Exercise 1 – Edge Detection

Scale-space theory

Differential operators

Task 1

$$\begin{cases} \tilde{L}_{vv} = & L_v^2 L_{vv} = & L_x^2 L_{xx} + 2L_x L_y L_{xy} + L_y^2 L_{yy} = 0, \\ \tilde{L}_{vvv} = & L_v^3 L_{vvv} = & L_x^3 L_{xxx} + 3L_x^2 L_y L_{xxy} + 3L_x L_y^2 L_{xyy} + L_y^3 L_{yyy} < 0. \end{cases}$$

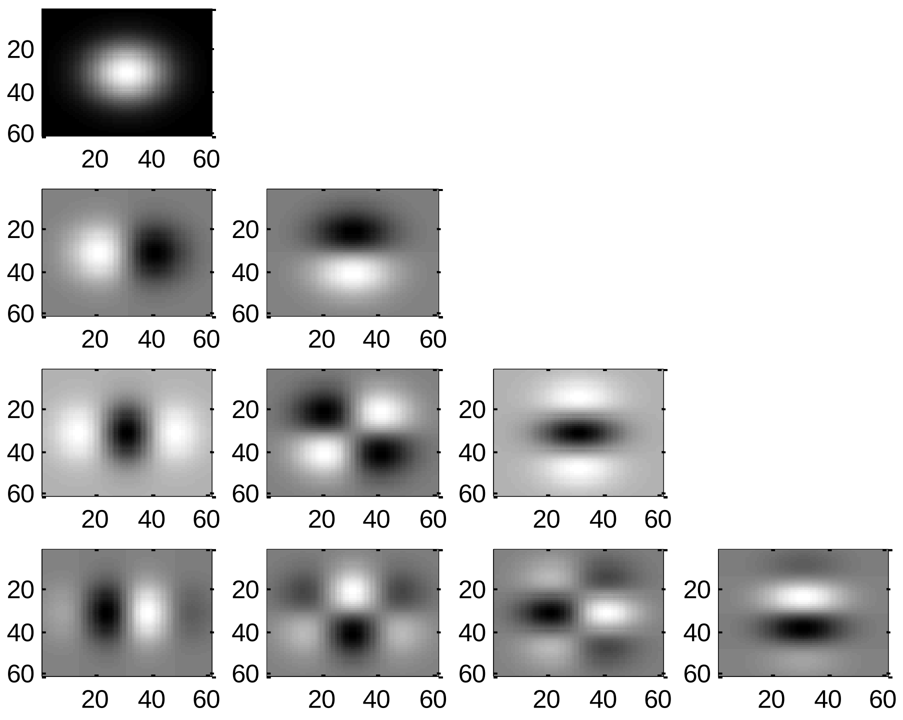
Computations

For $G(x,t) = \frac{e^{-\frac{x^2}{2t}}}{\sqrt{2\pi t}}$ and $L(x,y,t) = G(x,t)G(y,t)$, let:

$$F_n(x,t) = \frac{\frac{\partial^n G(x,t)}{\partial x^n}}{G(x,t)}$$

Then:

$$\frac{\partial^{n+m} L(x,y,t)}{\partial x^n \partial y^m} = F_n(x,t) F_m(y,t) G(x,t) G(y,t)$$



We precompute:

$$G(x,t)$$

$$F_1(x,t) = -\frac{x}{t}$$

$$F_2(x,t) = -\frac{1}{t} + \left(\frac{x}{t}\right)^2$$

$$F_3(x,t) = \frac{3x}{t^2} - \left(\frac{x}{t}\right)^3$$

Which is sufficient to compute the other derivatives.

Zero-crossings

```
LvvP=(Lvv>0); % Pixels with positive values
circshift(LvvP,[0,1]) % Pixels which left neighbors have positive values
circshift(LvvP,[1,0]) % Pixels which top neighbors have positive values
xor(LvvP,circshift(LvvP,[0,1])) % Pixels which have different signs than left
neighbours
xor(LvvP,circshift(LvvP,[1,0]))% Pixels which have different signs than top
neighbours
Lvv0=xor(LvvP,circshift(LvvP,[0,1])) | xor(LvvP,circshift(LvvP,[1,0]));
```

Filter radius

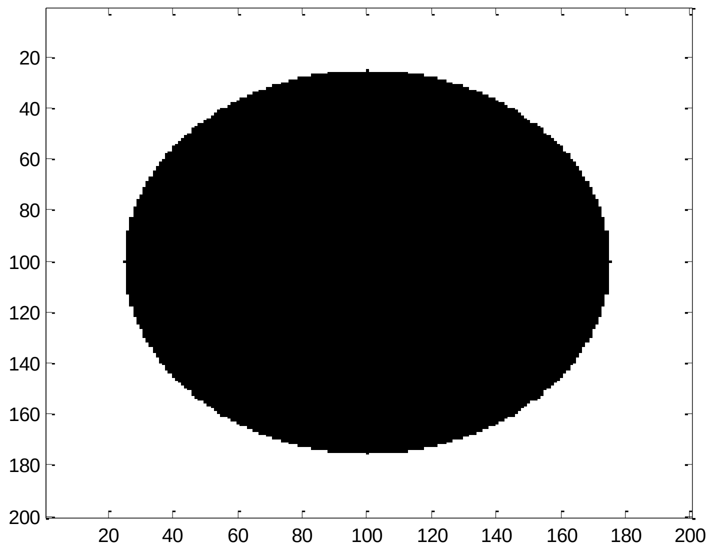
$$\frac{G\left(3\sqrt{t},t\right)}{G(0,t)} = e^{-\left(\frac{9}{2}\right)} \simeq 0.01$$

