Matlab-työkirja Konenäkö-kurssille

Tiivistelmä

Tässä dokumentissa opetetaan Matlabin perustoimintojen (muuttujat, matriisit, M-tiedostot) käyttöä sekä kuvankäsittelyssä ja konenäössä tarvittavien funktioiden käyttöä. Perehdyttäminen Matlab-funktioihin tehdään esimerkein. Konenäköön liittyvistä toiminnoista käydään läpi kuvanluku ja esitys, värimuunnokset, kohinan lisäys, suodatus ja geometriset muunnokset. Dokumentti kehittynee kurssin edetessä.

Sisällysluettelo

1. Johdanto

Tässä dokumentissa esitetään lyhyesti Matlabin käyttöä, erityisesti kuvankäsittelyyn liittyen. Osa käytetyistä Matlab-komennoista edellyttää Image Processing -toolboxin asentamista.

Konenäkö-kurssilla kuvankäsittelyharjoitukset tehdään Matlabilla lukuun ottamatta älykameraharjoituksia. Lähes kaikki tehtävät kannattaa toteuttaa M-tiedostoina, koska tiedosto toimii samalla dokumenttina, muutokset ovat helppoja toteuttaa ja uudelleenkäytettävyys on helppoa.

Luvusta 6, Kuvien käsittely Matlabissa, lähtien käsitellään kurssilla ja harjoituksissa läpikäytäviä asioita esimerkein. Tämän dokumentin avulla voi täten myös itsenäisesti tehdä harjoitustehtäviä sekä myös valmistautua perioditenttiin.

Dokumentti kehittynee kurssin edetessä, mutta valmista siitä tuskin tulee yhdellä kertaa. Parannusehdotukset otetaan vastaan kiitollisina.

2. Matlab help

F1-painikkeella saat help-dokumentaation auki. Yksittäisen komennon dokumentaation saat näkyviin `help` *`komento`* -käskyllä. Esim. `help imread` tulostaa kuvanlukukomennon käyttöohjeen. `help images` näyttää kaikki Image Processing Toolbox -pakkauksen komennot.

Tässä ohjeessa tuodaan esille hyödyllisten funktioiden nimiä ja niiden käyttötapoja. On kuitenkin huomattava, että useimmat funktioiden nimet ovat kuormitettuja eli niillä on erilaisia parametriyhdistelmiä. Kannattaa tutustua kunkin funktion toimintoihin tarkemmin help-dokumentaatioiden avulla.

3. Matlabin perusteet

Tässä luvussa käsitellään muun muassa muuttujia, niiden määrittelyä ja tyyppejä, vektoreita, matriiseja, moniulotteisia matriiseja ja matriisilaskutoimituksia. Aluksi toimitaan komentoikkunassa (Workspace).

3.1. Muuttujien käsittelyä

Muuttuja luodaan antamalla sille arvo, esim:

```
>> A=1;
```

Muuttujan tyyppiä ei siis tarvitse ilmoittaa, mutta tyypin määrittäminen on kuitenkin monesti tarpeen. Pienet ja isot kirjaimet ovat eri merkkejä muuttujien nimissä.

Mikäli komento päätetään puolipisteeseen (`;`) komennon tulosta ei tulosteta komentoikkunaan. Tätä ominaisuutta kannattaa käyttää, koska esimerkiksi kuvien tulostus numeroina voi viedä varsin kauan. Mikäli komennon suoritus halutaan keskeyttää, käytetään näppäinyhdistelmää *Ctrl+C*.

Kaikki luodut muuttujat saadaan listattua `who`-komennolla:

```
>> who

Your variables are:

A
```

Tarkempaa tietoa muuttujista saadaan `whos`-komennolla:

```
>> whos
  Name      Size                    Bytes  Class

  A         1x1                         8  double array

Grand total is 1 element using 8 bytes
```

Havaitaan, että muuttujan `A` tyyppi on `double array` eli liukulukuvektori.

Mikäli haluttaisiin luoda muuttaja, johon tallennetaan 8-bittisiä etumerkittömiä lukuja, kuten digitaalisen kuvan intensiteetit yleensä on, muunnetaan lukutyyppi seuraavasti:

```
>> B=uint8(255);
```

Nyt muuttujien listaus antaa:

```
>> whos
  Name      Size                    Bytes  Class

  A         1x1                         8  double array
  B         1x1                         1  uint8 array

Grand total is 2 elements using 9 bytes
```

Muuttujien arvoja voidaan muuttaa päällekirjoittamalla:

```
>> B=uint8(155);
```

Muuttujan arvon saa tulostettua kutsumalla muuttujan nimeä:

```
>> B

B =

  155
```

Muuttujan voi tuhota `clear` *`muuttuja`* -komennolla.

Muuttujilla voidaan tehdä laskutoimituksia, esim.

```
>> C=A+B

C =

  156

>> whos
  Name      Size                    Bytes  Class

  A         1x1                         8  double array
  B         1x1                         1  uint8 array
  C         1x1                         1  uint8 array
  ans       1x1                         1  uint8 array

Grand total is 4 elements using 11 bytes
```

Havaitaan, että `double`- ja `uint8`-tyyppisten muuttujien yhteenlasku antaa `uint8`-tyyppisen vastauksen. Erityistä huolellisuutta vaatii seuraavanlaiset tilanteet:

```
>> D=B+B-100

>> D =

  155
```

Nyt 155 + 155 – 100 = 155 (eikä 210) eli ylivuodon sattuessa (155 + 155) tulos pyöristyy lukualueen ääriarvoon (255, jolloin 255 – 100 = 155). (Esim. Javassa ylivuoto tapahtuu *modulo*-aritmetiikalla.) Tilanne voidaan korjata laskujärjestystä muuttamalla tai vaikkapa tyyppimuunnoksilla:

```
>> D=uint8(double(B)+double(B)-100)

D =

  210
```

Tyyppimuunnokset ovat siis itse asiassa funktiokutsuja.

