TNM059 – Grafisk teknik
Lab 1 – Introduktion till MATLAB och Digitala Bilder

DEL 2 Laboration

Introduktion till MATLAB och digitala bilder

Laborationen innehåller ett antal uppgifter som för det mesta ska lösas med hjälp av MATLAB. Era svar bör skrivas i det avsedda dokumentet *Lab_1.2_Laboration_Svar.docx*, där ni dessutom infogar erforderliga bilder.
För att spara bilder, använd MATLAB funktionen ***imwrite*** eller ***imsave***. Se till att spara bilderna i ett okomprimerat format, liksom *.**tif*** eller *.**png***. **Spara** svarsdokumentet som *.**pdf*** innan ni lägger ut det på Lisam.
För uppgifterna i detta dokument behöver ni inte lämna in några m-filer, men vi rekommenderar starkt att ni sparar era experiment i en m-fil, ifall ni behöver gå tillbaka och rätta till något senare. Ibland, kan ni återanvända era koder i senare uppgifter.
Alla bilder och funktioner som ni kommer att behöva finns på Lisam under *Kursdokument/Labbar/Lab1* och kan även nås via fliken **Laboration** på kurssidans vänstermeny.

Observera att ibland krävs det att ni ska skriva ett antal MATLAB kommandon som svar. Detta kan enkelt göras genom kopiera-klistra-in från MATLAB till svarsdokumentet.
Observera också att ni kan skriva matriserna antingen genom att använda ”Equation i words” och skapa matriser där, eller skriva dem precis som man definierar en matris i MATLAB, eller på ett annat lämpligt sätt för att tydligt visa en matris.

1) Variabeltyper och bildvisning:

Variabler kan sparas i MATLAB i form av olika datatyper. De två vanligaste är `uint8` (8-bitars icke-negativa heltal) och `double` (64-bitars reella tal). Eftersom *uint8* bara använder 8-bitar för att spara ett heltal, kan den bara representera heltal mellan 0 och 255 (**Läs avsnitt 2.1 i kompendiet, sida 3**). Funktionen ***whos*** i MATLAB kan användas för att bl.a. visa en variabels datatyp.
Låt oss nu göra några experiment med olika datatyper i MATLAB.

```
>> a=5;
>> whos a
 Name      Size            Bytes  Class     Attributes
 a         1x1                 8  double
```

I exemplet ovan har vi skapat en variabel *a* i MATLAB, vilket automatiskt fått datatypen double. Dessutom framgår från funktionen ***whos*** att *a* är en 1×1 matris, vilket betyder att den är en skalär, samt att den använder 8 bytes (dvs. 64 bitar). Det framgår också att den är av datatypen *double*.

Nu skapar vi en annan variabel *b* utifrån variabeln *a* (ovan), med hjälp av funktionen ***uint8*** som byter datatypen av en variabel till *unit8*.

```
>> b=uint8(a);
>> whos b
Name      Size          Bytes      Class     Attributes
 b          1x1             1      uint8
```

Som ni ser, kommer *b* att bara behöva 1 byte (dvs. 8 bitar) och att den har datatypen *uint8*.

1.1.

Vi kör följande i MATLAB.

```
>> a/4
ans = 1.2500

>> b/4
ans = 1
```

Varför är b/4 inte helt korrekt?
Vad får vi ut genom b/12? Och Varför?
Skriv era svar i svarsdokumentet på avsedd plats.

Funktionen ***imread*** kan användas i MATLAB för att läsa in bilder. Eftersom bilden 'kvarn.tif' är sparad med 8 bitars precision kommer den skapade variabeln (matrisen) i MATLAB att ha datatypen *uint8*, se nedan:

```
>> k=imread('kvarn.tif');
>> whos k
 Name        Size              Bytes    Class    Attributes
 k         512x512            262144    uint8
```

Som ni ser ovan, har variabeln *k* (vår bild i MATLAB) storleken 512×512, vilket betyder att den är en matris (bild) som innehåller 512 pixlar i höjd (antalet rader i matrisen) och 512 pixlar i bredd (antalet kolumner i matrisen). Dessutom använder den variabeltypen *uint8*, vilket betyder att varje pixel i bilden representeras av 8 bitar (eller 1 byte). Eftersom bilden är 512×512, består den totalt av $512 \times 512 = 262144$ pixlar och eftersom bilden är gråskalebild och varje *uint8* pixelvärde behöver 1 byte, blir det totala minnet som krävs för denna bild 262144 bytes (som ni också ser ovan).