Code

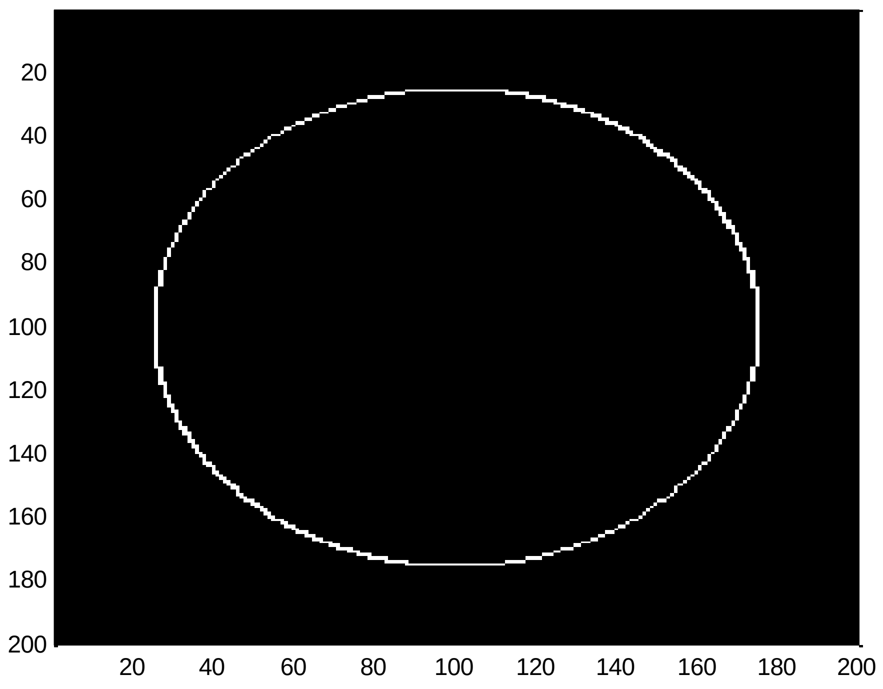
```
function edges = getEdges(im, t)

r = 3 * round(sqrt(t));
x = [-r:r];

uniDimGauss = exp(-(x.^2)/(2*t))/sqrt(2*pi*t);
uniDimGaussDiffx = -(x/t);
uniDimGaussDiffxx = ((-1/t) + (x/t).^2);
uniDimGaussDiffxxx = ((3*x/(t^2)) - ((x/t).^3));
uniDimGaussDiffy = uniDimGaussDiffx(1 + 2*r:-1:1)';
uniDimGaussDiffyy = uniDimGaussDiffxx(1 + 2*r:-1:1)';
uniDimGaussDiffyyy = uniDimGaussDiffxxx(1 + 2*r:-1:1)';

L = uniDimGauss(ones(1, 2 * r + 1),:) .* uniDimGauss(ones(1, 2 * r + 1),:)';

Lx = uniDimGaussDiffx(ones(1, 2 * r + 1),:) .* L;
LxConv = filter2(Lx, im);
Lxx = uniDimGaussDiffxx(ones(1, 2 * r + 1),:) .* L;
LxxConv = filter2(Lxx, im);
```

```
Lxxx = uniDimGaussDiffxxx(ones(1, 2 * r + 1),:) .* L;
LxxxConv = filter2(Lxxx, im);

Ly = uniDimGaussDiffy(:,ones(1, 2 * r + 1)) .* L;
LyConv = filter2(Ly, im);
Lyy = uniDimGaussDiffyy(:,ones(1, 2 * r + 1)) .* L;
LyyConv = filter2(Lyy, im);
Lyyy = uniDimGaussDiffyyy(:,ones(1, 2 * r + 1)) .* L;
LyyyConv = filter2(Lyyy, im);

Lxy = uniDimGaussDiffx(ones(1, 2 * r + 1),:) .* Ly;
LxyConv = filter2(Lxy, im);
Lxxy = uniDimGaussDiffy(:,ones(1, 2 * r + 1)) .* Lxx;
LxxyConv = filter2(Lxxy, im);
Lxyy = uniDimGaussDiffx(ones(1, 2 * r + 1),:) .* Lyy;
LxyyConv = filter2(Lxyy, im);


LvvConv = (LxConv.^2) .* LxxConv + 2 * LxConv .* LyConv .* LxyConv + (LyConv.^2)
.* LyyConv;
LvvvConv = (LxConv.^3) .* LxxxConv + 3 * (LxConv.^2) .* LyConv .* LxxyConv + 3 *
(LyConv.^2) .* LxConv .* LxyyConv + (LyConv.^3) .* LyyyConv;


LvvP=(LvvConv>0);
Lvv0=xor(LvvP,circshift(LvvP,[0,1])) | xor(LvvP,circshift(LvvP,[1,0]));
edges=Lvv0 & (LvvvConv < 0);

figure
colormap('gray');

imagesc(edges);

end
```

