserien basert på denne skalaverdien. Teknikken vart først introdusert av Robert Andrle i 1994 [10] og etterkvart nytta i fleire ulike fagfelt innan analyse av måleseriar.

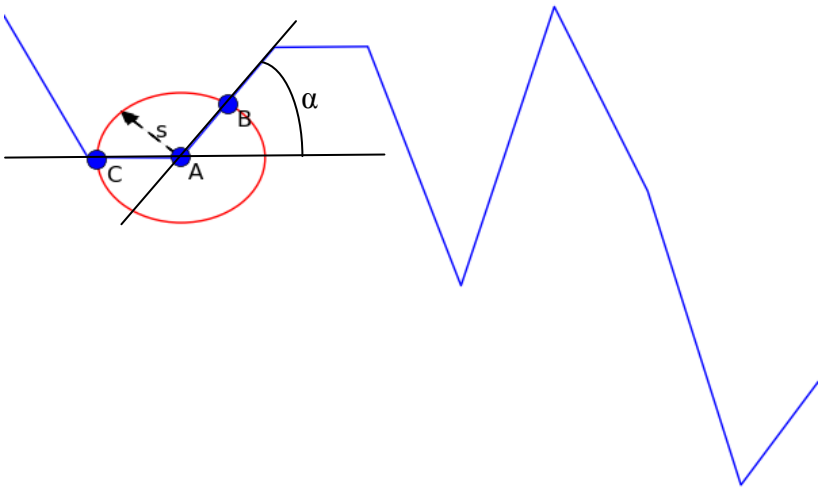
Å bruke AMT på bilete vart trekt fram som eit døme av Esbensen et. al. [11]. I tillegg viste Kvaal et. al [12] at AMT passar godt til å hente ut eigenskapar i ei teksturanalyse.

AMT vel tilfeldige stader på måleserien og lagar då ein sirkel med radius s med senter i punkt A på signalet, som vist i Figur 2.17. På skjeringspunkta mellom måleserien og sirkelen set ein punkta C og B. Dermed kan ein måle vinkelen til CAB og supplement vinkelen (180°-CAB) som er resultatet. Dette gjer ein for like mange punkter som det er ynskjelig å lage ein skala for. Mengda, n, tilfeldige punkt bør vera tilstrekkelig for å forklare signalet.

$$MA(s) = \frac{1}{n}\sum_{i=1}^{n}(\alpha_i)_s \qquad \text{Formel 2.9}$$

Formel 2.9: Gjennomsnittsvinkelen (MA) for kvar radius (s) er gjennomsnittet av vinkel α funnet for n-punkt med radius s.

Vidare aukar ein radius s opp til eit førehandsbestemt maks radius som ikkje kan vera større enn lengda på måleserien, etter kor mange målepunkt ein har teken med. Resultatet frå måleserien er oftast gjennomsnittsvinkel (MA) omtalt i Formel 2.9. Sidan ein får eit målepunkt per radius s er det maks radius som bestemmer kor mange målepunkt AMT vil gje. Andre måleeiningar frå måleserien kan vera den gjennomsnittlig endringa i x-(MDX) eller y-retning(MDY). Ved dei to siste måleeiningane er det forskjellen mellom punkta C og B frå Figur 2.17 i høvesvis x- og y-retning som vert registrert i staden for vinkelen α i Formel 2.9.



Figur 2.17: Illustrasjon på arbeidsmåten til AMT der signalet ein ser på er den blå streken. Etter eit tilfeldig valt punkt A trekker ein ut (raud) sirkel med radius s og plasserer punktane C og B i skjeringspunkta mellom sirkelen og signalet. Supplement vinkelen til BAC, α.

2.7.3 Grey Level Co-occurrence Matrix (GLCM)

Haralick [13] publiserte i 1973 ein generell metode for å hente ut teksturinformasjon om eit bilete omtalt som Grey level Co-occurence Matric (*GLCM*).

GLCM er målingar basert på ei ny matrise som baserer seg på naboskap av piksler i gråtonebilete. Matrisa er bygd opp med å legge til kor mange piksler med ein bestemt intensitet som er nabo med ein annan intensitet. Ved oppteljing vert desse lagt inn i ein matrise. Til dømes vil ein sjå på piksel x, y som har intensitet i, og nabopikslen x2,y2 som har intensiteten j. Dermed vil GLCM matrisa V ha ein auke i punktet V(i,j). Etter at alle naboskap i bilete er kartlagt, vert det utført fleire målingar på den nye matrisa P [14].

Etter kartlegging av naboskapet verdt det utført fleire utrekningar på matrisa for å hente ut informasjon om tekstureigenskapar. Forklaringar og formlar på dei ulike eigenskapane er henta frå ei god læringssida om GLCM som er skrive av Beyer [15].

Det første er at ein normaliserar naboskapmatrisa V med Formel 2.10.

$$P_{i,j} = \frac{V_{i,j}}{\sum_{i,j=0}^{N-1} V_{i,j}} \qquad \text{Formel 2.10}$$

Formel 2.10: Normalisering av GLCM matrisa (V) til normalisert form(P), som samstundes er svært nært til sannsynsmatrisa for pikselintensiteter, der N er mengden pikselintensitetar som er mogleg, oftast 256.

Ein konsekvens av normaliseringa i Formel 2.10 er at matrisa vert veldig nært ein matrise for sannsyn til ulike pikselintensitetar i bilete.

Angular Second moment (ASM) fortel noko om kor uniform fordelinga av pikselintensitetane er i bilete. ASM fortel noko om kor mykje orden og system det er i biletet.

$$ASM = \sum_{i,j=0}^{N-1} P_{i,j}^2 \qquad \text{Formel 2.11}$$

Formel 2.11: Utrekning på kor uniform eit bilete er ved hjelp av GLCM

Kontrast(Contrast) gir eit mål på kor homogent bilete er ved hjelp av varians i naboskapet. Dersom bilete har store forskjellar i pikselintensitetar vil naboskapmatrisa ha god spreiing og dermed vil og kontrasten vera høg som gitt i Formel 2.11:

$$Contrast = \sum_{i,j=0}^{N-1} P_{i,j}(i-j)^2 \qquad \text{Formel 2.12}$$

Formel 2.12: Utrekning av kontrasten på sansynsmatrisa i GLCM, der i og j er rad og kolonne id i matrisa og N er mengden intensitetar som eg mogleg

Korrelasjon(Correlation) forklarar den lineære avhengnaden for pikselintensiteten i bilete. Dvs kor mykje ein pikselintensitet kan påverke ein annan pikselintensitet. Ved høg korrelasjon er det lettare å gjette kva pikselintensiteten i eit punkt vil vera gitt den samsvarande pikselintensiteten. Korrelasjonen i matrisa vert utrekna med Formel 2.13.

$$Corr = \sum_{i,j=0}^{N-1} P_{i,j}\left[\frac{(i-\mu_i)(j-\mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}}\right] \qquad \text{Formel 2.13}$$

Formel 2.13: Korrelasjonen mellom pikselintensitetane opp mot rad og kolonne

Inverse difference moment(IDM) forklarar kor homogent bilete er med tanke på pikselintensitetane. Ved mykje av dei same verdiane i eit område vil ikkje spreiinga få så store utslag og ein vil få ein høg verdig ved eit homogent felt som beskrive i Formel 2.14.

$$IDM = \sum_{i,j=0}^{N-1} \frac{P_{i,j}}{1+(i-j)^2} \qquad \text{Formel 2.14}$$

Formel 2.14: Utrekning av IDM frå GLCM matrisa

Entropi(Entropy) er kor mykje kaos det er innan bilete og ved høge verdiar er det mykje kaos, det vil seie at pikselintensitetane er spreidd godt utover intensitetsskalaen som ein kan sjå ut ifrå Formel 2.15.

$$Entropy = \sum_{i,j=0}^{N-1} P_{i,j}(-\ln(P_{i,j})) \qquad \text{Formel 2.15}$$

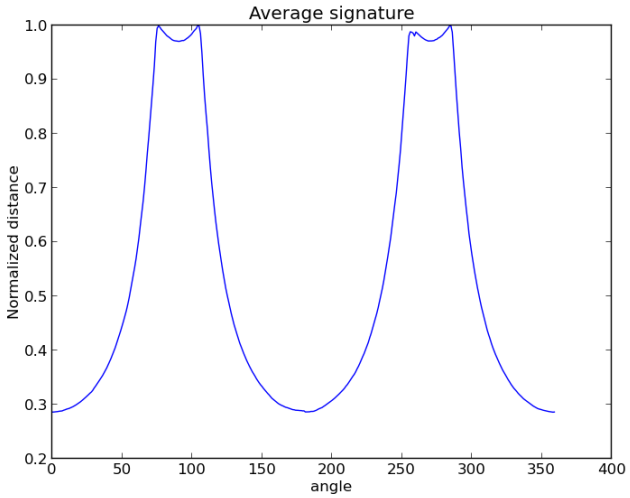
Formel 2.15: Entropi utrekning basert på GLCM, der sansynet for ein pikselintensitet vert vektlagt med si eiga logaritme. Dette vil seie at ved mange ulike pikselintensitetsgrupperinger vil det vera mykje kaos i bilete og dermed høg entropi.

Ved hjelp av Formel 2.10 - Formel 2.15 kan ein få fram ganske store skilnadar i teksturen til dei ulike bileta ved å sjå på dei ulike eigenskapane til kvart objekt. GLCM er mest brukt for å forklara nettopp strukturelle eigenskapar på teksturen og overflata til eit bilete.

2.7.4 Signatur frå massesenteret

Massesenteret og omkrinsen er allereie utrekna frå partikkel analysen (avsnitt 2.7.1) og dermed ligg alt til rette for å rekne ut distansen for kvar vinkel frå massesenteret og ut til omkrinsen. Dette kan vera særs nyttig sidan ein murstein så og seie alltid er firkanta og bør derfor ikkje variere noko særlig her ved ein frisk stein.

Dermed skal resultatet vera ein måleserie som aukar mot sidekantane og minkar på langsidene når målingspunktet nærmar seg massesenteret i midten av steinen som ein kan sjå i Figur 2.18.



Figur 2.18: Gjennomsnittleg og normalisert signatur for murstein

2.7.5 K-means clustering

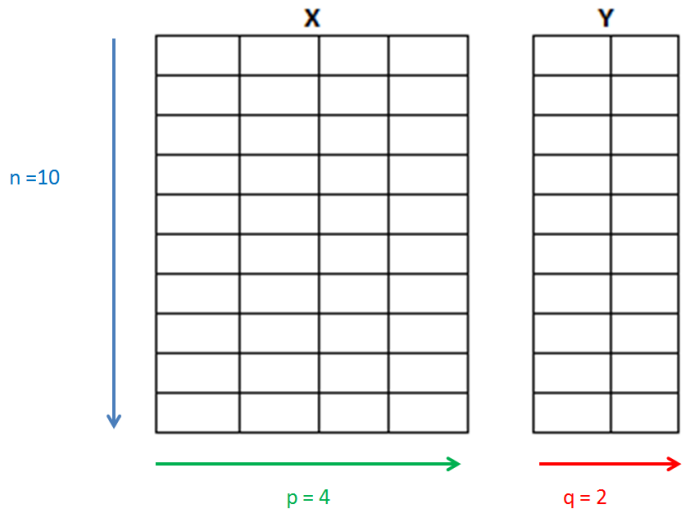
K-means clustering er ein teknikk for å finne regionar i eit bilete basert på fleire dimensjonar. Desse dimensjonane er ulike fargerom som kvar for seg spenner ut eit gråtonebilete. Teknikken nyttar terskling på kanalane og leitar etterpå kva for nokre regionar som naturleg høyrar saman over alle tilgjenglege dimensjonar. Regionane er difor danna på eit breiare

grunnlag der utgangspunktet er at regionar med relativt uniforme pikselverdiar høyrar saman. Dei unike regionane som vert oppdaga vert så slått saman heilt til ein står igjen med k regionar som naturleg høyrar saman basert på gjennomsnittlege pikselverdiar [16].

2.8 Statistisk analyse

Multivariat analyse er å sjå samanheng mellom fleire ulike eigenskapar og faktorar i målingar og observasjonar. Desse observasjonane vert ofte kalla objektar og inneheld fleire eigenskaper som vart målt. Målingar som det er enkelt å få tak frå til dømes utrekningar på data vert kalla x-variablar, medan vanskelege målingar tekne av avansert og dyrt utstyr vert kalla y-variable.

X-, og y-variablane vert ofte lagt inn i matriser der radene er observasjonane og kolonnane er dei ulike eigenskapane til observasjonen. Mengda observasjonar er n-radar, medan variablane/eigenskapane i matrisa X vert kalla for p og i Y vert dei kalla q for å lett kunne skilja dei frå kvarandre. Matrisene og dataoppsettet er illustrert i Figur 2.19.



Figur 2.19: Skisse over komponentane til datamatrisane i ei multivariat analyse.

Det viktigaste er at utvalet av variablar og eigenskapar har relevans til målet med analysen. For å kunne nyttiggjere seg av det innsamla datasettet er det viktig å kjenne til korleis ein kan nytta multivariat analyse på datasettet ved å leite etter samanhengar med vanlige statistiske utrekningar som gjennomsnitt, standardavvik, varians og korrelasjon.

Det er fleire teknikkar for å finne dei skjulte strukturane i datasettet som kan openbare seg på ulike måtar. Det som er viktig er å stille seg sjølv dei rette spørsmåla på kva det er ein er ute etter. Døme på arbeidsoppgåver kan vera å leite etter eit mønster, skilje ut grupperingar eller lage prognoser og spådommar i datasettet. [17]

Ved klassifisering vil ofte Y-variablane vera såkalla dummyvariable med verdien 0 eller 1 for å fortelje om ein eigenskap er aktiv eller ikkje.

2.8.1 Multivariat analyse

I ein multivariat analyse prøvar ein å forklare enten ein enkel datasett eller ulike samanhengar ved å nytte mange variable og observasjonar. Ved å leite etter ulike mønster, samanhengar og utviklingar vil ein prøva å predikera nye verdiar basert på eit nytt datasett, basert på erfaringar frå tidlegare datasett.

Ved vanleg multilineær regresjon vil ein ofte få problem med at ulike observasjonar og variable korrelere slik at desse observasjonane ikkje er uavhengige av kvarandre. Dette igjen førar til at modellen kan få ei svær uheldig vekting av forklaringsvariable som gjere at modellen vert ubrukelig for framtidige datasett.

2.8.2 Prinsipal komponent analyse(PCA)

Prinsipal Komponent Analyse(*PCA*) er ein multivariat analyse som fjernar fleire problem som kan oppstå under vanlig multivariat analyse. Ved PCA nyttar ein bare X-matrisa av datasettet til å leite etter mønster og samanhenger.

PCA er tidsreduserande for å finne samanhengar ettersom algoritmen leitar etter den største variansen/spreiinga i datasettet på tvers av variablane. Påfølgande aksar frå PCA vil leggje seg på den neste største variansen som ligg ortogonalt på allereie identifiserte prinsipal komponentar. Dermed vert det danna eit nytt aksesystem og modell som ofte kan syne nye samanhenger i datasettet.

I dette nye koordinatsystemet kjem det og fram nye prosjekteringar av datasettet, der variablane vert kalla ladningane(*loadings*) og observasjonane vert kalla for scorane. I det nye koordinat systemet nyttar ein ulike prinsipal komponentar som best syner den samanhengen ein er ute etter.

Modellen PCA er bygd opp på er at X-matrisa består av ein del struktur og ein del støy som kan uttrykkast som Formel 2.16 [17].

$$X = TP^T + E = struktur + støy \qquad \text{Formel 2.16}$$

Formel 2.16: Grunnmodellen til PCA der X er datasettet som består av T gange P-transponert matrisa som er strukturen i datasettet og E som består av restverdiane, dvs støy.

Scorer (T)

Scorene i er T-matrisa i Formel 2.16 og er projeksjonen av observasjonane i datasettet X i det nye aksesystemet som spenner seg ut i PCA. T-matrisa er altså projeksjonen til observasjonane for dei ulike prinsipal komponentane som saman dannar eit nytt koordinatsystem [17].

Ladningar (P)

Ladningar er P-matrisa i Formel 2.16 og er projeksjonen av plasseringa til variablane til datasettet. Desse projeksjonane visar samanhengen mellom dei ulike variable spreiar seg ut i det nye koordinatsystemet. På denne måten kan ein enkelt sjå kva for nokre variable som har ein tett korrelasjon og eventuelt er motsatt korrelerte. Saman med scorene kan ein difor nytta P-matrisa til å sjå kva variable som fortel mest om dei ulike observasjonane [17].

Residual

Residualane er differansen mellom modellen og observasjonane og vil vera definert som støy i datasettet. Dersom ein har høge residualverdiar vil modellen vera dårleg til å forklare observasjonane og strukturen i datasettet er dårleg [17]. Ein kan difor nytta residualene til å finne observasjonar som ikkje høyrar heime, eller på andre måtar skil seg ut frå datasettet.

2.8.3 NIPALS - algoritmen

NIPALS står for *Nonlinear Iterative Partial Least Squares* og vart utvikla av Herman Wold [18]. Algoritmen gjere det enklare ganske enkelt å hente ut struktur delen i datasettet X som forklart i Formel 2.16 saman med støydelen E.

Det første ein må gjere er å sentrere datasettet ved å nytte Formel 2.17.

$$x_{ik} = x_{ik} - \bar{x}_k \qquad \text{Formel 2.17}$$

Formel 2.17: Sentrering av verdiane i matrisa X der i representerer radene/observasjonane og k er kolonnane/variablane

I mange tilfelle vil det og vera hensiktsmessig å standardisere datasettet. Dette vert då gjort med dele den sentrerte verdien med standardavviket som gjort med Formel 2.18. På denne måten kan ulike einingar i datasettet samanliknast mot kvarandre. Herifrå vert matriser skrive med stor bokstav i fet skrift, medan vektorar vert skrive med små bokstavar og fet skrift. Framgangsmåten er henta frå både Esbensen [17] og Lohringer [18].

$$x_{ik} = \frac{(x_{ik} - \bar{x}_k)}{n} \sqrt{\sum_{i=1}^{n} (x_{ik} - \bar{x}_k)^2} \qquad \text{Formel 2.18}$$

Formel 2.18: Standardisere datasettet i matrisa X

Etter førebuingane med datasettet er unnagjort byrjar ein med å velje ei tilfeldig kolonne i **X** som i Formel 2.19. I utgangspunktet bør ein velje den kolonnen med størst verdiar. I tillegg vil ein telje kor mange iterasjonar ein har nytta i variabelen *f* som startar på 1(*f*=1).

$$t_{fo} = \mathrm{X_f} \qquad \text{Formel 2.19}$$

Formel 2.19: Tilfeldig valt kolonne i X-matrisa

Deretter prosjekterer ein **X** ned på den valte kolonnen **t** for å finne ladningane **p**i Formel 2.20.

$$p_f = \frac{X^T t_{fo}}{|X^T t_{fo}|} \qquad \text{Formel 2.20}$$

Formel 2.20: p vert projeksjonen til X for kolonne t, dvs at p vert dei ladningane til X

Deretter oppdaterar ein scorene **t** ved å prosjektere datasettet på dei nye aksesystemet som spenner ut prinsipal komponenten *f* der **p** vert referansesystemet for scorene i Formel 2.21.

$$t_{fn} = Xp_f \qquad \text{Formel 2.21}$$

Formel 2.21: Normaliserer vektor p til å ha lengda 1

Etter å ha funnet både ladningene og scorene sjekkar ein om differansen mellom røynda og modellen er innanfor ynskjeleg storleikorden med Formel 2.22.

$$|t_{fn} - t_{fo}| < c \qquad \text{Formel 2.22}$$

Formel 2.22: Sjekkar om forskjellen mellom ny og gammal projeksjon er mindre enn kriterium c

Dersom Formel 2.22 ikkje er sann set ein $t_{fo} = t_{fn}$ og reknar ut Formel 2.20 - Formel 2.22 til kriteriet er oppfylt. Då har NIPALS algoritmen kome til ein stabil løysning som gjere at prinsipal komponent nummer *f* har funnet den største variansen i datasettet.

Dermed vert feilkjeldene i datasettet rekna ut med Formel 2.23.

$$E_f = X_f - t_f p_f^T \qquad \text{Formel 2.23}$$

Formel 2.23: Lagarar feilkjeldene til modellen som er differansen til observasjonane.

Då har ein funne prinsipal komponent nummer *f*. Ein aukar *f* med 1 (*f=f+1*) og startar på nytt med neste kolonne i matrisa til X, men denne gangen nyttar ein **E** i staden for **X** sida ein vil finne den attverande strukturen som ikkje er forklart enda. Til slutt vil enten **E** vera svært liten eller ein har funnet det optimale mengda med prinsipal komponentar [17].

2.8.4 PCR

Prinsipal komponent regresjon (PCR) nyttar scorane frå PCA i staden for den vanlege X-matrisa til å lage ein prognose på datasettet. Dette førar til at datasettet ikkje kollapsar grunna lineære samanhenger i X-matrisa sidan scorane frå PCA per definisjon er lineært uavhengige(ortogonale). Teknikken til PCR ligger i å skifte ut X-matrisa for regresjonen med ei ny matrise T som vil gje datasett der observasjonane er lineært uavhengige av kvarandre per definisjon.

2.8.5 PLS

Partial least squares regression (PLS-R) ein annan metode for å lage prognoser/ predikera data basert på datasettet. Forskjellen frå PCR er at i PLS tar med Y-matrisa med inn i NIPALS algoritmen slik at ein får tilpassa X-matrisa til Y-matrisa. Ein oppnår dei same fordelane med lineært uavhengige observasjonar, men med den fordelen at variansen i X-matrisa nå passar betre med variansen i Y-matrisa. Det vert då lettare å forklare variansen i Y-matrisa ut ifrå X-matrisa sidan desse har verte tilpassa mot kvarandre enn i PCR scorene T vert utrekna på X-matrisa uavhengig av Y-matrisa.

2.8.6 Lineær diskriminantanalyse (LDA)

Ein metode for å skilje ulike grupperingar i eit datasett frå kvarandre er lineær diskriminantanalyse(LDA). I motsetning til PCA, som leitar etter nye strukturar i variansen til datasettet, leitar LDA etter kva som skilje dei ulike kategoriane frå kvarandre. Derfor vil ikkje LDA kunne gi ein god gradering av mellom ulike klasser. I staden forsøker LDA å plassere dei ulike objekta i sine respektive klassar.

LDA antar at fordelinga innanfor kvar gruppe er normalfordelt eller at kovariansen er lik. Det er dette som vil identifisere dei ulike gruppene og gje dei eit massesenter.
For å finne ut kva klasse eit nytt objekt høyrer til køyrer ein dei same utrekningane på dette og ser kva for eit senter i denne dimensjonen objektet passar best til. Derfor egnar LDA seg som klassifisering, men klarar ikkje heilt å gje ein informasjon om graderinga mellom ulike klassar dersom dette var ynskjelig.

I følgje Martínez og Kak [19] krev LDA større datasett for i tilstrekkelig grad kunne bestemme ulike klassar klart frå kvarandre.

2.8.7 Kalibrering

Kalibrering er å lage modeller som skal nyttast til å predikere nye verdiar. Dette vert gjort for å prøve å finne samanhenger mellom X-data som er lett å få tak og Y-data som er vanskelig å få tak i. På denne måten prøvar ein å lage modellar som forklarar Y-eigenskapar ved hjelp av eigenskapar i X-data. Då er det viktig at datasettet som vert nytta til kalibrering er stort nok og ikkje minst at ynskjelige resultat ligger i datasettet. Det er særs vanskelig å predikera nye verdier som skal treffe dersom datasettet leitar etter andre verdiar.

2.8.8 Validering

Validering er å teste modellen som har verte kalibrert. Dette vert gjort for å sjekke om modellen er gyldig for fleire ulike datasett og for å gjere korreksjonar for å gjere modellen meir treffsikker. Det er difor viktig å ha eit anna datasett tilgjenglig for å sjekke kor treffsikker den kalibrerte modellen eigentleg er.

For å sjekke om modellen kan vera pålitelig allereie i startprosessen nyttar ein kryssvalidering. Kryssvalidering er å ta ut ein eller fleire prøver for å deretter lage modell på dei resterande prøvane og teste den eller dei utelatne prøvane på den nye modellen. Dette må

ein gjera for alle prøvane. Dermed får ein sett kor mykje til dømes ei enkelt prøve kanskje avviker i førehald til resten av datasettet. Og ikkje minst kor mykje enkelte prøvar moglegvis skapar ei innverknad på modellen.

Dersom ein bare tar ut ein og ein prøve for å teste ny kalibrert modell på resten av datasettet opp mot den utelatne prøven vert det kalla full kryssvalidering.

Etter at modellen ser ut til å vera god bør heile modellen testast på eit anna datasett for å sjekke om modellen taklar litt andre parameterverdiar og likevel klarar å gje pålitelege svar.

2.9 Fotograferingsteknikk

Ved å ta bilete med same innstillingar for heile veggen, gjer at forholda vert så å seie like. Dette vil føre til at ein kan setje saman resultat av både analyse og bilete til å få eit meir heilskapleg resultat frå veggen. Difor kan det gjere til at sluttresultatet vert auka forståing på både skadeomfang og skadespreiing på veggen.

Ein annan ting som er viktig ved fotografering er å sjå på histogrammet. Histogrammet kan gje viktig informasjon om kva eksponering bilete vil få. Ved høg eksponering vil det vera mykje lyse, og lite mørke, område i bilete som gjere at det kan vera vanskeleg å sjå detaljane. Undereksponering er det motsatt med for mørkt bilete. Dette kan ein sjå ved at histogrammet ikkje dekker heile intervallet det har tilgjengeleg, men heller klumpar seg saman i eit mindre området. Då vert det vanskelegare å skilje dei ulike områda frå kvarandre. Ved å følgje med på histogrammet under fotografering kan ein prøve å få god eksponering ved at ein får utnytte heile intervallet i moglege pikselintensitetar.

Eit digitalt kamera tek bilete ved å dele opp synsområdet (Til dømes grønt område i Figur 2.15) i små rektangel. Kvart av desse små rektangla vil då vera ein piksel og den gjennomsnittlege lyseksponeringa i det rektangelet vil utgjere fargeintensiteten til kamera.

Dersom kameraet kan lagra bilete i såkalla RAW(rå)-format bør dette veljast, sidan det vil liggje mykje meir informasjon i bilete om kva eksponering, og kva pikselverdi kvar piksel inneheld. Jpeg-formatet er eit desktruktivt format for å kunne spare plass, og såleis mistar ein spesielt informasjon om ulike skilje i bilete [6].

3 Metode

3.1 Val av komponentar for prototypen

Python / Jython

Python er eit svært fleksibelt og ryddig programmeringsspråk med ei brei støtte for å nytta programvarebibliotek frå eit breitt spekter. I tilegg er det eit kort og konsist språk som egnar seg godt til å vera eit bindeledd mellom fleire program og programmeringsspråk. Python er bygd opp som eit objekt-orientert programmeringsspråk, samstundes som at det er enkelt å køyre det interaktivt eller skrive koden som ein prosedyre. Den store utbreiinga til Python kan kanskje forklarast best av opphavsmannen, Guido van Rossum:

> « From one perspective you can say Python is a very high-level scripting language. From another perspective you can say it's a high-level programming language that happens to be implemented in a way that emphasizes interactivity. » [20]

Frå slutten av 1980-tallet starta Guido van Rossum på det som etterkvart skulle verte Python og tok då med seg erfaringar frå si tid som medutviklar i eit språk kalla ABC. I tillegg til dette ville han ha moglegheit for å nytte seg av dei allereie eksisterande programbiblioteka skrive i C, dermed fekk Python ein særs fleksibel modulbasert struktur. Desse tankane i starten er nok mykje av grunnlaget for at Python i stadig større grad vert nytta som scriptspråk i eksterne program og dataspel[20].

At Python vart valt som hovudprogrammeringsspråk i oppgåva er tredelt. For det første er dette det programmeringsspråket kandidaten meinar han meistrar best og dermed vil nytta minst tid på å tilegne seg. Det andre er at Python er veldig universelt og dermed kan fungere særs godt som eit bindeledd mellom fleire ulike deler av prosessen frå fotografi til ferdig rapport. Det siste er at den kraftige pakken for behandling av bilete støttar Jython.

Jython[21] er Python-språket implementert på Java plattforma. Dette inneberer at ein kan skrive Python kode og køyre denne i eit Javamiljø der ein eigentlig burde ha skrive koden i programmeringsspråket Java. Ein konsekvens av dette er at Python kan importere Java moduler på lik linje som vanleg Python kan importere kode skrive i C/C++.

ImageJ

ImageJ er eit bilete prosesseringsprogram som har sitt opphav av Wayne Rasband frå National Institue of Health. Programvaren er basert på programmeringsspråket Java. I tillegg har ImageJ ein open struktur som oppfordrar bidragsytarar til skrive tilleggsprogram for ekstra funksjonalitet. Dette har ført til at det er mange funksjonar innanfor prosessering, analysering og datainnsamling på bilete.

Ei samlepakke av ImageJ saman med særdeles mange av tilleggsfunksjonane er laga under namnet Fiji [22]. Fiji står for Fiji is just Imagej. Denne samlinga av tilleggsprogram gjer at Fiji-pakken vert ståande som ei særdeles komplett pakke for prosessering av bilete. Fleire scriptsspråk er inkludert for å kunne lage eigne rutinar, macroar og tillegg, som til dømes

Jython. Heile rammeverket har ein særdeles god dokumentasjon for API og er bygd godt rundt gjenbruk av funksjonar

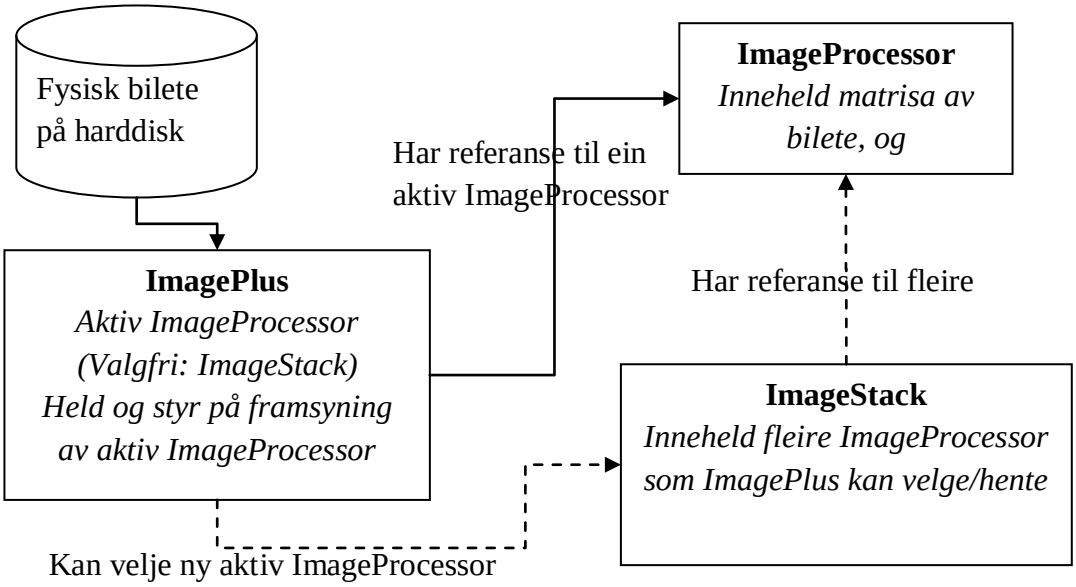
I ImageJ er det spesielt to instansar ein må vera obs på og skilje frå kvarandre og det er difor høveleg med ei oppsummering av desse:

ImagePlus er ImageJ sin referanse til sjølve fila og inneheld informasjon *om* fila som til dømes EXIF-informasjon, filbane og storleik. I tillegg inneheld ImagePlus informasjonen om den synlege delen av bilete frå brukaren sitt synspunkt med kva for eit vindauge bilete har. ImagePlus held og styr på kva for ein ImageProcessor som høyrer til bilete. Dersom bilete er ein biletestack, har den oversikt over dette og veit kva ImageProcessor som er aktiv til ein kvar tid.

ImageProcessor held styr på sjølve matrisa og pikselverdiane som representerer bilete. Operasjonar på matrise og pikselnivå av bilete vert difor utført på denne klassen. ImageProcessor er eigentlig ein abstrakt klasse for å klargjere kva funksjonar som er forventa for andre ulike representasjonar av eit bilete som til dømes desimaltal (FloatProcessor) istaden for heiltall eller for fargekanalar(ColorProcessor).

ImageStack er eigentleg nesten det same som ImageProcessor, bare at ImageStack inneheld ein liste av fleire ImageProcessor med same dimensjon og type.