1.2.

Konvertera den här bilden (*k* ovan) till double med funktionen ***double***. Kalla resultatbilden för *k2*. Hur mycket mer minne behöver *k2* jämfört med *k* (skriv hur du räknar)?
Skriv era svar i svarsdokumentet på avsedd plats.

Funktionen ***imshow*** kan användas i MATLAB för att visa bilder. Om bilden som ***imshow*** ska visa är av datatypen *uint8*, då antar ***imshow*** att 0 är svart och 255 är vit. Om bilden som ***imshow*** ska visa är av datatypen *double*, då antar ***imshow*** att 0 är svart och 1 är vit.

1.3.

Visa nu bilderna *k* och *k2* m.h.a *imshow*. Medan bilden *k* visas som en korrekt bild, visas *k2* däremot som en helvit bild. Förklara varför.
Skriv era svar i svarsdokumentet på avsedd plats.

1.4.

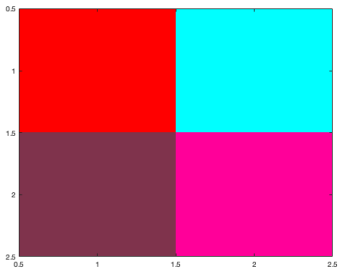
Dela nu båda bilderna (*k* och *k2*) med 255 och visa dem igen, dvs. kör ***imshow(k/255)*** och ***imshow(k2/255)***. Medan den första visas som en helt svart bild, visas den andra som en korrekt bild. Förklara varför. **Observera** att *k/255* består bara av 0:or och 1:or p.g.a. att *k* är av datatypen *uint8* och därför blev resultaten avrundade till närmaste heltal.
Skriv era svar i svarsdokumentet på avsedd plats.

2) RGB färger och colormap:

Olika färger kan beskrivas med hjälp av deras RGB värden i en vektor. T.ex. [0 0 0] är svart, [1 1 1] är vit, [1 0 0] är röd, och [0.5 0.5 0.5] är grå.
Om vi t.ex. kör följande i MATLAB,

```
>> map=([1 0 0; 0 1 1; 0.5 0.2 0.3; 1 0 0.6])
>> image([1 , 2 ; 3, 4])
>> colormap(map)
```

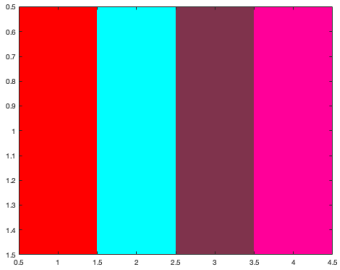
Får vi följande bild.



Och om vi kör

```
>> map=([1 0 0; 0 1 1; 0.5 0.2 0.3; 1 0 0.6])
>> image([1 , 2 , 3, 4])
>> colormap(map)
```

får vi följande bild.



Observera att semikolon (;) används i MATLAB för att separera rader. Skillnaden mellan dessa två visade bilder kan förklaras genom att jämföra de andra raderna i varje kod med varandra.

2.1.

Använd ovanstående exempel för att visa följande sex färger som ett 2×3 rutmönster i ett fönster (dvs. 2 rader och 3 kolumner).

1. Gult (Röd + Grön), upp till vänster
2. Röd, upp-mitten
3. Cyan (Grön + Blå), upp till höger
4. Mörkgrå, ner till vänster
5. Magenta (Röd + Blå), ner-mitten
6. Ljusgrå, ner till höger

Skriv er kod i svarsdokumentet (det krävs bara tre-rader kod som exemplen ovan). Observera att ni kan kopiera dessa tre rader från MATLAB och klistra in. **Infoga** även bilden i svarsdokumentet.
Skriv era svar och infoga bilden i svarsdokumentet på avsedd plats.