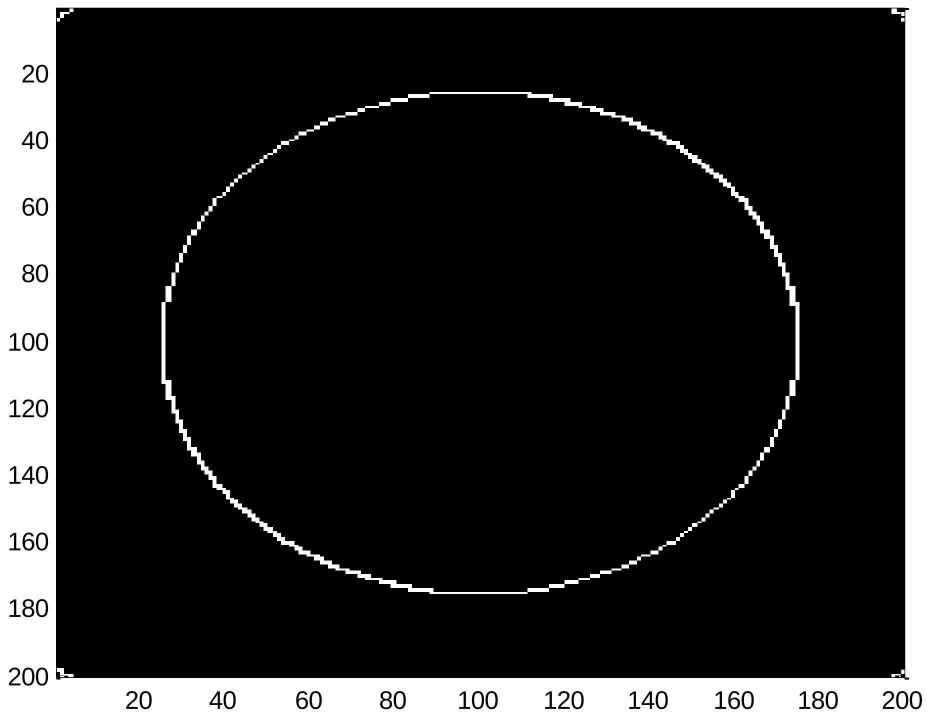
Task 2



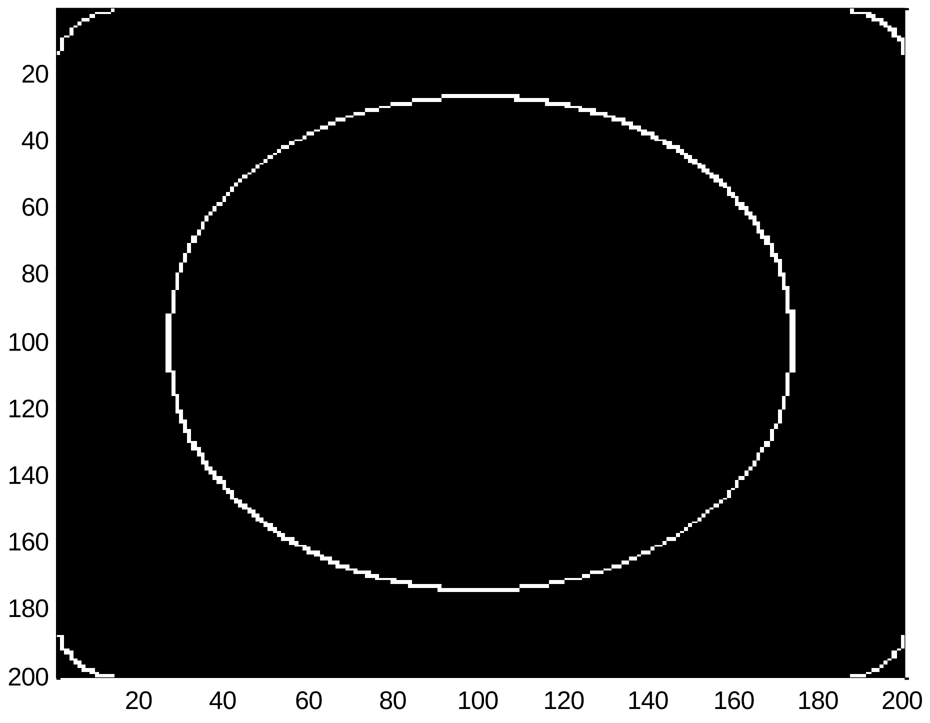
$t = 1$


20
40
60
80
100
120
140
160
180
200
20
40
60
80
100
120
140
160
180
200

$t = 10$

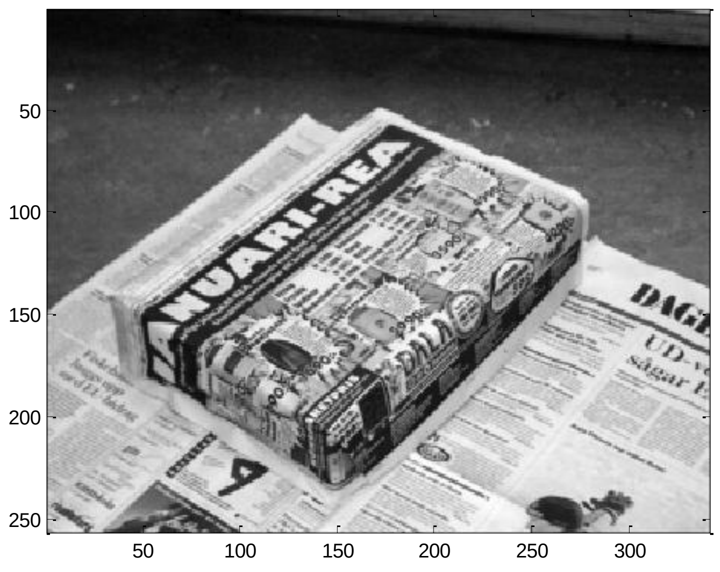

20
40
60
80
100
120
140
160
180
200
20
40
60
80
100
120
140
160
180
200

$t = 100$

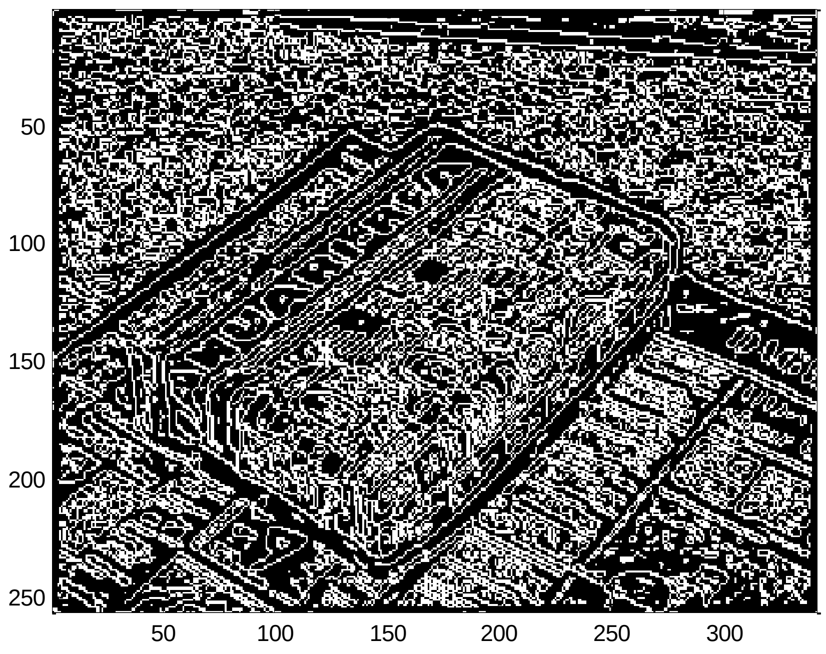


At large scale, the white area is also an edge.

