Løsningsforslag

UNIVERSITETET I OSLO

Det matematisk-naturvitenskapelige fakultet

Eksamen i:	INF2310 — Digital bildebehandling
Eksamensdag:	Onsdag 1. juni 2015
Tid for eksamen:	14:30 – 18:30
Løsningsforslaget er på:	**13 sider**
Vedlegg:	**Ingen**
Tillatte hjelpemidler:	**Ingen**

- Det er 6 oppgaver i dette oppgavesettet.

- Les gjennom hele oppgavesettet før du begynner å løse oppgavene !
Kontroller at oppgavesettet er komplett før du begynner å besvare det.
Dersom du savner opplysninger i en oppgave, kan du selv legge dine egne forutsetninger til grunn og gjøre rimelige antagelser,
så lenge de ikke bryter med oppgavens "ånd".
Gjør i såfall rede for forutsetningene og antagelsene du gjør.

- Det er tilsammen 18 deloppgaver. **Hver deloppgave teller like mye.** Det lønner seg derfor å disponere tiden slik at man får besvart alle deloppgavene. Hvis du står fast på en deloppgave, gå videre slik at du får gitt et kort svar på alle deloppgavene.

- **Alle svar skal begrunnes.** Gjør rede for bruken av eventuelle teoremer, prinsipper eller forutsetninger slik at en tredjeperson kan følge dine resonnementer.

1. Tema: Kompresjon og koding

Gitt følgende sekvens av 14 symboler: ABCDEAADEEDADA

a) Finn Huffmann-kodeboken for symbolsekvensen.
Hvis vi ser bort fra kodeboken, hvor mange biter vil vi bruke på å kode symbolsekvensen med Huffmann-koding?
Hva er gjennomsnittlig biter per symbol?

Løsningsforslag: NB! Her er det flere muligheter, men om man følger algoritmen gitt i foilene får man følgende kodebok:

Symbol	**Kode**	**Sannsylighet**	**Antall**
A	0	5/14=0.36	5
B	1110	1/14=0.07	1
C	1111	1/14=0.07	1
D	10	4/14=0.29	4
E	110	3/14=0.21	3

Uten å lagre kodebok ville vi brukt 30 biter for symbolsekvensen.

b) UTF8 koding bruker 8 biter per symbol.
Hvilke kompresjonsrate har vi oppnådd sammenlignet med om vi hadde brukt UTF8 koding til å kode symbolsekvensen?

Entropien for sekvensen er ≈ 2.
Hva er kodingsredudansen vi får ved å bruke Huffman-koding?
Løsningsforslag:
UTF8-koding av symbolsekvensen ville brukt 14*8=112 biter.
Kompresjonsraten vi har oppnådd er da 112/30 = 56/15 = 3.73.

Siden entropien er ≈2 og gjennomsnittlig antall biter per symbol (fra a) er 2.14 er kodingsredudansen vår 2.14 – 2 = 0.14.

c) Kan du foreslå en annen kodingsteknikk som generelt sett gir en enda høyere kompresjonsrate?
Hvordan skiller denne teknikken seg fra Huffman-koding?
Nevn fordeler og ulemper med den alternative teknikken sammenlignet med Huffman-koding.
Løsningsforslag:
Aritmetisk koding kunne gitt høyere kompresjonsrate. Aritmetisk koding skiller seg fra Huffman-koding ved at hele sekvensen kodes som ett tall. Det lages altså ikke kodeord for enkeltsymboler. Implementasjonene av aritmetisk koding er relativ regnetunge og krever høy regnepresisjon for lange symbolsekvenser. En annen ulempe ved aritmetisk koding er at vi ikke får noen delresultat av kompresjonen før hele sekvensen er behandlet. Det kan også ses på som en ulempe at de fleste aritmetisk koding-implementasjonene er patenterte.

2. Konvolusjon

a) Du får oppgitt en kombinert konvolusjonsoperator O som bygger på to vektorer A og B:

$$O = -\frac{1}{16}\left(A * A * B * B^T * B^T * B^T \; + \; A^T * A^T * B * B * B * B^T\right)$$
$$A = \begin{bmatrix}1 & 0 & -1\end{bmatrix}, \quad B = \begin{bmatrix}1 & 2 & 1\end{bmatrix}$$

Hvilken operator er O, og hva bruker vi den til?
Bruk egenskaper ved konvolusjon til å begrunne svaret uten å utføre konvolusjonen!