Figur 2.7 synar korleis relasjonen mellom desse ulike klassane heng saman. Dette er viktig å ha kontroll på når ein skal programmere opp mot bileta og halde styr på kva tid ein bør referere til dei ulika klassane. Legg merke til at det ikkje er referansar oppover i hierarkiet. Referansane kan hente opp ein instans frå lågare nivå, men det er ikkje naudsynt at verken ImageStack eller ImageProcessor trenger ImagePlus, så lenge ein sjølv held kontroll på instansen internt i programvaren [6].



Figur 3.1: Skisse på nivå- og arbeidsinndelinga innad i ImageJ for å representere eit bilete.

Piwigo

For å lagre og halde styr på bilete av mursteinsvegger som kjem inn til analyse er det viktig å ha ein god basis i form av eit fotogalleri. Etter å ha testa fleire ulike system for dette, landa valet på Piwigo[23] føre bl.a. Gallery og Coppermine. Piwigo er eit av fleire fotogallerisystem som er skrive i PHP og nyttar MySQL for lagring av info ved sidan av å lagre bileta som vanlege filer. Piwigo har eit godt fundament og mange bidragsytarar til å lage forskjellige tilleggsprogram og malar. Dette saman med Piwigo sin fleksible struktur vart eit godt val av fundament for visning og opplasting av bilete. I tillegg er Piwigo lagt opp til at det er mogleg å ha fleire underalbum. Noko som kan vera hending ved å lagra bilete i ein større struktur som til dømes By->Adresse->Bygning->Vegg. Ein annan ting er at det er mange forskjellige metodar for å gruppere bileta slik at ved vidare bruk kan det vera lettare å samanlikne ulike vegger på tvers av album.

Eit design med namn *SimpleNG* gjorde nettsida kjapt til ei mobilvennleg side med responsivt design og siste nytt med HTML5.

Numpy og SciPy

Numpy[24] er eit tilleggsmodul til Python for matematiske operasjonar på matriser og vektorar medan SciPy[25] brukar Numpy til statistiske analyser, signal prosessering eller andre meir kompliserte matematiske operasjonar. Saman utgjere desse ein samling for vitskapelig bruk av Python. Numpy gjere ein god jobb med tal i form av lister og matriser saman med eit stort utval av matematiske funksjonar og operasjonar på dette. Begge desse pakkane vert nytta til vidareprosessering og analysering av talmaterialet som kjem etter prosesseringa av bilete.

Diverre kan ein ikkje nytte desse direkte i ImageJ miljøet med Jython sidan desse nyttar Python 2.7 som basis, medan Jython er basert på ein eldre kodebase som ikkje er kompatibel med desse modulane.

Matplotlib

Matplotlib er ein kraftig og fleksibel modul til Python for å teikna grafar og figurar i både 2D og 3D i mange forskjellige variantar. Det var i si tid starta som eit prosjekt av John D. Hunter [26]. Matplotlib har støtte for mange ulike bakgrunnssystem som gjer at programvarepakken oppførar seg særs likt i ulike miljø som til dømes interaktivt og prosedyre.

pandas - Python Data Analysis Library

Pandas er ein Python modul for å hente inn og oppbevare data frå ulike kjelder på ein grei måte. Forskjellen frå reine matriser er at pandas i tillegg kan halde orden på namna til både objekta/radene og kolonnane. Dette gjer det veldig enkelt å plukke ut og bruke data frå datainnsamlinga[27]. Pandas er ein kraftig pakke for dataanalyse som har som mål «å verte det kraftigaste og mest fleksible analyseverktøyet med open kjeldekode » [28].

Hoggorm: a chemometrics package in Python

Hoggorm[29] er ein modul for PCA (Sjå avsnitt 2.8.2 for innføring i PCA) i Python der ein har nytta NIPALS algoritmen forklart i avsnitt 2.8.3. Implementeringa er laga av Oliver Tomic og gjer det enkelt å køyra prinsipal komponent analyse på ulike deler av datasettet. I tillegg er det lagt inn støtte for kryssvalidering til å sjekke ut kor pålitelig modellen er.

IJ Plugins: k-means Clustering

Eit tilleggsprogram frå IJ Plugins vart nytta til å køyra k-means clustering[30] på bilete som ein alternativ metode å skilje murstein frå fuge på ein mursteinsvegg. Denne modulen er ein del av ei større pakke for segmentering av bilete i ImageJ.

Lineær diskriminatanalyse i Matlab

Matlab script laga av Ulf Indahl for å klassifisere ein modell basert på Lineær diskriminatanalyse(avsnitt 2.8.6). Scriptet er delt opp i ein klassifisering for å lage ny modell og ein predikeringsfunksjon for å predikere nye variabler ved å endre input. Dette kan ein nytte til å lage og teste ein modell. Etterpå vil ein nytte denne modellen til å predikera nye Y-verdiar/klasser basert på nye X-data.

Xvfb

Xvfb[31] emulerar eit grafisk lag på maskinen som tek unna alle applikasjonar som krev eit grafisk grensesnitt, utan at det er naudsynt å trykkje på det. Dermed kan ein nytte vindauge med eit eige API gjennom scripting i staden for å trykkje på det grafiske grensesnittet. Diverre har nokre av desse applikasjonane som vert nytta i denne oppgåva eit krav om at grensesnittet skal køyre, minst i bakgrunnen, difor må ein nytta program som xvfb til å behandle desse operasjonane i internminnet til datamaskinen for å kunne nyttast utan skjerm.

Xvfb startar eit grafisk miljø med shellkodar som er viktig for å kunne nytte bl.a. RoiManager i ImageJ. Tipset om denne metoden kom frå Fiji si side om å køyre kode utan grafisk grensesnitt [32].

jsDraw2DX

For å kunne velje eit kurant utsnitt til kalibrering av forskyvingar i bilete trengtes det ein grensesnitt for å velje eit utsnitt av bilete utan at ein var fastlåst til ein fast rektangel med 90 graders hjørne. jsDraw2DX [33] teiknar opp figurer innanfor angitt område ved hjelp av SVG. Dermed kan dei aller fleste moderne nettlesarar nytte seg av denne programvaren for å teikne polygon over bilete i Piwigo. Det vart lagt inn ein modifikasjon med *filltype=opacity0* for å kunne skape transparente bakgrunn inni polygonet.

Ubuntu

Ubuntu er eit operativsystem med utspring frå Debian som igjen er ein fullverdig distribusjon av GNU/Linux kjernen. Operativsystemet Ubuntu vart starta i 2004 for å laga ein Linux distribusjon med eit enkelt og brukarvennleg grensesnitt. Eit opent fellesskap og enkle løysningar har gjort at Ubuntu er eit av dei mest brukte operativsystema basert på open kjeldekode [34].
Som basissystem for den utvikla programvaren vert Ubuntu 12.04 nytta.

VMware Player

Ubuntu vert køyrt som ei virtuell maskin for å kunne flyttast til ulike maskinar ved behov. For virtualisering av datamaskinen vert det nytta VMware Player. VMWare Player emulerar ein datamaskin inni den allereie eksisterande maskinen. Fordela med dette er at ein kan velje kor mykje ressursar maskinen skal få, samstundes som at det er relativt enkelt å flytte den virtuelle maskinen til andre fysiske datamaskiner ved behov.
VMware Player er laga av VMWare og er ein enklare programvare for virtualisering. VMware Player er gratis for ikkje-kommersiell verksemd [35].

LAMP

LAMP er ein fellesbetegnelse på ein programvarepakke som består av Linux, apache, MySQL og PHP. Linux er operativsystemet som i dette tilfellet er Ubuntu. Apache[36] er ein webserver for å kunne tilby nettsider på eige maskin og er såleis bindeleddet mellom serveren og brukaren for nettinnhald. MySQL[37] er ein database for å lagre i hovudsak tekstbasert innhald medan PHP er eit av dei mest brukte programmeringsspråka for webapplikasjonar. Saman utgjere dette sjølve motoren i fleire webapplikasjonar, som til dømes Piwigo køyrer på.

UMB ImageJ plugins

Frå Universitetet for Miljø- og Biovitskap si eiga biletgruppe vert tilleggsprogramma AMT_spectra og GLCM_spectra [38] implementert til å kunne nyttast i programvaren. Desse tilleggsprogramma implementerer teknikken som vart omtala i avsnitt 2.7.2 og 2.7.3. Tilleggsprogramma er skrive for ImageJ og funger då såleis i Fiji.

3.2 Arbeidsprosess

Ynskje med oppgåva er å svare på om det er mogleg å lage ein rutine for å avdekke frostskader på ein mursteinsvegg ved hjelp av eit fargebilete. I tillegg kan det vera andre ting med mursteinsveggen som kan gje rom for samanlikningar. Det er difor viktig å skissere opp prosessen frå og med sjølve fotograferinga og fram til framsyning av resultata etter analysen. Etter at ein interessant mursteinsvegg er funne, vert aktivitetane identifisert til arbeidsprosessen vist i Figur 3.2.

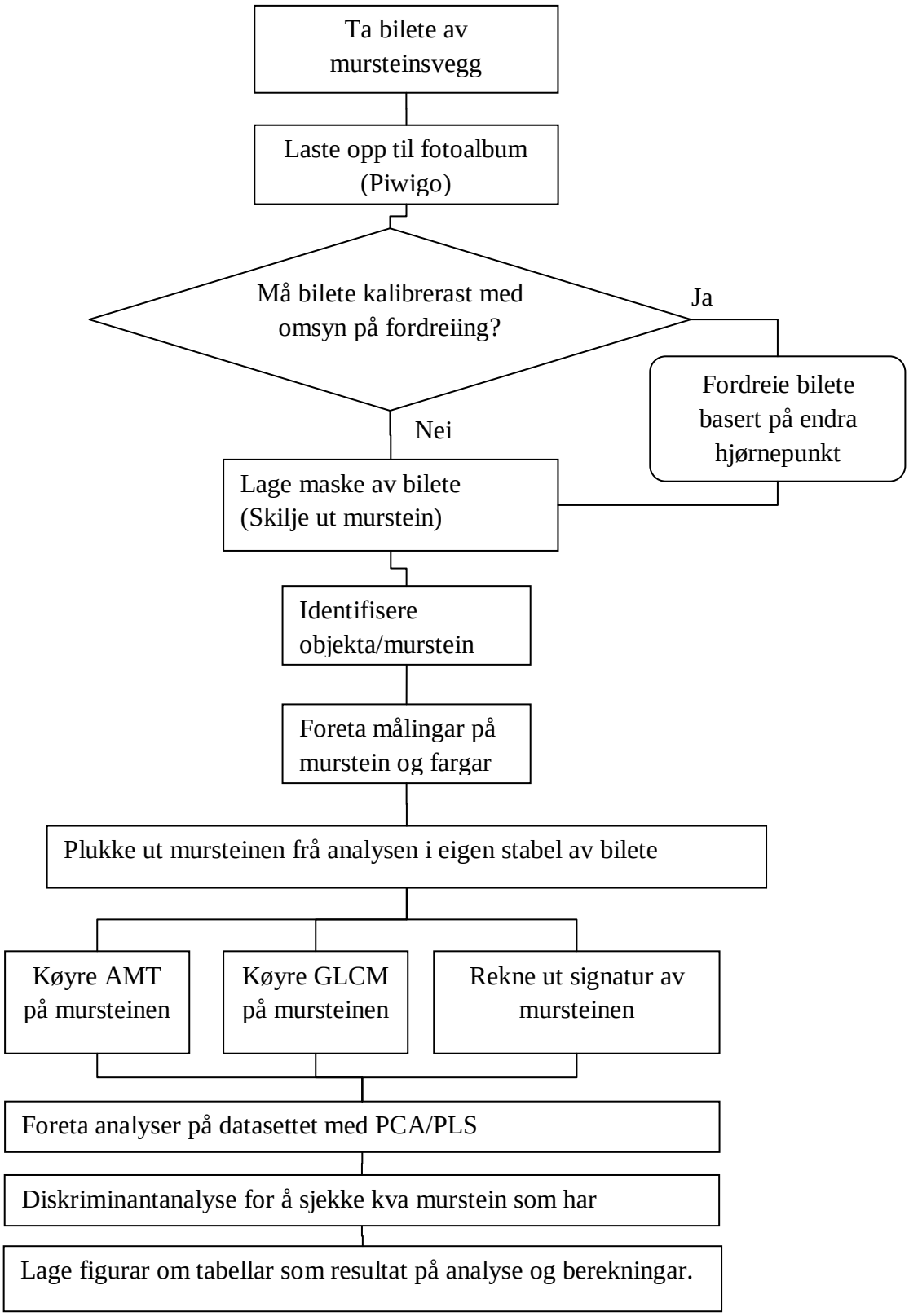
For interaktiv bruk av programvaren kan ein velje kva for nokre av dei følgjande rutinane ein vil nytta. Sjå vedlegg F) på korleis ein kan gjere dette gjennom Jython Interpreter i Fiji.

Aktivitetane frå Figur 3.2 er identifisert ut frå kva for nokre ledd som må verte gjennomført før neste ledd kan byrja for sørgje for at heile prosessen frå fotografi til resultat av mursteinsvegg vert ivaretatt. Både før og undervegs i prosessen er det naudsynt å oppnå best mogleg bilete for datainnsamling og analyse. Fundamentet for godt resultat til slutt er biletet som fotografen tek i starten. Fotografen må difor prøve å ta eit bilete som utnyttar mest mogleg av det tilgjenglige intensitetsrommet i pikselverdiar som mogleg. Desto større spenn og utfolding det er i pikselverdiane, desto lettare vert det å skilje mellom dei ulike delane i biletet. Programvaren klarar ikkje å skape meir forskjell enn det som allereie finnest i grunnlaget.

Programvaren kan derimot hjelpe til med å korrigere bilete dersom det trengs forskyvingar på høgdeforholdet eller frå sida som i Figur 2.15. Det kan vera aktuelt å endre perspektivet sidan det fører til at mursteinen kan samanliknast på likt grunnlag mot dei andre mursteinane i same biletet. Ulike fordreiingar i bilete vil slå ut på storleiken til mursteinen i form av areal og omkrins.

Etter at bilete er godkjent og korrigert av fotograf og programvare, kan sjølve identifiseringa av murstein starta. Først må ein nytte ein metode for å skilje fuge og murstein frå kvarandre. Desse unike mursteinane vil deretter verte gjenstand for grundig informasjonsinnsamling. Programvaren kan etter denne kartlegginga hente inn mykje informasjon om kvar einskilde murstein. Då er det viktig å plukke ut relevant informasjon alt etter kva som best forklarar sluttresultatet.

I hovudsak handlar denne analysen om å identifisera frostskader på murstein ved hjelp av innsamla data, ulike utrekningar og vidareforedling av denne informasjonen. Etter at prosessen er gjennomført er det mest sannsynleg at fotografen er interessert i svara frå analysen. Det er derfor viktig å lage rapportar og figurar som framstillar resultata på ein interessant og høveleg måte.



Figur 3.2: Arbeidsprosessen programvaren må ta omsyn til frå fotografi til resultat

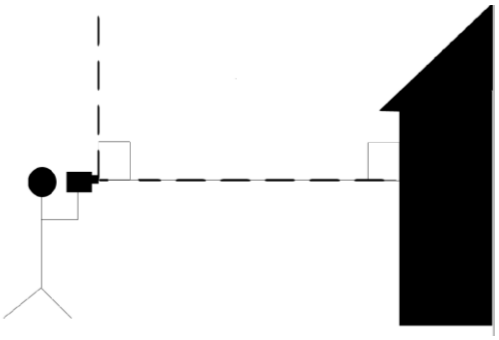
3.3 Fotografering og prosessering av bilete

Det er ikkje alltid like lett å ta bilete av høge bygningar. Det er ofte ulike hindre i vegen for å kunne ta bilete normalt på veggen (Illustrasjon: Figur 3.3), som er det optimale. Det er ikkje alltid ein får stå på den mest gunstige plassen for å ta eit godt bilete. Døma på hindra er mange som trær, liten plass mellom bygningar og ei rekkje andre moment.

I første instans er det fotograf og utstyr som gjev fundamentet for analysen av bilete. Det er difor naudsynt å nytta tida godt på å ta bilete der perspektivet og fordreiingar i bilete vert redusert til eit minimum. Det er viktigare å utnytte intervallet til pikselverdiar i bilete framfor å få til eit godt perspektiv. Det er lettare å endre forskyvingar i perspektivet enn å erstatta manglande pikselverdiar til å spenne ut intensiteten. For å hjelpe fotografen med perspektivendringar bør programvaren leggje til rette for dette dersom det er ynskjelig.

Korrigering av perspektivet kan gjerast ved å velje kvar perspektivet eigentleg skal vera gjeldande ved å endre på hjørnepunkta. Deretter vert det laga ein transformasjonsmatrise som gir programvaren instruks til korleis resten av biletet skal justere seg.

Etter at biletet enten er tatt normalt på veggen som i Figur 3.3, eller korrigert fram til same resultat, er det klart for å plukke ut og identifisere kvar einskilde murstein på veggen.



Figur 3.3: Fotografere normalt på mursteinsveggen

3.4 Lage maskebilete av mursteinen før analyse

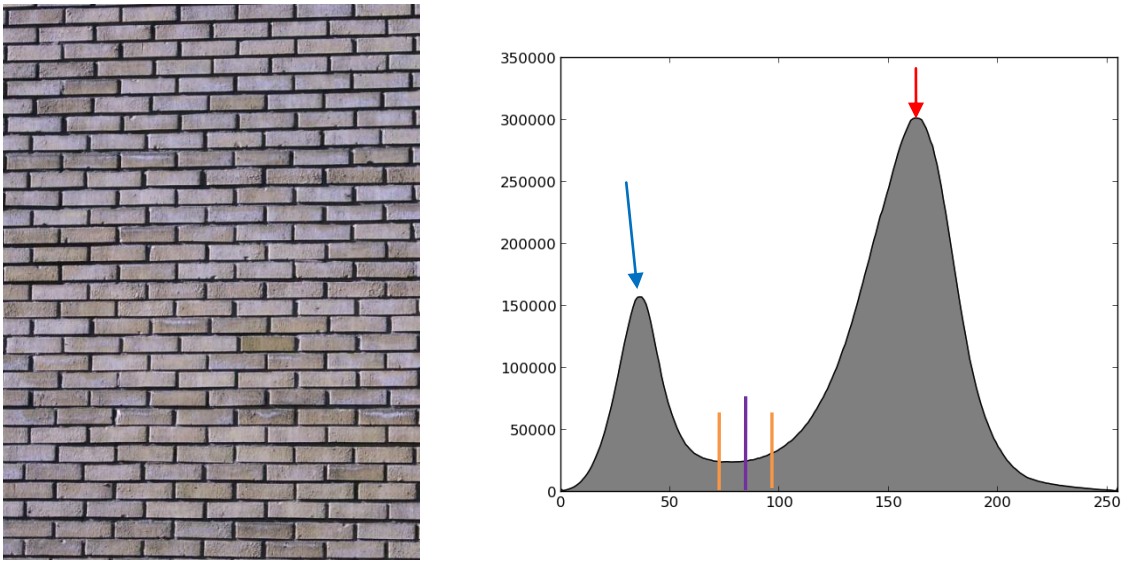
Biletet skal nå vera klart til å skilje fuge frå murstein og såleis få fram forma på kvar murstein i biletet. Dette kan verte gjort på fleire måtar, der nokre metodar er raskare enn andre. Dei treigare metodane er ofte meir treffsikre på å skilje ut mursteinen.

I denne oppgåva vert det tilbod om to ulike variantar. Ein vert å finne den beste pikselverdien som vert nytta som tersklingspunktet for å skilje fuge og støy frå resten av mursteinen. Den andre metoden stepper inn for tersklinga og slår saman og identifiserar regionar som liknar kvarandre. Etter eit par gjennomkøyringar er målet at bilete vert delt inn i to typar regionar som skal vera fuge og murstein kvar for seg.

3.4.1 Terskling på bestemt pikselverdi

Det har vorte utvikla ein metode for å estimere kva pikselverdi som best vil skilje fuge og murstein frå kvarandre. Ofte vil det då følgje med litt støy rundt både murstein og fuge, slik at dette må fjernast og reduserast etterpå.

Aller først vert det samla inn heile 10 histogram for bilete. Desse histogramma er for gråtonebilete, intensiteten til fargekanalane i RGB, HSB og L*a*b*. Etter dette vert det gjort ei kort analyse på kvart histogram for å finne beste tersklingspunkt på best eigna fargekanal. Figur 3.4 viser eit godt døme på korleis eit godt og enkelt histogram som programvaren leitar etter. Det har to klart definerte toppar, og ein dal i mellom der det kan utførast terskling på.
I fleire bilete av mursteinsvegger er det svært tydleg at i minst ei av desse 10 histogramma er det to topper som representerer fuge og murstein fråskilt av eit botnpunkt i mellom. Det er derfor ynskjelig å identifisera denne dalen mellom to klare toppunkt for å finne det beste tersklingspunktet i midten som vist i Figur 3.4.
Derivering vert nytta for å sjå på endringa i histogrammet frå 0 til 255. Der maksima og minima er definert som topp eller botn for punktet. Dvs. at punktet er mindre enn det førre og større enn neste for topp. Og mindre enn det førre og større enn det neste for botn. Sidan histogrammet ikkje naudsynt er heilt kontinuerlig er det viktig å sjekke om dette makspunktet er det største innanfor ein større serie av nabopunkt. Difor vert det sjekka for punkter i ein distanse på 10 punkter i kvar retning. Figur 3.4 visar eit døme på dette utvida området med pikselverdien i midten med lilla strek, og ytterpunkta markert med kvar sine oransje streker. Ved å traversere gjennom heile histogrammet på denne måten kan ein oppdaga alle topp- og botnpunkt for kvart histogram.



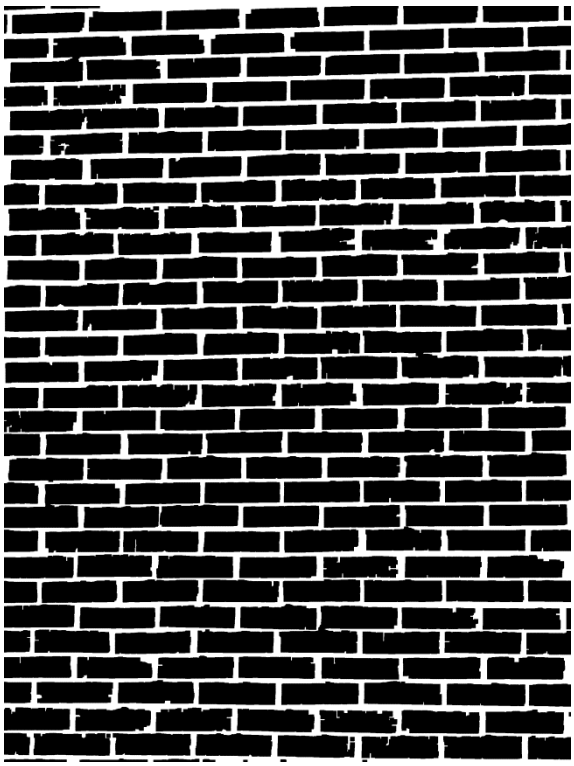
Figur 3.4: Vegg frå Stavangergata til venstre, Foto: Andreas Fløː Til høgre: Gråtonehistogrammet av bilete visar to klare toppar der toppen markert med raud pil stammar frå alle mursteinane som har ein lysare farge enn den lille toppen, markert med blå pil, som visar den mørkare fugen.

Etter at alle toppunkta er identifisert er det på tide å finne toppar som tilsvarar dei markerte punkta i Figur 3.4, då det er ynskjeleg å finne dalen mellom desse. Sidan det helst bare er ein topp for fuge og ein topp for murstein er det viktig å skilje desse to toppane frå kvarandre.

Dalen vil vera det minste punktet mellom toppane, og dette er pikselverdien tersklinga skal utførast på. I tillegg vert det sjekka kva side av tersklinga som inneheld flest pikslar ved å summere histogrammet frå tersklingspunktet og ut til kvar side. Den sida som har flest pikslar

må innehalde murstein, då mursteinen inneheld eit større areal enn fugene. Dette bestemmer då om bilete skal inverterast eller ikkje. På denne måten vert sluttresultatet uansett eit maskebilete med kvite fuger og svarte murstein som vist i Figur 3.5 nedanfor.

For å få fram Figur 3.5 etter terskling må det reinskast for støy ved å køyra tre prosesser til, nemlig utviding, erosjon og fjerning av støy. Etterpå fyllest holrom innanfor objekta igjen for å få fram klare mursteinar. Dette vert gjort for å reinska bilete mest mogleg for støy og småpartiklar slik at bilete vert klart for vidare bruk.



Figur 3.5: Maskebilete av Figur 3.4 der mursteinen er så å seie skilt frå fugene.

3.4.2 K-means clustering

Den andre metoden er å hente ut regionar som liknar på kvarandre og til slutt vert satt saman til k regionar. I dette tilfelle vil ein bare ha to regionar, enten murstein eller fuge og ein set difor k til å vera to for å oppnå 2 regionar.
Ein startar på same måte som med tersklinga ved å hente ut dei fargekanalane som har eit histogram med 2 toppar. I staden for å velje den beste fargekanalen samlar ein saman alle fargekanalar som har to topper og køyrer regionsøk på alle samla. Dette gjere at ein får eit større grunnlag til å skilje ut stein frå fuge.
Ved identifisering av stein vert det køyrt terskling for å skilje region 1 frå region 0. Etterpå vert det her og sjekka kva for ein region som er størst for å finne ut om biletet skal inverterast. Til slutt vert dette biletet reinska for eventuell støy ved hjelp av utviding, erosjon, fjerning av støypartiklar og fylla igjen holrom.

3.5 Innhenting av informasjon frå maskebilete og originalbilete

Då maskebilete er klart er det på tide å trekkje ut informasjon frå regionane som er skilt mellom murstein og fuge. Då skal ein partikkelanalyse verte køyrt på maskebilete forklart i avsnitt 3.4. Partikkelanalysen i ImageJ går igjennom maskebiletet som er omtala i avsnitt 2.7.1. Etter at regionane er identifisert, hentar ein fram arealet til alle regionane og standardiserer desse med formlane under:

$$\sigma = \frac{1}{n}\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \qquad \text{Formel 3.1}$$

Formel 3.1: Formel på standardavvik (σ) på datasett x, der $\bar{x}$ er gjennomsnittet til x, n er antall objekt.

$$X = \frac{(x_i - \bar{x})}{\sigma} \qquad \text{Formel 3.2}$$

Formel 3.2: Standardisering (X) av datasettet x

Då sit ein igjen med areala med verdiar sentrert om 0 og med standardavviket som ytterpunkt. Dette gjere at ein kan ta vekk regionar som ikkje passar inn fordi dei er for store eller for små. Dette vil vera typiske mursteinar som ligg i kanten av biletet og difor ikkje får med heile steinen, eller murstein som ikkje har vore skilt tilstrekkelig i maskebiletet slik at fleire steinar heng saman. Ved å fjerne mursteinar som ikkje passar inn, får ein i større grad sett på dei mursteinane som faktisk er interessante. For små og store steinar vil skape unødig støy i dei vidare analysane.

Etter fjerninga av støyen hentar ein inn statistikk på pikselintensiteten for dei 10 fargekanalane gråtone, RGB, HSB og Lab. Sjå vedlegg A) for tabell over kva eigenskapar som vert henta inn på objekta.

3.5.1 Signatur av steinen

Ein metode som vart utvikla er "signatur av steinen" etter ide frå rettleiar. Med signatur på steinen meinar ein at denne måleserien kan fortelja noko unikt for kvart objekt. Signaturen fortel korleis steinen ser ut eller på andre måtar klarar å trekkje fram det spesielle ved kvart objekt.

For å få fram ein signatur av mursteinen vert det føreteken ein utrekning av distansen mellom midten av steinen, og ut til omkrinsen. For kvart objekt går programvaren gjennom kvart pikselpunkt i omkrinsen og reknar ut vinkelen og avstanden ved hjelp av Formel 3.3 og Formel 3.4.

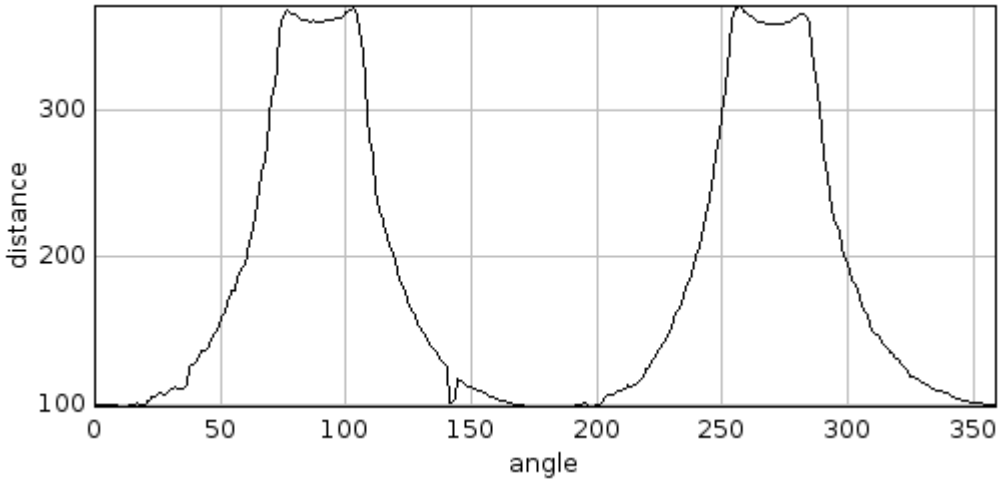
$$\alpha = \arctan\left(\frac{x_c - x_o}{y_c - y_o}\right)$$ Formel 3.3

Formel 3.3: Vinkel(α) mellom to punkter i xy-planet der alfa er vinkelen mellom punkt O og C. O står for punkt i omkrins og C står for massesenteret.

$$d = \sqrt{(x_c - x_o)^2 + (y_c - y_o)^2}$$ Formel 3.4

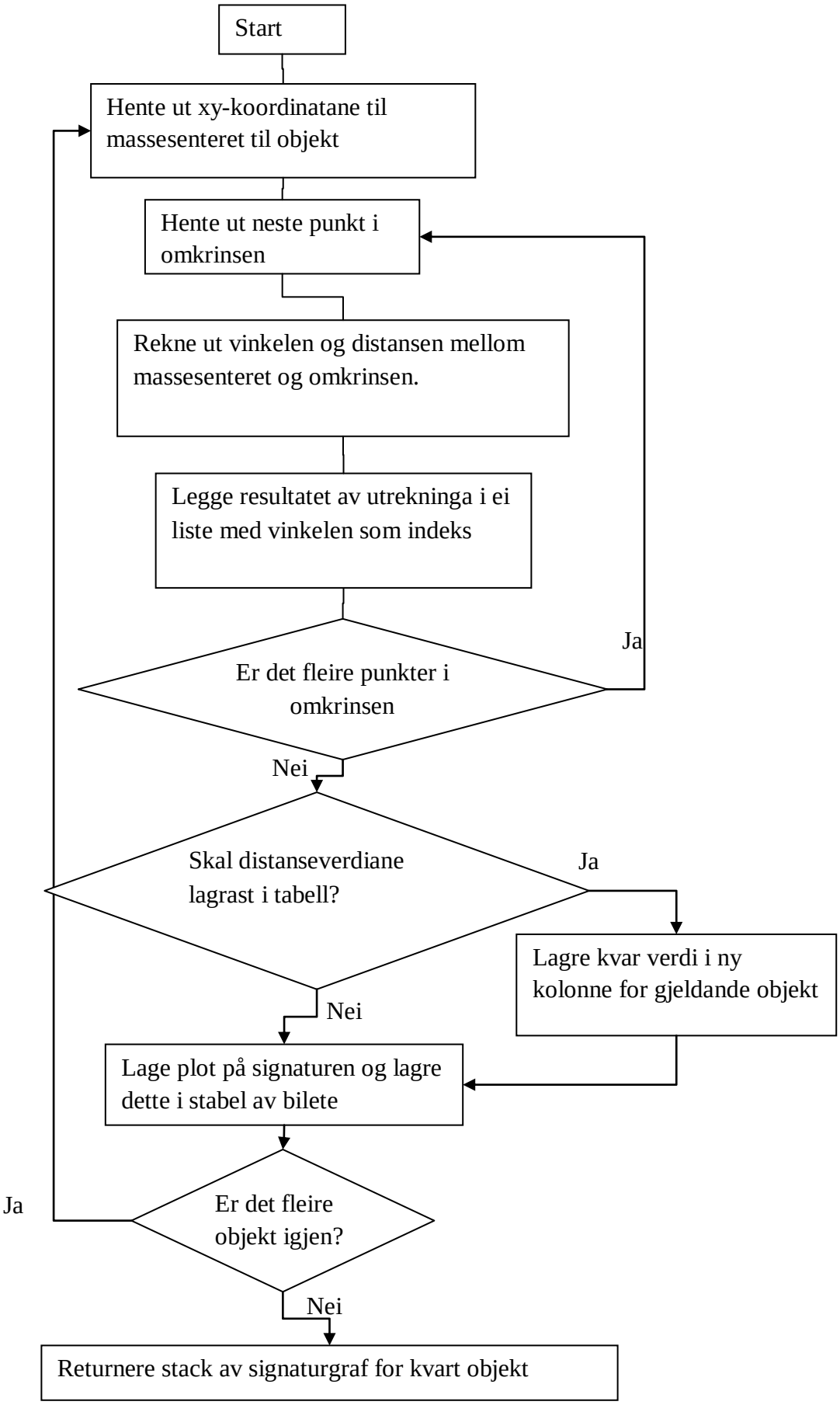
Formel 3.4: Distanse(d) mellom to punkter i xy-planet der O står for punkt i omkrins og C står for massesenteret.