3) Matriser och punktvis operation:

I följande exempel förklarar vi hur man skapar olika vektorer/matriser i MATLAB.

>> v1=[1 2 3 4 5]; %Skapar följande radvektor: $v1 = [1, 2, 3, 4, 5]$

>> v2=[2 3 1 4 2]; %Skapar följande radvektor: $v2 = [2, 3, 1, 4, 2]$

>> v3=[2; 6; 8]; %Skapar följande kolumnvektor: $v3 = \begin{bmatrix} 2 \\ 6 \\ 8 \end{bmatrix}$

>> m1=[2 3 4; -1 3 -2]; %Skapar följande 2×3 matris: $m1 = \begin{bmatrix} 2 & 3 & 4 \\ -1 & 3 & -2 \end{bmatrix}$

Om man har en matris *M* i MATLAB, visar kommandot *M(p, q)* elementet på rad *p* och kolumn *q* i *M*. Om man skriver *M(p, :)*, returneras rad *p* i matrisen och om man skriver *M(:, q)* returneras kolumn *q* i matrisen. Skriver man t.ex. *M(1:2:end, :)* returneras varannan rad i matrisen. Låt oss ta upp några exempel:

>> M=[1 2 3 4 5; 6 7 8 9 10; 11 12 13 14 15]; vilket är $M = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \end{bmatrix}$

>> M(:, 2) ger kolumn 2 i matrisen, vilket är $\begin{bmatrix} 2 \\ 7 \\ 12 \end{bmatrix}$

>> M(1:2, 2) ger rad 1 till 2 från kolumn 2, vilket är $\begin{bmatrix} 2 \\ 7 \end{bmatrix}$

>> M(1:2, 1:2:end) ger rad 1 till 2, och kolumn 1, 3 och 5 (varannan kolumn), vilket är $\begin{bmatrix} 1 & 3 & 5 \\ 6 & 8 & 10 \end{bmatrix}$

>> M(1:4, :) kommer resultera i ett felmeddelande eftersom matrisen bara har 3 rader.

3.1.

Vi definierar en matris *N* enligt nedan:

```
>> N=[1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15];
```

Skriv vad följande kommandon resulterar i. Försök att lista ut resultaten utan att köra dem i MATLAB. Sedan får ni gärna dubbelkolla svaren genom att köra dem i MATLAB.

```
>> N(1:3:end, 1)
>> N(1:3:end, :)
```

Skriv era svar i svarsdokumentet på avsedd plats.

Matematiska operationer mellan matriser kan göras på två olika sätt i MATLAB, den vanliga matematiska operationen och punktvis operation. Låt oss förklara detta med två exempel. Anta följande tre matriser:

```
>> m1=[2 3 4; -1 3 -2]; % en 2 × 3 matris
>> m2=[1 2; -1 2]; % en 2 × 2 matris
>> m3=[3 1; 4 -1]; % en 2 × 2 matris
```

Om man kör $m2 * m3$, utför man matrismultiplikationen mellan $m2$ och $m3$, precis som ni har lärt er i linjär algebra, vilket blir: $m2 * m3 = \begin{bmatrix} 11 & -1 \\ 5 & -3 \end{bmatrix}$. Som ni förhoppningsvis kommer ihåg från linjär algebra; för att matrismultiplikationen ska kunna utföras måste antalet kolumner i matrisen till vänster vara lika med antalet rader i matrisen till höger. T.ex. kommer $m1 * m2$ att ge ett felmeddelande eftersom *m1* har tre kolumner medan *m2* har två rader. Däremot går $m2 * m1$ att beräkna och resulterar i en 2×3 matris.
För att genomföra elementvisa operationer i Matlab används punkt (**.**) innan operatorn, t.ex. **.*** för elementvis multiplikation och **./** för elementvis division. Därför, resulterar $m2.* m3$ i $\begin{bmatrix} 3 & 2 \\ -4 & -2 \end{bmatrix}$. För att elementvisa operationer ska kunna utföras måste matriserna vara exakt lika stora och resultatet blir självklart lika stort. T.ex. $m1.* m2$ kommer att resultera i ett felmeddelande eftersom *m1* är 2×3 medan *m2* är 2×2.