Løsningsforslag:
Operatoren O kan omformes til (ligningen er en del av løsningsforslaget)

$$O = -\frac{1}{16}\left((A * B^T) * (A * B^T) + (A^T * B) * (A^T * B)\right) * \left(B * B^T\right)$$

($A*B^T$) og (A^T*B) er de to konvolusjons-filtrene i Sobel-operatoren som vi bruker til å estimerer gradienten (førstederiverte) i hhv x- og y-retning.
*($A*B^T$) * ($A*B^T$) og (A^T*B)* (A^T*B) gir oss da estimater av den andre-deriverte i hhv x- og y-retning. Så langt svarer operatoren til en Laplace-operator realisert ved hjelp av Sobel-operatoren.*
Dette konvolveres så med 1/16($B*B^T$), som er et lavpass Gauss-filter.
Altså er operatoren et LoG-filter, "Laplacian of Gaussian".
Brukes til (2. derivert) kantdeteksjon med innebygget støyreduksjon.

b) Hvor stor er filtermatrisen O ovenfor?
Begrunn svaret uten å utføre konvolusjonen!

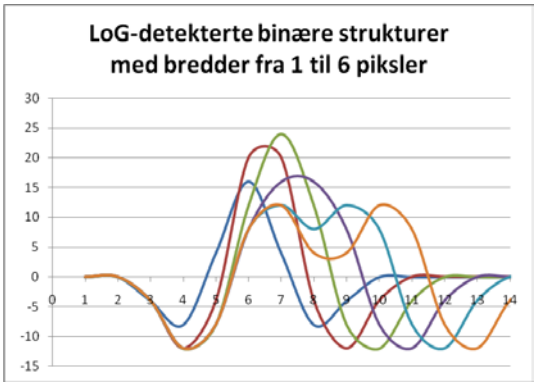
Løsningsforslag:
Operatoren O = -($A*A*B^T*B^T$ + A^T*A^T*B*B)($B*B^T$) kan skrives som*
O = -(($A*A*B$)($B^T*B^T*B^T$) + ($A^T*A^T*B^T$)*($B*B*B$))*
($A*A*B$) og ($B*B*B$) er radvektorer som er [7x1].
($A^T*A^T*B^T$) og ($B^T*B^T*B^T$) er kolonnevektorer som er [1x7].
O er da en matrise som er [7x7].

c) Hvor brede diskrete strukturer kan dette filteret detektere med korrekt plassering av kantene? Begrunn svaret.

Løsningsforslag:
LoG-kjernen i et 7x7 filter er omtrent 3 piksler. Dette er en figur fra forelesningsnotatene:

$$LoG_{7\times7} = -\nabla^2_{5\times5} * G_{3\times3} = \begin{bmatrix} -2 & -8 & -14 & -16 & -14 & -8 & -2 \\ -8 & -24 & -24 & -16 & -24 & -24 & -8 \\ -14 & -24 & 30 & 80 & 30 & -24 & -14 \\ -16 & -16 & 80 & 160 & 80 & -16 & -16 \\ -14 & -24 & 30 & 80 & 30 & -24 & -14 \\ -8 & -24 & -24 & -16 & -24 & -24 & -8 \\ -2 & -8 & -14 & -16 & -14 & -8 & -2 \end{bmatrix}$$

Tommelfingerregel for (diskrete) strukturer (fra forelesningen):
LoG-kjernen må være smalere enn strukturen.
Hvis strukturen er mindre enn halvparten av LoG-kjernen vil nullgjennomgangene bestemmes av filteret og falle utenfor strukturen.
Hvis strukturen er større enn halvparten av LoG-filteret vil nullgjennomgangene falle nøyaktig på kantene til strukturen.