Det vert lagra ein distanse frå midten av objektet og ut til omkrinsen for kvar vinkeleining mellom 0-359°. Ein ser at det då vert den siste distansen for kvar vinkel som vert lagra. Denne framgangsmåten gjer at ein får ei like lang serie for kvart objekt. I tillegg er det lett å samanlikne signaturane direkte mot kvarandre, sidan dei bør endre seg på dei same stadane.



Figur 3.6: Signatur frå ein murstein. Dei to toppane er karakteristisk for dei smale sidene på endene og at ved vinkel på 180 og 0 grader er dei nærast senteret av steinen.

Framgangsmåten til algoritmen forklart i Figur 3.7 og implementeringa er vedlagt i Vedlegg C under metoden *runSignature*.



Figur 3.7: Arbeidsprosessen for å lage "signatur" av omkrinsen og massesenteret til objekt

Avvik i signaturen

Vidare prosessering av signaturen vert gjort med å normalisere signaturane til å liggje mellom 0 og 1. Deretter lagar ein eit gjennomsnittssignatur for alle mursteinane i biletet. Her vil eventuelle avvik vera minimerte og signaturen vil vera svært nær ein ideell frisk stein, så sant ikkje alle objekta har akkurat same skade på akkurat same plass.

Med normaliserte verdiar for både gjennomsnittet og kvar enkelt objekt kan ein finne avvik ved å trekkje frå den gjennomsnittlege signaturen. Dermed kan ein finne ut kor dei største avvika ligg i signaturen for kvart objekt. Etter dette vil skader i steinen få ei større innverknad på signalet, og dermed forhåpentlegvis vera lettare for klassifiseringa å oppdaga frostskade.

Ein stor fordel med å leite etter normalisert avvik i steinen er at det betyr ikkje noko kva storleik mursteinen har så lenge den framleis er firkanta. Vinklane vil sørgje for at signaturen av steinen uansett vert lik for alle steinar.

Snu traverseringa på signaturen

Ein annan metode for å finne avvik i signaturen er å snu traverseringa frå å gå med klokka til å gå mot klokka. Sidan det bare er den siste distanseverdien som vert lagra i måleserien er det rom for at det eigentleg skulle vera andre verdiar i måleserien. Ved å traversere omkrinsen i motsett rekkefølgje kan ein sjå om signaturane får store nok forskjellar til at denne kan forklara eventuelle frostskader i objektet.

3.5.2 AMT

I programvaren vert det mogleg å køyra AMT på enten pikselverdiar til den ytre randsonen av objekta, signaturen som er beskrive i avsnitt 3.5.1, histogramma, eller eit sjølvvalt signal. Dei to mest aktuelle er pikselverdiar og signaturen.

Etter at måleserien er plukka ut vert det køyrt gjennom tilleggsprogrammet AMT_spectra til ImageJ laga av biletgruppa på UMB [38]. Etterpå kan alle resultata for kvart objekt lagrast i tabellen, samt grafen av AMTresultatet visast fram som ein biletestack.

Pikselverdiane vert henta ut som ein spiral der ein tek utgangspunkt i storleiken på kvart objekt i stacken og hentar ut 50 % av pikselverdiane frå det yttarste rektanglet til objektet. På denne måten får ein med seg mest mogleg varians i objektet som kan nyttast til å forklara ulike eigenskapar til objektet.

3.5.3 GLCM

GLCM vert køyrde på biletestacken av objekta der stabelen først vert konvertert til gråtonebilete. Når prosesseringa til gråtone er overstått, vert tilleggsprogrammet GLCM_spectra frå [38] køyrt. Resultatverdiane frå køyringa kan lagrast for kvart objekt i programvaren si samletabell for resultata.

3.6 Vidare prosessering av analysedata

Undervegs i den første delen av analysen vert all innsamla data lagra. Vidare vart dette sendt til eit eige script som føretek PCA (på form, farge, GLCM, AMT og signatur). Deretter vert dei innsamla data overført vidare til matlab for lineær diskriminantanalyse. Her vert mursteinen sjekka om den har frostskader eller ikkje basert på tidlegare kalibrert modell. Ved identifisert frostskade vert skadeomfanget utrekna basert på innsamla data om mursteinen. For å skape gode resultat og rapportar, vert alle figurar tilpassa høvelege formater og lagra i eit bestemt arbeidsmappe.

3.7 Framsyning av resultata av analysen

Etter at heile prosessen er køyrd vert alle høvelege, ferdige resultat lagra i ei mappe der Piwigo har tilgang til å hente dei fram. Deretter kan resultata frå analysen visast for bileta som har vore køyrd gjennom analysen. Her vert det framstilt ein indeks som forklarar kor mykje frostskader dette biletet er vorte estimert til å ha.

4 Resultat og diskusjon

4.1 Programvaren

4.1.1 Generelle utfordringar i programvareutviklinga

For å lage ei verktøykasse og prototype på programvaren vart det viktig å tenke todelt under utviklinga. På ei sida vert det laga ei rutine som kan køyre frå A til Å og gi brukaren ei god rapport på skadeomfanget til mursteinsveggen. Samstundes skal ein ivareta kravet om ei verktøykasse, og då er det viktig å ha fleire moglege alternativ på analyse og datainnsamling på mursteinsveggen. Dette har resultert i eit funksjonsrikt tilleggsprogram med førehandsdefinert køyring for hurtig gjennomkøyring og estimering av frostskadeomfanget.

Python / Jython

Det er fleire utfordringar som må løysast for å få programvaren til å vera samarbeidsvillig. Ei stor utfording under utviklinga er kva verdiar listene for lagring av resultat, bilete og andre lister internt i programma nyttar som startverdi. For å hente ut eit element frå ei liste trengs det ein indeksverdi som representerer det unike elementet i lista. Utfordringa besto i at listene ikkje er konsekvente på om første element i lista startar på 0 eller 1. Det gjeldt difor å vera oppmerksam på kva liste i programmet ein nytta for å hugse kva plasseringar dei ulike elementa har på lista. Det mest vanlege er å nytte startverdi på 0 som vert nytta i Python sine interne lister, JavaArray, internt i RoiManager og internt i ResultsTable. Derimot nyttar både ImageStack og Roi nummereringa startverdien 1 som første element saman med lister i Matlab. Matlab koder er uansett skilt ut frå resten, og er dermed lettare å forhalde seg til. Blandinga av startindeks mellom ImageStack, ResultsTable og RoiManager i same programmeringsspråk har ført til ein del forvirring på korrekt bruk av indeksverdiar for korrekt objekt.

Python er eit svært fleksibelt programmeringsspråk med tanke på gjenbruk av andre program og bibliotek og limet mellom dette. Samstundes fører denne interaktive kompileringa og til at programmet kan virke svært treig ved store operasjonar, spesielt på lister. For vanlig Python kan mykje av dette forbedrast ved å nytte Numpy, medan denne muligheita ikkje eksisterer i Jython. Ein merka difor ei klar forbetring frå å nytta vanlig Jython kode til AMT før ein gjekk over til å nytta jAMT i AMT_spectra. Denne koden er optimalisert i Java og køyrte mykje raskare enn Jython. Som døme kan ein trekkje fram at Python brukte ca 1 min per murstein, noko som resulterte i at den sto og jobba i 2 timar. Java brukte derimot bare 2 min på 100 murstein og gjer det mykje greiare å teste ut ulike parameter.

Enkelte tilleggsprogram i ImageJ er laga direkte mot det grafiske brukargrensesnittet og gjer det difor vanskelig å bruke desse gjennom terminal. Det programmet som tilsynelatande ikkje har nokon veg rundt ser ut til å vera konvertering av eit RGB-bilete til ei liste av dei tre fargekanalane i CIELab. Problemet er at lista av bilete i LAB-fargerommet visast fram i staden for å returnere ein referanse tilbake til programvaren. Dette gjere at ein ikkje får tak i det nylig genererte vindauget med bilete i Lab-fargerommet.

Piwigo

Har veldig god basisfunksjon, men å utvikle tilleggsfunksjonar er litt meir klundrete. Det som underteikna ikkje likte var at for å leggje til ny funksjonalitet måtte det lagast små funksjonar som la til kvar sine små deler i såkalla events. Dette resulterer i ei mengde med små funksjonar for å knytte saman alt. I tillegg var dokumentasjonen litt rotete, men dette kan ha samanheng med overgang til Piwigo 2.5 i tida dette var aktuelt for oppgåva.
Men uansett gav denne måten å leggje til funksjonalitet tilgang til heile HTML-koden, og i så måte kunne ein øydeleggje alt anna dersom ein ville. Men sjølv om strukturen og API ikkje var like utviklarvennleg som ImageJ var det greit å sjå korleis andre tilleggsprogram hadde løyst oppgåvene. For å få fram fleire felt i sjølve visninga av bilete måtte ein leggje direkte i malane for design og layout, noko som vil gjere det litt vanskelegare å oppdatere og skifte mal.

4.2 Rutinen og køyring av programvaren

Figur 4.1: Arbeidsdelegeringa til dei ulike programmeringsspråka etter kva tid dei faktisk utførar arbeid i analysen frå brukargrensesnittet i fotogalleriet til siste figur er laga.

Figur 4.1 visar samanhengen mellom dei ulike programmeringsspråka og kva tid i arbeidsprosessen dei startar kvarandre. Alle overgangane som er illustrert med ein grøn pil er kommandoar køyrd gjennom shellkodar til neste program.

4.2.1 Fotoalbum og den magiske starten

Den ferdige prototypen som er omtala i kapitel 0 startar med eit fotoalbum. Fotoalbumet nyttar Piwigo som grunnlag, mykje grunna at standardoppsettet gir rom for ein nærmast uendelig rekke av underfotoalbum. Ideelt sett kan dette derfor nyttast til å lage samlingar av bilete på fleire ulike nivå, dersom dette er ynskjelig. Døme på dette kan då vera øvste nivå er eit bustadkompleks med fleire ulike bygningar. Deretter kan ein bryte det ned til kvar einskilde bygning og til slutt kva himmelretning veggen er orientert til, dette kan ha påverknad på kor mykje frostskader det vil vera på ein mursteinsvegg jfr. kapittel 2.1. Her ligger alt til rette for behandling og lagring av bilete på ein fornuftig måte, derfor var første stopp på vegen å lage eit tilleggsprogram, eller ei utviding til Piwigo. Tilleggsprogrammet vert kalla Brickwall som eit gjennomgangsnamn på programvaren i oppgåva.

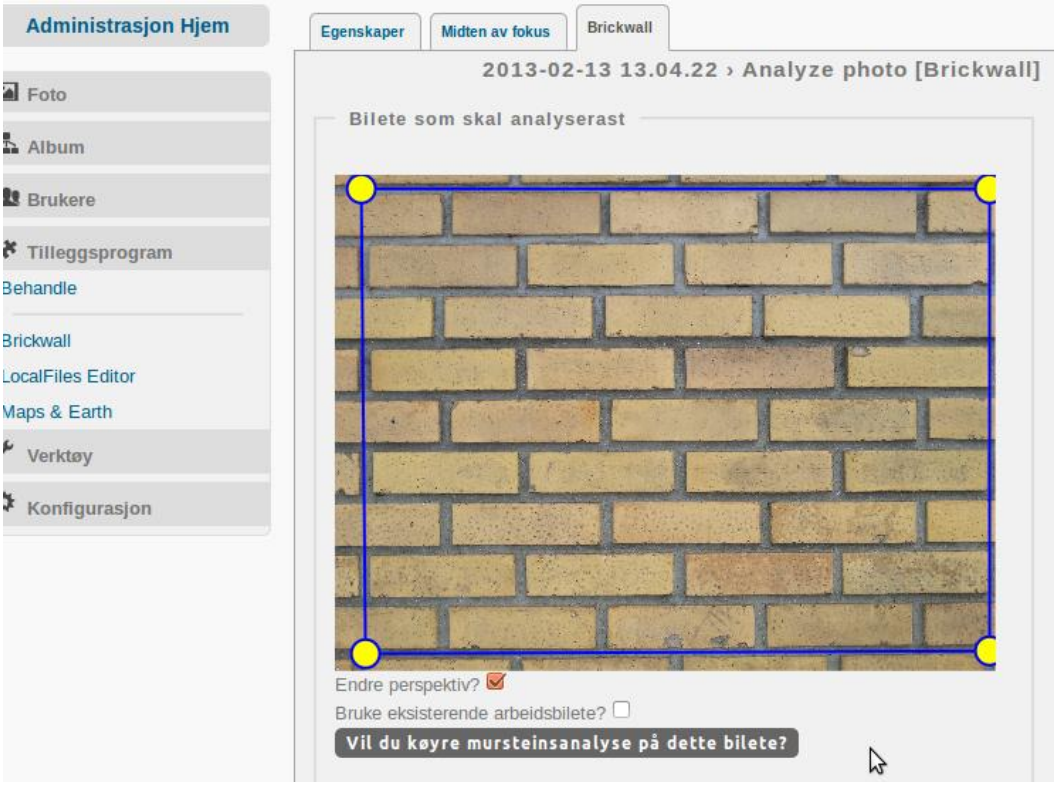
I første omgang har administrator tilgang til å starta analysen på enkelbilete. Før analysen startar, får brukaren eit val om bilete skal forskyvast for å rette opp eventuelle skeivskap i bilete. Dermed kan fotografen korrigere bilete slik at perspektivet vert sjåande ut som det er

teken normalt på mursteinsveggen. Brukargrensesnittet ved å leggje eit polygon er modifisert ut ifrå eit script frå jsFiction [33]. Modifikasjonen var å leggje til at bakgrunnen til polygonet kunne vera transparent.

Sjølve algoritmen for å dreie perspektivet er henta frå Solem [9] der ein lagar ein transformasjonsmatrise ved hjelp av endringa for dei 4 ulike hjørnepunkta. Grunnen til at valet falt på eit polygon var for at ein måtte kunne endre hjørnepunkta uavhengig av kvarandre. Dei fleste andre grensesnitta var enten fastlåst til eit fast rektangulær form eller så var det sjølvstendige punkter utan god forankring til endeleg utsnitt.

For at det skal vera greitt for brukaren å ha oversikt på biletet ved endring av hjørnepunkt, bør ikkje biletet overskride storleiken til nettlesarvindauget. Dette vert gjort ved at biletet bare kan strekkje seg ut til tilgjengelig område på administratorsida. Etter at sida er lasta inn, veit ein difor kor bredt bilete kan vera for å fylla ut nettlesaren. Deretter nyttar ein javascript og jQuery til å finne bredda på det skalerte bilete og kva ratio bilete vert skalert ned til. Dette er viktig for å korrigere punkta ein får ut frå grensesnittet tilbake originalverdiane for at transformasjonsmatrisa skal ta høgde for heile storleiken av bilete. For å oppnå tilbake til korrekt storleik må ein dividere kvar punktverdi med ratioen. Då vil dei nye hjørnepunkta tilsvare same punkt for den nedskalerte versjonen og for originalbilete.

Figur 4.2 visar korleis brukargrensesnittet er for å starta analysen på eit valt bilete. Ein kan velje å korrigere bilete før analysen, eller sende det inn som det er.



Figur 4.2: Utsnitt av starten av mursteinsanalysen, dei gule sirklane kan ein nytte for å kalibrere bilete til å vera tilnærma normalt på synsvinkelen. Den nye ramme i blått verdt så dratt ut til å romme heile bilete. Det er difor viktig å prøve å få ramma til å gå beint med mursteinen og fugene som ein kan sjå i døme ovanfor. Dei blå strekane følgjer parallelt med fugene til venstre og både i øvre og nedre kant.

Dersom det allereie finnes eit kalibrert bilete *(dvs. eit bilete ved namn work_image.tif allereie eksisterar i mappa)* vert det mogleg å nytte dette utan å transformere det på nytt.

Php sender så ein kommando til python gjennom terminalen at den skal starte scriptet "start_analyze.py". I same kommando informerar programvaren for kva arbeidsmappe bilete har, kvar bilete er lagra, eventuelle nye punkter og om ein skal bruke det gamle bilete som ligg der.

Grunna at php har innebygd ein timeout, vert køyringa mot terminalen avslutta for Php sin del, og all respons i form av tekst på terminalen vert lagra i ei logg-fil kalla out.log i arbeidsmappa. På denne måten er det mogleg å la terminalen få køyre sitt eige liv, medan php trur at jobben er gjort. Dette er viktig med tanke på prosessering av bilete som ofte tek ein del tid. Spesielt kalibreringa av bilete visar seg som tidkrevjande i Python.

Foreløpig lagast det ein ny mappe med ID'en til bilete i ei førehandsbestemt arbeidsmappe som kan endrast i konfigurasjonen av Brickwall.

Startfila køyrar korrigering av bilete basert på fordreiing som forklart i avsnitt 2.6.3 dersom dette er ynskjelig og hjørnepunkta har flytta på seg. Etterpå startar scriptet opp ein ny prosess der Fiji vert starta gjennom grafikkemulatoren Xvfb med arbeidsmappe og bilete som inngangsparametra. Det er viktig å starte denne kommandoen med Xvfb for å kunne nytta RoiManager seinare i Fiji som nyttar eit grafisk miljø i botn som det ikkje var mogleg å kome seg vekk ifrå. RoiManager er viktig for å halde styr på grensene til kvart objekt i bilete som vert identifisert ved hjelp av partikkelanalysen.

Startscriptet kan ein sjå i sin heilskap i vedlegg B).

4.2.2 Prosessering av bilete i Fiji

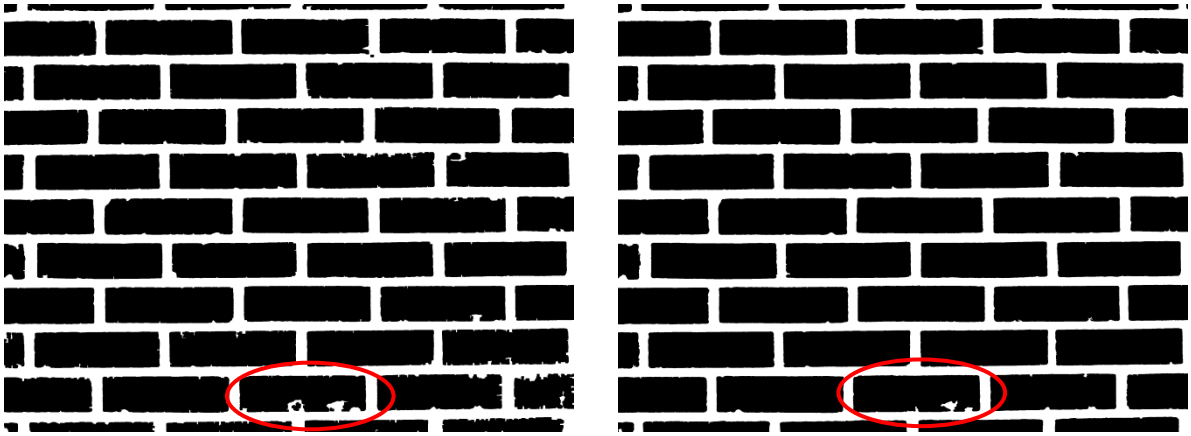
Rutinen i Fiji er laga som ein klasse med fleire metodar for datainnsamling på mursteinsveggen. Sjølve rutinen er førehandsbestemt i funksjonen ***run*** som vert drøfta her.

Klassen kan og brukast interaktivt gjennom Jython interpreter der ein sjølv vel kva deler av rutinen som skal utførast. I vedlegg H) kan ein sjå korleis ein nyttar programvaren interaktivt i Fiji ved hjelp av Jython. Kjeldekoden til den utvikla rutinen i ImageJ kan sjåast i vedlegg C).

For å få eit breiast mogleg datagrunnlag vert det henta inn histogram frå fleire ulike fargekanalar. Sidan tilleggsprogrammet "*RGB to CIELab*" vel å lage ei ny instans av ImagePlus og vise den fram grafisk, er det vanskelig å hente ut informasjon frå denne biletestacken utan eit fungerande grafisk miljø som forsvann ved å køyre koden gjennom terminalen. Difor vert bare 7 histogram samla inn og nytta vidare rutinen, datainnsamlinga og analysen.

Figur 4.3 visar maske bilete av ein mursteinsvegg *(Datasettet i* Figur 2.5*b))* der to ulike teknikkar for framstilling er nytta. Jamt over ser det ut til at teknikken med terskling gjev meir hakkete murstein og dermed litt meir rufsete utsnitt. *K-means clustering* med 2 grupper gir

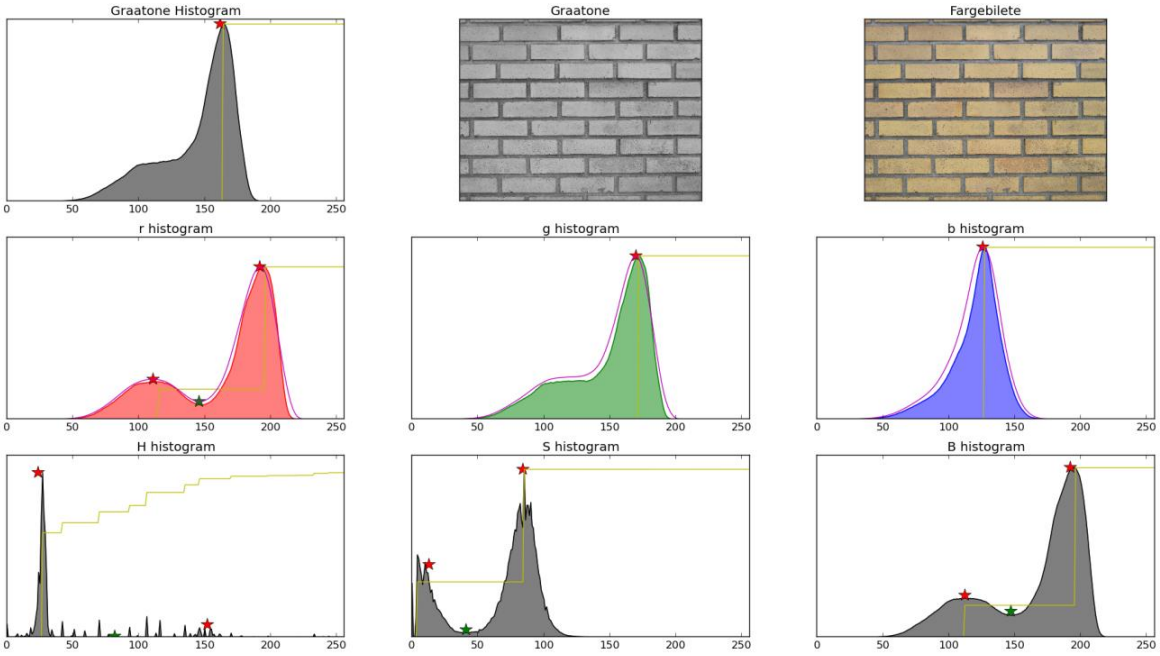
derimot eit klarare skilje mellom fuge og murstein, men ser og ut til å tette litt igjen på hakk inn i mursteinen. Dette bør då korrigerast for seinare i rutinen for å finne frostskader.



Terskling k-means clustering

Figur 4.3: Samanlikning av maskebilete basert på terskling og k-means clustering. Ein kan sjå at dei ulike metodane finner mursteinen på forskjellige måtar.

For begge metodane er det analyse av histogramma som sørgjer for grunnlagsmaterialet. Forskjellen er at terskling bare nyttar ein fargekanal, medan k-means clustering nyttar alle godkjente fargekanalar. Dei godkjente fargekanalane er som ein kan sjå i Figur 4.4. Der leitar programvaren etter to toppar og ein dal i midten. Toppane vert markert med ei raud stjerna. Dersom programvaren klarar å identifisere 2 toppar vert den grøne stjerna plassert på minimumspunktet i mellom. Dei raude stjernene finn toppunkta, men til dømes i histogrammet for fargetone er det ganske mange toppar, slik at det ikkje er naudsynt at denne toppen er representativ for å skilje fuge frå murstein. Elles kan ein sjå at for histogramma gråtone, grøn fargekanal og blå fargekanal er det bare ein einslig stjerne på toppunktet. Her har ikkje algoritmen funne noko klart punkt nummer 2, dermed klarar heller ikkje programvaren å seie kva som er murstein eller fuge ut ifrå desse fargekanalane.



Figur 4.4: Histogram for gråtonebilete og fargekanalane RGB samt HSB. Raude stjerne marker beste toppunkt og grøn markerar beste botnpunkt dersom dette eksisterar i histogrammet.

Det kan difor og diskuterast med omsyn på histogramma om ulike fargekanalar moglegvis bør ha ulike reglar for godkjenning som eit godt histogram. I denne prototypen vert alle histogramma samanlikna på likt grunnlag, noko ein ser spesielt fargetonen ikkje passar heilt inn i med si spisse og taggete histogram.

I tillegg til å finne ulike topp og botnpunkt i histogrammet, vert det og sjekka om desse punkta er gode nok til å skilje murstein frå fuge. Desse reglane er laga etter fleire køyringar for å sjå kva som må til for å skilje dei ulike toppane frå kvarandre. Reglane ein har enda opp med er gitt i Tabell 4.1.

Regel	Godkjent/Ikkje godkjent
Ikkje eksisterer eit botnpunkt	Ikkje godkjent
Verdien til botnpunktet er større enn minste toppunktet	Ikkje godkjent
Mindre enn 20 punkter mellom toppunkta	Ikkje godkjent
Mindre enn 10 punkter mellom minimum og toppunkt	Ikkje godkjent
Dersom verdien på toppunkta er mellom halvparten til halvannan av det andre toppunktet.	Godkjent
Toppunkta er større enn botnpunktet og 70% av det minste toppunktet framleis er større enn botnpunktet.	Godkjent

Tabell 4.1: Regler for kva tid eit histogram er godkjent for å verte nytta til terskling. (Ser i ettertid at dette var noko krunglete)

Ein annan ting som er interessant er kva ein skal gjere etter tersklinga for å fjerne støy og reinske bilete til å framstille mursteinen på ein god måte. Tabell 4.2 visar ulike måtar å reinske maskebilete for støy etter terskling. Ein må finne ein god måte å fjerne støy i bilete

ved å endre ulike parameter. Desse parameterane er storleiken på lovlege partiklar og å bestemme kor mange køyringar ein skal gjennomføre av erosjon, etterfølgt av like mange utvidingar.

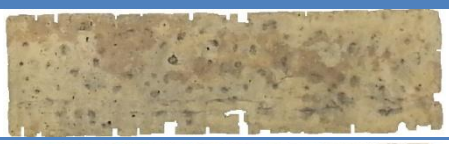
Det er ein god ting å køyre utviding like mange gonger som ein har køyrt erosjon sidan tersklinga allereie i dei fleste samanhenger har funne grensa mellom fuge og murstein.

Ved erosjon reduserer ein denne grensa med ein pikseleining kvar gong slik at murstein i maskebiletet vil krympe. Ved å køyre utviding etterpå vil ein kome ut igjen til den opphavlege grensa. Erosjonen fjernar derimot støy som er for små ved siste krymping, då forsvinn desse områda og då er det heller ikkje noko å utvide i det same området. Difor er gjentakande erosjon og utviding ei grei metode for å fjerne små partiklar og støy.

Frå Tabell 4.2 kan ein leggje merke til at ved høg radius på støyfjerning vert mursteinane svært avrunda i hjørna som ein kan sjå i rad 2. Ein kan ikkje sjå bort frå at eventuelle frostskader vil liggja i eit hjørneområde. Ein bør difor ikkje bevisst ta vekk deler av mursteinen tidleg i prosesseringsfasen.

Eit anna slåande fenomen ved for mange erosjon med påfølgande utvidingar, er at programvaren klarar å utnytte eit lite hakk i steinen til å komme seg eit godt stykkje innpå steinen som skapar meir støy enn det fjernar.

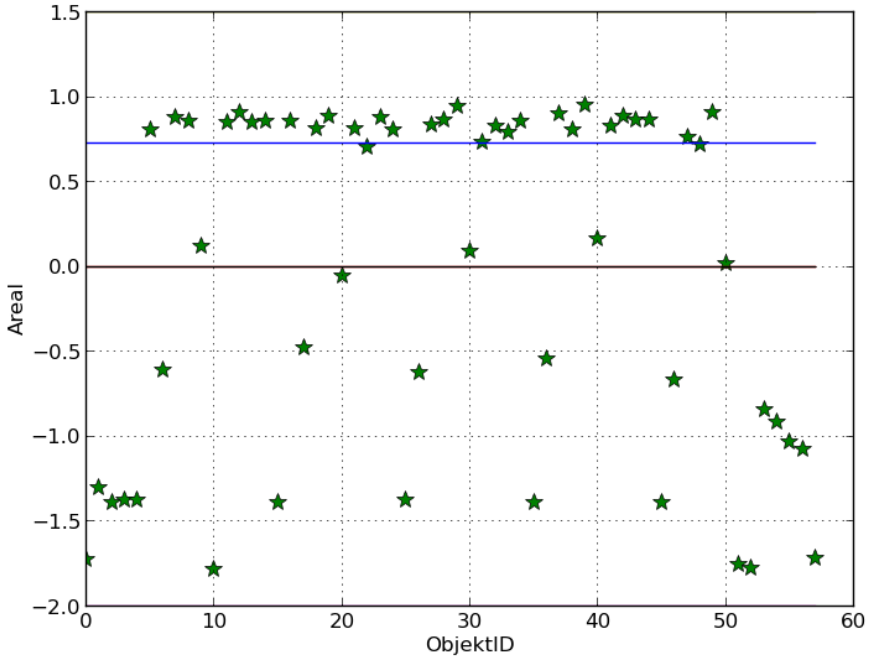
Basert på desse steinane ser det ut til at ein bør ha eit balansert forhold til begge metodane og nytte dei forsiktig ved å køyre ein mellomting frå ytterpunkta som i den første rada.

Parameter	Murstein 1	Murstein 2
Radius 10 erosjon/utviding 6		
Radius 20 erosjon/utviding 2		
erosjon/utviding 8		

Tabell 4.2: Bilete av to mursteinar etter ulike prosesseringar av bilete. Radius angir kva radius ein har nytta i for å fjerna for små partiklar, medan talet etter erosjon/utviding er kor mange gjentakingar desse funksjonane har hatt.

Innsamling av data på objekta

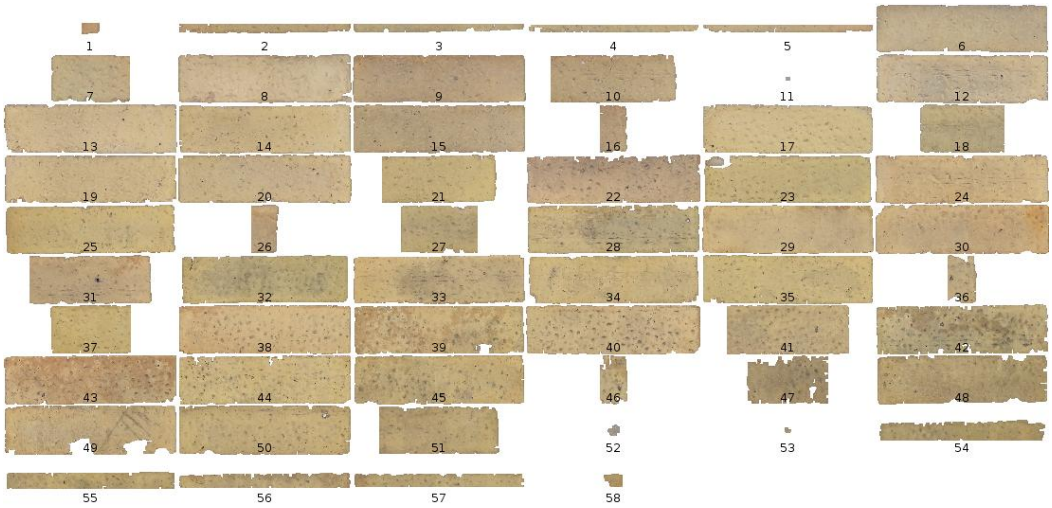
Når maskebilete er klart, vert det køyrt partikkelanalyse på biletet. Som forklart i avsnitt 2.7.1 identifiserer partikkelanalysen regionane i bilete.