3.2. Vektorit ja matriisit

Pystyvektori luodaan esim. seuraavasti:

```
>> A=[1; 2; 3]

A =

     1
     2
```

```
     3
```

Vaakavektori puolestaan:

```
>> B=[3 2 1] % tai B=[3, 2, 1]

B =

     3     2     1
```

Matriisi puolestaan saadaan luotua:

```
>> M=[1 2 3; 3 2 1; 2 0 2]

M =

     1     2     3
     3     2     1
     2     0     2
```

eli puolipisteellä erotetaan matriisin rivit. Muuttujien listauksesta nähdään nyt mm. vektoreiden ja matriisien koot:

```
>> whos
  Name      Size                    Bytes  Class

  A         3x1                        24  double array
  B         1x3                        24  double array
  C         1x1                         1  uint8 array
  D         1x1                         1  uint8 array
  M         3x3                        72  double array
  ans       3x3                        72  double array

Grand total is 26 elements using 194 bytes
```

Muuttujan koon saa selville myös `size(`*`muuttuja`*`)`-komennolla, ja mitat voi tallettaa muuttujiin jatkokäyttöä varten:

```
>> [rows colums]=size(A)

rows =

     3


colums =

     1
```

yksittäisen dimension pituuden saa `size(`*`muuttuja, dim`*`)`-funktiolla.

Matriisin yksittäisiin elementteihin viitataan seuraavasti: *`matriisi`*`(`*`rivi, sarake`*`)`, esim:

```
>> M(1,2)

ans =

     2
```

Voidaan viitata myös useampaan elementtiin yhtä aikaa. Seuraavassa poimitaan vasemmasta yläkulmasta 2×2-alimatriisi:

```
>> m=M(1:2,1:2)

m =

     1     2
     3     2
```

Koko toinen rivi saataisiin vastaavasti:

```
>> m2=M(2,:)

m2 =

     3     2     1
```

Voidaan myös viitata esim. vain joka toiseen sarakkeeseen:

```
>> M(2,1:2:size(M,2))

ans =

     3     1
```

Matlabissa on valmiina tehokkaat matriisioperaatiot, seuraavassa joitakin yleisimpiä.

Skalaarilla kertominen:

```
>> 2*M

ans =

     2     4     6
     6     4     2
     4     0     4
```

Transpoosi:

```
>> A'

ans =

     1     2     3
```

Yhteenlasku:

```
>> A+B'

ans =

     4
     4
     4
```

Kertolasku:

```
>> B*A

ans =

    10
```

Determinantti:

```
>> det(M)

ans =

   -16
```

Käänteismatriisi:

```
>> inv(M)

ans =

   -0.2500    0.2500    0.2500
    0.2500    0.2500   -0.5000
    0.2500   -0.2500    0.2500
```

Kertolasku elementeittäin:

```
>> N=[2 2 2; 1 1 1; 0 0 0]

N =

     2     2     2
     1     1     1
     0     0     0

>> M.*N

ans =

     2     4     6
     3     2     1
     0     0     0
```

Vastaavasti tehdään jakolasku elementeittäin `./`-operaattorilla. (Muista nollalla jakamisen ongelma.)

3.2.1. Erityisiä matriiseja

Matriisi, jonka kaikki elementit ovat nollia luodaan komennolla `zeros(rows, columns)`.

Matriisi, jonka kaikki elementit ovat ykkösiä luodaan komennolla `ones(rows, columns)`.

Matriiseihin voidaan lisätä rivejä tai sarakkeita, mutta se ei ole laskennallisesti kovin tehokasta. Mikäli matriisin koko tiedetään etukäteen, kannattaa se luoda esim. `zeros`-komennolla ja täyttää sitten arvoilla. Lisätään `M`-matriisiin `A`-vektori uudeksi sarakkeeksi:

```
>> M=[M A]

M =
```

```
     1     2     3     1
     3     2     1     2
     2     0     2     3
```

3.3. Useampiulotteiset matriisit

Useampiulotteisia matriiseja tarvitaan esim. värikuvien käsittelyssä. Luodaan 3-ulotteinen matriisi, jonka arvot ovat ykkösiä:

```
>> RGB=ones(5,5,3);
```

Muuttujan koko on nyt:

```
>> size(RGB)

ans =

     5     5     3
```

Jos halutaan vastaavankokoinen satunnainen värikuva, käytetään `rand([rows, columns, dim3])`-funktiota:

```
>> RGB=uint8(255*rand(size(RGB)));
```

4. M-tiedostot ja muuttujien näkymisestä

M-tiedostot ovat Matlabiin lisättäviä omia funktioita tai scriptejä (komentosarjoja). Tee oma hakemisto M-tiedostoille ja lisää se Matlabin polkuun (File -> Set Path).

Uusi M-tiedosto luodaan esim. Matlabin omalla editorilla (File -> New -> M-File). M-tiedoston nimeksi annetaan funktion nimi ja päätteeksi *.m*.

M-tiedosto voi olla scripti tai funktio. Scripti on jono komentoja. Kutsuttaessa scriptiä M-tiedoston nimellä, suoritetaan kaikki M-tiedostossa olevat komennot. Scripteillä ei ole paikallisia muuttujia vaan ne näkevät komentoikkunan muuttuja-avaruuden.

Funktioilla on oma muuttuja-avaruus eli funktiot käyttävät paikallisia muuttujia. Ne eivät siis näe komentoikkunan muuttujia eikä komentoikkunasta näe funktion muuttujia. Poikkeuksen muodostavat `global`-komennolla yhteisiksi määritellyt muuttujat. Lisäksi on huomattava, että funktion muuttujat ovat myös kutsukohtaisia, jolloin joka kerta funktiota kutsuttaessa alustetaan uudet muuttujat. Tämäkin ominaisuus voidaan muuttujakohtaisesti ohittaa määrittämällä muuttuja `persistent`-komennolla säilyttämään arvonsa funktionkutsujen välillä.

Funktiolle annetaan kutsussa parametreja ja se voi palauttaa muuttujia. Funktio määritellään esim. seuraavasti:

```
function [tulos]=suodin(kuva, a, b)

          tulos=a*kuva+b;
end
```

Funktiossa ei siis ole `return`-lausetta vaan palautettavat arvot kirjoitetaan palautusmuuttujiin. Yllä olevaa funktiota kutsuttaisiin (komentoikkunasta, scriptistä tai funktiosta) seuraavasti:

```
kuva1=imread('Janne.jpg');
a1=2;
b1=10;
A=suodin(kuva1, a1, b1);
```

Funktion palauttama arvo tallentuu muuttujaan A.

Mikäli funktio palauttaa usean muuttujan, kutsutaan sitä seuraavasti (kaksi muuttujaa):

```
[A B]=funktio1(parametrit);
```

5. Kontrollirakenteet

Matlabissa on useita ohjelmointikielille yleisiä kontrollirakenteita, joista seuraavassa `if`, `while` ja `for`. Kontrollirakenteita voi käyttää komentoikkunassa, mutta lienee luontevampaa toteuttaa ne M-tiedostoissa.