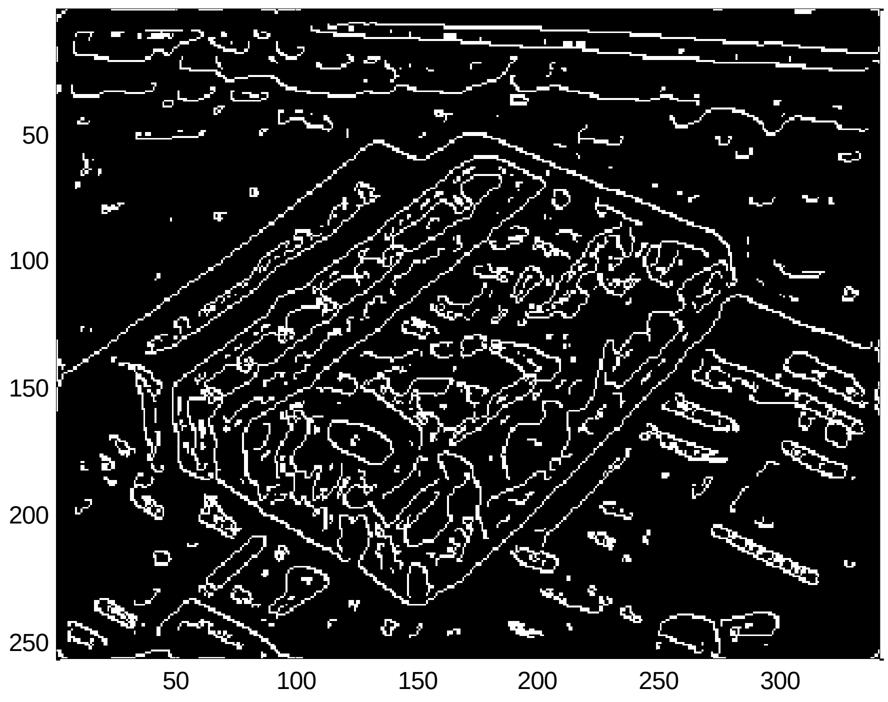
Taks 3



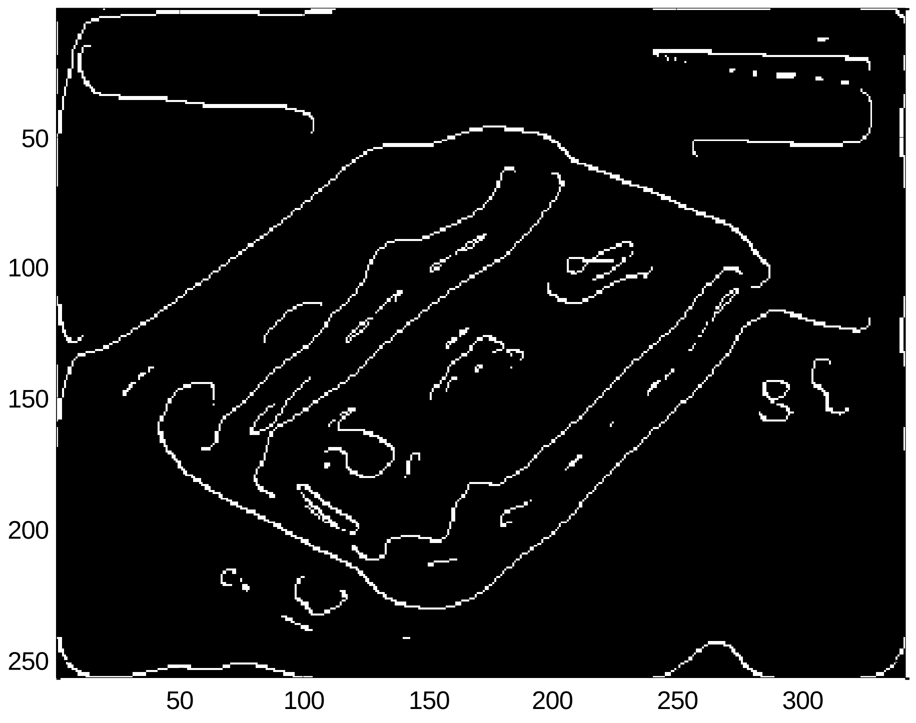
$t = 1$


50
100
150
200
250
50
100
150
200
250
300

$t = 10$


50
100
150
200
250
50
100
150
200
250
300

$t = 100$



Exercise 2 – Snakes

Active Contour Models

Parametric contour.

External force: potential derived.

Internal force: constraint to stretching and bending.

Formula

$$P(x,y) = -w_e \left|\nabla[G_\sigma(x,y) * I(x,y)]\right|^2$$

$$\begin{aligned}
\gamma \frac{\boldsymbol{X}_i^n - \boldsymbol{X}_i^{n-1}}{\Delta t} \quad = \quad & \frac{1}{h^2}[\alpha_{i+1}(\boldsymbol{X}_{i+1}^n - \boldsymbol{X}_i^n) - \alpha_i(\boldsymbol{X}_i^n - \boldsymbol{X}_{i-1}^n)] \\
& -\frac{1}{h^4}[\beta_{i-1}(\boldsymbol{X}_{i-2}^n - 2\boldsymbol{X}_{i-1}^n + \boldsymbol{X}_i^n) \\
& -2\beta_i(\boldsymbol{X}_{i-1}^n - 2\boldsymbol{X}_i^n + \boldsymbol{X}_{i+1}^n) \\
& +\beta_{i+1}(\boldsymbol{X}_i^n - 2\boldsymbol{X}_{i+1}^n + \boldsymbol{X}_{i+2}^n)] + \boldsymbol{F}_{\text{ext}}(\boldsymbol{X}_i^{n-1})
\end{aligned}$$

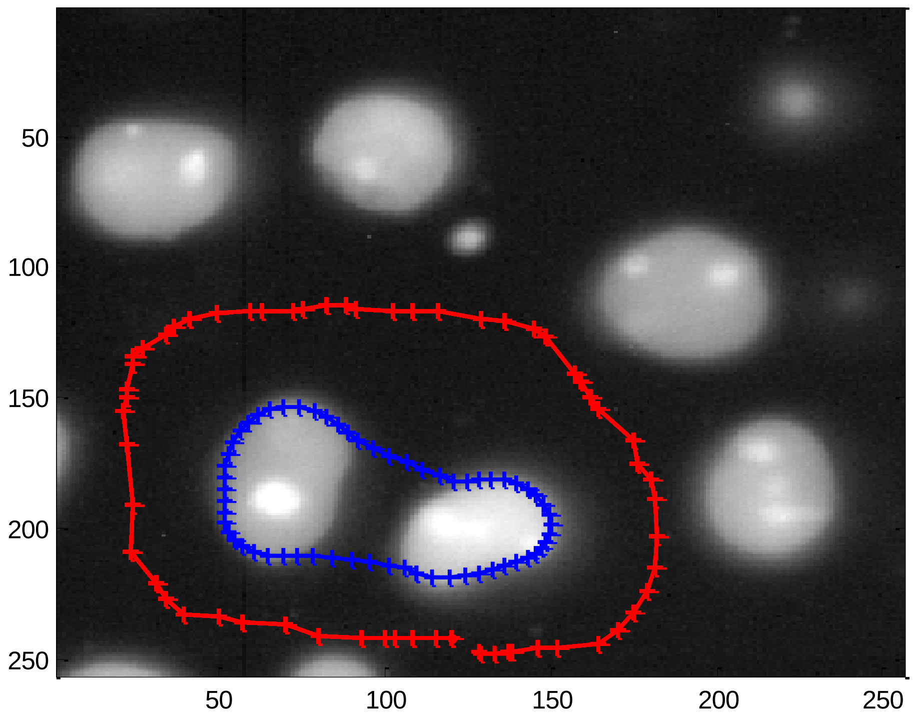
$$\boldsymbol{X}^n = (\boldsymbol{I} - \tau \boldsymbol{A})^{-1}[\boldsymbol{X}^{n-1} + \tau \boldsymbol{F}_{\text{ext}}(\boldsymbol{X}^{n-1})]$$

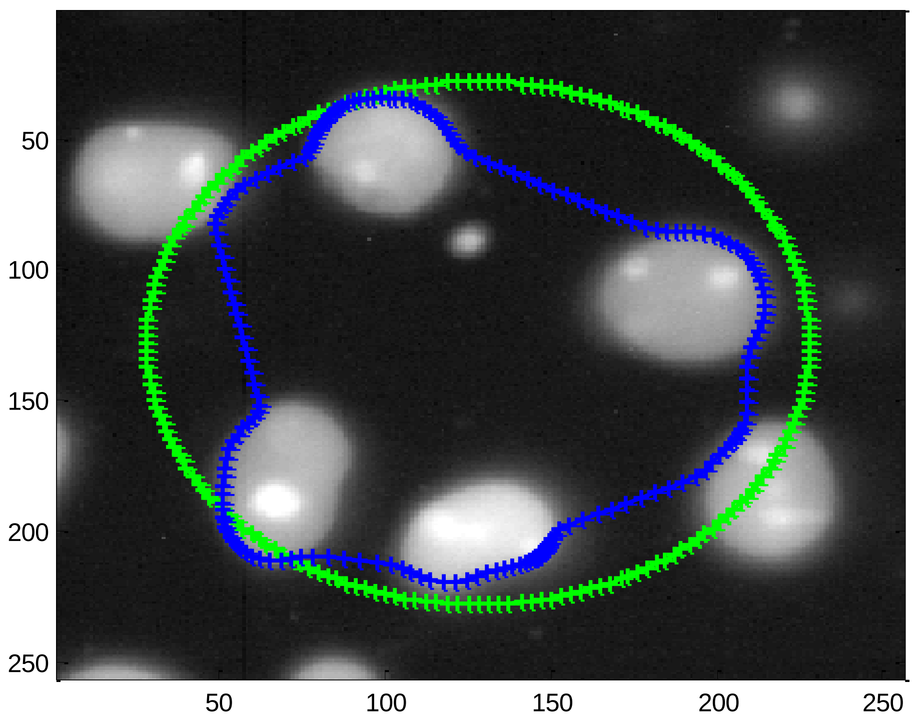
Code

```
function snake = evolveSnake(im, snake, weighting, deviation, alpha, beta)

n = size(snake, 1);
[H, W] = size(im);
gauss = fspecial('gaussian', 3 * round(deviation),deviation);
gaussConv = filter2(gauss, im);
[Px, Py] = gradient(gaussConv);
P = -weighting * (Px.^2 + Py.^2);
[Fx, Fy] = gradient(P);
Fx = -Fx;
Fy = -Fy;

d0 = -2 * alpha - 6 * beta;
d1 = alpha + 4 * beta;
d2 = -beta;


A = diag(d1 * ones(n - 1, 1), 1) + diag(d2 * ones(n - 2, 1), 2);
```