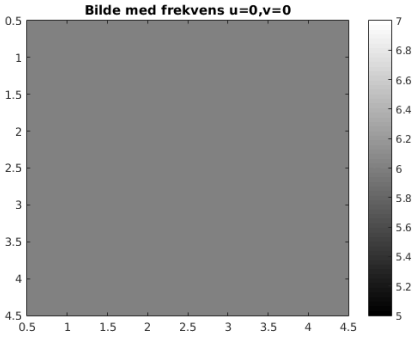


Så svaret må bli at for alle diskrete strukturer som er tilstrekkelig brede vil LoG-filteret detektere strukturen med korrekt plassering av kantene, bare de er bredere enn halvparten av filteret, altså 3 piksler eller bredere.
For strukturer som er 1 og 2 piksler vil bredden bli detektert som ca 3 piksler.
Dette er illustrert i figuren til høyre, hvor venstre kant ligger fast, mens bredden øker.

3. Fouriertransform

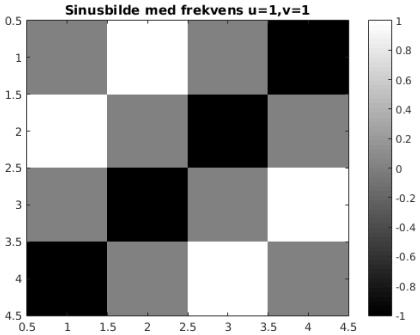
a) Nedenfor følger tre 4x4 gråtonebilder.
Tegn Fourierspekteret til hvert av de tre bildene. Bruk såkalt nullindeksering i Fourierspekterne, altså slik at frekvensen til DC komponenten er (0,0).

Det første bildet er et bilde med kun en gråtoneverdi:



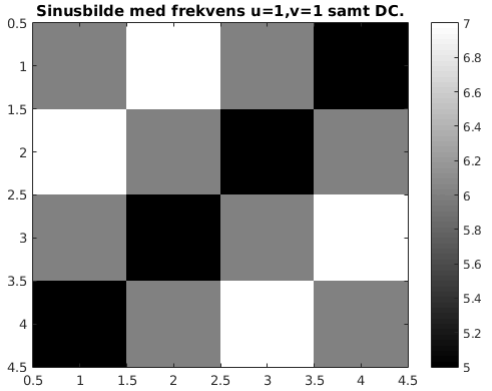
6	6	6	6
6	6	6	6
6	6	6	6
6	6	6	6

Det andre bildet er et sinusbilde med frekvens u = 1, v = 1:



0	1	0	-1
1	0	-1	0
0	-1	0	1
-1	0	1	0

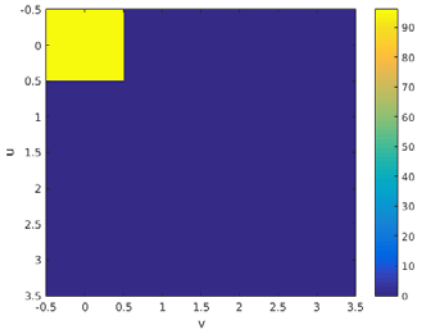
Det tredje bildet er summen av de to første bildene:



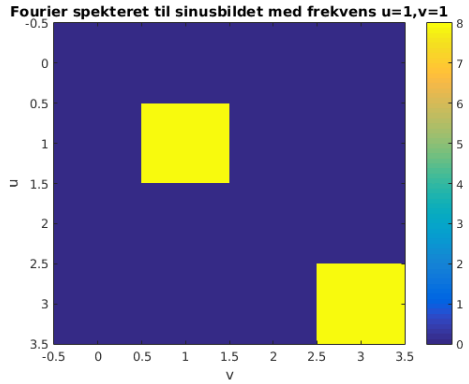
6	7	6	5
7	6	5	6
6	5	6	7
5	6	7	6

Løsningsforslag:

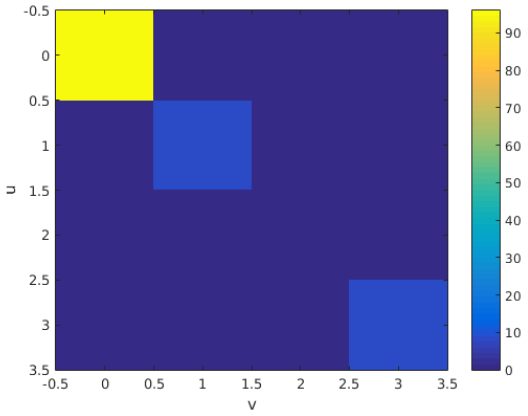
Det skal være relativt greit å finne disse Fourierspekterne. Det første spekteret har verdier kun for DC komponenten, denne verdien skal være N*M*2 = 16*6 = 96.