Matlabissa ei ole Boolean-tyyppistä muuttujaa. Lausekkeiden arvo on ’1’, jos ehto on totta ja ’0’ muutoin. Kokeillaan:

```
>> 1==1

ans =

     1
>> 1<=0.9

ans =

     0
```

5.1. Ehtolause (haarautuminen)

```
a=4;
b=5;
if a>b
    c=a-b
else
    c=b-a
end
```

5.2. Alkuehtoinen toisto

Kysytään käyttäjältä tietoa `input('kysymys')`-komennolla `while`-silmukassa, kunnes annettu tieto vastaa haluttua:

```
vastaus='k';
% Verrataan merkkijonoja strcmp-funktiolla.
while not(strcmp(vastaus,'e'))
    % Parametrin 's' ansiosta merkkijonot käsitellään sellaisenaan
eikä
    % muuttujaviittauksina
    vastaus=input('Jatkataanko (k/e):','s');
end
```

5.3. Määrätty toisto

Käydään matriisista `M` läpi jokaisen rivin joka toinen sarake `for`-silmukoilla. Kasvatetaan niiden arvoja yhdellä:

```
for y=1:size(M,1)
    for x=1:2:size(M,2)
        M(y,x)=M(y,x)+1;
    end
end
```

6. Kuvien käsittely Matlabissa

Matlabissa kuvat esitetään yleensä matriiseina. Matlabissa on valmiiksi tehokkaita matriisioperaatioita, mikä on hyväksi kuvankäsittelyssä. Harmaasävykuvat ovat kaksiulotteisia matriiseja ja värikuvat kolmiulotteisia.

6.1. Kuvan luku ja esitys

Kuvan voi lukea tiedostosta esim. `imread('tiedostonimi.pääte')`-komennolla. Tiedostoa etsitään nykyisestä hakemistosta, jonka saa selville `cd`-komennolla. Hakemisto vaihdetaan samalla komennolla: `cd('hakemistopolku')`. Luetaan kuva muuttujaan ja tutkitaan sen kokoa:

```
kuva=imread('paletti.jpg');
size(kuva)

ans =

         803        1200           3
```

Kuvassa on siis kolme värikanavaa, yleensä RGB. Lisäksi `whos`-komento paljastaa, että tietotyyppinä on `uint8`:

```
kuva        803x1200x3                       2890800  uint8 array
```

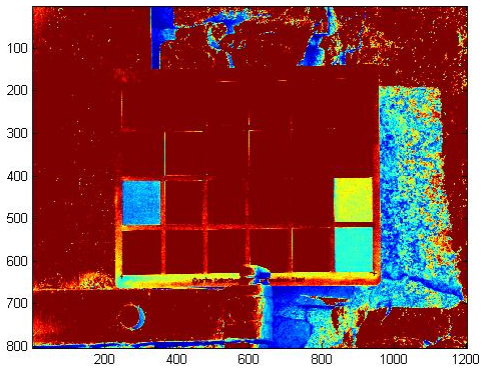
Kuvan esittäminen on kätevää vaikkapa `image`-komennolla. Uuden tulostusikkunan saa luotua `figure`-komennolla. Esitetään äsken luettu kuva:

```
figure;
image(IM)
```

Kuva 1. Kuva Matlabin figure-ikkunassa.

Kokeillaan seuraavaksi tulostaa vain punainen värikanava:

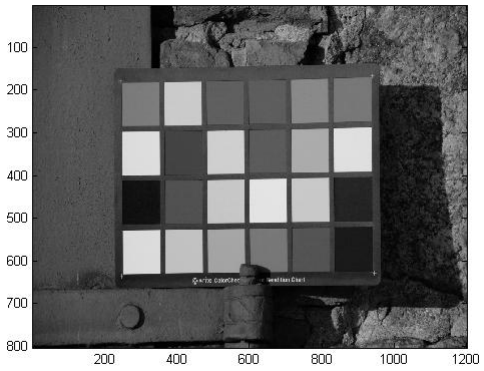
```
figure;
image(kuva(:,:,1))
```



Kuva 2. Monokromaattinen kuva käyttäen oletusarvoista väriavaruutta.

Tulos ei ole halutunlainen, koska monokromaattisia kuvia esitettäessä pitää kuvan väriavaruus valita erikseen `colormap`-komennolla. Väriavaruus muuntaa pikselin arvon (intensiteetin) väriksi. Esitetään punainen kanava vaikkapa harmaasävyillä.

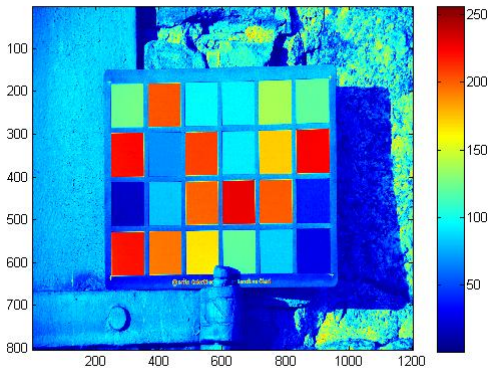
```
colormap(gray(256))
```



Kuva 3. Monokromaattinen kuva harmaasävyväriavaruudessa esitettynä.

Hyödyllinen väriavaruus on myös `jet`, joka skaalaa intensiteetit sinisestä vihreän ja keltaisen kautta punaiseen. Sitä voi käyttää esim. 2-ulotteisten tulosten esittämiseen:

```
colormap(jet(256))
```



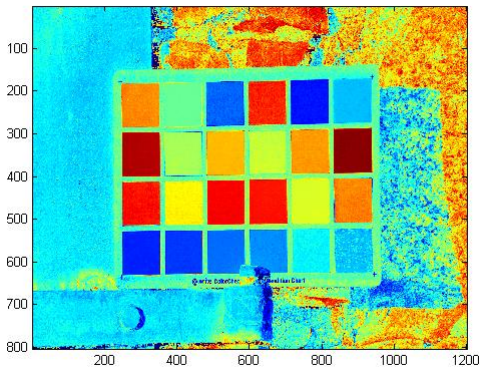
Kuva 4. Monokromaattinen kuva väriavaruudessa `jet`(256), jossa on siis 256 eri väriä.. Oikeassa reunassa väriavaruuden skaala.