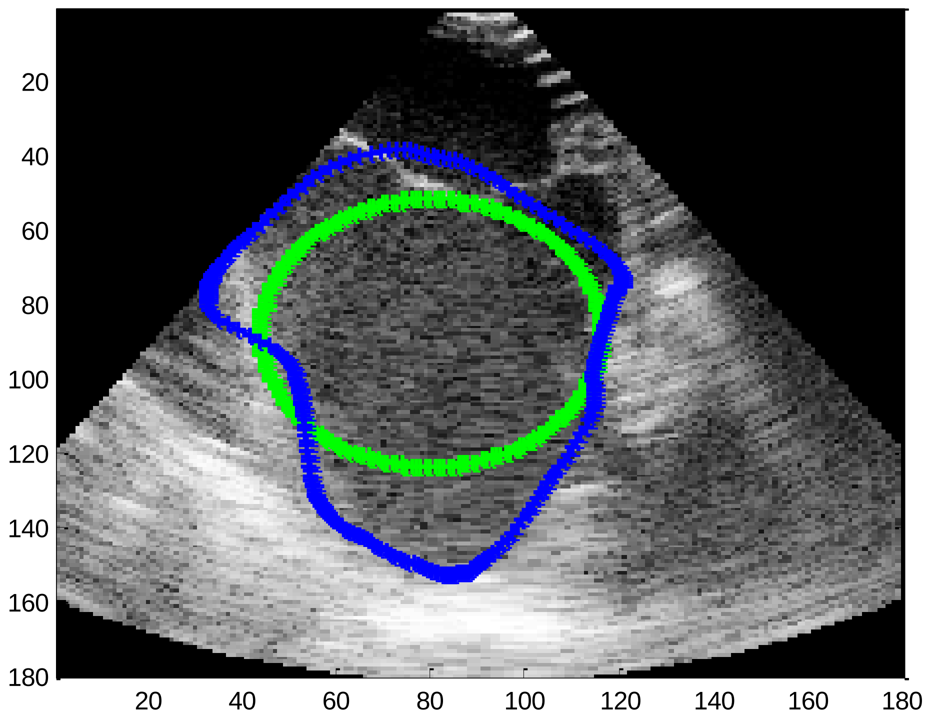
```
A(1,n) = d1;
A(1,n-1) = d2;
A(2,n) = d2;
A = A + A' + diag(d0 * ones(n, 1));
M = sparse(eye(n) - A);
Minv = inv(M);

isConverged = false;
i = 0;
while not(isConverged)
    XY1d=round(snake(:,2))+size(Fx,1)*(round(snake(:,1))-1);
    Fext=[Fx(XY1d) Fy(XY1d)];

    snake = Minv * (snake + Fext);
    snake = [max(min(snake(:,1), W), 1), max(min(snake(:,2), H), 1)];
    if i == 10000
        isConverged = true;
    end
    i = i + 1;
end

end
```

Tests





50
100
150
200
250
50
100
150
200
250



20
40
60
80
100
120
140
160
180
20
40
60
80
100
120
140
160
180

Exercise 3 - Statistical Shape Modelling

Formulas

We use the complex representation of the points.

Result 3.2 (Kent, 1994) *The* **full Procrustes mean shape** $[\hat{\mu}]$ *can be found as the eigenvector corresponding to the largest eigenvalue of the* **complex sum of squares and products matrix**

$$S = \sum_{i=1}^{n} w_i w_i^* / (w_i^* w_i) = \sum_{i=1}^{n} z_i z_i^*, \qquad (3.9)$$

where the $z_i = w_i / \|w_i\|$, $i = 1, \ldots, n$, *are the* **pre-shapes**.

We first center and scale the shapes (as compex vectors); this gives the so-called pre-shapes. Using formula 3.9, we get the full procrustes mean shape.

The **full Procrustes fits** or **full Procrustes coordinates** of $w_1, \ldots, w_n$ are

$$w_i^P = w_i^* \hat{\mu} w_i / (w_i^* w_i), \; i = 1, \ldots, n, \qquad (3.12)$$

Formula 3.12 gives the full procrustes fits of the shapes to the mean shape.

A principal component analysis is performed on these shapes. Using the fact that an eigen value of the covariance matrix of the series of shapes is equal to the varaince along the corresponding eigen direction, we determine that 6 modes are necessary to account for 95% of the global variance.

Code

Function annotate

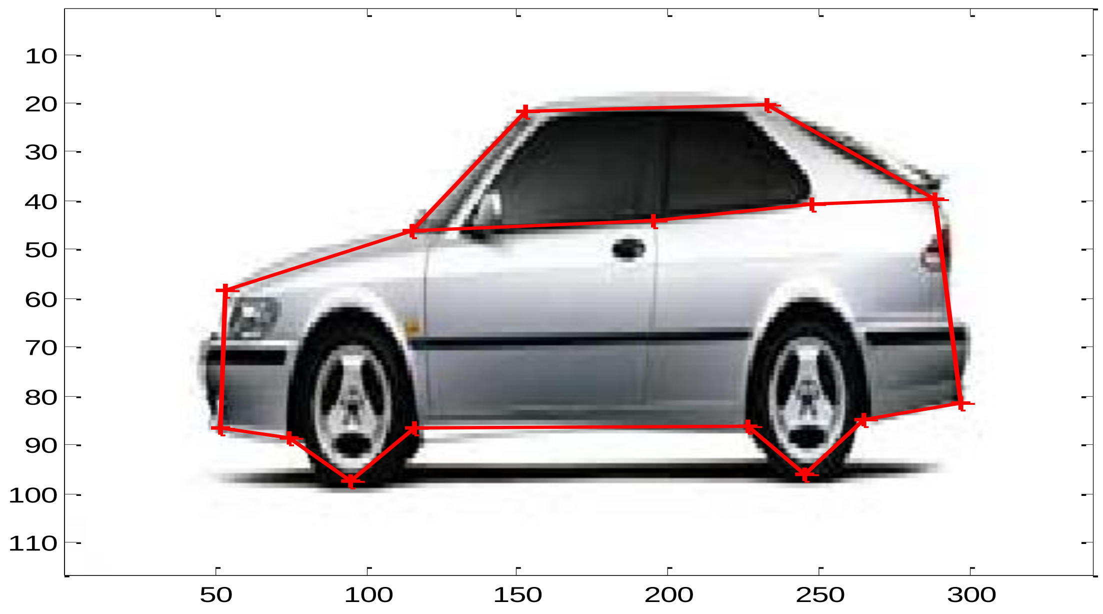
```
function v = annotate(im)

figure
colormap('gray');
imagesc(im)
u = MarkShape();
v = u(:,1)+i*u(:,2);

end
```