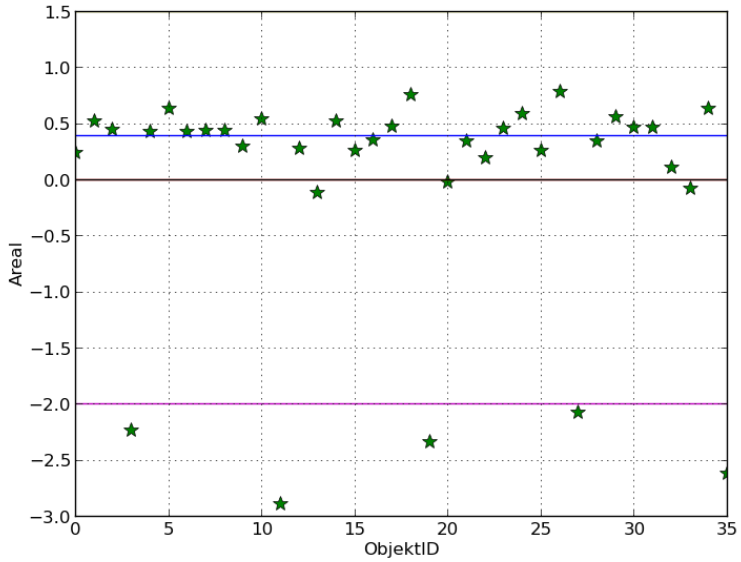
Figur 4.5: Spreiing av identifiserte murstein der y-aksen visar standardisert areal. (Obs, ObjektID startar på 0, pluss derfor på med 1 for å samanlikne med Figur 23)

For å bare hente ut dei interessante mursteinane vert det gjort ei grovsortering på dei identifiserte mursteinane. Figur 4.5 visar at det er mange av mursteinane som er mykje mindre enn normalt. Dei unormale mursteinane er lette å oppdaga i Figur 4.6 der dei skilje seg klart ut som for små.

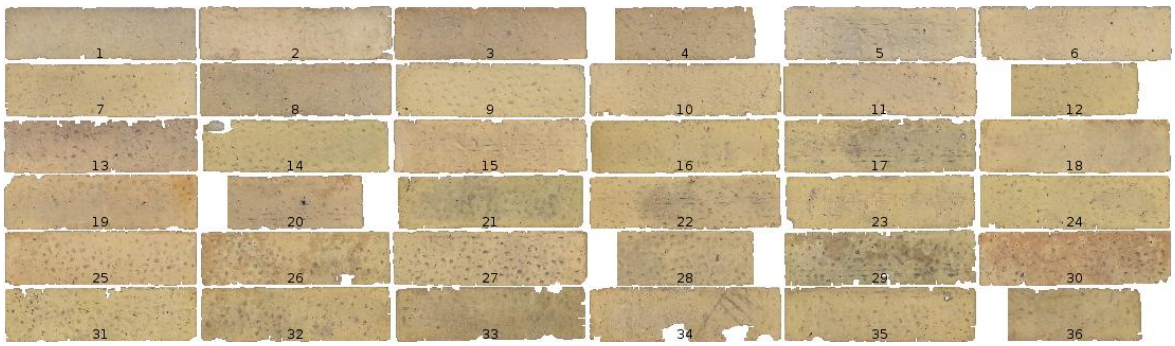
Ei unormal stein er definert som ein stein med standardisert areal under -0,25 eller større enn 1,5. Desse verdiane har gode slingringsmonn for å få fram ei homogen storleik på dei aktuelle steinane. Som ein ser i Figur 4.5 ligg dei aller fleste interessante steinane mellom 0.5 og 1, medan det er mange små som ligger under -0.5. Frå Figur 4.6 kan ein sjå at fleire av desse små mursteinane har lege i kanten av bilete og dermed har ikkje heile steinen fått vera med. Ved å fjerna mursteinar som er definert som unormale får ein eit meir homogent felt som vist i Figur 4.8 og Figur 4.35 der ein klart ser at storleiken på mursteinen er lettare å samanlikne mot kvarandre.



Figur 4.6: Identifiserte mursteinar før fjerning av unormalt små og store mursteinar

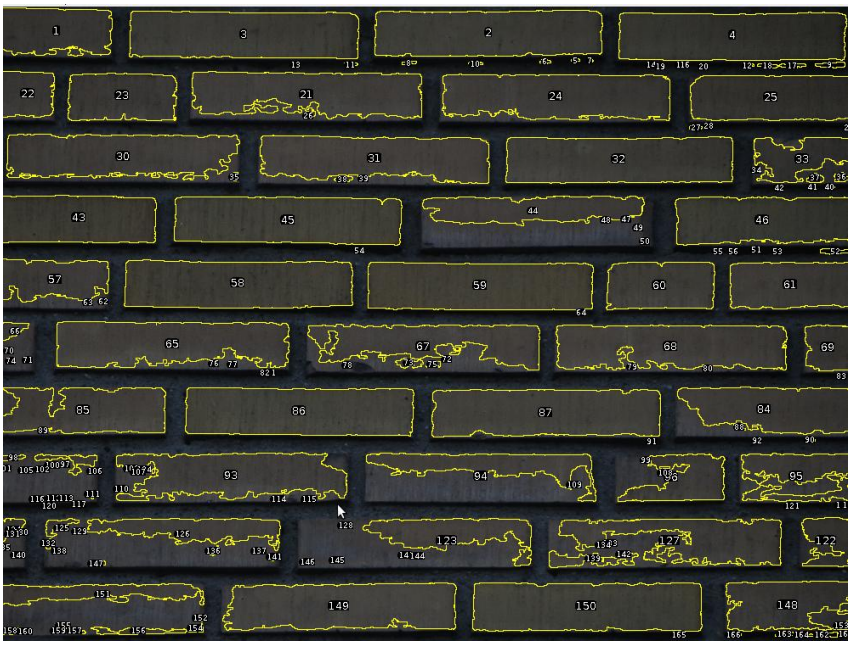


Figur 4.7: Fordelinga på identifiserte murstein basert på standardisert areal etter fjerning av unormal storleik på mursteinen. (Obs, ObjektID startar på 0, pluss derfor på med 1 for å samanlikne med Figur 25)



Figur 4.8: Identifiserte mursteinar etter fjerning av unormalt små og store mursteinar

Diverre kan denne teknikken fjerne mursteinar i bilete som eigentleg høyrar heime. På grunn av støy og vanskelege forhold for å skilje steinar frå kvarandre kan for mange små regionar føre til at standardiseringa slår feil ut. Figur 4.9 visar eit tilfelle der programvaren har hatt store problem med å skilje ut regionar og det kjem difor svært mange små regionar i nedre venstre hjørne. Samstundes ser ein fleire steinar som har den rette forma til å vera ein murstein. Etter å fjerna unormale steinar frå Figur 4.9 vert dei korrekt identifiserte steinane fjerna sidan dei vert sett på som unormale i denne samanhengen, som ein kan sjå i Figur 4.10. Det gjenstår dermed svært få regionar igjen i bilete. I tillegg er dei identifiserte regionane svært mangelfulle med tanke på å få med heile mursteinen. Ein stor grunn til dette er den varierande pikselverdien innan den enkelte murstein som for fargeteljarmetodane terskling og k-mean clustering vert vanskelig å skilje frå fugen.



Figur 4.9: Identifiserte grenser på mursteinen før fjerning av ikkje standardiserte steinar, der kvart nummer er ein eigen regionar i bilete.

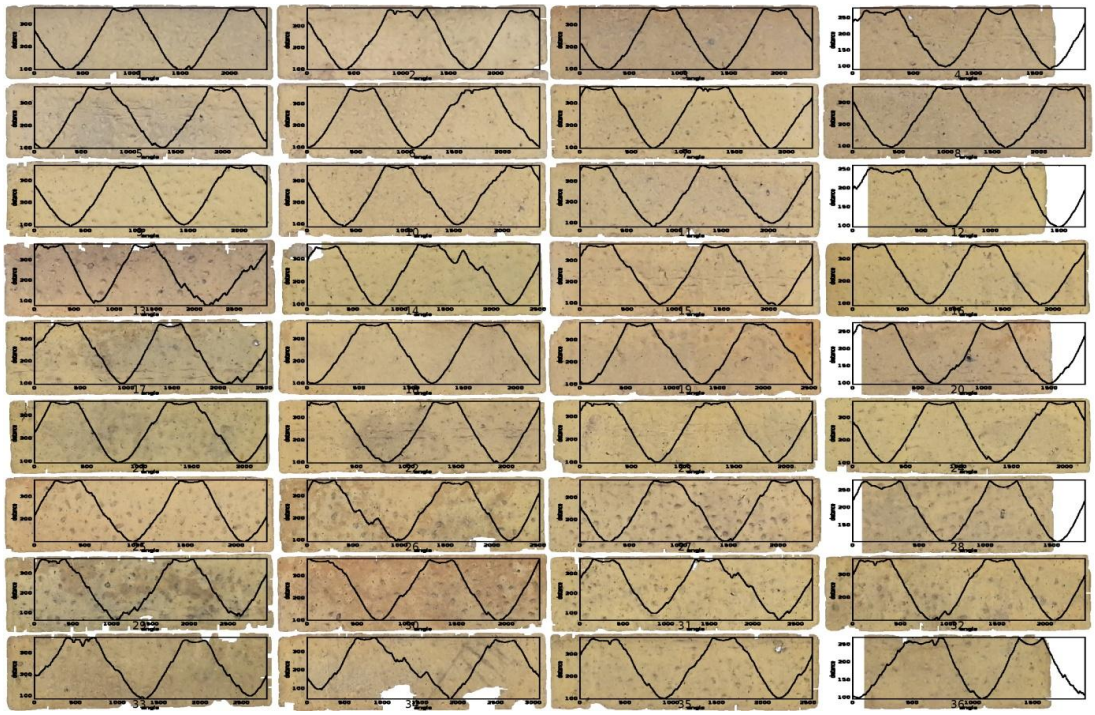
Figur 4.10: Identifiserte regionar etter fjerning av standardiserte steinar i bilete

Utplukk og identifisering av murstein er svært avhengig av eit godt maskebilete som allereie der har klart å skilje ut fuge og støy til kvit bakgrunnsfarge og murstein til svart forgrunnsfarge for at vidare identifiseringar skal ha ein nyttig effekt.

4.2.3 Signatur innsamling og drøfting

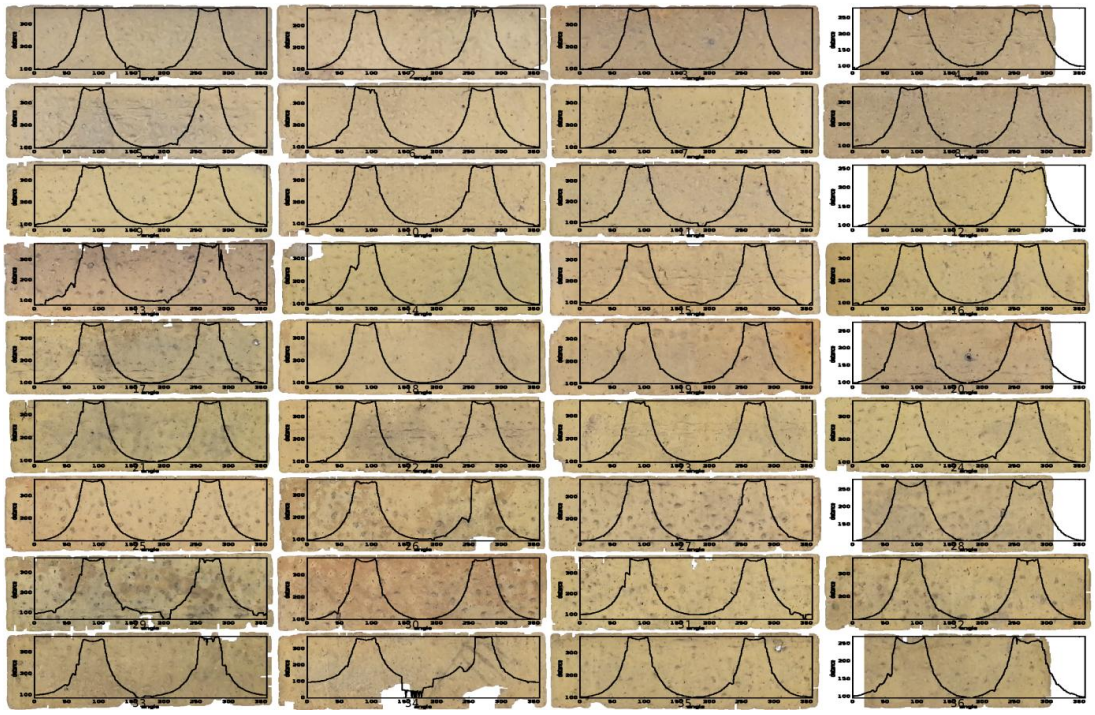
Punkta på omkrinsen er lagra i RoiManager etter partikkelanalysen, det er difor enkelt å hente ut x- og y-koordinater for grensa til objekta i bilete. Av Figur 4.11 kan ein sjå at det er litt vilkårlig kvar grenseverdiane byrjar, og dette gjer at det kan verte vanskelig å samanlikne mursteinane mot kvarandre. Toppnivåa representerer dei korte sidene av mursteinen, då det er størst distanse mellom midten og ut til desse sidene.

Sidan toppnivåa forskyver seg for dei ulike objekta kan det vera vanskelig å samanlikne signaturen på likt grunnlag. Difor har ein prøvd å bare lagra ein distanse per vinkel for å oppnå 360 målepunktar.



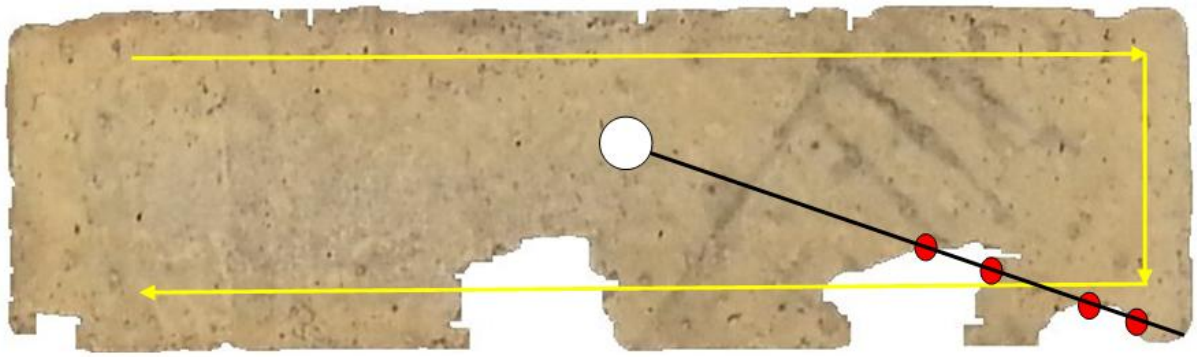
Figur 4.11: Mursteinen med signalet henta ut med distansen per piksel i omkrinsen. Ein kan sjå at det er litt varierande kvar omkrinsen startar, då toppunkta i signaturen er lokalisert på forskjellige stadier

Figur 4.12 visar korleis signaturane vert plassert ved å plassere dei etter vinkelen α frå Formel 3.3.



Figur 4.12: Signatur på mursteinen der signala er lagt inn med ein distanse per vinkel mellom 0 og 359 grader i omkrinsen. Ser at toppane legg seg i meir ordna rekkefølgje. Dermed veit ein kvar toppane skal vera for alle mursteinar.

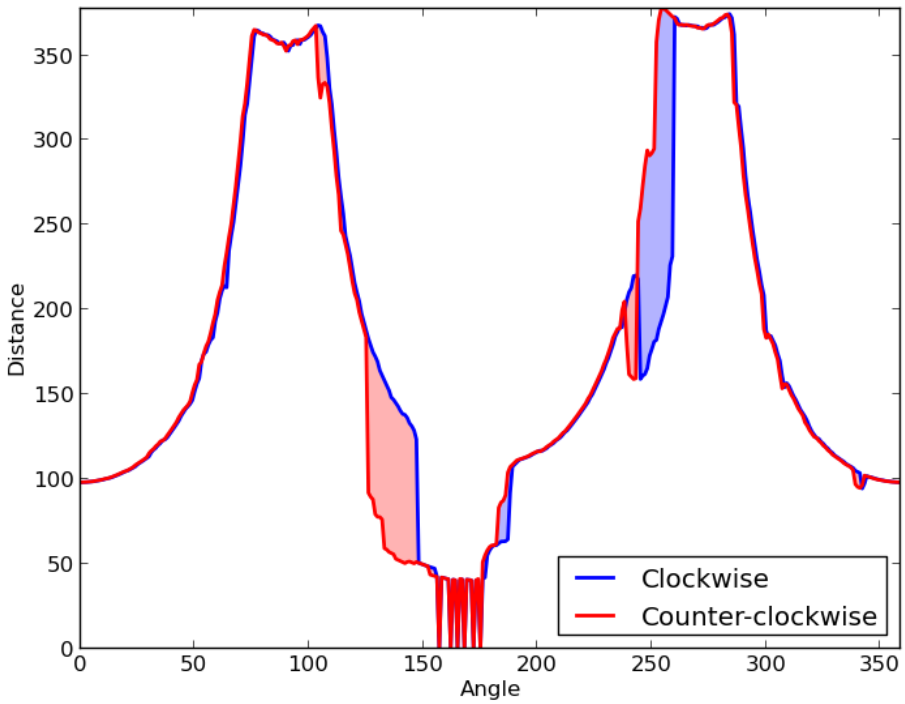
For at signaturen skal vera samanliknbar for alle steinane vert signaturen lagra med ei distanse for kvar vinkel. Eit problem som kan oppstå på grunn av denne måten er at ein overskriv informasjon som kan vera viktig. Figur 4.8 visar ein murstein med store frostskader. Algoritmen for signatur går igjennom omkrinsen på steinen i same retning som den gule linja. Frå nedre hjørne kan ein sjå at for same vinkel vil ein kome borti 4 ulike distanser.



Figur 4.13: Døme på samanstilling av gruppering der dei raude punkta alle har same vinkel, men forskjellige distanse. Algoritmen byrjar i øvste venstre hjørne og går så mot høgre før den går ned langs kanten i høgre som gul pil. Gul pil markerer kva retning algoritmen går igjennom omkrinsen.

Frå Figur 4.13 kan ein sjå at ein mistar informasjon om steinen ved at det same punktet i signaturen eigentleg skulle ha representert 4 forskjellige punkt langs grensa til mursteinen. Ved å snu traverseringa av omkrinsen til å gå mot klokka vil ein difor få ei anna signatur på

mursteinen. Signaturen på mursteinen for begge retningar kan sjåast i Figur 4.14, der det raude plotet er mot klokka, medan det blå plotet er signaturen utrekna med klokka.
Det skraverte feltet i raudt og blått er forskjellen mellom desse to ulike signaturane for dei same vinklane. Dersom mursteinen hadde vore heilt perfekt utan skader ville forskjellen mellom desse to retningane ha vore svært liten, sjølv om det framleis ville ha vore konflikt ved å skalere ned ein omkrins på over 1000 punkter til ei signatur med 360 punkter.

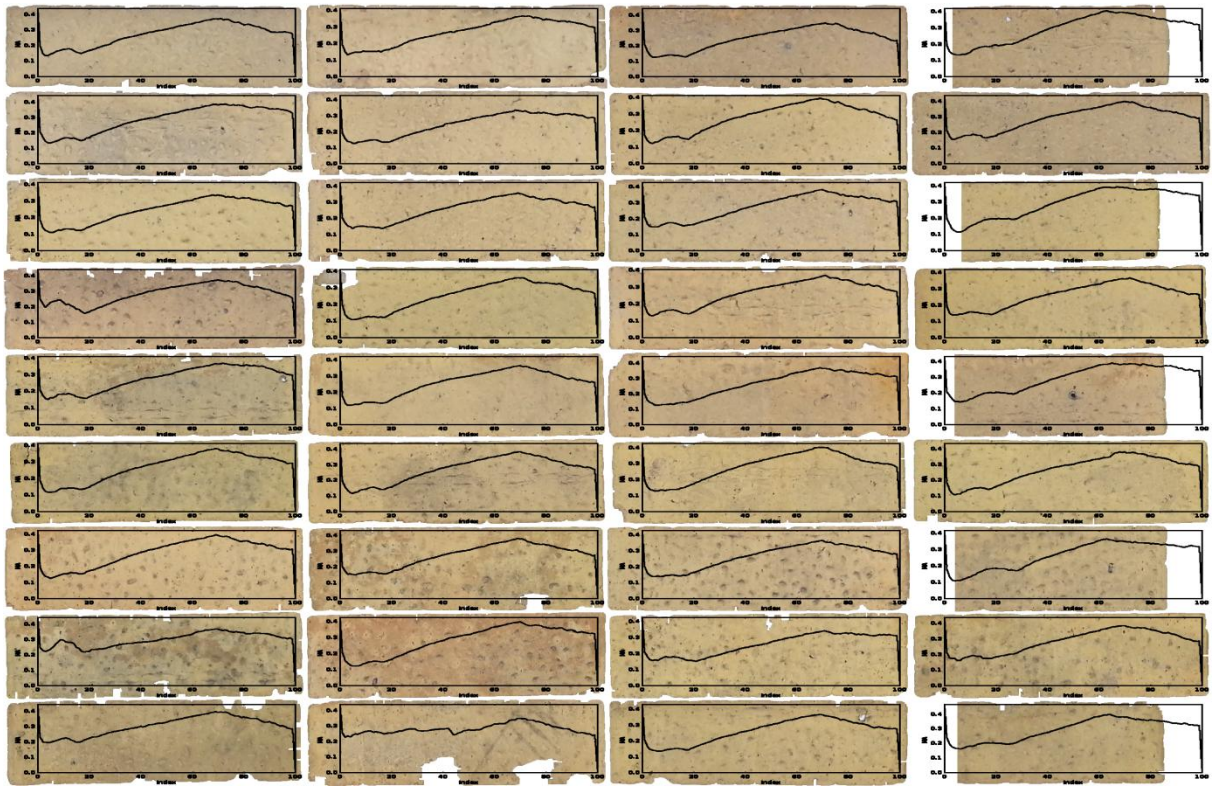


Figur 4.14: Signaturen til mursteinen i Figur 4.13. Den blå streken representerer signaturen ved å traversere omkrinsen med klokka, medan den raude er signaturen ved å traversere mot klokka. Dei skraverte felta er differansen mellom desse to signaturane.

Det ser ut til at val av retning på traverseringa vil kunne hente ut klare tal på skadeomfang i stein i form av holrom. Desse funna vil vera naturleg å diskutere vidare ved klassifisering av frostskader på mursteinane.

4.2.4 Køyring av AMT

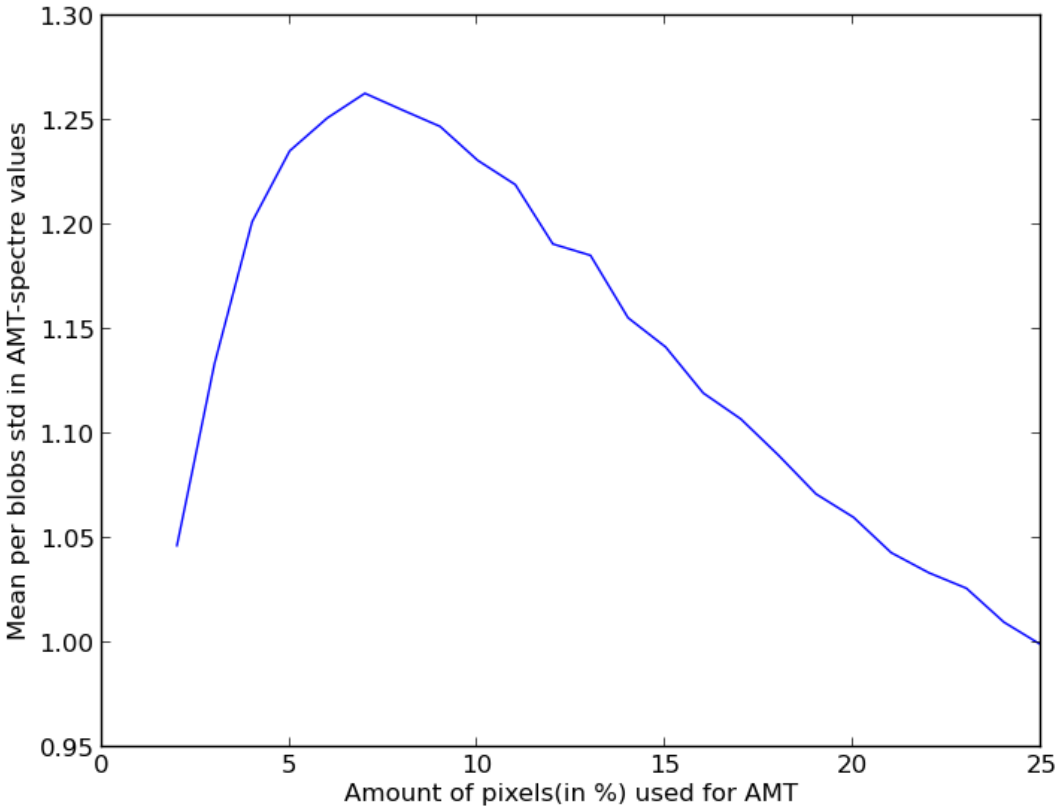
For AMT er det mogleg å køyre på både signaturen og på pikselverdiane i objekta. Begge metodane er tilgjengeliggjort for å sjå om det er mogleg å hente ut meir informasjon om kvart objekt. Grunnen til at ein kan velje datagrunnlag på denne måten er fordi det er Jython som kontrollerer prosessen frå utplukk av måleseriar til å nytte enkeltfunksjonar frå tilleggsprogramvaren AMT_spectra. I staden for å servere ein biletestack av mursteinen rett til AMT_spectra, kan Jython prosessera ein vektor som skal analyserast. Dette mogleggjer at ein til dømes kan sende inn signaturen inn i AMT for å leite etter nye mønster som vist i Figur 4.15. Ein kan allereie her sjå at den frostskadde mursteinen i Figur 4.13, oppfører seg på ein heilt annan måte enn "normalen" innan dette datasettet.



Figur 4.15: AMT-spekter køyrt med signaturen som inputsignal for mursteinane i bakgrunnen.

Ved å køyre AMT på pikselverdiane i bilete kan ein hente ut informasjon om korleis pikslane endrar seg. For å få ytterkantane i objekta i ein samanheng, vert pikselverdiane i bilete henta ut i ein spiral som går med klokka. I denne samanhengen ser ein at dei fleste mursteinane ikkje når heilt ut til kanten, og at det difor vil vera naturlig å ta vekk eit par pikslar frå kanten. Dvs, at i løpet av dei fire første rundane rundt objektet vil ikkje pikselverdiane vera tekne med. Etter dette vert det testa kor mange pikslar ein skal hente etterpå for å hente ut mest mogleg informasjon som helst kan skilje steinane frå kvarandre.

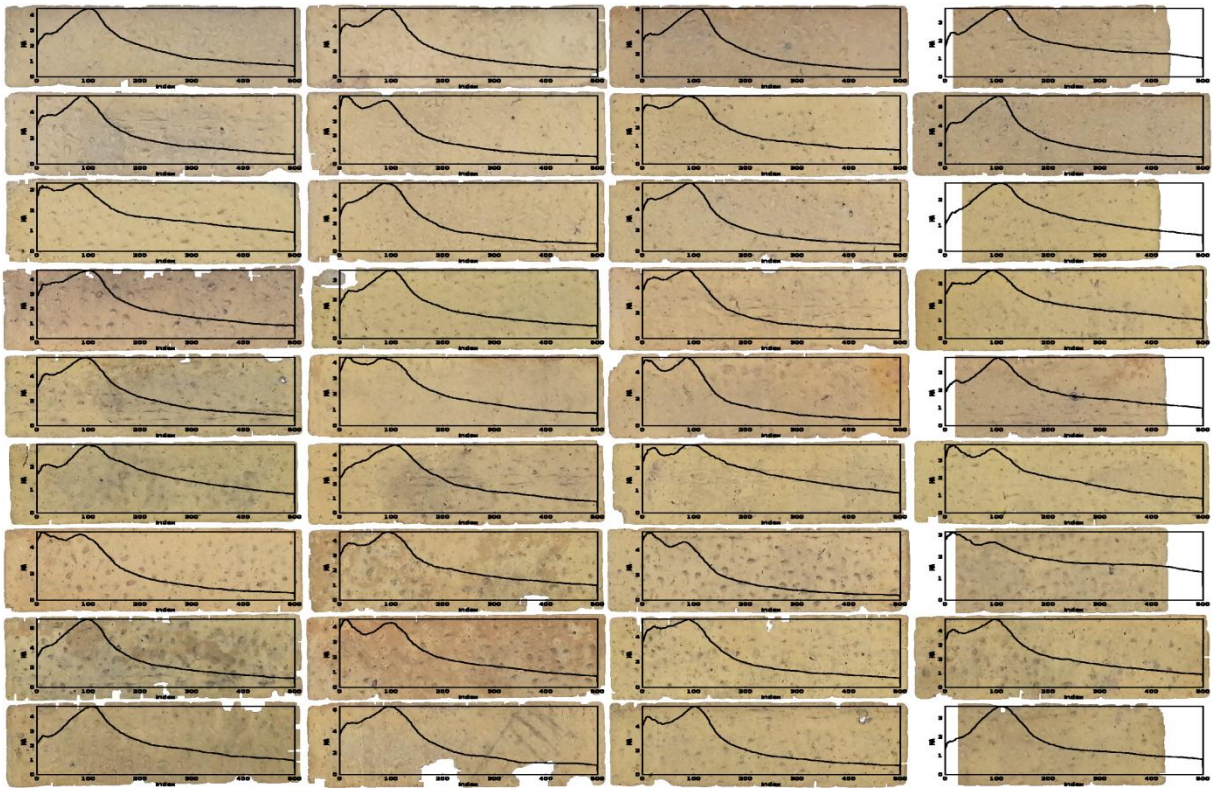
Dette vert gjort ved å hente ut 500 prøvar av eit utval på 2000 tilfeldige pikslar frå kvart objekt. Ein finn det gjennomsnittlege standardavviket(spreiing i datasettet) for 36 objekt ved å auke mengda av mursteinen som vart nytta for å få ein profil som i Figur 4.16. Mengda er auka f.o.m. 2% opptil og med 25%. Ut ifrå profilen i Figur 4.16 kan ein sjå at den gjennomsnittlege variansen er størst ved å hente ut 7% av pikselverdiane i objektet. Altså her er det størst varians innanfor datasettet, noko som kan tyde på at ved å nytte seg av 7% av pikslane etter dei 4 første rundane har størst innverknad på endringar av pikselverdiar. Figur 4.17 visar med raud markering kva for nokre pikselverdiar som vert nytta ved å hente ut 7% av steinen til å skaffe AMT-spekteret. Dette AMT-spekteret kan ein sjå i Figur 4.18. Figuren visar forskjellige spekter, men det er vanskelig å gje ein klar peikepinne på kva spekter som tilhøyrer dei ulike skadane på objekta.



Figur 4.16: Spreiingsprofil ved å auke mengda av pikselverdiane fleire objekt til å skaffe AMT-spekter. Ved størst varians, vil det og liggje størst forskjell innan datasettet.

Figur 4.17: Murstein der det raude feltet er 7% av pikselverdiane som vert nytta til å lage AMT-spekteret av steinen

Sidan mursteinen har ulik storleik vert det mykje kvite område rundt steinen slik at alle får same dimensjon. Dette gjev eit utslag som ein tydeleg kan sjå i dei minste mursteinane til høgre i Figur 4.18. I motsetning til dei "store" mursteinane som nesten fyllar ut plassen sin, får dei små mursteinane eit definert dump i starten og har dermed eit anna spekter å målast på. Samstundes kan ein ikkje skrelle vekk alle pikselane for å bare sitte igjen med ein uinteressant klump i midten av mursteinen. Dei aller fleste mursteinane ser ut til å ha frostskadane sine på ytterkanten av steinen.



Figur 4.18: AMT-spekter for mursteinane ved å benytte seg av 7% steinen til å laga spekteret.