6.2. Väriavaruuden vaihto

Valmiita väriavaruudenvaihtofunktioita ovat esim. `rgb2hsv`, `hsv2rgb` ja `rgb2gray`.

Tutkimaan, miten esimerkkikuvamme värisaturaatio näyttää kuvana. 1. Normalisoidaan RGB-kuva, 2. tehdään muunnos rgb → hsv, hsv-arvot ovat normalisoituja, 3. kerrotaan saturaatioarvot (s) 255:llä, 4. esitetään värikylläisyysarvot kuvana:

```
kuva_norm=double(kuva)/255;
kuva_hsv=rgb2hsv(kuva);
figure
image(255*kuva_hsv(:,:,2));
colormap(jet(256));
```



Kuva 5. Testikuvan HSV-muunnoksen tuloksesta esitetty värikylläisyyskanava väriavaruudessa `jet(256)`.

6.3. Kohinan lisäys kuvaan

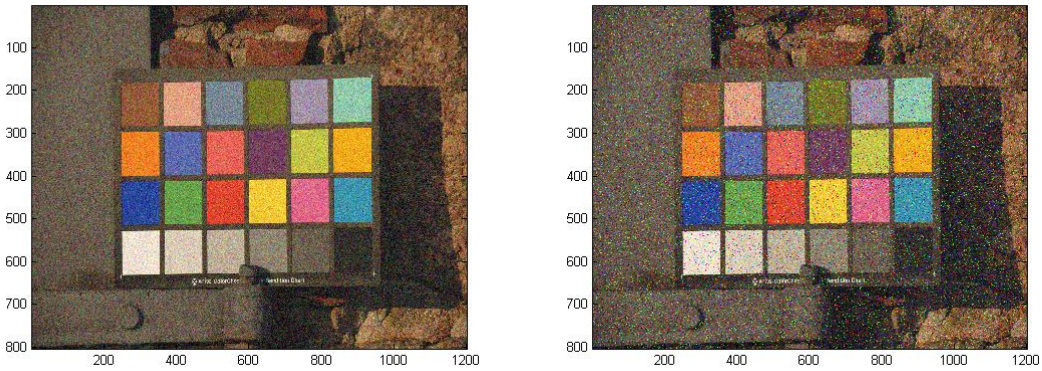
Matlabilla voi tehdä satunnaislukuja mm. taulukon 1 mukaisilla funktioilla. Kohinaa voi lisätä käyttämällä näiden tuottamia satunnaislukuja.

Taulukko 1. Satunnaislukufunktioita.

Nimi	Toiminta
rand	Tasajakauma [0,1]
randn	Normaalijakauma N(0,1)
random	Erilaisia jakaumia käytössä

Image Processing Toolboxissa on myös valmis funktio kohinan lisäämiseen kuvaan.

```
% Tehdään kohinainen kuva
% Käytetetään valkoista normaalijakautunutta kohinaa sekä suola ja
pippuri
% -kohinaa.
% imnoise käyttää normalisoituja yksiköitä, joten keskiarvo 0.1 vastaa
n.
% intensiteettiä 25.
keskiarvo=0.1; % Kuva siis vaalenee keskimäärin
varianssi=0.01;
kohinakuva=imnoise(kuva,'gaussian', keskiarvo, varianssi);
figure
image(kohinakuva)
kohinakuva=imnoise(kohinakuva,'salt & pepper',0.05);
figure
image(kohinakuva)
```



a) b)

Kuva 6. Kohinaisia kuvia. a) Gaussista kohinaa, b) Gaussista ja suola ja pippuri -kohinaa lisätty.

6.4. Kuvan suodatus

Kaksiulotteinen FIR-suodatus onnistuu `filter2(`*`kertoimet, kuva`*`)`-komennolla. Tehdään kuvalla keskiarvoistus 3×3-kokoisella ikkunalla:

```
w=7;
h=ones(w,w)/(w*w);
kuva_mean_3_3=kuva;
kuva_mean_3_3(:,:,1)=filter2(h,kuva(:,:,1),'same');
kuva_mean_3_3(:,:,2)=filter2(h,kuva(:,:,2),'same');
kuva_mean_3_3(:,:,3)=filter2(h,kuva(:,:,3),'same');
figure
image(kuva_mean_3_3)
```

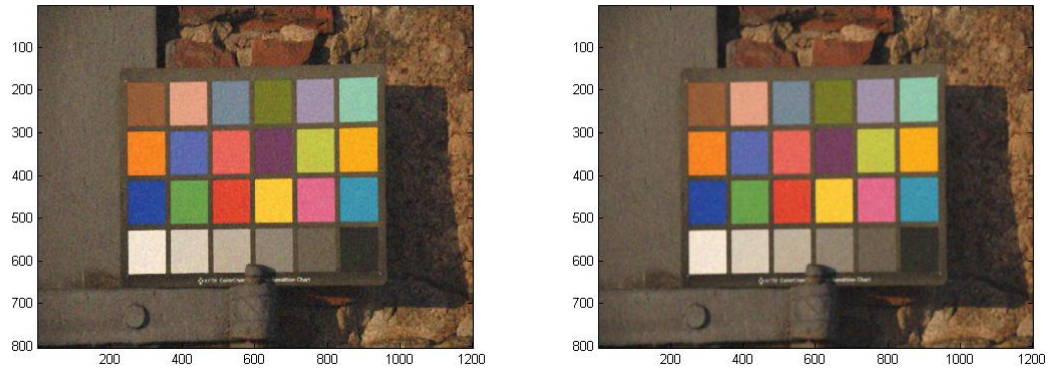
Monipuolisempi vaihtoehto on käyttää `imfilter(`*`kuva,H`*`)`-funktiota, jolloin reuna-alueiden käsittelyssä voidaan valita esim. peilausvaihtoehto. `fspecial` tarjoaa joitakin hyödyllisiä FIR-kertoimia, kuten gaussisen keskiarvosuotimen:

```
% Suodin
H=fspecial('gaussian',[5 5],2)
kuva_gaussian=imfilter(kuva,H,'symmetric');
figure
image(kuva_gaussian)

H =

    0.0232    0.0338    0.0383    0.0338    0.0232
    0.0338    0.0492    0.0558    0.0492    0.0338
    0.0383    0.0558    0.0632    0.0558    0.0383
    0.0338    0.0492    0.0558    0.0492    0.0338
    0.0232    0.0338    0.0383    0.0338    0.0232
```