Main program

```
load cars;
v = [];

%% Annotations

s = whos('-file', 'cars');
for k=1:length(s)
```

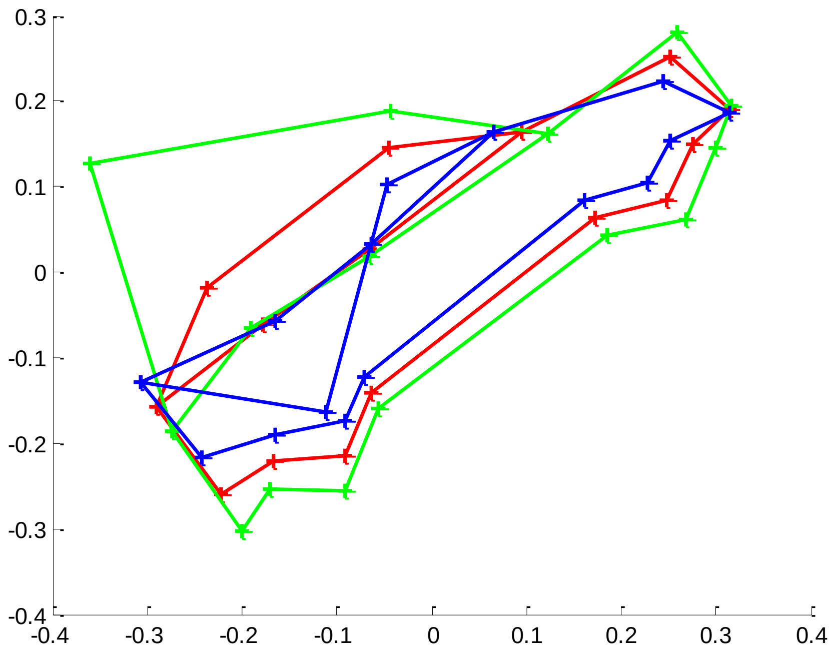
```matlab
    eval(sprintf('v = [v, annotate(%s)];', s(k).name));
end

%% Centering + scaling

centers = sum(v, 1) / size(v, 1);
vc = v - centers(ones(1, size(v, 1)), :);
ns = sqrt(diag((conj(vc)') * vc)');
vs = vc ./ ns(ones(1, size(v, 1)), :);

%% GPA

S = vs * (conj(vs)');
[V, D] = eigs(S);
m = V(:, 1);
va = (ones(size(v, 1), 1) * ((conj(vs)') * m)') .* vs;

%% PCA

C = cov(va');
[V, D] = eig(C);
[d, i] = sort(diag(D)', 'descend');
V = V(:, i);

s = d * triu(ones(size(D))) / sum(d);
n = find((s >= 0.95), 1);

%% Visualization

for k=1:3
    figure
    drawCarShape(m);
    drawCarShape(m - 3 * sqrt(d(k)) * V(:, k), 2) ;
    drawCarShape(m + 3 * sqrt(d(k)) * V(:, k), 3) ;
end
```

Annotations

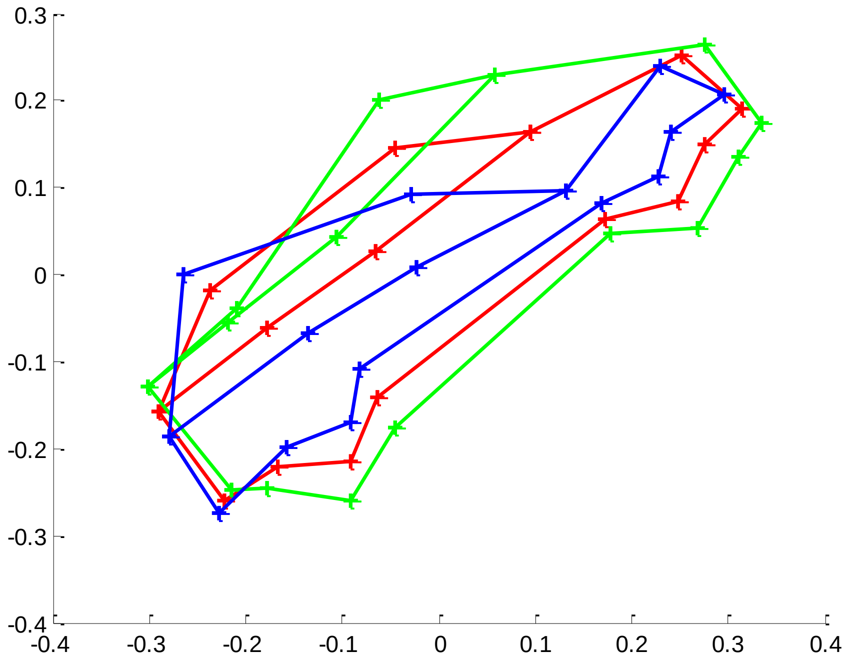


First three modes

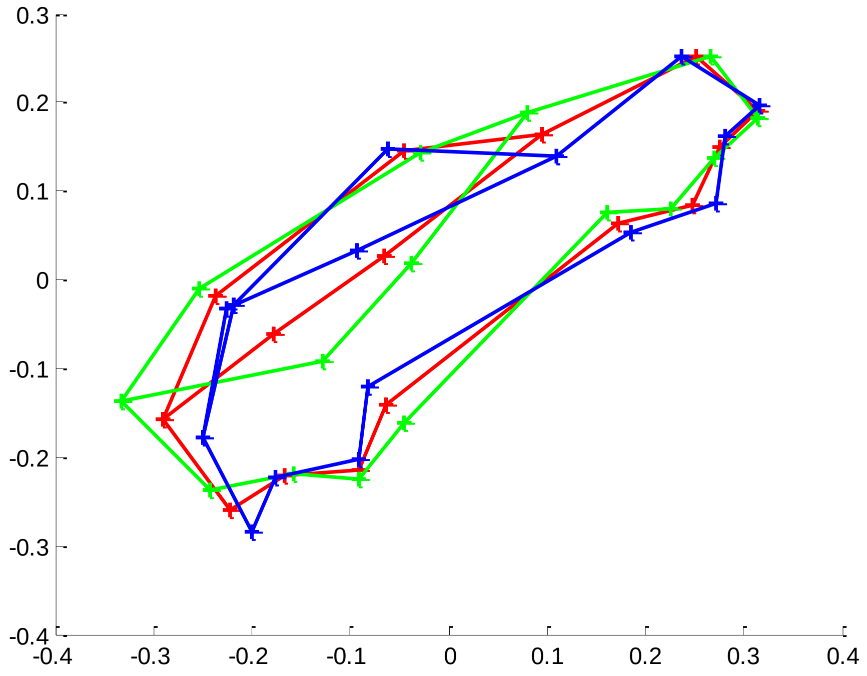
To represent a mode we plot the mean shape along with this shape plus or minus 3 standard deviations (which is the square root of the eigen value).