Det andre spekteret har verdier kun for frekvens u=1,v=1 og u=3,v=3 grunnet symmetri. Denne verdien skal være N*M/2=8.



Det siste spekteret er spekteret til summen av de to tidligere bildene. Altså vil spekteret også være summen av de to tidligere spekterne og vi får:



b) I figuren nedenfor er det gitt den nullutvidede Fourier-spekteret til et 5x5 konvolusjonsfilter. Resonner deg frem til hvilke konvolusjonsfilter dette kan være ved å analysere frekvensresponsen til filteret.

Løsningsforslag:

Dette er et lavpassfilter. Et lavpassfilter vil glatte lokale variasjoner og støy, men vil også glatte kanter. Dette er altså et 5x5 middelverdifilter:

1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25
1/25	1/25	1/25	1/25	1/25

c) Prewitt's horisontale gradientoperator er gitt ved:

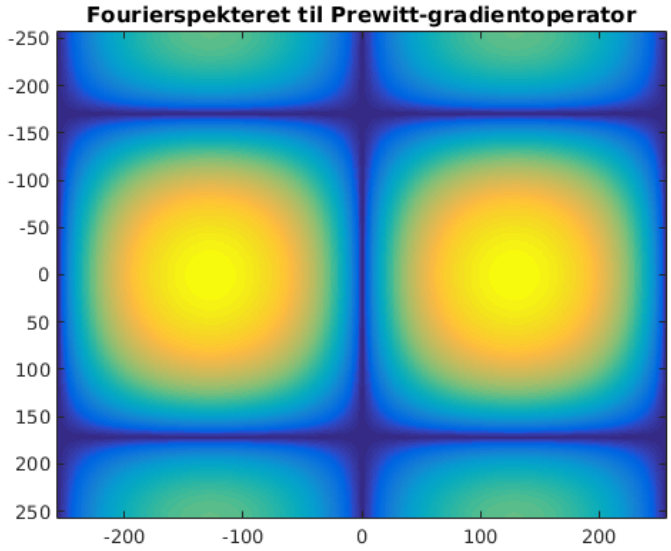
1	0	-1
1	0	-1
1	0	-1

Denne brukes som kjent til å finne gradienter i den horisontale retningen i bildet. Forklar hvilke frekvensegenskaper dette filteret har, og tegn en skisse av Fourierspekteret.

Løsningsforslag:

En mulig forklaring: Gradienten har høy verdi for kanter i bildet. Kanter i bildet inneholder mange frekvenser og vil derfor ha høye koeffisienter for høye frekvenser.

Siden det oppgitte filteret skal finne gradienten må det derfor være et høypassfilter, og siden den finner gradienten i horisontal retning må Fourierspekteret gjennspeile dette ved å kun finne høye frekvenser i horisontal retning.

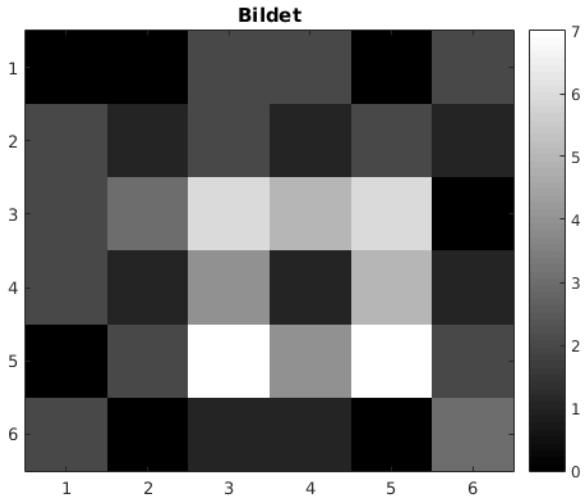


4. Segmentering ved terskling

Vi har følgende 6x6 gråtonebilde med 3 bits gråtoneskala:

0	0	2	2	0	2
2	1	2	1	2	1
2	3	6	5	6	0
2	1	4	1	5	1
0	2	7	4	7	2
2	0	1	1	0	3

Bildet som tallverdier

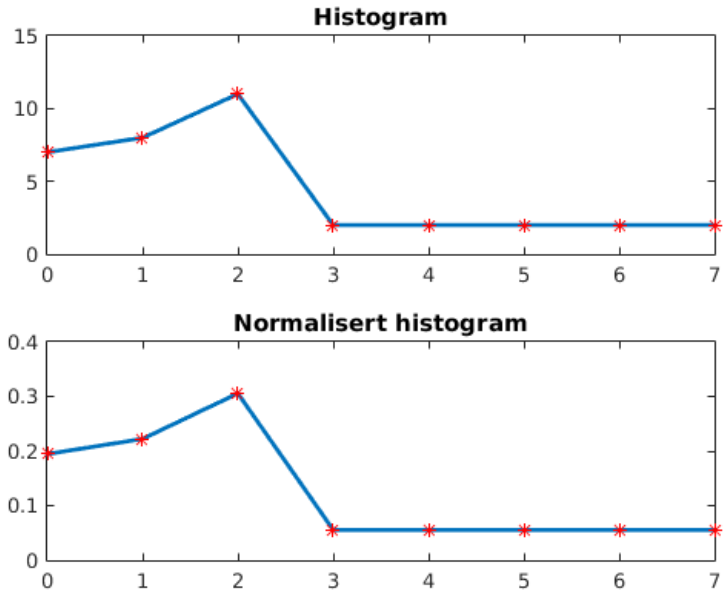


Bildet med pikelsverdier indikert med gråtoner.

a) Finn histogrammet og det normaliserte histogrammet til gråtonebildet.

Løsningsforslag:

Histogram:
h = [6, 9, 11, 2, 2, 2, 3, 1]
Normalisert histogram:
p = [0.17, 0.25, 0.31, 0.06, 0.06, 0.06, 0.8, 0.03]

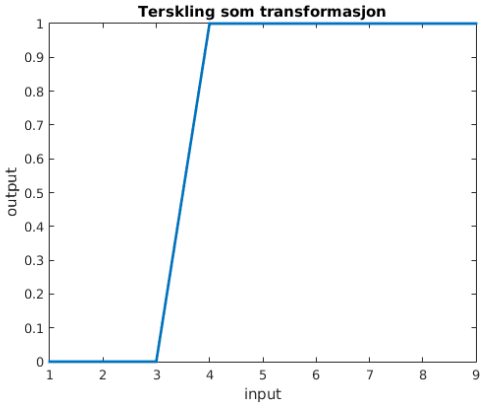


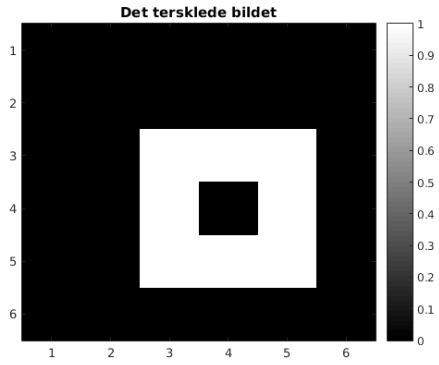
b) Bruk histogrammet og Ridler Calvard's metode (enkel tersklingsalgoritme) til å finne en passende terskel for bildet. Vis fremgangsmåten. Tegn tersklingen som en transformasjon og tegn det resulterende bildet etter terskling.

Løsningsforslag:
Middelverdien i bildet er t = 2.2, avrundet til 2. I Ridler Calvard's metode starter vi med middelverdien som en initiell terskel. Så gjør vi en iterativ prosess hvor vi finner middelverdien (u1(t)) av de pikslene som er mørkere enn terskelen, og middelverdien av de piklsene (u2(t)) som er lysere enn terskelen. Dette gir oss en ny terskel t = (u1(t)+u2(t))/2. Dette gjøres iterativt intill terskelen ikke flytter seg mer.

I dette eksemplet er som sagt den initielle terskelen t = 2. Det er 10 piksler som har verdier større enn 2, og summen av disse er 50, så middelverdien over terskelen er 5. Det er 26 piksler under eller på terskelen, med sum 30, så middelverdi under terskelen er 1,15. Den nye terskelen blir da 3,08 avrundet til 3.
Det skaffer oss to nye piksler under eller på terskelen (begge med verdi 3), så summen blir 30+6 = 36 på 26+2 = 28 piksler, middelverdi 1,29. Over terskelen er det nå 10- 2 = 8 piksler med sum 50-6 = 44, middelverdi 5,5. Og ny terskel er blir da 3,39 avrundet til 3, som er samme terskelverdi som vi hadde, og vi er framme.