Kaksiulotteisen mediaanisuodatuksen voi tehdä `medfilt2`-funktiolla. Seuraavaksi yritetään kohentaa gaussista sekä suola ja pippuri -kohinaa sisältävää kuvaa (Kuva 6 b). Ensiksi tehdään mediaanisuodatus (5 × 5 ikkunalla) suolan ja pippurin poistamiseksi. Toiseksi gaussisella keskiarvosuotimella suodatetaan mediaanisuotimen tulosta.

```
% Mediaanisuodatus
kuva_medfilt2=kohinakuva;
kuva_medfilt2(:,:,1)=medfilt2(kohinakuva(:,:,1),[5 5]);
kuva_medfilt2(:,:,2)=medfilt2(kohinakuva(:,:,2),[5 5]);
kuva_medfilt2(:,:,3)=medfilt2(kohinakuva(:,:,3),[5 5]);
```

```
figure
image(kuva_medfilt2);
% Gaussinen suodin
H=fspecial('gaussian',[5 5],2)
kuva_gaussian=imfilter(kuva_medfilt2,H,'symmetric');
figure
image(kuva_gaussian)
```



a) b)

Kuva 7. a) Tulos mediaanisuodatuksen jälkeen. b) Tulos mediaani- ja gaussisen suotimen jälkeen.

6.5. Geometriset muunnokset

Käydään läpi Matlab-esimerkit affiinimuunnoksesta ja yleisestä ei-kiinteän kappaleen muunnoksesta.

6.5.2. Affiinimuunnos

1. Määritetään muunnosmatriisi, jossa skaalaus ja y-suuntainen leikkaus:

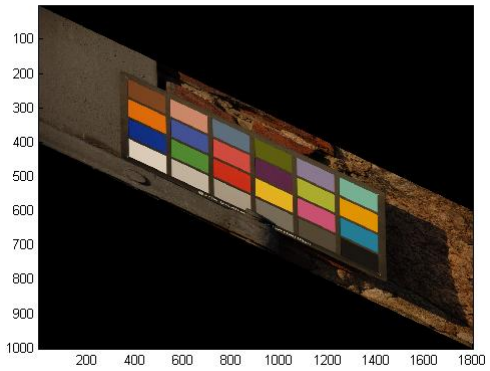
```
cx=1.5;
cy=0.5;
sy=0.5;
Aff=[cx 0 0; sy cy 0; 0 0 1]';
% Affiinimatriisi täytyy transponoida (’), koska luentomateriaalissa
muunnosmatriisi on esitetty toisin, kuin mikä on Matlabin esitystapa
```

2. Tehdään muunnosmatriisista `TFORM`-struktuuri:

```
Aff_TFORM=maketform('affine',Aff);
```

3. Suoritetaan muunnos:

```
kuva_aff=imtransform(kuva, Aff_TFORM);
figure
image(kuva_aff)
```



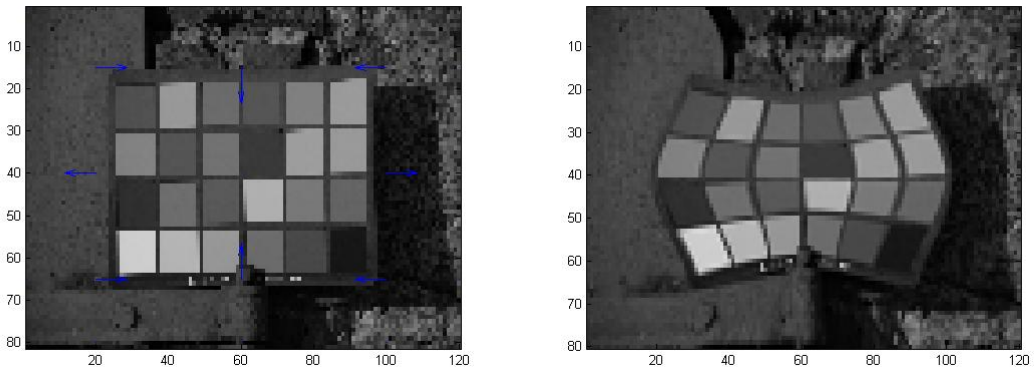
Kuva 8. Kuva muunnettuna affiinimuunnoksella, jossa skaalaus ja y-leikkaus.

6.5.3. Yleinen muunnos

Käytetään uudelleennäytteistettyä harmaasävykuvaa prosessointiajan pienentämiseksi kokeiluvaiheessa:

```
% Käytetään uudelleennäytteistettyä harmaasävykuvaa prosessointiajan
% pienentämiseksi kokeiluvaiheessa
kuva_gray=rgb2gray(kuva);
kuva_gray=kuva_gray(1:10:size(kuva_gray,1), 1:10:size(kuva_gray,2));
%   Määritetään   kontrollipisteet,   joiden   siirtymä   tunnetaan,
säännöllisinä
% matriiseina
x=[1 20 60 100 120];
y=[1 15 40 65 80]';
[X Y]=meshgrid(x,y)
% X ja Y matriisien mukaisten pisteiden siirtymät x ja y suuntiin
Dx=[0 0 0 0 0; 0 5 0 -5 0; 0 -5 0 5 0; 0 5 0 -5 0; 0 0 0 0 0;];
Dy=[0 0 0 0 0; 0 0 5 0 0; 0 0 0 0 0; 0 0 -5 0 0; 0 0 0 0 0;];

% Interpoloidaan Dx and Dy kuutiollisilla splineillä

% Määritetään ensiksi interpolointialue
intp_x_min=X(1,1);
intp_y_min=Y(1,1);
intp_x_max=X(1,size(X,2));
intp_y_max=Y(size(Y,1),1);
% Määritetäät pisteet XI ja YI, joissa siirtymän arvo halutaan tietää
[XI,YI] = meshgrid(intp_x_min:intp_x_max,intp_y_min:intp_y_max);
%  Interpoloidaan  interp2-funktiolla,  koska  X  ja  Y,  joissa  arvot
tiedetään
% ovat tasavälein määriteltyjä
DxI = interp2(X,Y,Dx,XI,YI,'cubic');
DyI = interp2(X,Y,Dy,XI,YI,'cubic');

% Luodaan tyhjät vektorit joihin talletetaan kunkin pikselin x ja y
% koordinaatit liukulukuina ja vastaava intensiteettiarvo
xt=zeros(size(kuva_gray,2)*size(kuva_gray,1),1);
yt=zeros(size(kuva_gray,2)*size(kuva_gray,1),1);
bt=zeros(size(kuva_gray,2)*size(kuva_gray,1),1); % intensiteetti

% Lasketaan pikseleille uudet koordinaatit ja talletetaan
% intensiteettiarvot muuttumattomina
for i2=intp_y_min:intp_y_max
    for i1=intp_x_min:intp_x_max
```

```
        xt(i1+size(kuva_gray,2)*(i2-1))=XI(i2,i1)+DxI(i2,i1);
        yt(i1+size(kuva_gray,2)*(i2-1))=YI(i2,i1)+DyI(i2,i1);
        bt(i1+size(kuva_gray,2)*(i2-1))=kuva_gray(i2,i1);
    end
end

% Interpoloidaan intensiteetit pisteisiin (XI, YI). Nyt täytyy käyttää
griddata-funktiota,
% koska pisteet (x,y) eivät enää ole tasavälein. Tulos pyöristetään 8
bittiin. Tuloksena syntyy
% kaksiulotteinen matriisi.
kuva_morphed=uint8(griddata(xt,yt,bt,XI,YI,'cubic'));

% Esitetään kuva
figure
image(kuva_morphed)
colormap(gray(256))

% Piirretään lisäksi alkuperäiseen kuvaan kontrollipisteiden
% translaatiovektorit (vektoreiden pituus mielivaltaisissa yksiköissä)
figure
image(kuva_gray)
colormap(gray(256))
hold on
quiver(X,Y,Dx,Dy,0.3)
```