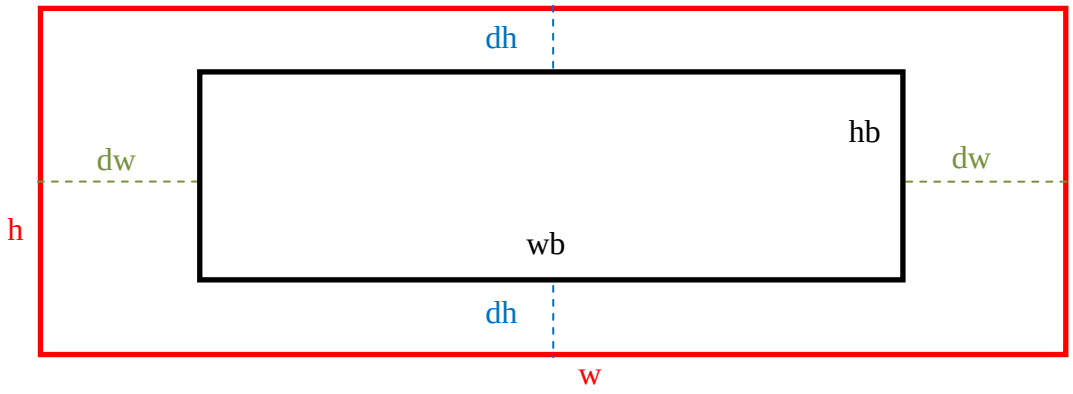
Problemet med å hente inn pikselverdiane på denne måten er at alle steinane eigentleg må vera homogene, og eigentleg fylle ut heile bilete. Ein kan til dømes sjå i Figur 4.19 at ein ikkje alltid vil klare å hente opp eigenskapane i kantane av steinen på denne måten. Det er tydeleg at kortsidene ikkje får innverknad på steinen ved denne metoden å hente pikseldata.

Figur 4.19: Pikselverdiar som vart nytta i AMT-spekteret er markert med raud strek for ein murstein som ikkje utnyttar potensialet i lista av bilete .

Ein kan gjere utfoldinga meir dynamisk ved å tilpasse området pikselverdiane vert henta i frå. Ved å redusere bredda og høgda til utvalet kan ein sørgje for at det faktisk er grenseverdiane som verkeleg vert tatt med. Frå partikkelanalysen har ein allereie bredda og høgda på kvart objekt og dermed kan ein finne differansen i boksen ved hjelp av formel 4.1:

$$dw = \frac{w - wb}{2}$$
$$dh = \frac{h - hb}{2} \qquad 4.1$$

4.1: Differansen(dw, dh) for høgde og bredde mellom murstein og stack. Bredda og høgda til stacken er w og h. Bredde og høgda til sjølve objektet er wb og hb.



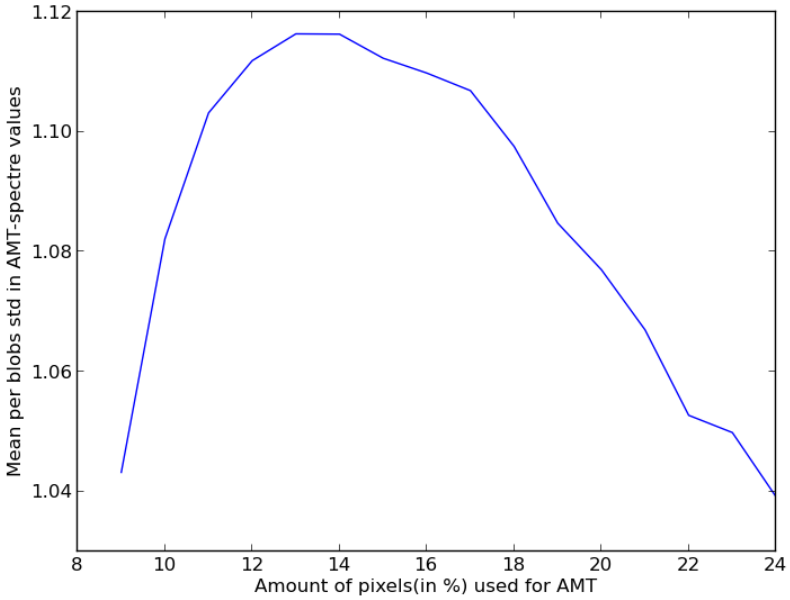
Figur 4.20: Skisse over korleis ulike storleiker heng saman med kvarandre. Raud er ytste dimensjon av stacken, svart er yttarste dimensjon til objektet inni stacken(denne varierar). Dei grøne og blå strekane vert rekna ut ved hjelp av formel 4.1

Med formel 4.1 og skissa i Figur 4.20 kan ein sjå at ein må justere posisjoneringa når pikselverdiane skal hentast før AMT-spekteret genererast. Ved å hente ut pikselverdiane i ein spiral må ein derfor innskrenke rektangelet ved å redusere bredda med *dw* for begge sider og *dh* for høgda i topp og botn. Sidan midtpunktet i stacken òg er midtpunktet i objektet vil ein difor sitje igjen med yttergrensa til objektet. Dette kan ein sjå på som ein meir dynamisk metode for å fjerne kantane som allereie drøfta.

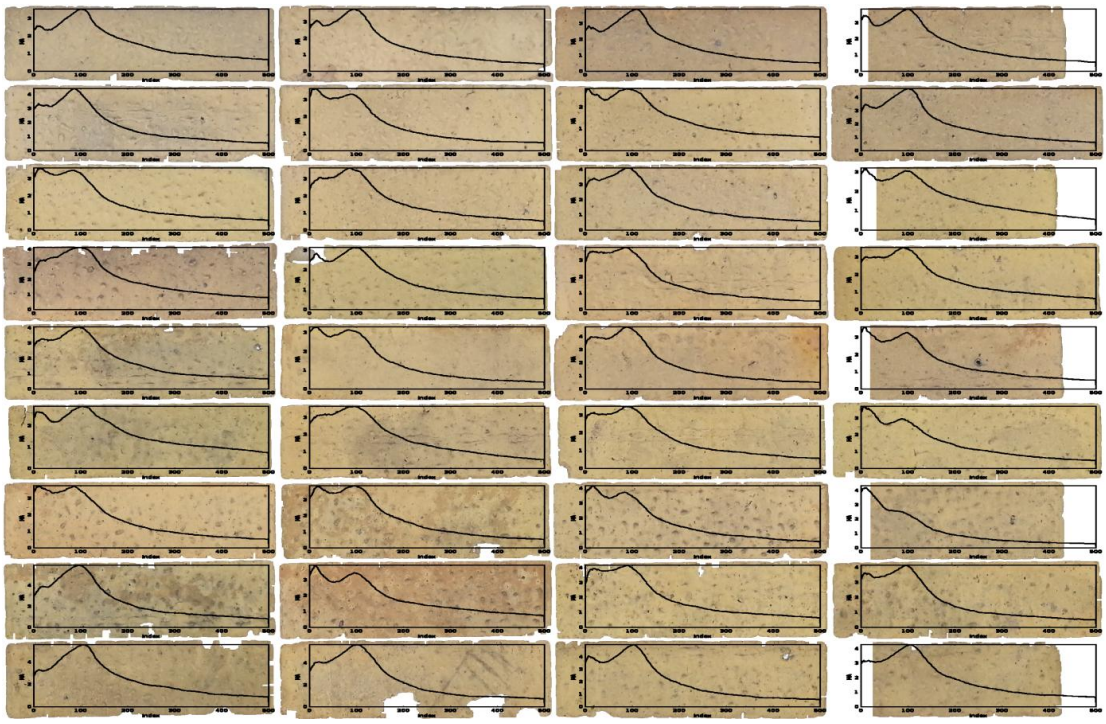
Ved å leggje til differansen i spiralen får ein henta ut pikselverdiane i området som vist i Figur 4.21.

Figur 4.21: Markerte pikselverdiar ved å hente ut 7% av steinen og redusere utvalsområdet basert på bredda og høgda til bloben, med utgangspunkt i stacken.

Med ein dynamisk boks som omtalt over er pikselverdiane meir tilpassa røynda. Ved å sjå på variansen i AMT-spektera kan ein finne ut kor mange prosent ein bør ta med av mursteinen for å få mest mogleg spreiing. Figur 4.22 visar denne nye spreiinga, der ein kan tydeleg sjå eit toppunkt på 13-14%. Ved å velje 14% av mursteinen i AMT-spekteret får ein fram spektra vist i Figur 4.23.



Figur 4.22: Spreiinga i AMT-spektra ved å benytte seg av dynamisk boks for å finne pikselverdiane i grensa til bloben.



Figur 4.23: AMT-spektra ved å hente ut 14% av mursteinen med dynamisk boks for å sikre at pikselverdiane er i randsonen av bloben.

Ein kan sjå at AMT-spekterene frå dei små mursteinane ikkje skil seg noko særleg ut frå andre tilnærma heile steinar. Dermed kan det vera at ein har løyst problemet med at storleiken spelar inn på skadane til steinen i AMT.

4.2.5 Køyring av GLCM

Tilleggsprogrammet GLCM_spectra frå biletegruppa på UMB tek inn ei liste av bilete i gråtone. Etter analysen vert dei utrekna variable omtalt i 2.7.3 lagra i samletabellen til programvaren.

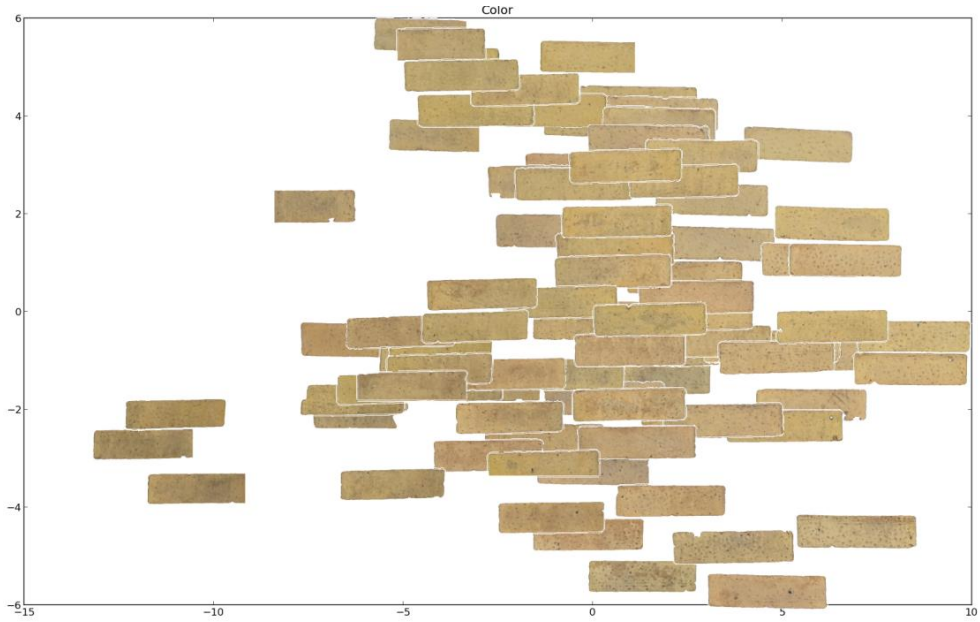
4.3 Avdekke eventuelle frostskader og andre observasjonar

Fiji lagrar innsamla data om mursteinane i ei *.csv* fil der kvar rad representerer kvar murstein og heile datasettet kan sjåast på som X-matrisa omtalt i Figur 2.19. Ved å velje ut grupperingar som AMT på piksler, signatur, form, farge og GLCM kan ein leite etter strukturar og forklaringar basert på eigenskapane til mursteinane.

4.3.1 PCA

Prinsipal komponent analyse vert køyrd med Hoggorm som er ei ny implementering av PCA i Python. Å køyre PCA analysen i Python gjev fleire fordelar der prototypen har tilgang til resultata direkte etter køyringa og dermed kan lage meir forklarande resultat med mursteinen i eit score-plot.

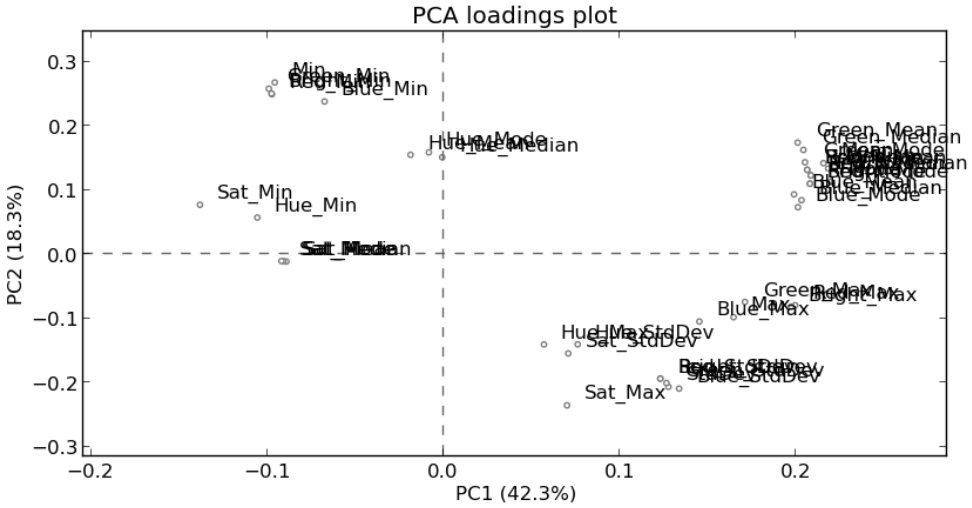
Fargeparameter



Figur 4.24: Scorene til PCA på fargeparameter

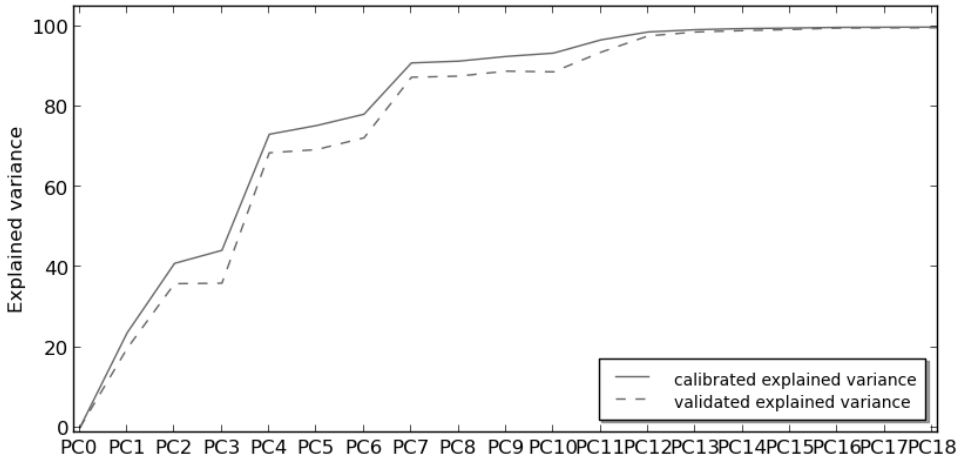
Ein kan sjå i Figur 4.24 at fargene kan skilje ulike mursteinar frå kvarandre. Ved å plotte prinsipal komponent(PC) 1 mot PC2 kan ein sjå at dei mørke fargane held til i nedre sjiktet av figuren. Den litt lysare gul fargen ser ein strekkjer seg meir opp mot toppen, medan den raudlege fargen ser ut til å trekkje seg meir mot nedre høgre hjørna.

I Figur 4.25 kan ein sjå ladningane etter PCA analysen. Det ser til dømes ut som om at høge fargekonsentrasjonar førar til at mursteinen og vil halde seg til høgre i Figur 4.24, noko som samsvarar med mørk mursteinar til venstre med låg fargemetning.



Figur 4.25: Ladningane til fargeparameterane ved prinsipal komponent 1 mot 2

Figur 4.26 visar at ein treng dei 7 første prinsipal komponentane for å kunne forklare ca 90% av variansen til fargeparametra.

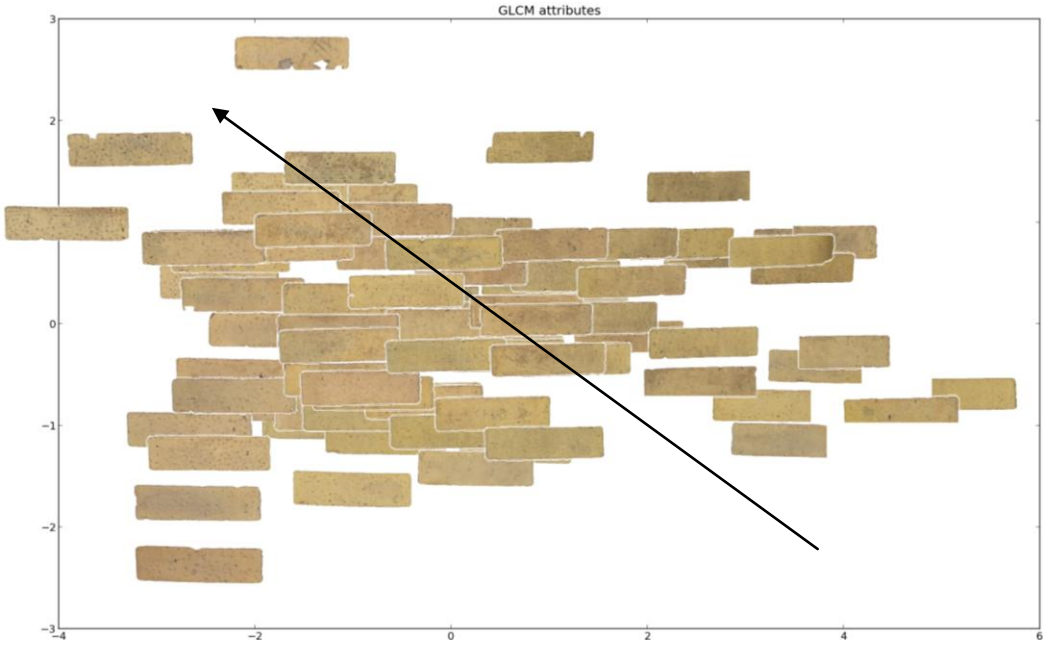


Figur 4.26: Forklarande varians per akkumulerte prinsipal komponent ved fargeparameter

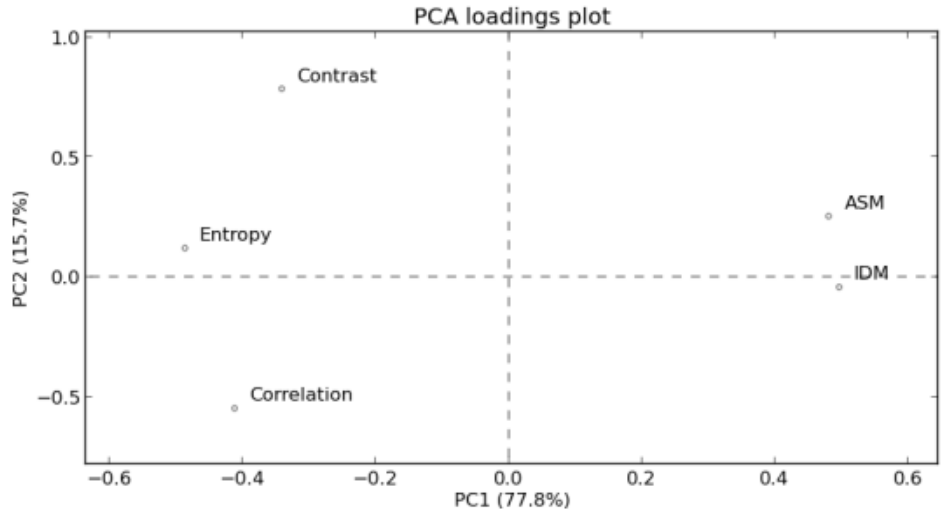
GLCM

Ved å køyre PCA analysen på GLCM verdiane, får ein sett korleis overflateteksturen til ei murstein breiar seg ut som vist i Figur 4.27. Til høgre i figuren ser det ut som om mursteinane har ein mattare overflate enn dei til venstre, spesielt i det øvre hjørna. Dette samsvarar med ladningane i Figur 4.28 der Entropi er til venstre, som tydar kaos og større forskjell blant pikselverdiane, medan det uniforme, matte ligger til høgre med høg IDM.

GLCM fungerar veldig godt for å forklare teksturen til objekta sidan dei ulike eigenskapane frå GLCM analysen gjev gode tilbakemeldingar på overflata til mursteinane. Diverre ser det ikkje ut til at det forklarar skadeomfanget primært, sjølv om dei store skadane ser ut til å plassera seg i øvre del av score-plotet.



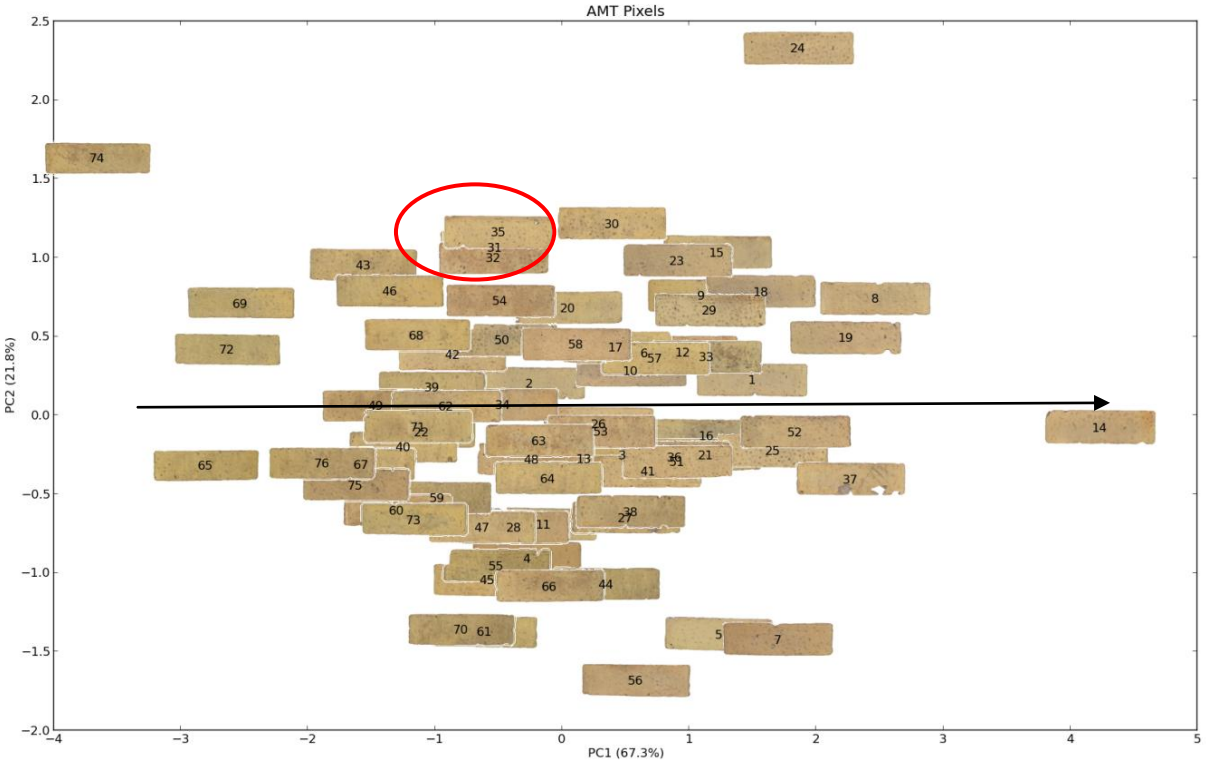
Figur 4.27: Scorene til GLCM på mursteinen, kan sjå ut som om graden av rufsete overfalte til steinane går i retninga til svart pil ved meir pigmenter i mursteinane i øvre venstre hjørne enn i motsatt.



Figur 4.28: Ladningane til GLCM variablane i PCA-analyse

AMT-spekter frå pikselverdiar

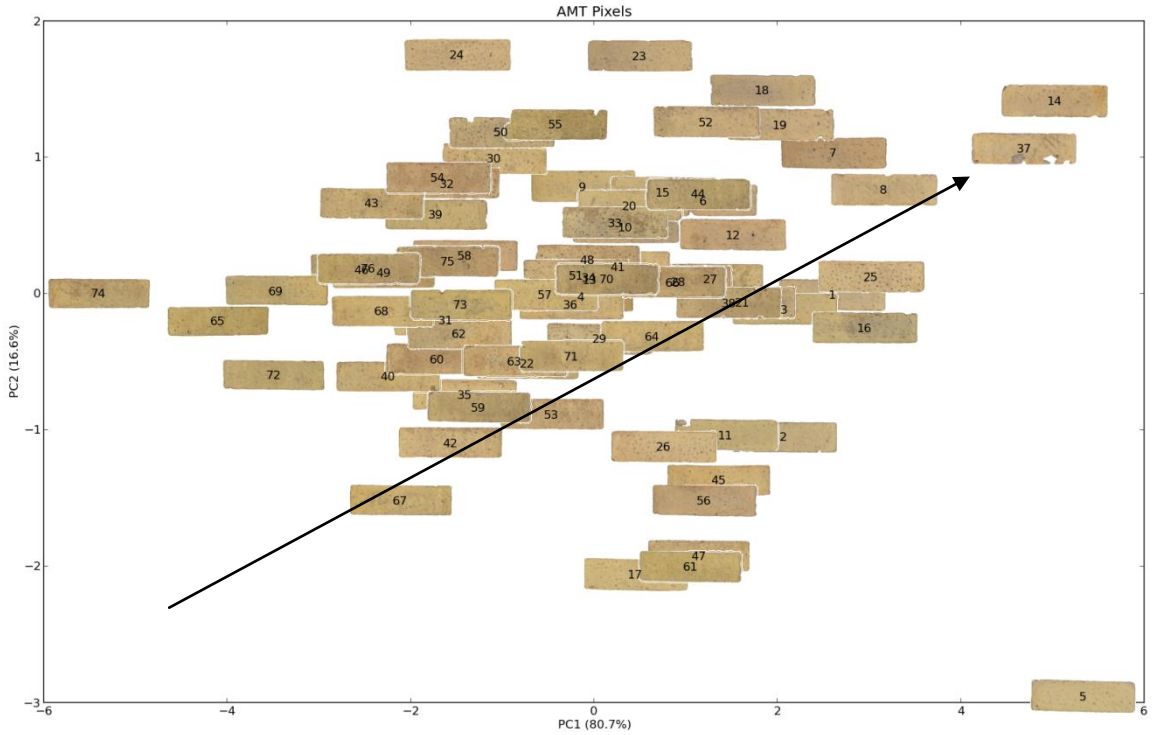
AMT-spekteret er diskutert i avsnitt 4.2.4 der ein kom fram til at ein burde nytta seg av ein dynamisk storleik for å få med den viktige ytterkanten til steinen. Samstundes skulle det vera mest spreiing i datasettet ved 14% av pikselverdiane til objektet som grunnlag for måleserie.



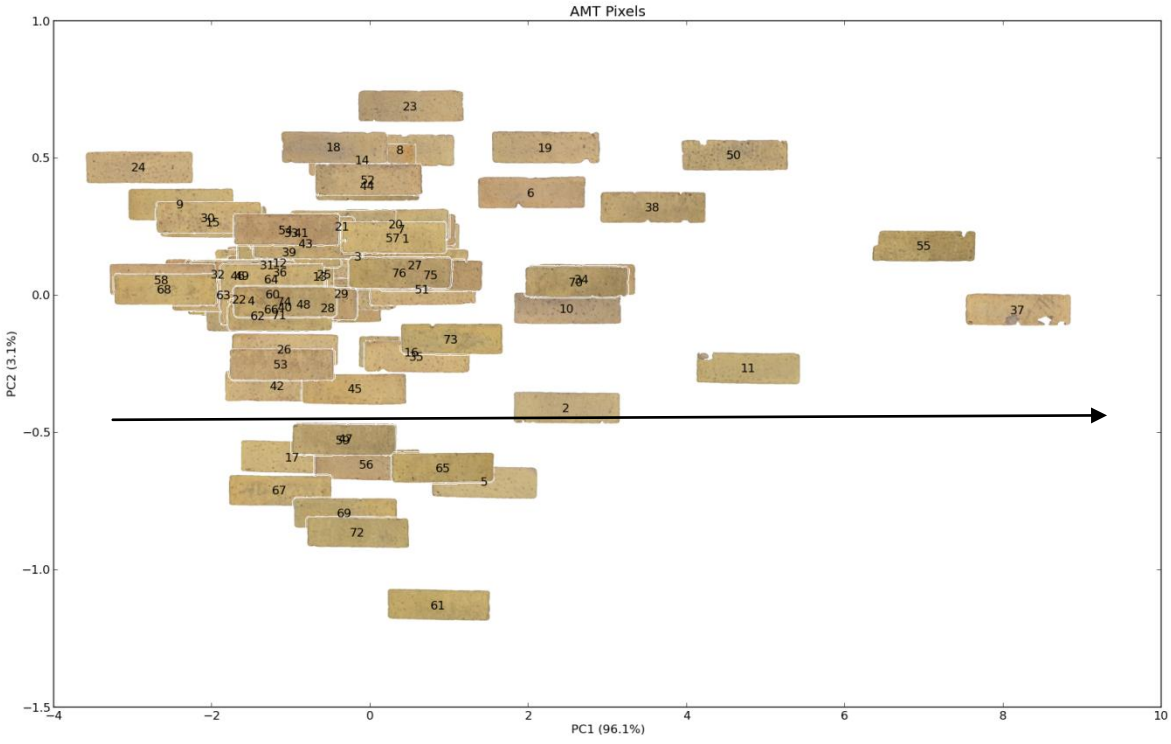
Figur 4.29: PCA på AMT-spekteret frå pikselverdiar.

Figur 4.29 visar resultatet av PCA på amt-spekteret ved desse føresetnadane. Fleire av dei skada mursteinane går mot høgre, medan dei friske går mot venstre. Samstundes kan det sjå ut som om nokre av dei skada mursteinane likevel dukkar opp midt i blant friske som til dømes nummer 35. Ved nærmare ettersyn kan det vera at teksturen på mursteinen har ei innverknad på biletet og på den måten gjev skaden mindre betyding i heilskapen. Ein tok derfor ei ny køyring av AMT-spekteret der gråtonen på pikselverdiane vert bytta ut med det binære maske bilete. Dermed vert kvart objekt representert med svart forgrunnsfarge og kvit bakgrunnsfarge. På denne måten håpar ein å sjå kva innverknad teksturen vil ha på resultatet. Kanten av mursteinen bør nå få aller høgaste prioritering i PCA.

Figur 4.30 visar resultatet av teksturendringa og det fyrste ein kan leggje merke til er at den forklarte variansen i PC1 har auka frå 67,3% til 80,7%. Den nye prinsipal komponenten har altså fått eit større bidrag til å forklare variasjonen i datasettet. Tendensen i figuren ser ut til å vera at skadane trekkjer seg opp i øvre høgre hjørne. På eit binært bilete med lite variasjon i ytterkantane av biletet kan det tenkjast at 14 % igjen vert for lite til å forklare heile historia om frostskader. Ein prøvar difor AMT-spekteret med 50% av pikselverdiane i objektet for å sjå om resultata vert tydlegare.



Figur 4.30: PCA-plot av murstein etter å ha fjerna teksturen frå objekta.



Figur 4.31: PCA-plot på amt-spekteret på pikselverdiar etter å ha fjerna teksturen og sett på pikselverdiar frå 50% av mursteinen. Ser at stein med frostskader legger seg langs PC1.

I Figur 4.31 ser ein enda tydlegare at stein med frostskade held seg til venstre i PCA-plottet. I tillegg har den forklarte variansen auka i prinsipal komponent 1. Ved å fjerna teksturen på biletet med ein binære overflata ser det ut til at ein klarar å skilje murstein med frostskader frå

friske og heile mursteinar. Ein kan sjå av figuren at dei skada steinane spenner seg ut frå klynga og opp mot den øvre delen. Men den største forklaringskomponenten til skada stein ser ut til å vera PC1.

Det skulle altså vera mogleg å skilje ut murstein med frostskade basert på PCA på AMT på pikselverdiar.

4.3.2 LDA

Ein annan metode for å klassifisere tilstanden til mursteinsvegger er å nytte lineær diskriminantanalyse. Ved å lage ei klassifisering med dummyvariable for kva mursteinar som har frostskade, kanskje for små skader og friske murstein. Desse verdiane vert så lagt inn i ei Y-matrise for klassifisering.

For å finne ut kva parametre og modell som fungerer til dette formålet må ein bruke både eit kalibreringsdatasett og eit valideringsdatasett for å sjekke om modellen fungerer i begge samanhenger. Difor bør ein først teste om modellen klarar å finne sine egne klasser før ein prøver nye.

Den manuelle klassifiseringa av mursteinen i Figur 4.32 er satt til at store synlege frostskader er satt i gruppe A, gruppe B er små frostskader som er vanskelegare å oppdaga medan gruppe C er friske mursteinar.