3.2.

Vilket värde får *s1* resp. *s2* om vi kör följande kod? Funktionen ***sum*** summerar alla element i en vektor. Om ni får felmeddelande i någon av raderna, förklara vad det bror på!

```
>> v1=[1, 1, 1, 1];
>> v2=[0.5, 0.25, 0, 0.25];
>> v3=[0.25, 0.5, 0, 0.5, 1];
>> s1=sum(v1 .* v2);
>> s2=sum(v1 .* v3);
```

Skriv era svar i svarsdokumentet på avsedd plats.

4) Logiska operationer

En variabel med datatypen *logical* kan bara anta två värden, nämligen 1 om det är sant eller 0 om det är falskt. Detta betyder att för att spara en sådan variabel behövs det bara 1 bit. Låt oss förklara detta med några enkla exempel.

```
>> a=5;
>> l1= (a == 5);
>> l2= (a >= 4);
>> l3= (a ~= 5);
```

Alla tre variabler *l1*, *l2* och *l3* ovan är logiska variabler (använd ***whos*** för att dubbelkolla). *l1* får värdet 1, eftersom *a* är lika med 5. *l2* får också värdet *1*, eftersom *a* är större än eller lika med 4. *l3* får däremot värdet 0, eftersom *a* inte är skilt från 5.

Det går också att kombinera flera logiska operationer:

```
>> a=5;
>> b=4;
>> l4= (a == 5 & b==4);
>> l5= (a ==5 | b==2);
>> l6= (a >3 & b<2);
```

l4 får värdet 1, eftersom *a* är lika med 5 **och** *b* är lika med 4. *l5* får också värdet 1, eftersom *a* är lika med 5 (observera att | betecknar den logiska operationen **eller**. Eftersom *a* är lika med 5, kommer påståendet att vara sant trots att *b* inte är lika med 2). *l6* får däremot värdet 0, eftersom *b* inte är mindre än 2 (och trots att *a>3* kommer inte hela påståendet att vara sant). Logiska operationer körs elementvis och därför kan enkelt användas mellan matriser. Låt oss titta på några enkla exempel:

```
>> m4=[2 3; -1 4];
>> m5=[-1 2; 3 8];
```

Om vi nu kör

```
>> l7= m4 > 3;
```

kommer vi att få $l7 = \begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}$, eftersom endast elementet i position *(2, 2)* (som har värdet $m4(2,2) = 4$) är större än 3.

Om vi nu kör

```
>> l8= (m4 >= 3) & (m5 < 8);
```

kommer vi att få $l8 = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}$. Övertyg er själva varför det blir så.

Om vi t.ex. kör

```
>> l9= m4 >  m5;
```

Kommer vi att få $l9 = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$. Detta betyder att i de positioner där *m4* har ett värde större än värdet på motsvarande position i *m5*, kommer *l9* att få värdet 1. T.ex. $m4(1,1) = 2$, vilket är större än $m5(1,1) = -1$. Därför kommer *l9* att få värdet 1 i position *(1, 1)*. Samma sak gäller för alla andra positioner. Observera igen att för att detta ska fungera måste *m4* och *m5* vara lika stora.

4.1.

Vi definierar två matriser enligt nedan:

```
>> m6=[1 2 3; 4 5 6; 7 8 9];
>> m7=[-2 3 1; 0 2 -7; 1 3 6];
```

Skriv vad följande logiska operationer resulterar i. Försök att lista ut resultaten utan att köra dem i MATLAB. Sedan får ni gärna dubbelkolla svaren genom att köra dem i MATLAB.

```
u1= m6>3;

u2= (m6 > 3 & m7==-7);

u3= (m6 > 3 | m7==-7);

u4= ( (m6+m7) >= 4 & m6>5);
```

Skriv era svar i svarsdokumentet på avsedd plats.