a) b)

Kuva 9. a) Alinäytteistetty kuva ja kontrollipisteiden translaatiovektoreiden suunnat, b) geometrisesti muunnettu kuva.

6.6. Hahmonsovitus (pattern matching)

Hahmonsovituksessa mitataan kuvien välistä samanlaisuutta/erilaisuutta jollakin hyvyysfunktiolla/etäisyysfunktiolla. Toinen verrattavista kuvista on mallikuva **M**, ja toinen tuntematon kohdekuva **A**. Usein mallikuvaaa kohtaa termillä ZOI (Zone-Of-Interest). Kun kuvien välistä etäisyyttä lasketaan, tulee niiden olla samankokoisia.

Yleensä tuntematon kuva **A** on mallikuvaa suurempi. Tunnistettava kohde voi olla missä tahansa kohtaa **A**:ta. Tällöin **M**:ää transloidaan eli liu'utetaan **A**:n alueella ja lasketaan hyvyysfunktion arvoa kullakin translaatiolla.

Käytännössä tehdään esim. seuraavasti: Olkoon translaatiovektori `t=[ix,iy]'` vastaten mallikuvan vasenta yläkulmaa. Leikataan kuvasta **M**:n kokoinen alue ROI (Region-Of-Interest):

```
[h,w]=size(ZOI); % Mallikuvan koko
ROI=kuva(iy:iy+h-1,ix:ix+w-1);
```

Jos hyvyysfunktiona on korrelaatio, voidaan ZOI:n ja ROI:n välinen samankaltaisuus laskea vaikkapa Matlabin `corr2(ROI,ZOI)`-funktiolla.

Neliöetäisyys (rms = Root-Mean-Squared) saataisiin esim: `sqrt(sum(sum((ROI-zoi).*(ROI-ZOI)))/(size(ZOI,1)*size(ZOI,2)));`

Translaatiot käydään läpi sisäkkäisissä `for`-silmukoissa, kuten luvun 6.6.4 esimerkissä.

6.6.4. Usean luokan hahmonsovitus ja paikannus

Seuraavassa Matlab-funktiossa etsitään useita (tai yhtä) mallikuvia:

```
% [korrelaatiokertoimet, x-koordinaatit, y-
koordinaatit]=(harmaasävykuva, mallikuvat)
function [r, x, y]=haeMalli(kuva, ZOIs)
    [H,W]=size(kuva);

    % Alustetaan ulostulomuuttujat
    r=zeros(size(ZOIs,3),1);
    x=zeros(size(ZOIs,3),1);
    y=zeros(size(ZOIs,3),1);

    % Käydään kaikki mallit läpi
    for malli=1:size(ZOIs,3)
        zoi=ZOIs(:,:,malli); % Zone-Of-Interest
        [h,w]=size(zoi); % Mallikuvan koko
        % [ix,iy]' = mallikuvan vasen yläkulma
        r(malli)=-1;
        for iy=1:H-h
            iy
            for ix=1:W-w
                % Leikataan kuvasta mallikuvan kokoinen alue, jota
                % verrataan mallikuvaan translaatiolla (ix, iy)
                ROI=kuva(iy:iy+h-1,ix:ix+w-1); % Region-Of-Interest
                % Lasketaan korrelaatio valmiilla funktiolla, vaikka
                % tehokkaampaa olisi käyttää omaa funktiota, jolloin
                % mallikuvan varianssin tarvitsisi laskea vain kerran.
                r_temp=corr2(ROI,zoi);
                %rms_temp=sqrt(sum(sum((ROI-zoi).*(ROI-
zoi)))/(size(zoi,1)*size(zoi,2)));

                % Jos uusi korrelaatiohuippu löytyi, päivitetään
                % koordinaatit ja korrelaatiokerroin
                if r_temp>r(malli)
                    r(malli)=r_temp;
                    % Muutetaan koordinaatisto mallikuvan
keskipisteeseen
                    x(malli)=ix+floor(w/2); % Pyöristetään alaspäin
kokonaislukuun: esim. floor(5/2)=2
                    y(malli)=iy+floor(h/2);
                end
            end
        end
    end
end % function
```

Jos haettavat mallikuvat ovat ZOI1, ZOI2 ja ZOI3, ne annetaan funktiolle 3-ulotteisena matriisina:

```
ZOIs(:,:,1)=ZOI1;
ZOIs(:,:,2)=ZOI2;
ZOIs(:,:,3)=ZOI3;
```

Funktiota kutsutaan esim:

```
[Rs, Xs, Ys]=haeMalli(kuva_gray, ZOIs);
```

Tuloksessa `Rs`, `Xs`, ja `Ys` ovat vektoreita, joiden alkiot vastaavat mallikuvien korrelaatioita ja (x,y)-koordinaatteja.