Gruppe A	Gruppe B	Gruppe C
10	15	51

Tabell 4.3: Oppteljing av murstein i kvar frostskade gruppe A, B og C, der A er store frostskader, B, er små frostskader og C er friske mursteinar

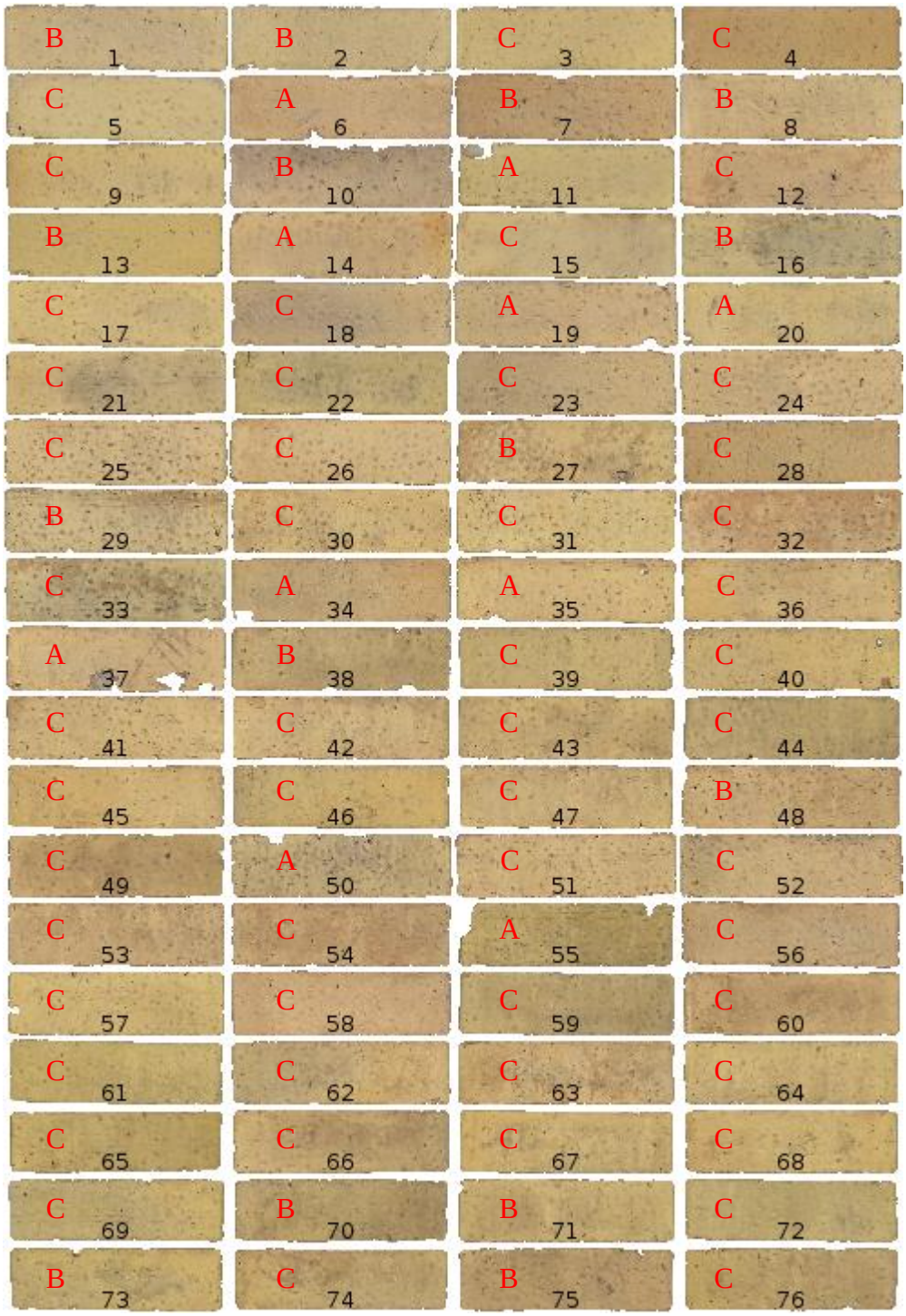
For å finne beste modellen i LDA må ein prøve denne for ulike parametere. Resultata av køyringane på datasettet er gjengjeve i Tabell 4.4.

For datasett nr 7:	Matlab (Frå Ulf)	
	LDA vanlig	LDA med PLS
amtpix	Feil	78 %
amtsign	Feil	84 %
color	88 %	72 %
glcm	68 %	65 %
shape	78 %	76 %
Normalisert signatur	Feil	100 %
Normalisert differanse i signatur	Feil	99 %

Tabell 4.4: Resultat av treffprosenten av klassifisering på LDA med ulike inngangsparametere, der feil er at modellen ikkje klarar å klassifisere datasettet.

Ut ifrå resultata i Tabell 4.4 bør ein sjå vidare på modellen LDA med enkel PLS før klassifisering. Parametera ein bør sjå på er *normalisert signatur* og *normalisert differanse i*

signatur for å lage modeller som så testast med nytt datasett. På denne måten får me sett kva modellar som fungerar betre på tvers av dei ulike datasetta.



Figur 4.32: Datasett for å kalibrere LDA-modell, Gruppe A har frostskade, B, har litt frostskadeskada og C er friske mursteinar

Modell frå 7:	Matlab LDA med PLS (Frå Ulf)	
	Datasett 6	Datasett 6 slått saman A og B
amtpix	53 %	53 %
amtsign	42 %	47 %
color	31 %	56 %
glcm	50 %	50 %
shape	31 %	47 %
Normalisert signatur	50 %	58 %
Normalisert differanse i signatur	58 %	61 %

Tabell 4.5: Resultata av klassifisering av skader med bruk av modell frå Tabell 4.4.

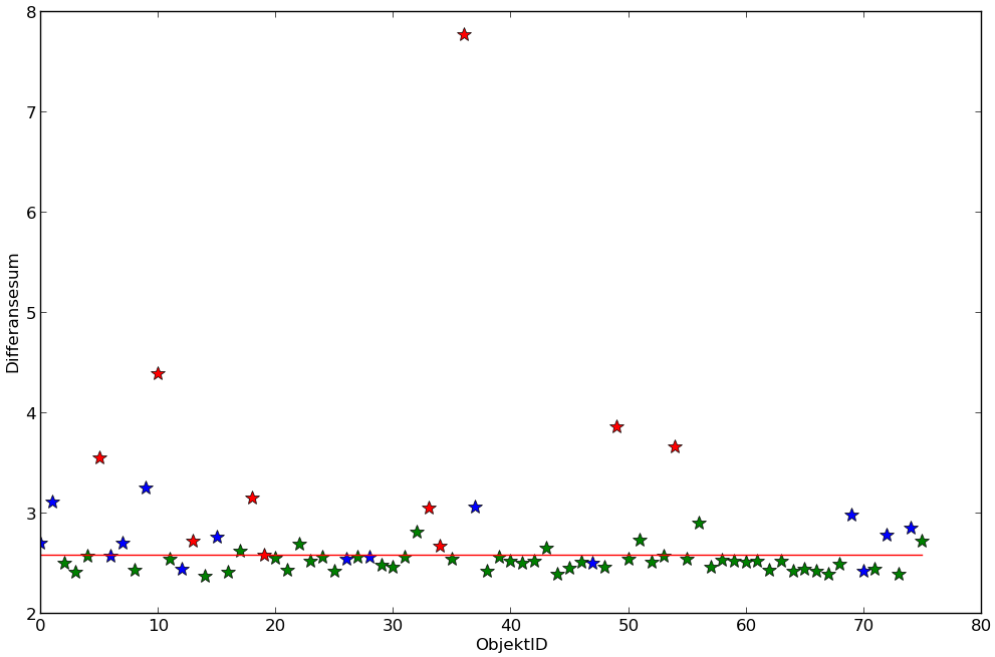
Resultata i Tabell 4.5 er litt under pari med tanke på dette er to svært like datasett sidan det er to bilete av same vegg. Noko av grunnen til avvika kan vera at den manuelle kartlegginga av skader ikkje er presis og konsekvent nok til at strukturen kjem godt nok fram i LDA.

4.3.3 Eigenutvikla frostskadedetektor basert på signatur

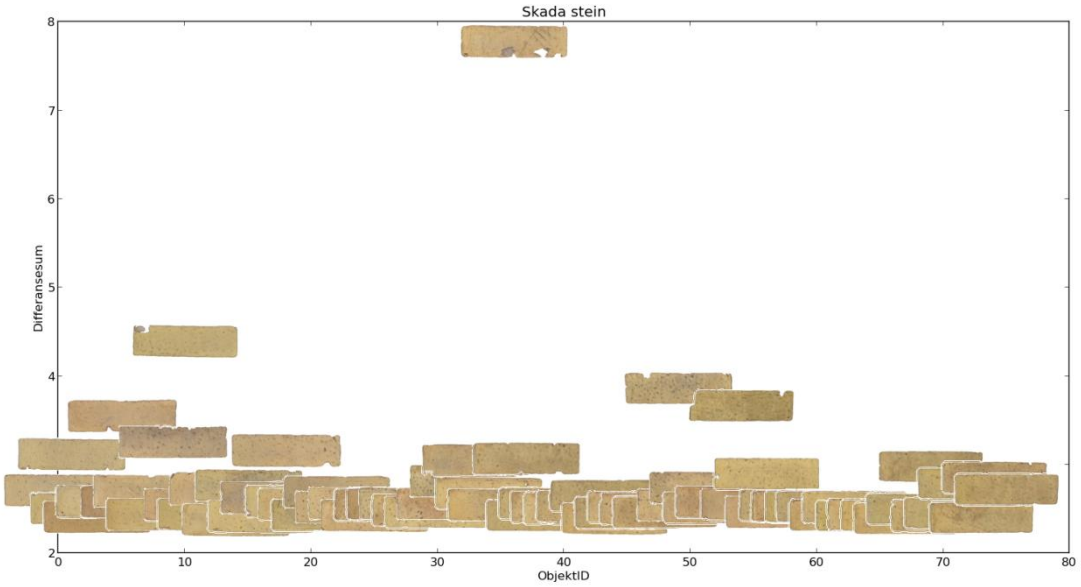
Ein annan metode for å leite etter skada stein er å nytta oppdaginga frå avsnitt 4.2.3 der det er tydeleg forskjell på kva veg traverseringa rundt omkrinsen vert gjennomført som ein kan sjå i Figur 4.14. Det vil alltid oppstå ein forskjell mellom desse to retningane for traversering. Spørsmålet er om den forskjellen i tillegg har eit klart skilje for murstein som har synlege frostskader sett opp mot murstein som ikkje har ein synleg skade? Ein må ta omsyn til at det er områder i biletet som har pikselverdiar som skil seg, og dermed skapar ein hump i bilete.

Den samla summen for differansen mellom desse to retningane vil uansett auka desto større differanse det er. Frå Figur 4.33 kan ein sjå at dei friske steinane (grøne stjerner) held seg i det nedre sjiktet, medan mykje frostskade (raude stjerner) held seg i det øvre sjiktet. Små frostskader (blå stjerner) er i mellomregionen. Til ein viss grad ser det ut til at differansen mellom retninga på signaturutrekning kan nyttast til å forklare frostskadane til mursteinen. Ein annan interessant ting er at storleiken på frostskaden ser ut til å auka med differansesummen, noko som er tydlegare i Figur 4.34.

Dersom ein nyttar den raude streken som utgangspunkt for å dele inn mursteinen i skada/ikkje skada murstein kan ein sjå at 13 mursteinar er utanfor sitt område, noko som tilseier at 83 % av steinane er klassifisert korrekt i skade eller ikkje skade.



Figur 4.33: Fordeling differansesummen mellom signaturretningane, der raud er store frostskader, blå er små frostskader og grøn er friske steinar.



Figur 4.34: Fordeling av differansesummen mellom signaturretningane med bilete av blobane

Ved å estimere ei grense for å skilje skada murstein frå frisk murstein ser det ut til at ein iallfall får skilt dei grove skadane frå dei heilt friske. Men støy i form av hakk i omkrinsen kan føre til feil klassifisering av mursteinen. Ved bruk av denne metoden for klassifisering er det altså svært viktig at programvaren klarar å identifisera mursteinen i bilete utan særleg støy. Fordelen med denne metoden er at den er objektiv og har ein ibuande effekt med å vera ein enkel skala for å gradera frostskaden på ein murstein.

4.4 Skadeindeks

Ein skadeindeks basert på rutinane og klassifiseringane nemnt tidlegare bør vera uavhengig av eining. Mykje av grunnen til dette er at bileta ikkje er garantert å ha same skala, faktisk vil det vera sjeldan at bileta er tatt frå same distanse og vinkel for alle tenkte veggar. Difor er det ikkje sikkert at dei ulike storleikane for mursteinen vil vera direkte samanliknbare.

Eit anna moment som bør betraktast er graden av frostskader. Det er forskjell på graden av frostskade ein murstein vil ha, som påverkar graden av omfang på skaden. Dermed bør ein større skade få ei større indeks.

Andre indeks- og karaktergivingar innan byggebransjen er energimerking som har ein karakterskala med bokstavar mellom A og G. Bygningar med energimerke A har lågt energiforbruk, medan karakter G tyder på eit høgt energiforbruk [39].

Ein kan tenkje seg at en liknande skala kan fungere for mursteinsvegger der A er ein frisk stein, medan F er ein murstein med mykje frostskader.
Etterpå kan ein moglegvis ta ein gjennomsnittleg karakter for heile veggen/biletet som ein endeleg indeks. Denne indeksen vil dermed kunne fortelje tilstanden for heile veggen for enkel samanlikning. Samstundes bør ein få varsel om kvar på veggen det er størst frostskade for å kunne utbetre dette før skadeomfanget vert for stort.

Eit mogleg grunnlag som ser ut til å kunne gje eit svært godt resultat er å ta differansen i signaturen på dei ulike retningane av traverseringa på omkrinsen. Då ser det ut frå resultata som om ein kan laga ein svært god indeksverdi basert på graden av skade. Her kan karaktergivinga gå frå ein frisk stein med karakter A, medan til dømes F står til store skader på stein og bør skiftast ut. Graden i mellom kan angi kor langt skadeomfanget er kome basert på auka i summen ein kan sjå i Figur 4.34.

For å setje karaktergivinga i perspektiv er det nok viktig å løfta skadeomfanget frå enkelt murstein og opp til biletet av veggen. Her må ein og hugse på at biletet ikkje naudsynt inneheld heile veggen, og det er difor eit spørsmål om kva del av veggen som er avbilda. For å kunne samanlikne skadeomfanga opp mot forskjellige bilete bør ein bruke gjennomsnittet av indeksen til kvar murstein for å danne ein karakter for heile bilete.

Tabell 4.6 gir eit forslag til estimering av karakterar basert på differansesummen til differansen i signaturretningane.

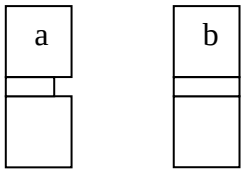
A	B	C	D	E	F	G
< 2.5	< 3.5	<4.5	<5.5	<6.5	<7.5	>7.5

Tabell 4.6: Forslag til grenseverdiar for ein karakterskala for frostskade på murstein basert på differansesummen til signaturen.

4.5 Datasett

Under leitinga av referansebygningar for oppgåva, vart det oppdaga ein tendens på mursteinsbygningar. Det såg ut til at det kom hyppigare, og moglegvis tidlegare, frostskader på mursteinsvegger der steinen stikker litt ut frå fugen. Tverrsnittet i Figur 4.35a) illustrerer korleis desse hyllene på veggen ser ut. Det kan sjå ut som om vindriven regn får ei større flate å leggje regnet på som kan drenere ned i det porøse materialet. Dermed vil det oppstå større sprekkdanningar og avskalling enn i Figur 4.35b) der vatnet får ei større hindring med å trenge inn i mursteinen.

Mursteinsvegger der mursteinen står utanfor fugene risikerer å få ein avskalling som følgje av frost skader. Men mursteinsvegger der fugene stikker like mykje ut som steinen oppstår det ikkje avskalling som følgje av frostskader/vatn som utvidast ved frysing av is.



Figur 4.35: Illustrasjon på tverrsnitt av murstein og fuge der a) stikker mursteinen ut i forhold til fugen, medan b) er fugen like langt ut som mursteinen.

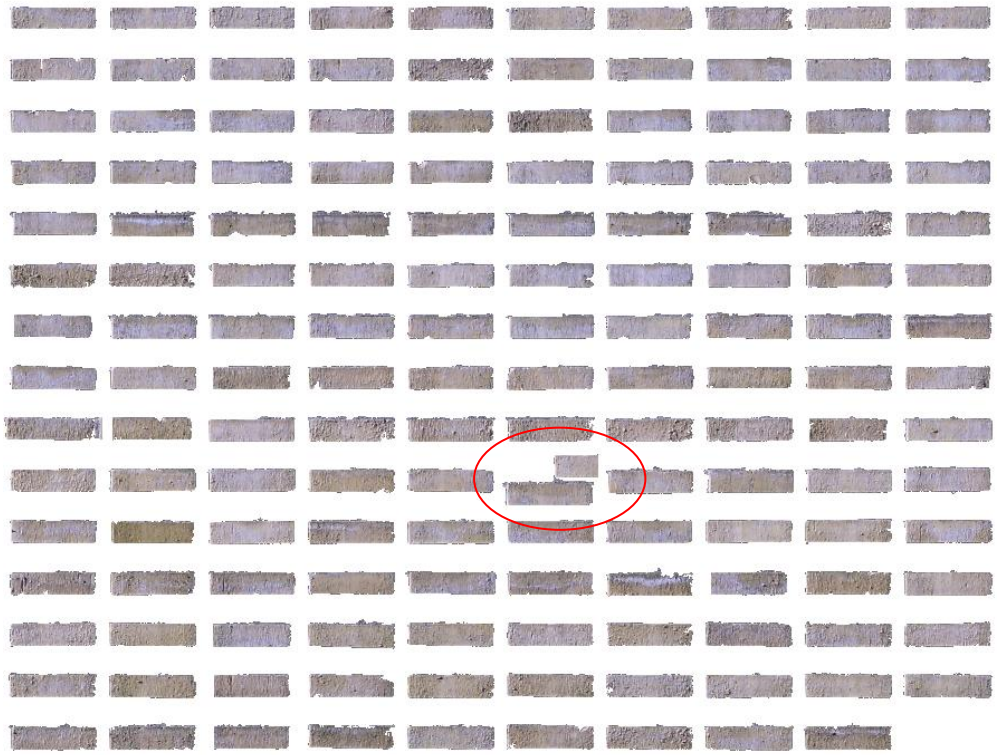
Referansebygningane omtala i avsnitt 2.1 gav fleire gode datasett som visar at den utvikla programvaren klarar å skilje ut murstein frå eit fargefotografi. Samstundes viser dei at protypen har rom for forbetring for å få fram steinen på ein god måte i fleire ulike høve. Dei påfølgjande resultata har alle nytta k-means clustering.

4.5.1 Resultat frå Stavangergata

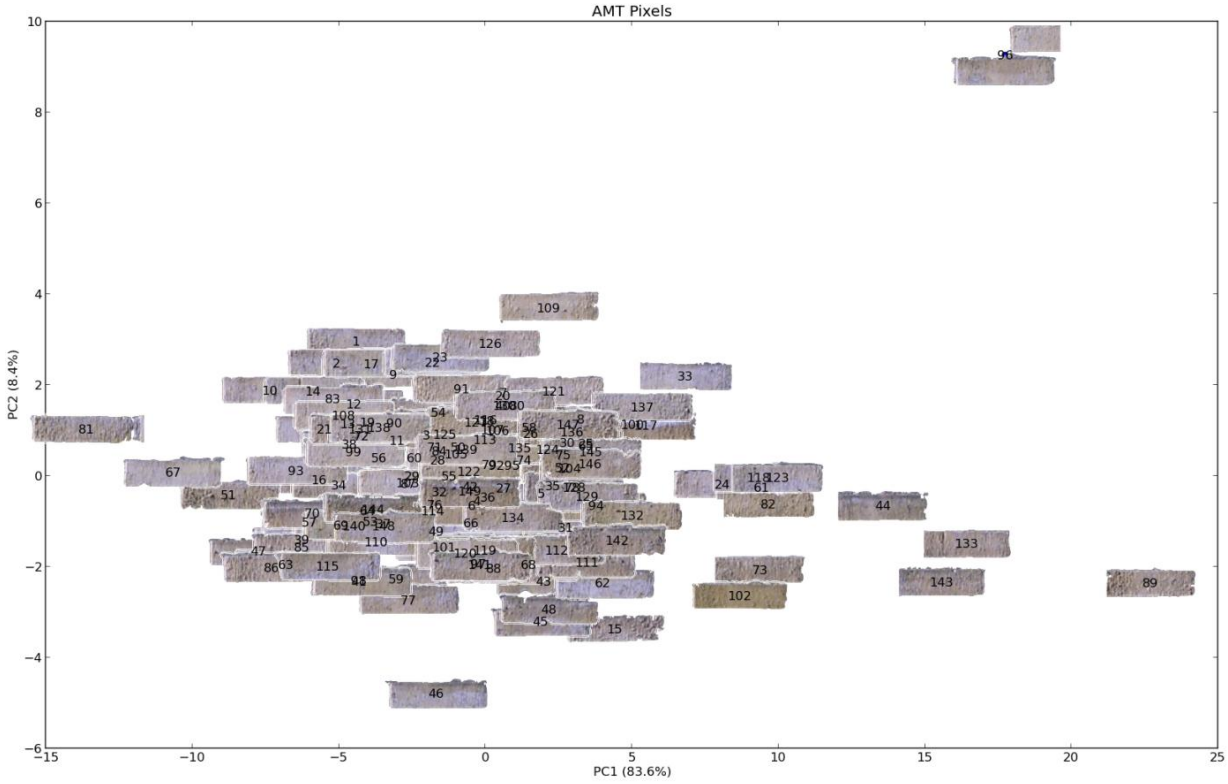
Figur 4.36 viser at programvaren har klart å skilje mursteinen frå veggen ganske godt, der det er tydeleg at mursteinane har fått si karakteristiske firkanta form. Men ein ser og frå montasjen av mursteinane at det er to steinar programvaren ikkje har skilt frå kvarandre og som heller ikkje har vore klassifisert som unormal nok til å fjernast. Dette øydelegg litt for storleiken til biletestacken og er skyld i at det vert mykje kvit rundt mursteinane. Ein annan ting som er tydleg i Figur 4.36 er at det ikkje har vore enkelt å skilje kanten av mursteinen frå fugen, då det er kanskje overkant mykje ruglete mursteinskantar.

Frå plotet av PCA-scorene i Figur 4.37 ser ein at tendensane er at frostskadde mursteinar ligg mot høgre i bilete. Men sida den gjengse mursteinen er litt meir ruglete i utgangspunktet vert det meir støy i dette plotet og vanskelegare å fast bestemme frostskaden til kvar stein.

Veggen som sett under eitt har med klassifiseringa gitt i Tabell 4.6 vert klassifisert som ein C vegg. Noko som frå hovudbilete ser ut til å stemme bra med ein vegg med litt frostskader.

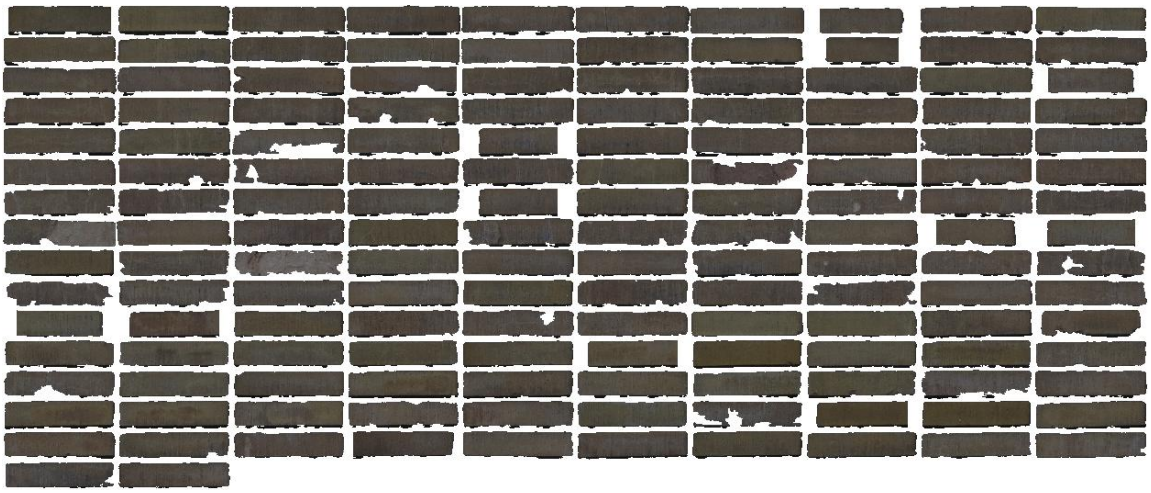


Figur 4.36: Dei fråskilte mursteinane etter køyring av programvaren på for Figur 2.3a). Ein kan sjå at det er ein murstein som skil seg ut frå mengda og som eigentleg burde ha vore oppdaga av programvaren.

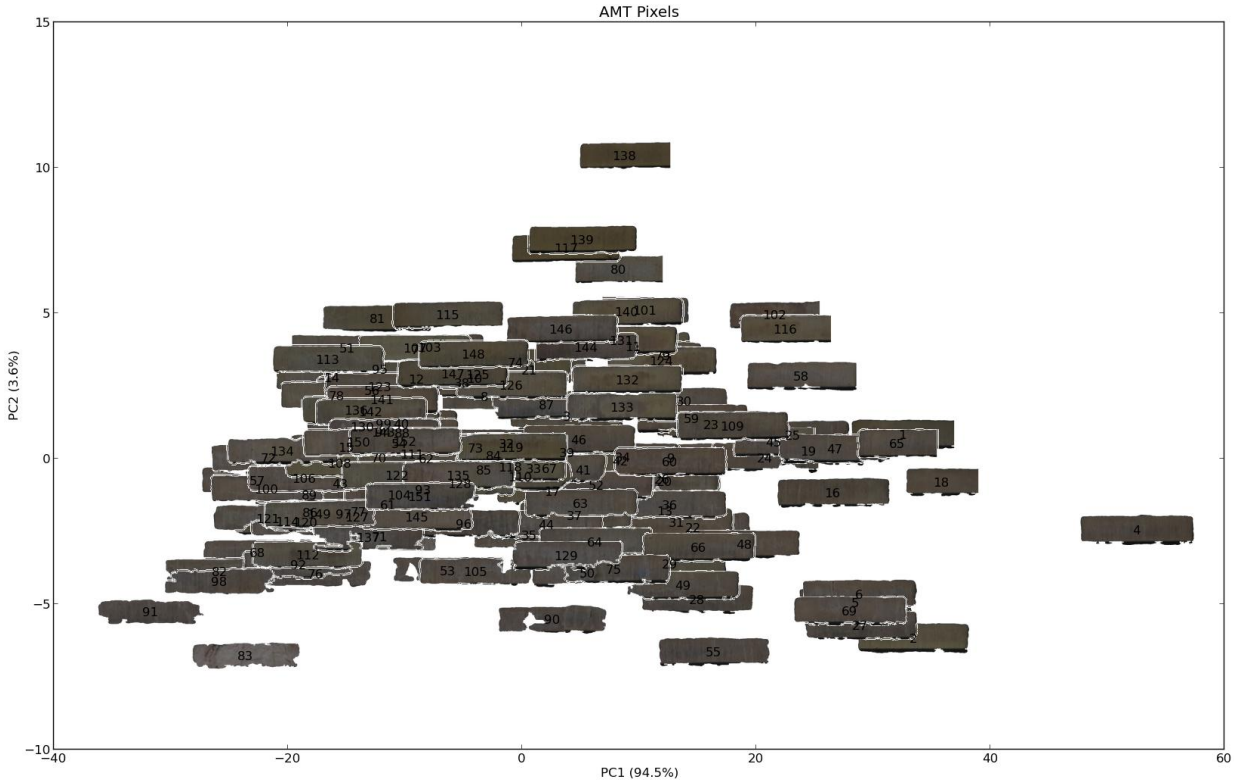


Figur 4.37: PCA-scorene på AMT-spekteret frå Stavangergata der tendensene ser ut til å vera overens med skada stein mot høgre.

For den andre veggen i Stavangergata kan ein sjå frå Figur 4.38 at det er verre å skilje murstein frå fugen ved eit undereksponert bilete. Det er fleire av mursteinane som har større hakk i steinen enn hovudbilete skulle tilseie. Då er det heller ikkje så rart at heile veggen vert klassifisert med karakteren D. Frå PCA-plotet i Figur 4.39 er det og tydleg at modellen får litt større problem med å separere ut frostskadde steinar. Steinar med størst skadeomfang ser ut til å trekkje seg nedover i plotet.



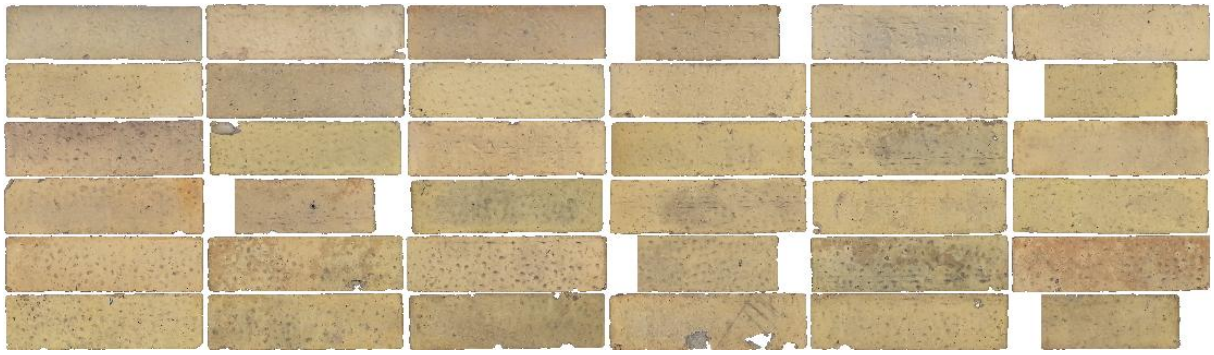
Figur 4.38: Montasje av mursteinen frå Figur 2.3b). Her kan ein sjå at det er fleire steinar det har vore vanskelig å skilje frå fugen.



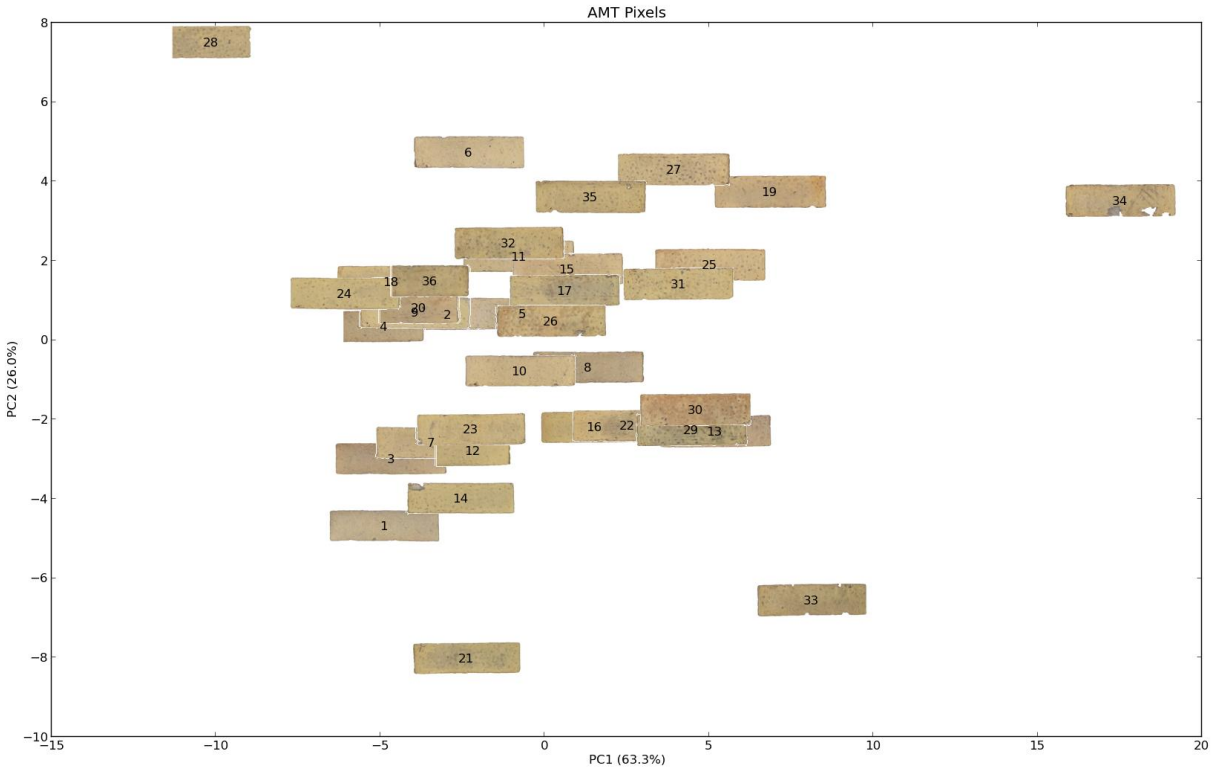
Figur 4.39: Sidan fleirtalet av steinane tilsynelatande har fått påvist frostskader så klarar ikkje PCA-plotet å skilje dei klart frå kvarandre. Ein kan sjå at dei største skadeomfanget heller nedover i plotet.