5) Färgbilder

Som diskuterades under avsnitt 1, kan funktionen ***imread*** användas för att läsa in en bild i MATLAB. Den skapade matrisen kommer innehålla lika många element som den inlästa digitala bildens pixlar. För bilder sparade i 8-bitarsformat (de flesta) kommer den resulterande matrisen att vara av datatypen *uint8*, och för att kunna behandla en sådan bild bör den konverteras till *double*. I de flesta bildbehandlingsoperationer antas det även att bildens

pixelvärden ligger mellan 0 och 1. Därför, måste bilden delas med 255 (om den inlästa bilden representeras med 8 bitar). Följande två kommandon kan användas för att läsa in en bild och normera den mellan 0 och 1.

```
>> bild = imread('image.tif');
>> bild = double(bild)/255;
```

Man kan också använda funktionen *im2double* enligt följande för att få bilden normerad mellan 0 och 1:

```
>> bild = imread('image.tif');
>> bild = im2double(bild);
```

Om bilden *image.tif* är en $M \times N$ pixlar stor **gråskalebild**, kommer matrisen *bild* att bli en $M \times N$ matris. Om *image.tif* är en $M \times N$ pixlar stor **färgbild**, kommer matrisen *bild* att bli en $M \times N \times 3$ matris. Dvs. matrisen *bild* kommer att ha tre dimensioner, där den tredje dimensionen representerar de olika färgkanalerna i färgbilden, dvs. kanal 1, 2 och 3 som representerar den röda, gröna resp. blå kanalen av färgbilden. Följande MATLAB-kommandon visar den gröna kanalen av en färgbild:

```
>> fargbild = imread('colorimage.tif');
>> fargbild = im2double(bild);
>> imshow(fargbild(:, :, 2))
```

Observera att *fargbild(:, :, 2)* betyder alla rader, alla kolumner och den andra kanalen (vilket är den gröna) i *fargbild*. Observera också att ***imshow*** visar den gröna kanalen som en gråskalebild, eftersom för ***imshow*** är *fargbild(:, :, 2)* en vanlig två dimensionell matris. Ljusare delar tyder på att det är mer grönt i originalbilden i de delar, och mörkare delar tyder på motsatsen. Om ni vill visa den gröna kanalen i dess rätta färg (dvs. grön) ska ni definiera en färgbild (tre dimensionell matris) där ni sätter dess andra kanal till *fargbild(:, :, 2)* och dess två andra kanaler till 0. Fråga gärna lärarna om ni är intresserade av att göra detta.

5.1.

Läs in färgbilden *'Butterfly.tif'*. **Glöm inte** att konvertera bilden till double och sedan normera. Konvertera sedan RGB bilden till en gråskalebild, genom t.ex. (R+G+B)/3, där R, G och B är den inlästa färgbildens röda, gröna respektive blå kanal. Döp den skapade gråskalebilden till *mygray*.

Skriv MATLAB kommandon ni har använt för att skapa *mygray* (går med bara 5 eller 6 rader kod) och infoga *mygray* i svarsdokumentet.

Ledning: För att spara en bild som heter *minbild* i MATLAB till en bild i *.png* format som ska heta *mypicture* kan ni t.ex. köra:

```
>> imwrite(minbild, 'mypicture.png');
```

Skriv era svar i svarsdokumentet på avsedd plats.

6) Nedsampling och uppsampling:

Här ska ni sampla ner en gråskalebild så att den blir hälften så stor i varje led på två olika sätt. Detta betyder att t.ex. en 500×300 pixlar stor bild blir 250×150.

6.1.

Sampla ner *mygray* från **uppgift 5.1** genom att ta varannan rad och varannan kolumn och döp resultatbilden till *b61*. Använd vad ni har lärt er i avsnitt 3. Detta kan skrivas med bara en-rad kommando.
Skriv MATLAB kommandot i svarsdokumentet och **infoga** *b61*.

6.2.