6.6.5. Geometriset muunnokset mallinsovituksessa

Edellisessä esimerkissä oli kaksi sisäkkäistä silmukkaa translaatiolle (*ix*, *iy*). Lisätään yksi sisäkkäinen silmukka lisää käymään läpi rotaatioita. Kierron resoluutioksi valitaan esim. 10 astetta. Mallikuvalle tehdään kyseinen kierto. Tämä silmukka kannattaa sijoittaa siten, että sama muunnos (rotaatio) tarvitsee tehdä vain kerran.

Rotaatiossa siirretään origo ensin translaatiolla haluttuun kiertopisteeseen, yleensä kuvan keskikohtaan. Rotaation jälkeen origo voidaan siirtää takaisin kuvan vasempaan yläkulmaan. Ongelman muodostaa vielä reunalle muunnoksessa kulmiin tulevat mustat pikselit, jotka vääristävät tulosta. Tilanne paranee, jos täytetään kulmat mallikuvan keskiarvolla.

```
for rot=0:10:350

        % Rotatoidaan mallikuvaa

        % Siirretään origo
        Aff=[1 0 -floor(w/2); 0 1 -floor(h/2); 0 0 1];
        % Rotaatio
        Aff=[cos(rot*pi/180) sin(rot*pi/180) 0; -sin(rot*pi/180) cos(rot*pi/180) 0; 0 0
1]*Aff;
        % Origo takaisin
        Aff=[1 0 floor(w/2); 0 1 floor(h/2); 0 0 1]*Aff;
        Aff=Aff'; % Matlabin vaatima transpoosi
        Aff_TFORM=maketform('affine',Aff);
        % Muunnos, jossa XData ja YData hakevat tuloksesta oikean kokoisen
        % kuvan oikeasta kohdasta. Täytetään reunat keskiarvolla
        rotzoi=imtransform(zoi, Aff_TFORM, 'bicubic','XYScale',1,'XData',[1
w],'YData',[1 h],'FillValues', mean2(zoi));

             for iy=1:H-h
             …
end
```

Täydellinen vaihtoehto on laskea korrelaatio vain aidoista pikseleistä. Tämä voitaisiin tehdä täyttämällä rotatoidun kuvan kulmat arvoilla `NaN` (Not A Number) (Huomaa tyyppimuunnos `double(zoi)`!)

```
rotzoi=imtransform(double(zoi), Aff_TFORM, 'bicubic','Size',size(zoi),
'FillValues', NaN);
```

ja tekemällä oma korrelaatiofunktio. Toinen yhtä hyvä vaihtoehto olisi käyttää mallikuvan muunnosten pohjana lopullista mallikuvaa suurempaa kuvaa, jolloin reunoihin tulisi aidot pikseliarvot.

Kokonaisuudessaan saadaan seuraavanlainen Matlab-funktio:

```
% Output: [R, r, x, y, rotation]=[korrelaatiokuva, optimin korrelaatiokerroin, optimin
x-koordinaatti, optimin y-koordinaatti, optimirotaatio]
% Input: (harmaasävykuva, mallikuva, x- ja y-translaatioiden resoluutio, rotaation
minimi, resoluutio ja maksimi)
```

```
function [R, r, x, y, rotation]=haeMalli_rot(kuva, zoi, trans_step, r_min, r_step,
r_max)
    [H,W]=size(kuva);

    % Alustetaan korrelaatiokuvan taulukko
    R=zeros(size(zoi));
    % Alustetaan optimikorrelaatio
    r=-1;
    [h,w]=size(zoi); % Mallikuvan koko


    ir=1; % Taulukon indeksi
    for rot=r_min:r_step:r_max
        rot
        % Rotatoidaan mallikuvaa

        % Siirretään origo
        Aff=[1 0 -floor(w/2); 0 1 -floor(h/2); 0 0 1];
        % Rotaatio
        Aff=[cos(rot*pi/180) sin(rot*pi/180) 0; -sin(rot*pi/180) cos(rot*pi/180) 0; 0 0
1]*Aff;
        % Origo takaisin
        Aff=[1 0 floor(w/2); 0 1 floor(h/2); 0 0 1]*Aff;
        Aff=Aff'; % Matlabin vaatima transpoosi
        Aff_TFORM=maketform('affine',Aff);
        % Muunnos, jossa XData ja YData hakevat tuloksesta oikean kokoisen
        % kuvan oikeasta kohdasta. Täytetään reunat keskiarvolla
        rotzoi=imtransform(zoi, Aff_TFORM, 'bicubic','XYScale',1,'XData',[1
w],'YData',[1 h],'FillValues', mean2(zoi));

        % Tarvittaessa voidaan tulostaa rotatoidut kuvat poistamalla
        % kommentit
        % figure
        % image(uint8(rotzoi))
        % colormap(gray(256))
        % drawnow

        % Translaation for-silmukat
        for iy=1:trans_step:H-h
            for ix=1:trans_step:W-w
                % Leikataan kuvasta mallikuvan kokoinen alue, jota
                % verrataan mallikuvaan translaatiolla (ix, iy)
                ROI=kuva(iy:iy+h-1,ix:ix+w-1); % Region-Of-Interest

                % Lasketaan korrelaatio
                r_temp=corr2(ROI,rotzoi);

                % Lisätään korrelaatio ulostulomuuttujaan
                R(ir,iy,ix)=r_temp;
                % Jos uusi korrelaatiohuippu löytyi, päivitetään
                % koordinaatit ja korrelaatiokerroin
                if r_temp>r
                    r=r_temp;
                    rotation=rot;
                    x=ix;
                    y=iy;
                end
            end % ix
        end % iy
        ir=ir+1;
    end % rot

    % Esitetään optimitranslaatio suorakaiteella kuvassa
    figure
    image(kuva)
    colormap(gray(256))
    hold on
    rectangle('Position',[x,y,w,h])
    rotation

    % Esitetään vielä optimaaliset zoi ja ROI.
    Aff=[1 0 -floor(w/2); 0 1 -floor(h/2); 0 0 1];
    Aff=[cos(rotation*pi/180) sin(rotation*pi/180) 0; -sin(rotation*pi/180)
cos(rotation*pi/180) 0; 0 0 1]*Aff;
```

```
    Aff=[1 0 floor(w/2); 0 1 floor(h/2); 0 0 1]*Aff;
    Aff=Aff';
    Aff_TFORM=maketform('affine',Aff);

    rotzoi=imtransform(zoi, Aff_TFORM, 'bicubic','XYScale',1,'XData',[1 w],'YData',[1
h],'FillValues', mean2(zoi));

    ROI=kuva(y:y+h-1,x:x+w-1); % Region-Of-Interest

    figure
    image(ROI)
    colormap(gray(256))
    drawnow

    figure
    image(uint8(rotzoi))
    colormap(gray(256))
    drawnow
end % function
```