4.5.2 Resultata frå Refsnes Gods

Refsnes Gods er eit godt bilete som difor og gjev eit godt resultat til slutt. Veggen vart klassifisert til ein B-vegg. Mykje av grunnen er nok at det er mange friske steinar som trekker gjennomsnittsskaden ned. Frå Figur 4.40 kan ein sjå at programvaren har klart å skilje ut mursteinen ganske godt. I tillegg skil frostskadde murstein seg ut i PCA-plotet og trekkjer seg vekk frå ein frisk kjerne.



Figur 4.40: Montasje av de identifiserte mursteinana på bilete av Refsnes Gods

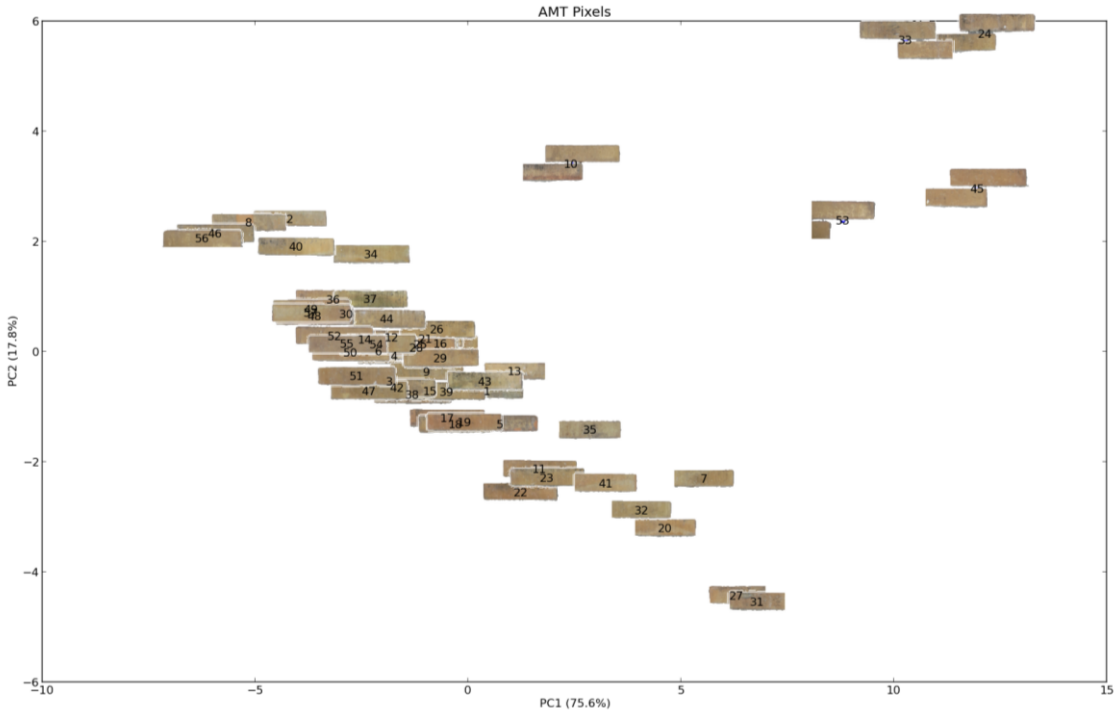


Figur 4.41: PCA-plotet for Refsnes Gods der frostskadde murstein legg seg i utkanten av kjernen.

4.5.3 Resultat frå KA-bygget, Campus Ås

Figur 4.42 visar at programvaren klarar å hente ut murstein i dei fleste samanhenger, men at det diverre av og til er vanskelig å skilje fugen frå mursteinen ved varierande mønster og farger på mursteinen. Grunna desse problema har klassifiseringa av veggen karakterisert den med karakteren F. Noko som må kunne seies å vera urettferdig med tanke på skadeomfanget. På varierande murstein er det tydelegvis vanskelig å skilje murstein frå veggen og dermed vert det og vanskelig å automatisk angi kva stein som har frostskader eller friske.

Figur 4.42: Det er tydleg fleire par med steinar som ikkje har klart å skilje seg frå kvarandre og det kan tyde på at programvaren ikkje likar stor variasjon på mursteinen.



Figur 4.43: PCA-plotet for KA-bygget der mursteinen som ikkje er skilt frå kvarandre bidrar til at det vert vanskelig å skilje ut frostskadene på ein god måte

5 Konklusjon

Frå resultata i denne oppgåva ser ein at det er mogleg å skilje ut murstein frå eit fargebilete. Men det er og tydleg at fotografiet må vera eksponert på ein god måte slik at fugen og mursteinen vert skilt frå kvarandre.

Forskjellen mellom dei ulike datasetta visar at bilete må tilpassast til programvaren ved fotografering. Det er tydeleg at resultatet vert best der fugen og mursteinen har klart definerte fargar som er forskjellig frå einannan. For at analysen skal verte god og samanliknbar krev programvaren gode bilete og at det er tydeleg forskjell mellom fuge og murstein.

Vidare ser det ut til at programvaren klarar å skilje ut frostskader enten med PCA eller med den eigenutvikla metoden for å sjå på signaturdifferansen. Men begge desse metodane heng tett saman med at det er viktig med eit godt bilete som grunnlag. Det viktigaste for å klassifisere murvegger er framleis eit godt utgangspunkt som her vil vera eit godt bilete som utnyttar tilgjengeleg intervall i eksponeringa for den interessante delen av veggen.

5.1 Vidare arbeid

Det område som ser ut til å kunne gjere rutinen og brukaropplevinga betre er å sjå på meir regionsbasert terskling som omtalt i [40] eller andre måtar å skilje ut mursteinen frå veggen. Dette er særlig aktuelt der det er små variasjonar mellom dei ulike fugene og mursteinen.

Ein annan ting er at sjølve brukaropplevinga frå fotogalleriet kan rustas opp og undervegs med dette arbeidet vart det oppdaga fleire moment som auka grensesnittet. Til dømes kan ein nemne RoiReader for Python [41] og html5 backend for matplotlib[42] som saman kan gi dynamiske figurar rett i fotogalleriet.

Litt meir direkte kan det nemnast at val av fargekanalar til kmeans clustering kanskje treng litt andre kriterium for utveljing av fargekanalar enn den som vert nytta for terskling.

Det ser og ut til at xvfb krev at det bare er ein instans om gangen, elles så er det andre uoppdaga problem som av og til stoppar opp ferdigstilling av ein analysekøyring. Mogleg det hadde passa betre med ei køordning for å køyre nye analysar.

Det gjenstår litt for å løfte programvaren frå ein prototype og opp til å vera klar som produksjonsmiljø.

6 Kjelder

1. **Kyllingstad, S, et al.** *Climate, environment and frost damage of architectural heritage.* Guimaraes, Portugal : Taylor & Francis, 2010.

2. **Hotell Refsnes Gods AS.** Refsnes Gods Jeløy. [Internett] 2013. [Sitert: 4 mai 2013.] http://www.refsnesgods.no/no/.

3. **Lisø, Kim R., et al.** A frost decay exposure index for porous, mineral building materials. *Building and Environment.* 2007, 42.

4. **MycoTeam.** Ekte hussopp. [Internett] MycoTeam. [Sitert: 17 april 2013.] http://www.mycoteam.no/emner/skadetyper/rate/kopi_av_hussopp.

5. **Hoff, Lars.** Soppen som kom inn fra kulden. [Internett] Apollon, 26 februar 2007. [Sitert: 11 mai 2013.] http://www.apollon.uio.no/artikler/2007/hussopp.html.

6. **Burger, Wilhelm og Burge, Mark James.** *Digital image processing : an algorithmic introduction using Java.* New York : Springer, 2010. 978-1-84628-379-6.

7. **Wyszecki, Günther og Stiles, W.S.** *Color science : concepts and methods, quantitative data and formulae.* New York : John Wiley & Sons, 2000. ISBN:0-471-39918-3.

8. **Guoping, Qiu, Xia, Feng og Jianzhong, Fang.** Compressing histogram representations for automatic colour photo categorization. *Pattern Recognition.* 2004, 37.

9. **Solem, Jan Erik.** *Programming Computer Vision with Python.* Sebastopol, Calif. : O'Reilly, 2012.

10. **Andrle, Robert.** The angle measure technique: a new method for characterizing the complexity of geomorphic lines. *Mathematical Geology.* 1994.

11. **Esbensen, Kim H., Hjelmen, Kent H. og Kvaal, Knut.** The AMT approach in Chemometrics - First forays. *Journal of Chemometrics.* 1996, 10.

12. **Kvaal, Knut, et al.** Multivariate feature extraction from textural images of bread. *Chemometrics and Intelligent Laboratory Systems.* 42, 1998.

13. **Haralick, R.M., Shanmugam, K. og Dinstein, I.** Textural Features for Image Classification. *IEEE Transactions on Systems, Man, and Cybernetics.* SMC-3, 1973, 6, ss. 610–621.

14. **Fongaro, Lorenzo og Kvaal, Knut.** Surface texture characterization of an Italian pasta by means of univariate and multivariate feature extraction from their texture images. *Food Research International.* 51, 2013, 2.

15. **Beyer, H.M.** The GLCM Tutorial, versjon 2.10. [Internett] 21 februar 2007. [Sitert: 6 mai 2013.] http://www.fp.ucalgary.ca/mhallbey/tutorial.htm.

16. **Jain, Anil K. og Dubes, Richard C.** *Algorithms for Clustering Data.* Englewood Cliffs, N.J. : Prentice Hall, 1988. 0-13-022278-X.

17. **Esbensen, Kim.** *Multivariate data analysis - in practice : an introduction to multivariate data analysis and experimental design.* Oslo : Camo, 2001. 8299333032.

18. **Lohninger, H.** Teach/Me Data Analysis. [Internett] Springer-Verlag, 1999. [Sitert: 13 mai 2013.] http://www.vias.org/tmdatanaleng/dd_nipals_algo.html.

19. **MartõÂnez, Aleix M. og Kak, Avinash C.** PCA versus LDA. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE.* 2001, 2.

20. **Venners, Bill.** The Making of Python, A Conversation with Guido van Rossum, Part I. [Internett] 13 januar 2003. [Sitert: 10 april 2013.] http://www.artima.com/intv/python.html .

21. The Jython Project. [Internett] 2013. [Sitert: 5 mai 2013.] http://jython.org/.

22. **Johannes Schindelin, Ignacio Arganda-Carreras, Erwin Frise, Verena Kaynig, Mark Longair, Tobias Pietzsch, Stephan Preibisch, Curtis Rueden, Stephan Saalfeld, Benjamin Schmid, Jean-Yves Tinevez, Daniel James White, Volker Hartenstein, Kevin Eliceiri, Pavel.** Fiji: an open-source platform for biological-image analysis. *Nature Methods.* 2012, 7, ss. 676-82.

23. **Piwigo.** [Internett] 2.5.0, 4 mars 2013. [Sitert: 5 mai 2013.] http://piwigo.org/.

24. **NumPy.** [Internett] 2013. [Sitert: 5 mai 2013.] http://www.numpy.org/.

25. **SciPy.** SciPy. [Internett] 2013. [Sitert: 5 mai 2013.] http://www.scipy.org/.

26. **Hunter, J. D.** Matplotlib: A 2D graphics environment. *Computing In Science & Engineering.* 2007.

27. **McKinney, Wes.** *Python for Data Analysis.* Sebastopol : O'Reilly Media, Inc., 2013. 9781449319793.

28. **pandas.** pandas: powerful Python data analysis toolkit. [Internett] 22 januar 2013. [Sitert: 12 april 2013.] http://pandas.pydata.org/pandas-docs/stable/.

29. **Tomic, Oliver (oliver.tomic@nofima.no).** Hoggorm: a chemometrics package in Python. [Internett] NOFIMA, 2013. vert lansert i 2013.

30. **ImageJ Plugins.** [Internett] 13 mars 2013. [Sitert: 4 mai 2013.] http://ij-plugins.sourceforge.net/plugins/segmentation/k-means.html.

31. **Wiggins, David P. og The Open Group, Inc.** XVFB. [Internett] 2013. [Sitert: 5 mai 2013.] http://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml.

32. **Fiji.** Headless - Fiji. [Internett] 30 oktober 2012. [Sitert: 10 april 2013.] http://fiji.sc/wiki/index.php/Headless.

33. **jsFiction.** jsDraw2DX. [Internett] 2012 November 2012. [Sitert: 10 April 2013.] http://jsdraw2dx.jsfiction.com/.

34. **Canonical Ltd.** Ubuntu. [Internett] Canonical Ltd., 2013. [Sitert: 2 mai 2013.] http://www.ubuntu.com/.

35. **VMware.** VMware Virtualization. [Internett] VMware, 2013. [Sitert: 02 mai 2013.] http://www.vmware.com/.

36. **The Apache Software Foundation.** Welcome to the Apache Software Foundation. [Internett] 2013. [Sitert: 2 mai 2013.] http://www.apache.org/.

37. **Oracle Corporation.** [Internett] Oracle Corporation. [Sitert: 2 mai 2013.] http://www.mysql.com/.

38. **Kvaal, Knut.** UMB ImageJ Plugins. [Internett] Norwegian University for Life Sciences UMB,, 25 februar 2013. [Sitert: 16 april 2013.] http://arken.umb.no/~kkvaal/eamtexplorer/imagej_plugins.html.

39. **Noregs Vassdrags- og Energidirektorat.** Energimerking.no: Karakterskalaen. [Internett] 13 april 2011. [Sitert: 10 mai 2013.] http://www.energimerking.no/no/Energimerking-Bygg/Om-energimerkesystemet-og-regelverket/Energimerkeskalaen/.

40. **Navon, Ety, Miller, Ofer og Averbuch, Amir.** Color image segmentation based on adaptive local thresholds. *Image and vision computing.* 2005, ss. 69-85.

41. **Coelho og Pedro, Luis.** [Internett] 2012. [Sitert: 13 mai 2013.] https://gist.github.com/luispedro/3437255.

42. **Ratcliffe, Simon og Schwardt, Ludwig.** [Internett] 2011. [Sitert: 13 mai 2013.] http://code.google.com/p/mplh5canvas/.

43. **ImageJ.** Analyze Menu. [Internett] [Sitert: 16 april 2013.] http://rsbweb.nih.gov/ij/docs/menus/analyze.html.

Vedlegg

A)

Variablar som kan lagrast i ResultsTable frå programvaren

B)

Start scriptet til Python *start_analyze.py*

C)

Hovudscriptet *brickwall_.py*

D)

Analysescriptet: *test_result2.py*

E)

Matlab scriptet: *script2.m*

F)

Framstilling over korleis ein kan nytte seg av brickwall_.py interaktivt

Vedlegg A)

Dei ulike parameterane ein får frå målingane av bilete

Namn	Forklaring
Area	Areal av murstein (antall piksler)
Mean	Gjennomsnittsintensiteten i gråtone
StdDev	Standardavviket til intensiteten(pikselverdien) i gråtonebilete
Mode	Typetallet til pikselverdien i intensiteten i gråtone bilete
Min	Minste pikselverdi i intensiteten i gråtone bilete
Max	Største pikselverdi i intensiteten i gråtone bilete
X	Senter av mursteinen, x-retning
Y	Senter av mursteinen, y-retning
XM	Massesenteret(basert på gråtoneintensiteten) til mursteinen i x-retning, relativt for bilete
YM	Massesenteret(basert på gråtoneintensiteten) til mursteinen i y-retning, relativt for bilete
Perim.	Omkretsen til mursteinen i piksler
BX	Starten av rektangel som omfavner heile mursteinen i x-retning (Hjørnepunkt (x,y))
BY	Starten av rektangel som omfavner heile mursteinen i y-retning (Hjørnepunkt (x,y))
Width	Bredden på mursteinen for å dekke ytterpunkta
Height	Høgden på mursteinen for å dekke ytterpunkta
Major	Lengda til hovudaksen eil ellipse-representasjon av mursteinen
Minor	Lengda til sekundæraksen til ein ellipse-representasjon av mursteinen
Angle	Vinkelen mellom ellipseretningen og x-aksen av bilete
Circ.	Sirkulæritet (4π*area/omkrets^2); 1.0 er

	perfekt sirkel
Feret	Lengste distansen mellom to punkter i mursteinen
Median	Medianen av pikselverdiene i intensiteten til gråtonebilete
FeretX	Startkoordinatene til Feret, x-retning
FeretY	Startkoordinatene til Feret, y-retning
FeretAngle	Vinkelen mellom Feret linja og x-aksen
MinFeret	minimum caliper diameter (?)
AR	Aspect ratio (major/minor)
Round	Rundskap (4*area/(π*major_axis^2))
Solidity	area/convex area (?)
Red_Mean	For raud fargekanal i stadenfor gråtonebilete RGB-fargerommet
Red_StdDev	
Red_Mode	
Red_Min	
Red_Max	
Red_Median	
Green_Mean	For grøn fargekanal i stadenfor gråtonebilete RGB-fargerommet
Green_StdDev	
Green_Mode	
Green_Min	
Green_Max	
Green_Median	
Blue_Mean	For blå fargekanal i staden for gråtonebilete RGB-fargerommet
Blue_StdDev	
Blue_Mode	
Blue_Min	
Blue_Max	
Blue_Median	
Hue_Mean	For Hue fargekanal i stadenfor gråtonebilete HSB-fargerom
Hue_StdDev	
Hue_Mode	
Hue_Min	
Hue_Max	
Hue_Median	
Sat_Mean	For Saturation fargekanal i stadenfor gråtonebilete HSB-fargerom
Sat_StdDev	
Sat_Mode	
Sat_Min	
Sat_Max	
Sat_Median	
Bright_Mean	For Brightness fargekanal

Bright_StdDev	i stadenfor gråtonebilete HSB-fargerom
Bright_Mode	
Bright_Min	
Bright_Max	
Bright_Median	
L_Mean	For L fargekanal i stadenfor gråtonebilete CIELab-fargerom
L_StdDev	
L_Mode	
L_Min	
L_Max	
L_Median	
a_Mean	For a fargekanal i stadenfor gråtonebilete CIELab-fargerom
a_StdDev	
a_Mode	
a_Min	
a_Max	
a_Median	
b_Mean	For b fargekanal i stadenfor gråtonebilete CIELab-fargerom
b_StdDev	
b_Mode	
b_Min	
b_Max	
b_Median	
Sign (0-360)	Distansen(i piksler) frå midten av objektet (X,Y) og ut til omkrinsen for kvar heile vinkel 0-359 grader. Traversert med klokka
SignR (0-360)	Distansen(i piksler) frå midten av objektet (X,Y) og ut til omkrinsen for kvar heile vinkel 0-359 grader. Traversert mot klokka
AMT+pixels(1-501)	AMT på pikslene til mursteinen, 500 målingar
ASM	ASM i GLCM måling
Contrast	Contrast i GLCM måling
Correlation	Korrelasjon i i GLCM måling
IDM	IDM i i GLCM måling
Entropy	Entropy i i GLCM måling
AMT+sign (1-101)	AMT på signal, 100 målingar

Kjelde: http://rsbweb.nih.gov/ij/docs/menus/analyze.html

Vedlegg B)

Start scriptet til Python start_analyze.py

```
#!/usr/local/bin/python
"""
Created on Sun Mar 31 20:49:03 2013

@author: Kristian S. Foerde

input:
    arg:
        [0] -> scriptfile
        [1] -> work and storage folder
        [2] -> path to image file
        [3] -> Ratio between original image and the new corner points
        [4] -> The points for new corners stored in pair of [(x,y),(x2,x2)..]
        [5] -> 1 if the analyze should use existing image work file

First it checks if it should calibrate or save a new work image.
Then it starts the brick wall analyze by running Fiji

Constants inside the script:
    IMAGEJ -> path to the Fiji application
    PY_FILE -> path to the Fiji Jython plugin
"""

import sys, subprocess, os


#Constants:
IMAGEJ = r'/home/inf300/Fiji.app/ImageJ-linux32'
PY_FILE = r'/home/inf300/Fiji.app/plugins/Master/brickwall_.py'

from scipy import ndimage
import homography
import numpy as np
import Image

#Warp perspective from Programming Computer Vision with Python
imname = sys.argv[2]

im = np.array(Image.open(imname))
h = im.shape[0]
w = im.shape[1]

ratio = float(sys.argv[3]) / w
x = eval(sys.argv[4])

if x is None or int(sys.argv[5]) == 1:
    warp = False
    im_all = im
else:
    fp = np.array([[p[1] / ratio, p[0] / ratio, 1] for p in x]).T
    tp = np.array([[0, 0, 1], [0, w, 1], [h, w, 1], [h, 0, 1]]).T

    warp = True
    for f, t in zip(fp[0:2, :], tp[0:2, :]):
        if int(f[0]) == int(t[0]) and int(f[1]) == int(t[1]) and \
           int(f[2]) == int(t[2]) and int(f[3]) == int(t[3]):
            continue
        else:
            break
    else:
        warp = False
        im_all = im
if warp:
    # estimate the homography

    H = homography.H_from_points(tp, fp)
```

```
    def warpfcn(x):
        """Helper function for geometric_transform based on
        the algorithm from Solem in
        Programming Computer Vision with Python"""
        x = np.array([x[0], x[1], 1])
        xt = np.dot(H, x)
        xt = xt / xt[2]
        return xt[0], xt[1]

    #warp each color channel with full perspective transform
    im_r = ndimage.geometric_transform(im[:, :, 0], warpfcn)
    im_g = ndimage.geometric_transform(im[:, :, 1], warpfcn)
    im_b = ndimage.geometric_transform(im[:, :, 2], warpfcn)
    im_all = im.copy()
    #Merge color channel back together
    im_all[:, :, 0] = im_r
    im_all[:, :, 1] = im_g
    im_all[:, :, 2] = im_b

if int(sys.argv[5]) == 0:
    Image.fromarray(im_all).save(sys.argv[1] + os.sep + '/work_image.tif')
    #For piwigo, since only Safari brwoser allow tiff
    Image.fromarray(im_all).save(sys.argv[1] + os.sep + '/work_image.jpg')

#Start Fiji and the analyze
command = IMAGEJ+' '+ PY_FILE +' '+ sys.argv[1] +' '+ \
          sys.argv[1]+os.sep+'work_image.tif'
command2 = 'xvfb-run '+command
print command2
p = subprocess.Popen(command2, shell=True, stderr=subprocess.PIPE, \
    stdout=subprocess.PIPE)
print 'SubPID:', p.pid
out = p.stdout.read().strip()
for line in out.split("\n"):
    try:
        print line
    except:
        pass
```

Vedlegg C)