Detta är egentligen del 3 av denna laboration. Skriv en MATLAB funktion som tar medelvärdet av varje 2×2 område i en bild och sparar värdet som ett pixelvärde i den nedsamplade bilden. Använd det färdiga MATLAB kodskelettet ***samplaner.m***. Observera att er kod ska fungera för alla gråskalebilder oavsett hur stora de är (Läs dokumentet *Lab_1.3_nersampling.pdf* och kommentarerna i kodskelettet ***samplaner.m*** för tydligare instruktioner). Använd din funktion för att sampla ner *mygray* och döp den nedsamplade bilden till *b62*. **Infoga** *b62*. Om ni ännu inte har skrivit koden, kan ni gärna fortsätta till nästa uppgift och infoga *b62* när er kod är färdigskriven.
Infoga bilden i svarsdokumentet på avsedd plats.

Som ni såg i förberedelseuppgifterna, kommer den nedsamplade bilden behöva mycket mindre minne än originalbilden. Detta kan ses som ett mycket enkelt sätt att komprimera en bild. Det är dock mycket viktigt att kunna rekonstruera originalbilden från den komprimerade. Funktionen ***imresize*** kan nu användas för att sampla ner eller upp en bild, se följande exempel:

```
>> b2= imresize(b1, 2, 'nearest');
```

b2 är nu dubbel så stor som *b1* i varje led, och närmaste granne metoden ('nearest') har använts för att sampla upp bilden.

6.3.

Sampla upp nu den nersamplade bilden i **uppgift 6.1**, dvs *b61*, till dess ursprungliga storlek med funktionen, ***imresize***. När det gäller interpolationen använd de tre interpolationerna *'nearest'*, *'bilinear'* och *'bicubic'*. Döp de tre bilderna till *b63_nearest*, *b63_linear*, resp. *b63_cubic*. Hur de här tre olika interpolationerna opererar är utanför kursens ramar, men ni får gärna googla dem eller fråga lärarna om ni vill veta mer. Skriv MATLAB kommandon för att skapa dessa tre bilder (det krävs bara en rad för varje bild). Infoga även de här tre bilderna och diskutera vilken av dessa tre bilder och på vilket sätt ser bättre ut och liknar originalet mest.

Skriv era svar och infoga bilderna i svarsdokumentet på avsedd plats.

Funktionen ***imresize*** kan även användas för att sampla ner/upp en färgbild på samma sätt som för gråskalebilder. För att göra följande uppgifter, läs in färgbilden *'Butterfly.tif'* och döp den till *mycolorimage*. **Glöm inte att konvertera bilden till double och sedan normalisera**.

6.4.

Använd närmaste granne metoden (*'nearest'*) för att sampla ner och upp *mycolorimage* med faktor 0.5 resp. 2. Döp den rekonstruerade bilden till *b64*. Observera att den rekonstruerade bilden ska vara lika stor som *mycolorimage*. Skriv MATLAB kommandon (max två rader behövs) och infoga *b64* och beskriv de tydliga skillnaderna mellan *mycolorimage* och *b64*.

Skriv era svar och infoga bilderna i svarsdokumentet på avsedd plats.

Som ni förhoppningsvis noterat i förberedelseuppgifterna, behöver de komprimerade (nedsamplade) bilderna enligt metoden ovan 25% minne av det som krävs för originalbilden. Kvaliteten av den rekonstruerade bilden är dessvärre inte så bra.
Nu ska vi försöka komprimera en färgbild på ett annat sätt. I **uppgift 6.4** ovan, när ni samplat ner färgbilden har ni faktiskt samplat ner bildens alla tre kanaler samtidigt. I uppgifterna 6.5 och 6.7, ska vi behålla en av tre kanalerna och sampla ner två av dem, och rekonstruera bilden igen. I uppgift 6.8 ska vi komprimera bilden på ett annat sätt.

6.5.

Separera bilden *mycolorimage* till dess tre R, G och B kanaler. Sampla ner R och B och sampla upp dem igen (samplingsfaktorn 0.5 resp. 2) med *'nearest'* och döp dem till *R2* respektive *B2*, men behåll *G* som den var. Använd *R2*, *G* och *B2* för att återskapa bilden till dess originalstorlek. Döp den nya färgbilden till *b65*. Skriv MATLAB kommandon och infoga *b65* och jämför bilderna *b64* och *b65*. Vilken av dem liknar originalet mest? Förklara varför.