Transformasjonen vil bli T = [0 0 0 1 1 1 1 1].





c) Gjør en løpelengdetransformasjon på det tersklede bildet. Gjør løpelengdetransformasjonen rad for rad i bildet.
Bruk formatet: (tall,løpelengde).
Hint: Utnytt gjerne at det tersklede bildet er binært.

Løsningsforslag (om man ikke utnytter at bildet er binært):
Linje 1: (0,6)
Linje 2: (0,6)
Linje 3: (0,2)(1,3)(0,1)
Linje 4: (0,2)(1,1),(0,1),(1,1),(0,1)
Linje 5: (0,2)(1,3)(0,1)
Linje 6: (0,6)

Løsningsforslag (om man utnytter at bildet er binært). Om vi antar at raden starter med sort verdi får vi følgende løpelengdetransformasjon (vi trenger da kun ta med løpelengden).
Linje 1: (6)
Linje 2: (6)
Linje 3: (2)(3)(1)
Linje 4: (2)(1),(1),(1),(1)
Linje 5: (2)(3)(1)
Linje 6: (6)

5. Morfologiske operasjoner

I læreboka og i forelesningen om morfologiske operasjoner på binære bilder antas det at piksler utenfor det binære bildet har verdien 0, og det er ofte bare ett objekt i bildet.

a) Forklar de ulike teknikkene man kan bruke til å håndtere "rand-problemet" ved filtrering, der deler av et filter fallet utenfor bildet, og kommentér spesielt hvor godt egnet de ulike teknikkene er i forbindelse med morfologiske operasjoner på binære bilder når det er flere objekter i bildet, og noen av dem åpenbart strekker seg utenfor bildet.
Anta at strukturelementet er 3x3 med origo i midten, og forklar resonnementet!

Løsningsforslag:
Det er tre hovedteknikker:
1. Sett randpikslene i ut-bildet til en konstant verdi (her 0 eller 1).
Uansett valg av konstantverdi vil dette være veldig synlig her hvor vi antar at noen objekter strekker seg utenfor bildet.

2. Sett randpikslene til originalverdiene i innbildet.
Siden det filtrerte bildet vil ligne mye på originalen, kan det gå bra.

3. Utvid innbildet på en av følgende tre måter:
a. Sette pikslene i utvidet område til en konstant verdi, for eksempel 0.
Sammenfaller med antagelsen nevnt i innledningen, men håndterer ikke objekter som strekker seg utenfor bildet..

b. Kopiere/speile kantpikslene utover "(reflected indexing") før operasjonen utføres.
Siden vi har et 3x3 element vil speiling og kopiering være ekvivalente.
Vil gi forholdsvis gode resultater.

c. Kopiere bildet periodisk ("circular indexing").
Vil gi uheldige effekter der høyre og venstre rand (øvre og nedre) har ulike verdier.

b) "Hit / miss" transformasjonen er definert ved

$$A(*)\,S = A(*)\,[S_1, S_2] = (A \theta S_1) \cap (A^C \theta S_2)$$

der A er et binært bilde, og strukturelementet S er definert ved et par $[S_1, S_2]$ av binære strukturelementer som ikke har noen felles elementer.
Anta at S1 og S2 er symmetriske og har felles origo.

Bruk dialiteten mellom erosjon og dilasjon til å uttrykke A(*)S ved bare A og S_1, altså uten å bruke A^C og S_2.

Løsningsforslag:
Det er bare andre ledd i snittet, $(A^C \theta S_2)$, som vi trenger å gjøre noe med.
Vi har dualiteten $A \oplus S_2 = \left(A^C \theta \hat{S}_2\right)^C$.
Siden S_2 er symmetrisk kan vi droppe refleksjonen.
Dessuten er komplementet til S_1 lik S_2, og dermed har vi

$$A(*)\,S = A(*)\,[S_1, S_2] = \underline{\underline{(A \theta S_1) \cap (A \oplus S_1^C)^C}}$$

c) Beskriv hva man kan oppnå med operasjonene lukking og åpning, forklar hvordan den ene operasjonen kan utføres ved hjelp av den andre hvis struktureringselementet er symmetrisk, og forklar hva nytten av dette er.