Hovudscriptet brickwall_.py

```
# -*- coding: utf-8 -*-
"""
Brickwall plugin and shellrun

@author: Kristian S. Førde

Brickwall contains 2 classes for brick wall analyze:
    Brickwall for collection of data from brick wall image
    RunShell for simple communication with Python

    This module needs 3 attributes if it is run from terminal:
        arg:
            [0] -> scriptfile
            [1] -> path to work folder
            [2] -> path to image
"""

from ij import IJ
import ij.io
from ij.process import ImageStatistics as IS
from ij import ImagePlus
from ij.process import ImageConverter
import ij.process.ImageProcessor
from ij.measure import ResultsTable
from ij.measure import Measurements
from ij.plugin.frame import RoiManager
from ij.plugin.filter import ParticleAnalyzer
from ij.io import FileSaver
from ij.gui import Plot
import os
import subprocess
import sys
import java.awt.Font as Font
import java.lang.Class as Class
import math
from ij.gui import GenericDialog
import random as rnd
sys.path.append('/home/inf300/Dropbox/Master_programfiler/Python')
from ksf_funk import *
from jarray import array

#Import custom plugins
libDir = "/home/inf300/Fiji.app/plugins/"
sys.path.append(libDir)
sys.path.append(r"/home/inf300/Fiji.app/plugins/ij-plugins_toolkit.jar")
import UMB.AMT_spectra as amt
from net.sf.ij_plugins.clustering import KMeans

FOLDER_SCRIPT2 = r'/home/inf300/Dropbox/Master_programfiler/Python/test_results2.py'

class Brickwall(object):
    """Brickwall is a class for analyze a brick wall image to seperate
    the bricks from the wall and gather several informations from them"""

    def __init__(self, imp):
        """
        Constructor:
            input: an instance of ImageJ's ImagePlus

            output:None

        Starts the class and starts :meth:`collectHistograms`
        """
        self._imp = imp
        self._ip = imp.getProcessor()
        self._stack = ij.ImageStack(self._imp.width, self._imp.height)
        self._histo = [None] * 7 #For gray, RGB and HSB, not LAB
```

```
        self._small_options = (IS.MEAN + IS.MEDIAN + IS.MIN_MAX + IS.MODE
                               + IS.STD_DEV)
        self._options = (IS.AREA + IS.CENTER_OF_MASS + IS.CENTROID
                         + IS.CIRCULARITY + IS.ELLIPSE + IS.FERET + IS.LIMIT
                         + IS.MEAN + IS.MEDIAN + IS.MIN_MAX + IS.MODE
                         + IS.PERIMETER + IS.RECT + IS.SCIENTIFIC_NOTATION
                         + IS.STD_DEV)

        self.collectHistograms()

    def collectHistograms(self):
        """Collects the the histogram for the main image in the class.
        The collected histograms is graytone, each channel in RGB and each
        channel in HSB colorspaces. Additionally it stores each channel in
        its own stack.

        The histograms for CIELab is commented out, due to trouble to
        collect the histograms during headless ImageJ through terminal

        input: None

        output: None
        """
        #Gray
        self._histo[0] = self._ip.getHistogram()
        imp2 = self._imp.createImagePlus()
        ip2 = self._imp.getProcessor().duplicate()
        imp2.setProcessor("gray copy", ip2)
        img_conv = ImageConverter(imp2)
        img_conv.convertToGray8()
        self._stack.addSlice('Grayscale', imp2.getProcessor())
        imp2.close()

        #RGB
        imp_rgb = self._imp.duplicate()
        rgb_ic = ImageConverter(imp_rgb)
        rgb_ic.convertToRGBStack()
        rgb_stack = imp_rgb.getStack()
        stackr = ij.ImagePlus('Test RGB stack',rgb_stack)

        for ch, colorspace in enumerate(['Red','Green','Blue']):
            stackr.setSlice(ch+1)
            stats = stackr.getStatistics(self._small_options)
            self._histo[(ch+1)] = stats.histogram
            self._stack.addSlice(colorspace,stackr.getProcessor())

        stackr.close(), imp_rgb.close()

        # Histogram for HSB
        imp_hsb = self._imp.duplicate()
        HSBStack = ImageConverter(imp_hsb)
        HSBStack.convertToHSB()
        hsb_stack = imp_hsb.getStack()
        stackh = ij.ImagePlus('Test HSB stack',hsb_stack)
        for ch, colorspace in enumerate(['Hue','Saturation','Brightness']):
            stackh.setSlice(ch+1)
            stats = stackh.getStatistics(self._small_options)
            self._histo[(ch+4)] = stats.histogram
            self._stack.addSlice(colorspace,stackh.getProcessor())
        stackh.close(),
        imp_hsb.close()
        #makeMontage(self._stack)
#
#         #Histogram fra CIELAB
#
#         imp_lab = self._imp.duplicate()
```

```
#         IJ.run(imp_lab, "RGB to CIELAB", "");
#         imp_lab2 = IJ.getImage()
#         IJ.run(imp_lab2, "8-bit", "");
#         for ch, colorspace in enumerate(['L','a','b']):
#             imp_lab2.setSlice(ch+1)
#             stats = imp_lab2.getStatistics(self._small_options)
#             self._histo[(ch+7)] = stats.histogram
#             self._stack.addSlice(colorspace,imp_lab2.getProcessor())
##         imp_lab2.close()
##         imp_lab.close()

    def threshold(self):
        """This method thresholds all the channel which was collected
        in :meth:`collectHistograms`
        To get the thresholded channels, see :meth:`getThresholdStack`
        input: None

        output: None
        """
        self._threshold_stack = ij.ImageStack(self._imp.width, \
                                              self._imp.height)
        for idx, hist in enumerate(self._histo,start=1):
            ymax, xmax, ymin, xmin = bestTwoPeaks(stigningstal(cumsum(hist)))
            ath, flip = findThresholdValues(ymax, xmax, ymin, xmin, hist)
            label = self._stack.getSliceLabel(idx)
            ip_stack = self._stack.getProcessor(idx).duplicate()
            ip_stack.threshold(ath)
            if flip:
                ip_stack.invert()
            self._threshold_stack.addSlice(label,ip_stack)

    def getThresholdStack(self):
        """Method to get the stack of thresholded color channels.

        input: None

        output: ImageStack with thresholded images from :meth:`threshold`"""
        try:
            return self._threshold_stack
        except AttributeError:
            self.threshold()
            return self._threshold_stack

    def findBricks(self,kMeans = False):
        """Method for start the identification of brick from the image.
        From the attributes it choose between mask from:
                    best threshold (:meth:`maskWithThreshold`)
                    or kmeans (:meth:`maskKMeans`)

        input: (bool) kMeans; True if k-means clustering should be used

        output: None
        """
        if kMeans:
            maskeim = self.maskKMeans()
        else:
            maskeim = self.maskWithThreshold()

        self.fixMaskImage(maskeim)

    def maskKMeans(self):
        """Runs k-means clustering with for 2 regions based upon a stack
        form approved color channels

        input: None

        output: ImagePlus as an mask image
```

```
        """
        analyze_order = [1,2,3,4,6,7,5]#,8,9,10]
        kstack = ij.ImageStack(self._imp.width, self._imp.height)
        for idx in analyze_order:
            hist = self._histo[(idx-1)]
            m,i,mi,im = bestTwoPeaks(stigningstal(cumsum(hist)))
            if testForThreshold(m, i, mi, im, hist):
                 kstack.addSlice(str(idx), self._stack.getProcessor(idx).duplicate())
        imp_kstack = ImagePlus('KMeansStack', kstack)
        IJ.run(imp_kstack, "32-bit", "");

        kstack2 = imp_kstack.getStack()
        raw_mask = self.KMeans(kstack2)
#        IJ.run(raw_mask, "Make Binary", "");
        raw_mask.getProcessor().dilate()
        raw_mask.getProcessor().erode()
        IJ.run(raw_mask, "Remove Outliers...", "radius=10 threshold=1 which=Dark");

        times = 1
        for i in xrange(times):
            raw_mask.getProcessor().erode()
        for i in xrange(times):
            raw_mask.getProcessor().dilate()

        IJ.run(raw_mask, "Fill Holes", "")
        return raw_mask
#        self.fixThresholdedImage(raw_mask)

    def KMeans(self,stack):
        """Container method for running the k-means clustering with
        two regions.

        input: ImageStack for identification of regions

        output: ImagePlus binary mask image
        """
        config = KMeans.Config()
        config.setNumberOfClusters(2)
        kmean = KMeans(config)
        svar = kmean.run(stack)
        svar.setMinAndMax(0, config.getNumberOfClusters());
        svar.threshold(0)
        khist = svar.getHistogram()
        if khist[0] < khist[255]:
            svar.invert()
        return ImagePlus('Test KMeans',svar)

    def maskWithThreshold(self):
        """Create an mask image after an analyze after the best color
        channel to perform a threshold on. The color channel Hue is set last
        in the order of analyze since the routine stops at the first
        suitable color channel

        input: None

        output: ImagePlus as an mask image
        """
        analyze_order = [1,2,3,4,6,7,5]
        for idx in analyze_order:
            hist = self._histo[(idx-1)]
            m,i,mi,im = bestTwoPeaks(stigningstal(cumsum(hist)))
            if testForThreshold(m,i,mi,im,hist):
                imp_thre = ImagePlus("threshould_image",self._stack.getProcessor(idx)
                ymax, xmax, ymin, xmin = bestTwoPeaks(stigningstal(cumsum(hist)))
                ath, flip = findThresholdValues(ymax, xmax, ymin, xmin, hist)
#                print 'Tersklingspiksel',ath
                imp_thre.getProcessor().threshold(ath)
```

```
            if flip:
                imp_thre.getProcessor().invert()
            imp_thre.getProcessor().dilate()
            imp_thre.getProcessor().erode()
            IJ.run(imp_thre, "Remove Outliers...", "radius=10 threshold=1 which=D
            times = 6
            for i in xrange(times):
                imp_thre.getProcessor().erode()
            for i in xrange(times):
                imp_thre.getProcessor().dilate()

            IJ.run(imp_thre, "Fill Holes", "")
            return imp_thre
            break

    def fixMaskImage(self, imp):
        """Identifies regions, in this context bricks, from an mask image.
        Afterwards deletes unnormal regions which either to big or to small.
        At the end it collects several statistics on the identified blobs
        like size and color values.

        input: ImagePlus mask image

        output: None
        """
        self._mask = imp
        self._table = ResultsTable()
        self._roim = RoiManager(True)
        NROI = self._roim.count
        if NROI > 0:
            self._roim.runCommand('reset')
        # Create a ParticleAnalyzer, with arguments:
        pa = ParticleAnalyzer(ParticleAnalyzer.ADD_TO_MANAGER + \
                              ParticleAnalyzer.SHOW_NONE, self._options, \
                              self._table, 100, \
                              float(self._imp.width * self._imp.height), \
                              0.0, 1.0)
        pa.setHideOutputImage(True)
        pa.setRoiManager(self._roim)

        if pa.analyze(imp):
            pass
        else:
            raise RuntimeError("There was a problem in analyzing the mask image")

        NROI = self._roim.getCount()

        #Clean out wrong areas (Perhaps make this as an option)
        areas = []
        for idx, roi in enumerate(self._roim.getRoisAsArray()):
                areas.append(self._table.getValue('Area', idx))

        areas = standardize(areas)
        drop_list = []
        for idx, a in enumerate(areas):
            #Const from ksf_funk
            if BW_LOWER_STD_AREA > a or BW_UPPER_STD_AREA < a:
                drop_list.append(idx)

        #Remove wrong rois
        print drop_list
        for idx in reversed(drop_list):
                self._roim.select(imp, idx);
                self._roim.runCommand('Delete')
                self._table.deleteRow(idx)

        for i, roi in enumerate(self._roim.getRoisAsArray()):
```

```
            self._imp.setRoi(roi)
            stats = self._imp.getStatistics(self._small_options)
            self._table.setValue('Mean',i,stats.mean)
            self._table.setValue('StdDev',i,stats.stdDev)
            self._table.setValue('Mode',i,stats.mode)
            self._table.setValue('Min',i,stats.min)
            self._table.setValue('Max',i,stats.max)
            self._table.setValue('Median',i,stats.median)
            # Get values for colors in RGB and HGB as well
            for idx, txt in enumerate(['Red', 'Green', 'Blue', 'Hue', \
                                       'Sat', 'Bright'],start=2):
                self.fixStats(self._stack.getProcessor(idx), roi, txt, i)


    def fixStats(self, ip, roi,txt,row):
        """
        Helper function to calculate statistics from color channel and save
        the statistics in the internal ResultsTable

        input: ImageProcessor ip for color channel
               (Roi) roi
               (string) txt Label addon in ResultsTable
               (int) row

        output: None
        """
        imp_temp = ImagePlus("color_count", ip)
        imp_temp.setRoi(roi)
        stats = imp_temp.getStatistics(Measurements.MEAN + Measurements.MEDIAN + Meas
        self._table.setValue(txt+'_Mean',row,stats.mean)
        self._table.setValue(txt+'_StdDev',row,stats.stdDev)
        self._table.setValue(txt+'_Mode',row,stats.mode)
        self._table.setValue(txt+'_Min',row,stats.min)
        self._table.setValue(txt+'_Max',row,stats.max)
        self._table.setValue(txt+'_Median',row,stats.median)

    def getTable(self):
        """Method to give reference to the internal ResultsTable

        input: None

        output: ResultsTable
        """
        try:
            return self._table
        except AttributeError:
            self.findBricks()
            return self._table

    def getHistograms(self):
        """Method to give reference to the internal lists of histograms

        input: None

        output: list with histograms
                [0] -> gray
                [1] -> Red
                [2] -> Green
                [3] -> Blue
                [4] -> Hue
                [5] -> Saturation
                [6] -> Brightness
        """
        try:
            return self._histo
        except AttributeError:
            self.collectHistograms()
```

```
        return self._histo

    def getRois(self):
        """Gives an list of all the coordinates of the boundaries from
        the identified regions

        input: None

        output: list of coordinates for each blob"""
        roiList = []
        try:
            for roi in self._roim.getRoisAsArray():
                roi_coords = [roi.getFloatPolygon().xpoints, roi.getFloatPolygon().yp
                roiList.append(roi_coords)
            return roiList
        except AttributeError:
            self.findBricks()
            return self.getRois()

    def makeBlobStack(self, filename=None, bb=False):
        """
        Makes a stack of the blobs, creates an stack of binaery blobs as
        well for internal use

        input: (string) filename if the stack should be saved
               (bool) bb stands for use bounding box instead of roi,
                      default: False

        output: ImagePlus stack of blobs
        """
        try:
            NROI = self._roim.getCount()
        except AttributeError:
            self.findBricks()
            NROI = self._roim.getCount()

        max_width = 0
        max_height = 0
        for i in xrange(NROI):
            width = self._table.getValue('Width',i)
            height = self._table.getValue('Height',i)

            if width>max_width:
                max_width = width
            if height>max_height:
                max_height = height

        empty_imp = IJ.createImage("TempBlob", "RGB White", int(max_width), int(max_h
        self._blob_stack = ij.ImageStack(int(max_width), int(max_height));
        if not bb:
            self._mask_blob_stack = ij.ImageStack(int(max_width), int(max_height));

        for i, roi in enumerate(self._roim.getRoisAsArray()):
            if bb:
                self._imp.setRoi(int(self._table.getValue('BX',i)),
                                 int(self._table.getValue('BY',i)),
                                 int(self._table.getValue('Width',i)),
                                 int(self._table.getValue('Height',i)))
            else:
                self._imp.setRoi(roi)
            self._imp.copy(False)

            imp2 = empty_imp.duplicate()
            imp2.paste()
            imp2.trimProcessor()
            self._blob_stack.addSlice(str(i+1),imp2.getProcessor())
```

```
            #Same for mask_blob
            if not bb:
                self._mask.setRoi(roi)
                self._mask.copy(False)
                imp2 = empty_imp.duplicate()
                imp2.paste()
                imp2.trimProcessor()
                self._mask_blob_stack.addSlice(str(i+1),imp2.getProcessor())

        if filename is not None:
            FileSaver(ImagePlus('BlobStack',self._blob_stack)).saveAsTiffStack(filena

        return ImagePlus('BlobStack',self._blob_stack)

    def saveBlobImages(self,folder,idx='all'):
        """Save the blobs as seperated images in folder

        input: (string) folder path to where to save images
                (list) or (string) list of index values of which blobs to
                save, default to 'all'
        output: None
                But it saves selected blobs as images in folder"""
        try:
            NROI = self._roim.getCount()
        except AttributeError:
            self.findBricks()
            NROI = self._roim.getCount()

        for i, roi in zip(range(1,NROI+1), self._roim.getRoisAsArray()):
             if idx == 'all' or i in idx:
                self._imp.setRoi(roi)
                self._imp.copy(False)
                blob_ip = self._ip.createProcessor(int(self._table.getValue('Width',i
                blob_imp = ImagePlus('Blob-'+str(i),blob_ip)
                blob_imp.paste()
                blob_imp.trimProcessor()
                blob_filename = os.path.join(folder, str(i) + ".tif")
                FileSaver(blob_imp).saveAsTiff(blob_filename)
                blob_imp.close()

    def saveMask(self,filename):
        """Save the mask image

        input: (string) filename path and name to save the mask image

        output:None
        """
        try:
            FileSaver(self._mask).saveAsTiff(filename)
        except AttributeError:
            self.findBricks()
            FileSaver(self._mask).saveAsTiff(filename)

    def saveStack(self, filename):
        """Saves the stack to selected path. The stack contains the color
        channels

        input: (string) filename path and name to save the imagestack

        output: None"""
        if filename is not None:
            FileSaver(
                ImagePlus('Stack', self._stack)).saveAsTiffStack(filename)

    def saveThresholdStack(self, filename):
        """Saves the stack to selected path. The stack contains the
        thresholed images for each color channel
```

```
        input: (string) filename path and name to save the imagestack

        output: None"""
        try:
            self._threshold_stack
        except AttributeError:
            self.threshold()
        if filename is not None:
            FileSaver(ImagePlus('ThresholdStack',self._threshold_stack)).saveAsTiffSt

    def saveRoiset(self, filename):
        """Saves the Roiset to selected filepath

        input: (string) filename path and name to save the imagestack

        output: None"""
        if filename is not None:
            self._roim.runCommand("Save",filename);

    def makeRandom(self, inlist):
        """Wrapper function to shuffle the order of an list

        input: list

        output: list
        """
        return rnd.shuffle(inlist)

    def runAMT(self, dataIn='sign', resultsToTable=False,
               dynamic_box = False, mask_stack = False):
        """Wrapper method for creating AMT-spectre. Uses the plugin from
        UMB's Imaging group, AMT_spectra

        input: dataIn, type of signal
               resultsToTable, default False, set to True if values should
                               be stored in internal ResultsTable
               dynamic_box, default False, set to True to use the
                            bounding box for each blob instead of stack
               mask_stack, default False, Set to True to use binary stack
                            instead of gray values.

        dataIn: sign -> AMT on signature
                pixels -> AMT on pixel values (50% of the blob)
                histograms -> AMT on histograms
                [list] to give own values to run AMT on

        output: Stack of plots from AMT-specter for each blob
        """
        try:
            if mask_stack:
                stack = ImagePlus('BlobStack',self._mask_blob_stack).duplicate()
            else:
                stack = ImagePlus('BlobStack',self._blob_stack).duplicate()
        except AttributeError:
            self.makeBlobStack()
            if mask_stack:
                stack = ImagePlus('BlobStack',self._mask_blob_stack).duplicate()
            else:
                stack = ImagePlus('BlobStack',self._blob_stack).duplicate()

        datas = []
        if dataIn in ['sign','pixels']:
            for idx, roi in enumerate(self._roim.getRoisAsArray()):
                if dataIn == 'sign':
                    ux = self._signatures[idx]#1D vektoren me skal sjå på
                    smax = 100
```

```
    nsamp = 200
elif dataIn == 'pixels':
    sip = stack.getImageStack().getProcessor(idx+1)
    IJ.run(stack, "8-bit", "");
    smax = 500
    nsamp = 2000
    h = stack.getHeight()
    w = stack.getWidth()
    pix =[None]*h
    dw = dh = 0
    if dynamic_box:
        hb =  self._table.getValue('Height',idx)
        wb = self._table.getValue('Width',idx)
        dw = int(math.floor((w-wb)/2.0))
        dh = int(math.floor((h-hb)/2.0))
    for j in xrange(h):
        pix[j]=[None]*w
        for i in xrange(w):
            d = sip.getPixelValue(i, j);
            pix[j][i] = d # er luminance verdien i eit RGB, ergo grå

    ux = [None]*(h*w)

    #Spiral
    ready = True
    n = 0
    i = 0
    limit = h*w
    start = 0#(h+w)*2*4
    samples = int(math.ceil((limit)*0.14))
    if samples+start>limit:
        samples = limit+start
    ux = []

    while ready:
        for j in xrange(i+dw,w-i-dw): #top
            if n>start and n<=start+samples:
                ux.append(pix[i][j])
            n+=1

        for j in xrange(i+dh,h-i-dh): #right
            if n>start and n<=start+samples:
                ux.append(pix[j][(w-i-1-dw)])
            n+=1

        for j in xrange(w-i-1-dw,i+dw,-1): #bottom
            if n<limit:
                if n>start and n<=start+samples:
                    ux.append(pix[(h-i-1)][j])
                n+=1
            else:
                break

        for j in xrange(h-i-1-dh,i+dh,-1): #left
            if n<limit:
                if n>start and n<=start+samples:
                    ux.append(pix[j][i+dw])
                n+=1
            else:
                break

        if n>=start+samples:
            ready = False
        i+=1

datas.append(ux)
```

```
        elif dataIn in ['histograms']:
            datas = self.getHistograms()
            smax = 100
            nsamp = 200
        else:
            if type(dataIn)==type([None]):
                datas = dataIn
                dataIn = 'CustomList'
                smax = int(len(datas)*0.1)
                nsamp = int(len(datas)*0.9)
            else:
                raise Exception('Couldn\'t recognize "dataIn"')
        self._amt_stack = IJ.createImage("AMTStack - "+dataIn, "RGB White", 528, 255,
        for idx, uX in enumerate(datas):
            IJ.showProgress(float(idx)/len(datas))
            rand_order = range(len(uX))
            rnd.shuffle(rand_order)
            number = len(uX)-1
            pwcor = 1
            statType = 1
            linearMethod = False
            uX = array(uX, 'd')
            rand_order = array(rand_order, 'd')
            distrib = amt().jAMT(smax, nsamp, number, pwcor, statType, linearMethod,
            mda = amt().meanArray(distrib, smax+1, nsamp);
            plot = Plot("AMT", "index", "MA", array(range(len(mda)),'d'), mda).getIma
            plot.setRoi(0, 0, 528, 255)
            plot.copy(False)
            self._amt_stack.setPosition((idx+1))
            self._amt_stack.paste()
            plot.close()
            if resultsToTable:
                for i in xrange(smax):
                        self._table.setValue('AMT+'+dataIn+'('+str(i+1)+')',idx,mda[i

        IJ.showProgress(1)
        return self._amt_stack

    def runGLCM(self, resultsToTable=False):
        """Wrapper method to run the GLCM plugin from UMB's Imaging group

        input: resultsToTable, default False, set to True to store values
                             from GLCM to the internal ResultsTable

        output: None"""

        try:
            stack = ImagePlus('BlobStack',self._blob_stack).duplicate()
        except AttributeError:
            self.makeBlobStack()
            stack = ImagePlus('BlobStack',self._blob_stack).duplicate()

        IJ.run(stack, "8-bit", "");
        IJ.run(stack, "GLCM features", "step=1 direction=[0 degrees]");

        if resultsToTable:
            GLCM_results = ResultsTable.getResultsTable()
            i = 0
            while GLCM_results.getColumnHeading(i) is not None:
                j = 0
                for c in GLCM_results.getColumn(i):
                    self._table.setValue(GLCM_results.getColumnHeading(i),j,c)
                    j+=1
                i+=1
        win = GLCM_results.getResultsWindow();
        if win!=None: win.close(False)
```

```
def runSignature(self, resultsToTable=False, reverse = False):
    """Calulates the signature for each blob

    input: resultsToTable, default False, set to True to store signatur
                        for each angle in the internal ResultsTable
           reverse, default False, set to True to calculate the distance
                        in counter-clockwise direction

    output: ImageStack, profile of signatur for each blob
                        """
    self._signatures = []
    angle = 0
    label = 'Sign'
    if reverse:
        label+='R'
    empty_imp = IJ.createImage("TempSignalPlot", "RGB White", 528, 255, 1);
    self._sign_stack = ij.ImageStack(528, 255);
    for idx,roi in enumerate(self._roim.getRoisAsArray()):
        ox = self._table.getValue('X',idx)
        oy = self._table.getValue('Y',idx)
        xt = None
        yt = None
        distance = [-1]*360
        roi_coords = zip(roi.getFloatPolygon().xpoints, roi.getFloatPolygon().ypo
        if reverse:
            roi_coords = reversed(roi_coords)
        for x,y in roi_coords:
            if xt is None:
                xt = x
                yt = y

            if abs(xt-x) > 0:
                for i in range(xt, x, signMultiplier(x-xt)):
                    angle =  math.degrees(math.atan2(ox-i, oy-y))
                    if angle < 0:
                        angle += 360

                    distance[int(angle)]=math.sqrt(((ox-i)**2)+((oy-y)**2))

            if abs(yt-y) > 0:
                for j in range(yt, y, signMultiplier(y-yt)):
                    angle = math.degrees(math.atan2(ox-x, oy-j))
                    if angle < 0:
                        angle += 360
                    distance[int(angle)]=math.sqrt(((ox-x)**2)+((oy-j)**2))

            xt = x
            yt = y
            angle =  math.degrees(math.atan2(ox-x, oy-y))
            if angle < 0:
                angle += 360
            distance[int(angle)] = math.sqrt(((ox-x)**2)+((oy-y)**2))

        self._signatures.append(distance)
        if resultsToTable:
            for sig_idx, dis in enumerate(distance):
                self._table.setValue(label+'('+str(sig_idx)+')',idx,dis)
        plot = Plot(label+"-"+str(idx+1), "angle", "distance", array(range(len(di
        plot.setRoi(0, 0, 528, 255)
        plot.copy(False)
        imp2 = empty_imp.duplicate()
        imp2.paste()
        imp2.trimProcessor()
        self._sign_stack.addSlice(str(i+1),imp2.getProcessor())
        self._sign_stack.setSliceLabel(label+'-'+str(idx+1),idx+1)
        plot.close()
```

```
        return ImagePlus(label+'Stack',self._sign_stack)

    def closeRoi(self):
        """Wrapper to close the internal RoiManager"""
        self._roim.close()

class RunShell(object):
    """Wrapper class to communicate with Python through the terminal"""
    def __init__(self, brickwall):
        """Input: instance of Brickwall"""
        self._im = brickwall

    def pca(self,pc1,pc2,folder):
        """Run PCA and other analysis on the saved roidata_k.csv
        Makes the command ready and start the command
        """
        self._script_list = ['python',r'/home/inf300/Dropbox/Master_programfiler/Pyth
                             str(folder),str(pc1),str(pc2)]
        print self._run()

    def _run(self):
        """Internal method to send command too the terminal"""
        process = subprocess.Popen(
                        self._script_list, stdout=subprocess.PIPE, stderr=subproces
                    )

        whole_out = ''
        while True:
            out = process.stdout.read(1)
            if out == '' and process.poll() != None:
                break
            if out != '':
                whole_out+=out
                sys.stdout.write(out)
                sys.stdout.flush()
        return whole_out


def getOptions():
    """Get options like folder to storage the files
    Accept either arg from terminal or GenericDialog

    output: path to work folder
    """
    if len(sys.argv) < 2:
        gd = GenericDialog("Options")
        gd.addStringField('savefolder','/home/inf300/www/Brickwall/analyze/test3')
        gd.showDialog()

        if gd.wasCanceled():
            print "User canceled dialog!"
        savefolder = gd.getNextString()
    else:
        print sys.argv
        savefolder = sys.argv[1]

    return savefolder

def run():
    """Run a predefined routine to identify bricks from image and collect
    data on the blobs by using the class Brickwall

    Uses current image if there are no arg
    prints 'Finito' at the end

    output: None
    """
```

```
    options = getOptions()
    if options is not None:
        FOLDER = options
        try:
            if len(sys.argv) > 1:
                imp = IJ.openImage(sys.argv[2])
            else:
                imp = IJ.getImage()
            im = Brickwall(imp)
            im.findBricks(True)
            im.saveMask(FOLDER+os.sep+'mask.tif')
            im.makeBlobStack(FOLDER+os.sep+'BlobStacks.tif')
            im.saveStack(FOLDER+os.sep+'Stack.tif')
            im.saveThresholdStack(FOLDER+os.sep+'ThresholdStack.tif')
            im.saveRoiset(FOLDER+os.sep+'RoiSet.zip')
            im.runSignature(True)
            im.runSignature(True, True)
            im.runAMT('pixels', True, True)
            im.runGLCM(True)
            im.runAMT('sign', True)
            im.getTable().saveAs(FOLDER+os.sep+'roidata_k.csv')

            run = RunShell(im)
            run.pca(1, 2, FOLDER)
        except ZeroDivisionError:
            imp.close()
            return

    print 'Finito'

if __name__ == '__main__':
    run()
```

Vedlegg D)

Analysescriptet: test_result2.py

```
# -*- coding: utf-8 -*-
"""
Runs PCA and creates figures of the collect data from Fiji

Arg:
    [0] -> script file
    [1] -> work folder to both save files and get files
    [2] -> First PC to get
    [3] -> Second PC to get
"""
import scipy.io
import sys
import os
import numpy as np
import matplotlib
matplotlib.use('Agg')
import pandas as pd

import pca
import Image



def array(typearray, liste):
    return liste

def normalize(signal):
    signatur = signal.copy()
    n = len(signal[:,0])
    #Normalize
    for i in xrange(n):
        signatur[i,:]=signal[i,:]/np.max(signal[i,:])

    means = np.mean(signal,axis=0)/np.max(np.mean(signal,axis=0),axis=0)
    sv = np.zeros((n,360))
    for i in xrange(n):
        sv[i,:]=signatur[i,:]-means

    return signatur, sv

def justNormalize(sign):
    signatur = sign.copy()
    n = len(sign[:,0])
    #Normalize
    for i in xrange(n):
        signatur[i,:]=sign[i,:]/np.max(sign[i,:])

    return signatur


#http://matplotlib.org/examples/pylab_examples/demo_annotation_box.html Denne som gje
from matplotlib.offsetbox import AnnotationBbox, OffsetImage, TextArea

folder = sys.argv[1] #

# Load data
sensory = pd.read_csv(folder+os.sep+'roidata_k.csv', index_col=0)
samples = sensory.index.tolist()
attributes = sensory.columns.tolist()
data = np.asarray(sensory.values)

datac = data[:,0]
data_check = (datac-np.mean(datac))/np.std(datac)
plt = pca.plt
plt.ion()
plt.figure()
plt.plot(data_check,'*g',markersize=10)
```

```
md = [np.mean(data_check)]*len(data[:,0])
med = [np.median(data_check)]*len(data[:,0])
ma10 = [np.mean(data_check)*4.0]*len(data[:,0])
per90 = [1.5]*len(data[:,0])
per10 = [-2.0]*len(data[:,0])
plt.plot(md,'r-',med,'b-',ma10,'k-',per90,'y-',per10,'m-')
plt.xlabel('ObjektID')
plt.ylabel('Areal')

plt.grid(True)
plt.savefig(folder+os.sep+'area_distrubution.png')
values_out_range = []
for idx,da in enumerate(data_check,start=1):
    if da > per90[0] or da < per10[0]:
        values_out_range.append(idx)

print values_out_range
mat = {}

shape_attr = ['Area','Perim.','Width','Height','Major',
                               'Minor','Circ.','AR','Round']
color_attr = ['Mean','StdDev','Mode','Min','Max','Median',
                               'Red_Mean','Red_StdDev','Red_Mode','Red_Min','Red_Max',
                               'Green_Mean','Green_StdDev','Green_Mode','Green_Min','G
                               'Blue_Mean','Blue_StdDev','Blue_Mode','Blue_Min','Blue_I
                               'Hue_Mean','Hue_StdDev','Hue_Mode','Hue_Min','Hue_Max',
                               'Sat_Mean','Sat_StdDev','Sat_Mode','Sat_Min','Sat_Max',
                               'Bright_Mean','Bright_StdDev','Bright_Mode','Bright_Min
amt_sign_attr = [x for x in attributes if x[0:9]=='AMT+sign(']
amt_pixels_attr = [x for x in attributes if x[0:11]=='AMT+pixels(']
glcm_attr = ['ASM','Contrast','Correlation','IDM','Entropy']
sign_attr = [x for x in attributes if x[0:5]=='Sign(']
signR_attr = [x for x in attributes if x[0:6]=='SignR(']

attr = shape_attr+glcm_attr+sign_attr

mat['shape']= np.asarray(sensory[shape_attr].values)
mat['color']= np.asarray(sensory[color_attr].values)
mat['sign']= np.asarray(sensory[sign_attr].values)
mat['signr']= np.asarray(sensory[signR_attr].values)
mat['glcm']= np.asarray(sensory[glcm_attr].values)
mat['amtpix']= np.asarray(sensory[amt_pixels_attr].values)
mat['amtsign']= np.asarray(sensory[amt_sign_attr].values)
mat['raw']= np.asarray(sensory.values)


if min(data.shape)>5:
    #Run PCA
    cvt = None#['loo']

    pca_data = np.asarray(sensory[attr].values)
    model = pca.nipalsPCA(pca_data, mode=True, cvType=cvt)

    print 'Ferdig med PCA1'
    pca_data = np.asarray(sensory[shape_attr].values)
    model_shape = pca.nipalsPCA(pca_data, mode=True, cvType=cvt)#['loo'])

    print 'Ferdig med PCA2'
    pca_data = np.asarray(sensory[glcm_attr].values)
    model_glcm = pca.nipalsPCA(pca_data, mode=True, cvType=cvt)#['loo'])

    print 'Ferdig med PCA3'
    pca_data = np.asarray(sensory[sign_attr].values)
    model_signal = pca.nipalsPCA(pca_data, mode=True, cvType=cvt)#['loo'])

    print 'Ferdig med PCA4'
    pca_data = np.asarray(sensory[color_attr].values)
```

```
    model_color = pca.nipalsPCA(pca_data, mode=True, cvType=cvt)#['loo'])

    print 'Ferdig med PCA5'
    pca_data = np.asarray(sensory[amt_sign_attr].values)
    model_amt1 = pca.nipalsPCA(pca_data, mode=False, cvType=cvt)#['loo'])

    print 'Ferdig med PCA6'
    pca_data = np.asarray(sensory[amt_pixels_attr].values)
    model_amt2 = pca.nipalsPCA(pca_data, mode=False, cvType=cvt)#['loo'])#['loo'])

    print 'Ferdig med PCA7'
#
    signatur, avvik = normalize(sensory[sign_attr].values)
    model_sign = pca.nipalsPCA(signatur, mode=True, cvType=cvt)#['loo'])

    print 'Ferdig med PCA8'
    model_avvik = pca.nipalsPCA(avvik, mode=False, cvType=cvt)#['loo'])

    print 'Ferdig med PCA9'
    pca_diff = np.abs(sensory[sign_attr].values-sensory[signR_attr].values)
    model_diff = pca.nipalsPCA(pca_diff, mode=False, cvType=cvt)#['loo'])

    print 'Ferdig med PCA10'


def showBrickGraph(model,pc1, pc2,blobstack_file,filename,title=None):
    """Creates a score plot for PCA where the scores is showed as bricks and
    numbers"""
    x = model.scores()[:,pc1]
    y = model.scores()[:,pc2]
    imageSize=(150,150)
    dpi = 72
    fig = plt.figure(dpi=dpi,figsize=(20,12))
    fig.clf()
    ax = fig.add_subplot(111)

    ax.plot(x,y,'bo')

    im1 = Image.open(blobstack_file)
    n = len(x)

    for idx in xrange(n):
        im1.seek(idx)
        im1.thumbnail(imageSize, Image.ANTIALIAS)
        #remove white background
        img = im1.convert("RGBA")

        pixdata = img.load()

        for i in xrange(img.size[1]):
            for j in xrange(img.size[0]):
                if pixdata[j, i] == (255, 255, 255, 255):
                    pixdata[j, i] = (255, 255, 255, 0)

        imagebox = OffsetImage(img, zoom=1.0)
        xy = [x[idx],y[idx]]

        bbox_props = dict(boxstyle=None, fc=None, ec=None,alpha=0.0)

        ab = AnnotationBbox(imagebox, xy,
                            xycoords='data',
                            boxcoords="data",
                            pad=0.0,
                            bboxprops=bbox_props

                            )
```

```
        ax.add_artist(ab)

    for idx in xrange(n):
        textbox = TextArea(str(idx+1), minimumdescent=False)
        xy = [x[idx],y[idx]]

        bbox_props = dict(boxstyle=None, fc=None, ec=None,alpha=0.0)

        ab = AnnotationBbox(textbox, xy,
                            xycoords='data',
                            boxcoords="data",
                            pad=0.0,
                            bboxprops=bbox_props

                            )
        ax.add_artist(ab)

    if title is not None:
        plt.title(title)

    XexplVar = model.calExplVar()
    ax.set_xlabel('PC{0} ({1}%)'.format((pc1+1), \
                    str(round(XexplVar[pc1],1))))
    ax.set_ylabel('PC{0} ({1}%)'.format((pc2+1), \
                    str(round(XexplVar[pc2],1))))
    plt.savefig(folder+os.sep+filename)



pc1 = int(sys.argv[2])-1
pc2 = int(sys.argv[3])-1

showBrickGraph(model,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_all_shape_attr_bric
showBrickGraph(model_shape,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_shape_attr_br
showBrickGraph(model_glcm,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_glcm_attr_bric
showBrickGraph(model_signal,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_signal_attr_
showBrickGraph(model_color,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_color_attr_br
showBrickGraph(model_amt1,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_amt_signals_at
showBrickGraph(model_amt2,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_amt_pixels_att
showBrickGraph(model_sign,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_signature_bric
showBrickGraph(model_avvik,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_avvik_bricks.
showBrickGraph(model_diff,pc1,pc2,folder+os.sep+'BlobStacks.tif','show_diff_bricks.pn

print 'Laga PCA plot'

sign_norm = justNormalize(mat['sign'])
signr_norm = justNormalize(mat['signr'])
sign_diff = np.abs(sign_norm-signr_norm)
sign_test = np.sum(sign_diff,axis=1)

y=sign_test
x=range(len(y))
fig = plt.figure()
ax = fig.add_subplot(111)
im1 = Image.open(folder+os.sep+'BlobStacks.tif')
n = len(x)
imageSize=(150,150)
ax.plot(x,y,'bo')
for idx in xrange(n):
    im1 = Image.open(folder+os.sep+'BlobStacks.tif')
    im1.seek(idx)
    im1.thumbnail(imageSize, Image.ANTIALIAS)
    #remove white background
    img = im1.convert("RGBA")

    pixdata = img.load()
```

```
    for i in xrange(img.size[1]):
        for j in xrange(img.size[0]):
            if pixdata[j, i] == (255, 255, 255, 255):
                pixdata[j, i] = (255, 255, 255, 0)

    imagebox = OffsetImage(img, zoom=1.0)
    xy = [x[idx],y[idx]]

    bbox_props = dict(boxstyle=None, fc=None, ec=None,alpha=0.0)
    ab = AnnotationBbox(imagebox, xy,
                        xycoords='data',
                        boxcoords="data",
                        pad=0.0,
                        bboxprops=bbox_props
                        )


    ax.add_artist(ab)
plt.xlabel('ObjektID')
plt.ylabel('Differansesum')
plt.title('Skada stein')
plt.savefig(folder+os.sep+'skada_stein.png')

mat['signnorm']=sign_norm
mat['signdiff']=sign_diff

scipy.io.savemat(folder+os.sep+'matlab.mat', mat)


#Classify damage on bricks
def score(value):
    if value < 2.5:
        return 'A'
    elif value < 3.5:
        return 'B'
    elif value <  4.5:
        return 'C'
    elif value <  5.5:
        return 'D'
    elif value <  6.5:
        return 'E'
    elif value <  7.5:
        return 'F'
    else:
        return 'G'

f = open(folder+os.sep+'grade.txt','w')
f.write(score(np.mean(sign_test)))
```

Vedlegg E)

Matlab scriptet: script2.m

```
clear all
load('work7/matlab') %Datasett 7
load('work7/matlabY') %Klassifiseringa til datasett 7

n = length(sign(:,1));
sign_norm = sign;
for i=1:n
    svar = sign(i,:)/max(sign(i,:));
    sign_norm(i,:)=svar;
end

n = length(signr(:,1));
signr_norm = signr;
for i=1:n
    svar = signr(i,:)/max(signr(i,:));
    signr_norm(i,:)=svar;
end

sign_diff = sign_norm-signr_norm;
sign_diff_abs = abs(sign_diff);

y = Y*(1:3)';

X = sign_diff;
y = Y*(1:3)';
ncomp = 9;
[beta, P, T, R, mX, mY] = simpls(ncomp, X, Y);
k = ncomp;
b = [beta(:,k,1) beta(:,k,2) beta(:,k,3)];

Yhat = X*b;

[muG, CovP, ng] = LDAparameters(Yhat, y);
% priors = [1 1 1]/3;
priors = sum(Y)/size(Y,1);
% priors = [0.4 0.2 0.4];


load('matlab') % datasettet som skal klassifiserast
n = length(sign(:,1));
sign_norm = sign;
for i=1:n
    svar = sign(i,:)/max(sign(i,:));
    sign_norm(i,:)=svar;
end

n = length(signr(:,1));
signr_norm = signr;
for i=1:n
    svar = signr(i,:)/max(signr(i,:));
    signr_norm(i,:)=svar;
end

sign_diff = sign_norm-signr_norm;
sign_diff_abs = abs(sign_diff);
```

```
X = sign_diff;


Yhat = X*b;
[yhat, pclass] = LDAclassify(Yhat, muG, CovP, priors);
y = Y*(1:3)';
load ('fasit11')

[confmat2, pcc, ncc] = forvirring(yfasit,yhat,1);
confmat2
ncc
pcc
```

Vedlegg F)

Framstilling over korleis ein kan nytte seg av brickwall_.py interaktivt

brickwall_.py må ligge i ei mappe under plugins i Fiji.

Start Jython Interpreter frå Plugins->Scripting->Jython Interpreter

For å hente inn brickwall_.py skriver ein inn:

import brickwall_ as b

Då skal ein ha tilgang til klasser og funksjonar frå *b*. (Sjå Vedlegg B for dokumentasjon og koden til brickwall_.py)

For å hente opp bilete som er opna i Fiji, skriv:

imp = IJ.getImage()

for å sende denne til Brickwall klassen skriver ein:

im = b.Brickwall(imp)

Då kan ein nytte alle metodane som ligg i klassen.

Til dømes kan ein skrive ut histogramverdiene

im.getHistograms()