Ledning: Det finns två sätt att skapa en färgbild om man har dess tre kanaler. Anta att vi vill skapa en färgbild *bb*, vars R, G och B kanaler representeras med matriserna *RR*, *GG* respektive *BB*. Man kan göra detta antingen genom följande tre rader:

```
>> bb(:, :, 1)=RR;
>> bb(:, :, 2)=GG;
>> bb(:, :, 3)=BB;
```

Eller enligt följande rad:

```
>> bb=cat(3, RR, GG, BB);
```

Skriv era svar och infoga bilderna i svarsdokumentet på avsedd plats.

6.6.

Hur mycket minne utryckt i megabytes (MB) krävs för att spara en 4000×2000 pixlar stor **färgbild** i uint8-format? Hur mycket minne krävs för den komprimerade bilden om vi samplar ner två av bildens färgkanaler till hälften så stor i varje led men behåller den tredje som den var?

Skriv era svar i svarsdokumentet på avsedd plats.

6.7.

Separera bilden *mycolorimage* till dess tre R, G och B kanaler. Sampla ner R och G och sampla upp dem igen (samplingsfaktorn 0.5 resp. 2) med *'nearest'* och döp dem till *R3* respektive *G3*, men behåll *B* som den var. Använd *R3*, *G3* och *B* för att återskapa bilden till dess originalstorlek. Döp den nya färgbilden till *b67*. Skriv MATLAB kommandon och infoga *b67* och jämför bilderna *b65* och *b67*. Håller ni med att *b65* liknar originalet mer än vad *b67* gör? Förklara varför! **Ledning:** Titta på originalbildens tre kanaler och se vilken av kanalerna G och B innehåller mer information och försök att resonera varför *b65* ser bättre ut.

6.8.

Separera bilden *mycolorimage* till följande tre bilder *bild1*=R+G+B, *bild2*=R-G och *bild3*=R+G-2B, där R, G och B är färgbildens tre kanaler. Sampla ner *bild2* och *bild3* och sampla upp dem igen (samplingsfaktorn 0.5 resp. 2) med 'nearest', men behåll *bild1* som den var. Återskapa nu den nya bildens RGB kanaler med hjälp av *bild1*, *bild2* och *bild3* och döp den till *bild68*. Skriv MATLAB kommandon och infoga *b68* och jämför bilderna *b65* och *b68*. Håller ni med att *b68* liknar originalet mer än vad *b65* gör?

Ledning: Observera att följande ekvation gäller för denna uppgift,

$$\begin{cases} R+G+B = bild1 \\ R-G = bild2 \\ R+G-2B = bild3 \end{cases}$$

Att återskapa R, G och B med hjälp av *bild1*, *bild2* och *bild3* innebär att uttrycka R, G och B med hjälp av *bild1*, *bild2* och *bild3* genom att lösa ovanstående ekvationssystem. Lösningen till ovanstående ekvationssystem är följande:

$$\begin{cases} R = \frac{bild1}{3} + \frac{bild2}{2} + \frac{bild3}{6} \\ G = \frac{bild1}{3} - \frac{bild2}{2} + \frac{bild3}{6} \\ B = \frac{bild1}{3} - \frac{bild3}{3} \end{cases}$$

Sammanfattning: Bilda *bild1*, *bild2* och *bild3* enligt ovan. Sampla ner *bild2* och *bild3* och sampla upp dem igen, men behåll *bild1* som den var. Skapa den nya bilden (dvs. *b68*) genom att hitta dess R, G och B kanaler med hjälp av ovanstående ekvationer.

6.9.

Anta att **färgbilden** *B1* (bestående av dess RGB-kanaler) är 4000×12000 pixlar stor och att varje pixel i varje kanal behöver 8 bitar. Hur mycket minne (uttryckt i MB) krävs för att spara denna bild (utan att komprimera bilden)? Hur mycket minne krävs för bild *B2* respektive B3, om:

```
>> B2=imresize(B1, 0.25, 'nearest');
>> B3=imresize(B1, 0.25, 'bilinear');
```

OBSERVERA att nedsamplingsfaktorn är 0.25.

Skriv era svar i svarsdokumentet på avsedd plats.