Løsningsforslag:
Lukking kan lukke en åpning mellom to strukturer som bare er adskilt med et lite gap, uten at de to strukturene vokser i noen betydelig grad, og fylle mindre hull og innbuktninger.
Åpning kan skape en åpning (mellomrom) mellom to strukturer som bare henger sammen ved en tynn «bro», uten å krympe de to strukturene i noen betydelig grad, og fjerne små og utstikkende strukturer.

Lukking er en **dual** operasjon til **åpning** med hensyn til komplementering og reflektering (180° rotering), og omvendt. Pga symmetri kan vi droppe reflektering.
Lukking kan utføres ved å komplementere bildet f, åpne det med strukturelementet S, og ta komplementet av resultatet: $f \bullet S = (f^C \circ S)^C$
Tilsvarende for åpning: $f \circ S = (f^C \bullet S)^C$

Vi kan altså utføre begge operasjonene med kode bare for den ene, hvis vi har kode for å komplementere et binært bilde.

6. Fargerom og fargebilder

a) Forklar tre måter å representere fargebilder på; RGB, RGBα, og en metode som bruker en fargetabell.
Angi hvor stor plass de vil ta per piksel, og hvor mange forskjellige farger de tillater i bildet.

Løsningsforslag:
1) Tre separate 2D-matriser av samme størrelse, en for R, en for G og en for B (”True color”).
Ett piksel med sine r/g/b-verdier tar samlet vanligvis 3x8=24 bit.
Ca 16,7 millioner farger (2^24).
2. ”True color” – pakket organisering. Én eneste 2d-matrise av piksler, hver piksel inneholder r/g/b-verdier.
Hvert piksel er en 32 bits integer, der de siste 8 bit brukes til transparens.
Også ca16,7 mill farger.
3. Indeksert – én 2D-matrise av heltall, ofte bare 8 bit. Brukes som indeks inn i farge-tabell som definerer hvilke farge alle piksler med den verdien skal ha.
Tillater 256 samtidige farger valgt fra ca 16,7 mill farger.
Ved utvidelse til 16 bits piksler kan fargetabellen inneholde 65.536 av ca 16.7 mill farger.
I tillegg til den plass bildet tar (8 eller 16 bit pr piksel) må vi regne plass til fargetabel-len, hhv 3*2^8 og 3*2^16 byte.

b) Gi en enkel definisjon av gråtoneverdien (intensiteten) til en RGB-verdi. Hvordan vil dette modifiseres hvis man skal ta hensyn til mengden av de tre typene detektorer i øyet eller i et kamera?

Løsningsforslag:
I utgangspunktet et rent gjennomsnitt: I = (r+g+b)/3.
Man modifiseres ved en veiet middelverdi som tar hensyn til at det er flest grønn-følsomme og færrest blåfølsomme detektorer, som for eksempel
Y= 0.299*R + 0.587*G + 0.114*B i transformen fra RGB til YIQ.

Vi krever ikke at man skal huske disse koeffisientene, men man bør huske er at grønn har vesentlig større vekt enn rød, som har vesentlig større vekt enn blå.

c) Beskriv kortfattet ”Median-Cut” algoritmen, og forklar hvordan den brukes.

Løsningsforslag:

1. Finn den boksen i RGB-rommet som omslutter alle fargene i bildet.
2. Sortér fargene i boksen langs den lengste RGB dimensjonen til boksen.
3. Del boksen i to ved medianen til den sorterte listen.
   Dermed blir boksen delt i to nye bokser med (omtrent) like mange piksler tilhørende hver nye boks.
4. Gjenta steg 2 og 3 for alle boksene som nettopp ble dannet.
   Stopp når du har for eksempel 256 bokser.
5. For hver boks, la midtpunktets RGB-verdier representere boksen og lag en 256-linjers LUT som inneholder disse midtpunktene.

Erstatt hver $3 \cdot 8$ biters pikselverdi med en 8 bit indeks som svarer til det boks-midtpunktet som ligger nærmest $3 \cdot 8$ biters pikselverdien i RGB-rommet.

Dette gir en 256-fargers tilpasning til de farger som finnes i bildet.

TAKK FOR OPPMERKSOMHETEN!