Mode 1 seems to represent the height of the roof.



Mode 2 seems to represent whether the front windshield is high and the back windshield low and vice versa.



Mode 3 seems to represent whether the windows are a big part of the car silhouette.

Exercise 4 - Graph Cuts for Markov Random Fields

Formulas

According to the following formula,

$$P\left(f_{ij}=K|f_{kl}\in\mathcal{N}_{ij},y_{ij}\right)=c(T)\exp\left(\frac{-U(y_{ij}|f_{ij}=K)+\beta n_{ij}(K)}{T}\right)$$

the one-clique energy associated to a site (I,j) and a label K is equal to $U(y_{ij}|f_{ij}=K)$ and the two-clique energy is $-\beta$ if the other site present to the clique is laberled K and 0 otherwise.

An initial distribution for the alpha-expantion algorithm can be obtained by finding the label corresponding to the minimum cost for each pixel.

Code

Main program

```
load brain
muCSF = 32;
muGM = 47;
sdCSF = 5;
sdGM = 4;
muWM = 55;
sdWM = 3;

[H, W] = size(brain);
v = reshape(brain, H*W, 1);

cost = [(log(sdCSF^2) + ((v - muCSF).^2) / (sdCSF^2)) / 2, (log(sdGM^2) + ((v -
muGM).^2) / (sdGM^2)) / 2, (log(sdWM^2) + ((v - muWM).^2) / (sdWM^2)) / 2];

%% 1-clique
sourceCost = max(0, cost(:,2) - cost(:,1));
sinkCost = max(0, cost(:,1) - cost(:,2));

TerminalWeights = [(1:H*W)', sourceCost, sinkCost];

EdgeWeights=[1, 2, 0, 0];

[Cut,Flow]=GraphCutMex(H*W,TerminalWeights,EdgeWeights);

result = ones(H*W, 1);
result(Cut', :) = 0;
result = reshape(result, H, W);
figure
colormap('gray')
imagesc(result)
```

```
%% 2-clique

beta = 3;

b = cost(:,2);
a = cost(:,1);

sourceCost = max(0, b - a);
sinkCost = max(0, a - b);

indices = (1:H*W)';

% Vertical cliques
xV = indices(mod(indices, H) ~= 0);
yV = xV + 1;

% Horizontal cliques
xH = indices((indices + H) <= W*H);
yH = xH + H;

% Solution
x = [xV; xH];
y = [yV; yH];

a = -beta * ones(size(x));
b = zeros(size(x));
c = zeros(size(x));
d = -beta * ones(size(x));

sinkCost(y) = sinkCost(y) + c - d;
sourceCost(x) = sourceCost(x) + c - a;
directCost = b + c - a - d;
reverseCost = zeros(size(x));

TerminalWeights = [indices, sourceCost, sinkCost];
EdgeWeights=[x, y, directCost, reverseCost];
[Cut,Flow]=GraphCutMex(H*W,TerminalWeights,EdgeWeights);

result = ones(H*W, 1);
result(Cut', :) = 0;
result = reshape(result, H, W);
figure
colormap('gray')
imagesc(result)

%% Alpha expansion

beta = 3;

indices = (1:H*W)';

% Vertical cliques
xV = indices(mod(indices, H) ~= 0);
yV = xV + 1;

% Horizontal cliques
```

```
xH = indices((indices + H) <= W*H);
yH = xH + H;

% Solution
x = [xV; xH];
y = [yV; yH];

result = init(cost);
for i=1:10
    for k=1:size(cost, 2)
        result = alphaexp(cost, k, result, x, y, beta);
    end
end

result = reshape(result, H, W);
figure
colormap('gray')
imagesc(result)
```

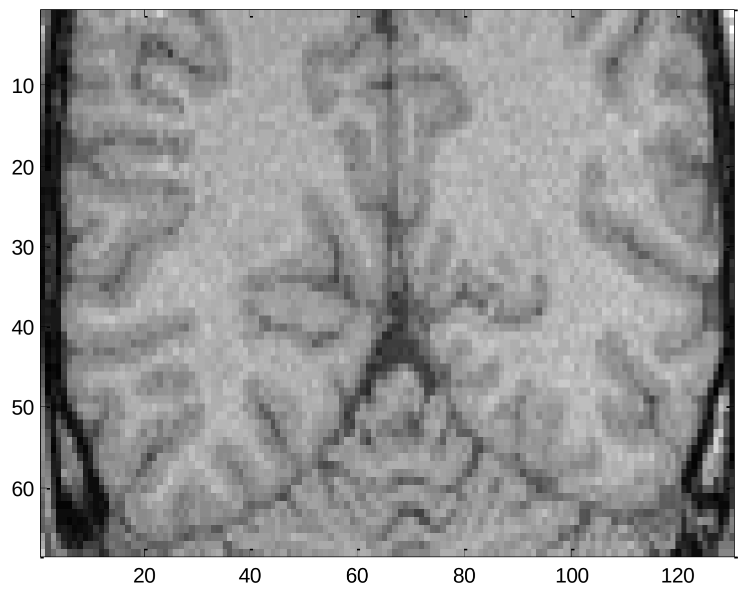
Function alphaexp

```
function result = alphaexp(cost, k, current, x, y, beta)

a = zeros(size(cost, 1),1);
for i=1:size(cost, 1),
    a(i,1) = cost(i, current(i,1));
end
a((current == k)) = inf;
b = cost(:, k);
b((current == k)) = 0;

sourceCost = max(0, b - a);
sinkCost = max(0, a - b);

a = -beta * (current(x) == current(y));
b = -beta * (current(x) == k);
c = -beta * (current(y) == k);
d = -beta * ones(size(x));

sinkCost(y) = sinkCost(y) + c - d;
sourceCost(x) = sourceCost(x) + c - a;
directCost = b + c - a - d;
reverseCost = zeros(size(x));

indices = (1:size(cost, 1))';

TerminalWeights = [indices, sourceCost, sinkCost];
EdgeWeights=[x, y, directCost, reverseCost];
[Cut,Flow]=GraphCutMex(size(cost, 1),TerminalWeights,EdgeWeights);

result = k * ones(size(current));
result(Cut) = current(Cut);
```