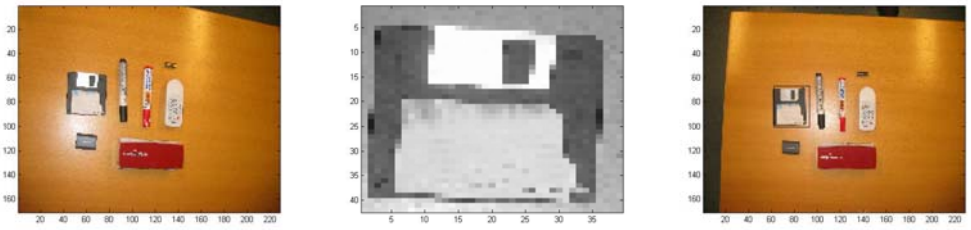
Kokeillaan vielä skaalausta. Nyt kolmas silmukka on oleellisilta osin seuraavanlainen:

```
        for ic=minc:resc:maxc
            % Skaalataan mallikuva
            ic
            Aff=[ic 0 0; 0 ic 0; 0 0 1]';
            Aff_TFORM=maketform('affine',Aff);
            czoi=imtransform(zoi, Aff_TFORM,
'bicubic','Size',floor(ic*size(zoi)), 'FillValues', mean2(zoi));
            figure
            image(czoi)
            colormap(gray(256))
            drawnow
            [h,w]=size(czoi);
```

Funktiolle annetaan tällä kertaa parametreinä hakualue (`minc,resc, maxc`) skaalaukselle. Testataan leikkaamalla mallikuva toisesta kuvasta (O1_2.jpg kurssin kuva-arkistosta) ja etsimällä samaa aluetta testikuvasta (O1_4.jpg). Tässä tarvittava skripti on seuraava:

```
% Opetuskuvan valinta
[FileName,PathName] = uigetfile(strcat('*','.jpg'),'Select training
image')
opetuskuva=imread(strcat(PathName,FileName));
ds=10;
opetuskuva=opetuskuva(1:ds:size(opetuskuva,1),
1:ds:size(opetuskuva,2),:);
f1=figure;
image(opetuskuva)
impixelinfo
[x y]=ginput(1);
X_MIN=round(x);
Y_MIN=round(y);
[x y]=ginput(1);
X_MAX=round(x);
Y_MAX=round(y);
close(f1)

% Mallikuvan leikkaus opetuskuvasta
opetuskuva=rgb2gray(opetuskuva);
mallikuva=opetuskuva(Y_MIN:Y_MAX, X_MIN:X_MAX);
```

```
f1=figure;
image(mallikuva);
colormap(gray(256))

% Haetaan mallikuvaa testikuvasta
[FileName,PathName] = uigetfile(strcat('*','.jpg'),'Select test
image')
testikuva=imread(strcat(PathName,FileName));
testikuva=testikuva(1:ds:size(testikuva,1), 1:ds:size(testikuva,2),:);
testikuva_gray=rgb2gray(testikuva);

% Suoritetaan haku
[R, r, x, y, c]=haeMalli3(testikuva_gray(20:140,20:140), mallikuva,
0.6, 1.0, 0.1);
x=x+20;
y=y+20;
[r x y c]

% Esitetään testikuva ja paikannettu kohta suorakaiteella
f2=figure;
image(testikuva)
rectangle('Position',[x, y, c*size(mallikuva,2), c*size(mallikuva,1)])
```



(a) (b) (c)

Kuva 10. Opetuskuva (a), siitä leikattu mallikuva (b) ja testikuva (c), jossa musta suorakaide osoittaa paikannuksen tuloksen (translaatio ja skaala).

6.6.6. Alipikselipaikannus funktion sovituksella

Olkoon `R` matriisi, joka sisältää korrelaatiot eri translaatioilla, maksimin (`x`, `y`) naapurustossa. Alipikselipaikannus saadaan seuraavasti:

```
function [u, v]=interpoloiMaksimi(R)
    [H W]=size(R);
    h=ceil(H/2);
    w=ceil(W/2);
    % Muutetaan matriisi R vektoriksi
    r_vect=zeros(H*W,1);
    % Design matrix
    A=ones(H*W,6);
    % Rivin indeksi: A ja r_vect
    i=1;
    for iy=1:H
        for ix=1:W
            r_vect(i)=R(iy,ix);
            % x^2 (x suhteessa keskipisteeseen)
            A(i,1)=(ix-w)^2;
            % y^2
```

```
            A(i,2)=(iy-h)^2;
            % xy
            A(i,3)=(ix-w)*(iy-h);
            % x
            A(i,4)=(ix-w);
            % y
            A(i,5)=(iy-h);
            i=i+1;
        end
    end
    % Lasketaan kertoimet (ax2, ay2, axy, ax, ay, a0) pienimmän
neliösumman
    % sovituksella
    k=inv(A'*A)*A'*r_vect;
    % Huippu suhteessa keskipisteeseen
    u=(2*k(2)*k(4)-k(3)*k(5))/(k(3)^2-4*k(1)*k(2));
    v=(2*k(1)*k(5)-k(3)*k(4))/(k(3)^2-4*k(1)*k(2));
end
```

Funktion palauttamilla arvoilla (u, v) korjataan tasapikselihuippua (x, y):

```
x=x+u;
y=y+v;
```

6.7. Tekstuurianalyysi

6.7.7. Yhteismatriisi

M-tiedostot:
http://lipas.uwasa.fi/~jako/vision/Tekstuuri_mean.m
http://lipas.uwasa.fi/~jako/vision/Tekstuuri_std.m
http://lipas.uwasa.fi/~jako/vision/Tekstuuri_entropia.m
http://lipas.uwasa.fi/~jako/vision/Yhteismatriisi_piirteet.m
http://lipas.uwasa.fi/~jako/vision/Yhteismatriisi.m (skripti).