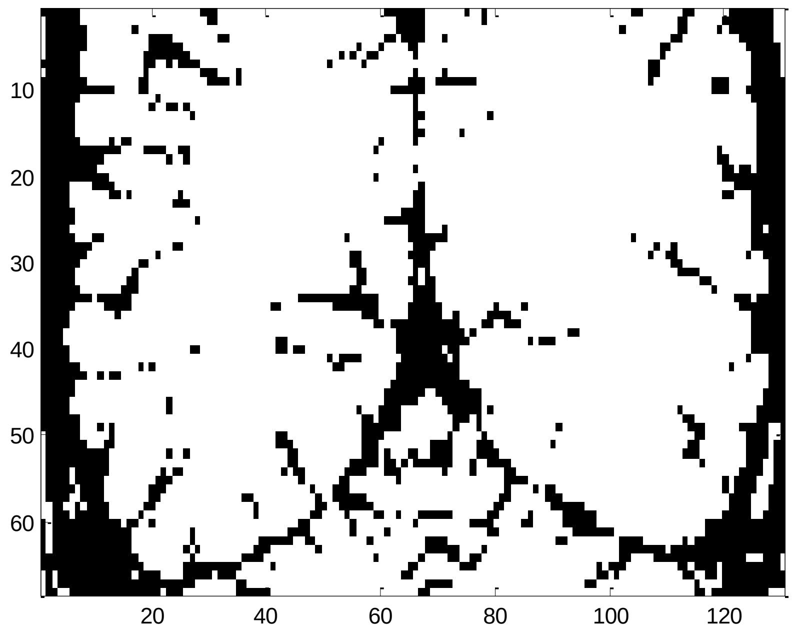
```
end
```

Results

Input image

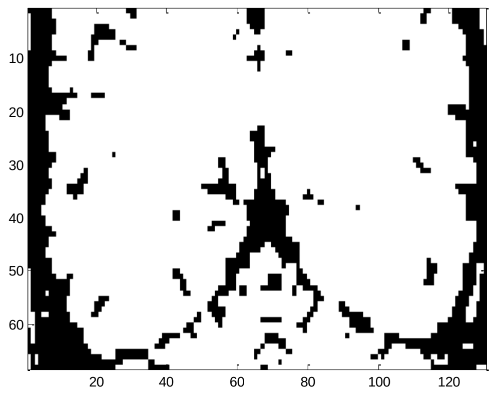


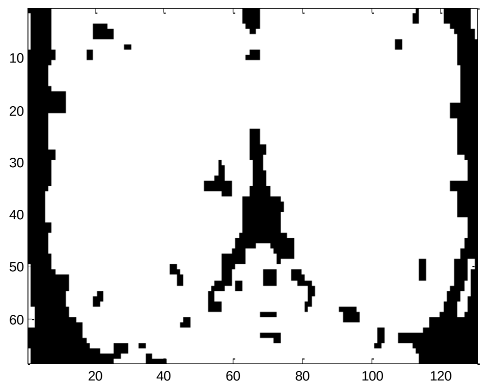
1-clique MRF

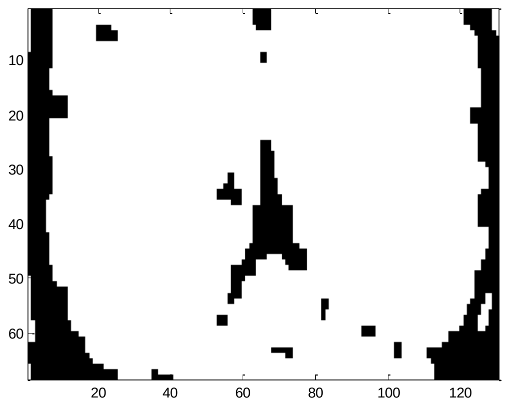


Although coherent, the result is affected by the input noise.

2-clique MRF with β set to 1, 3 and 5.

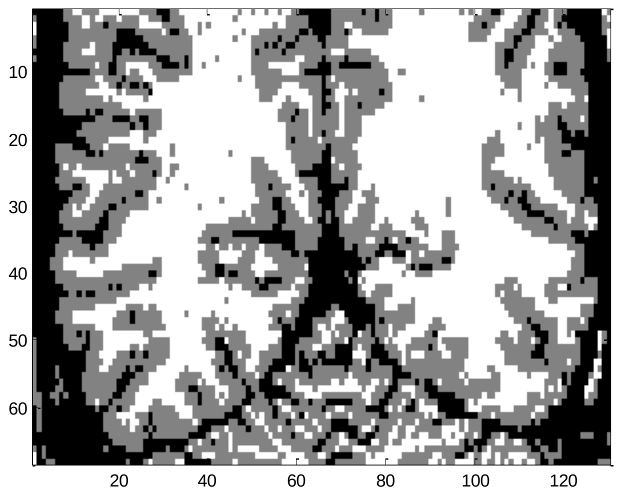






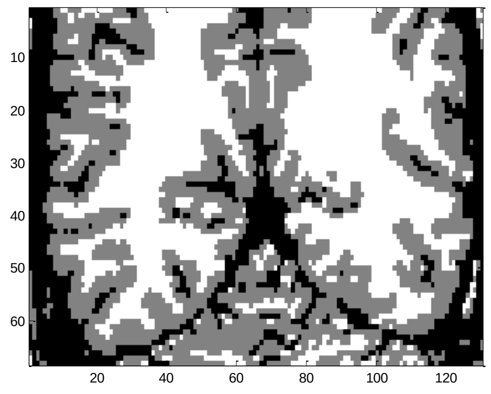
The second choice seems to be an appropriate amount of smoothing that doesn't remove to much relevant information.

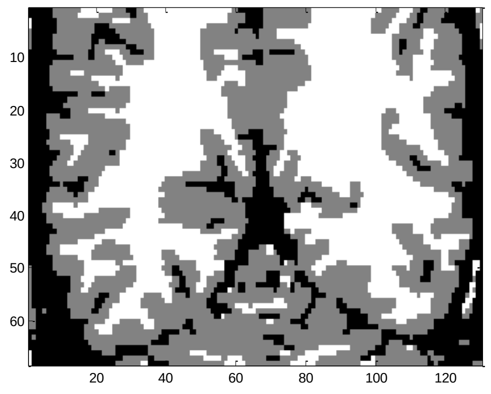
Minimum-cost based initial labeling for three labels

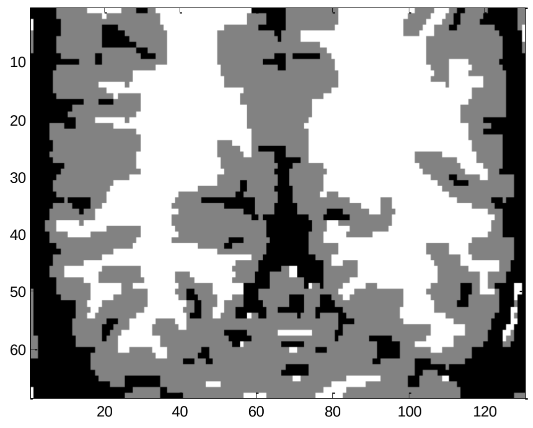


Similarly to the 1-clique MRF, the result is coherent but is by the input noise.

2-clique MRF with alpha-expansion with β set to 1, 3 and 5.







Once again, the second choice seems to be an appropriate amount of smoothing that doesn't remove to much relevant information.

The alpha-expansion algorithm seems to be converged after 10 expansions of each label.

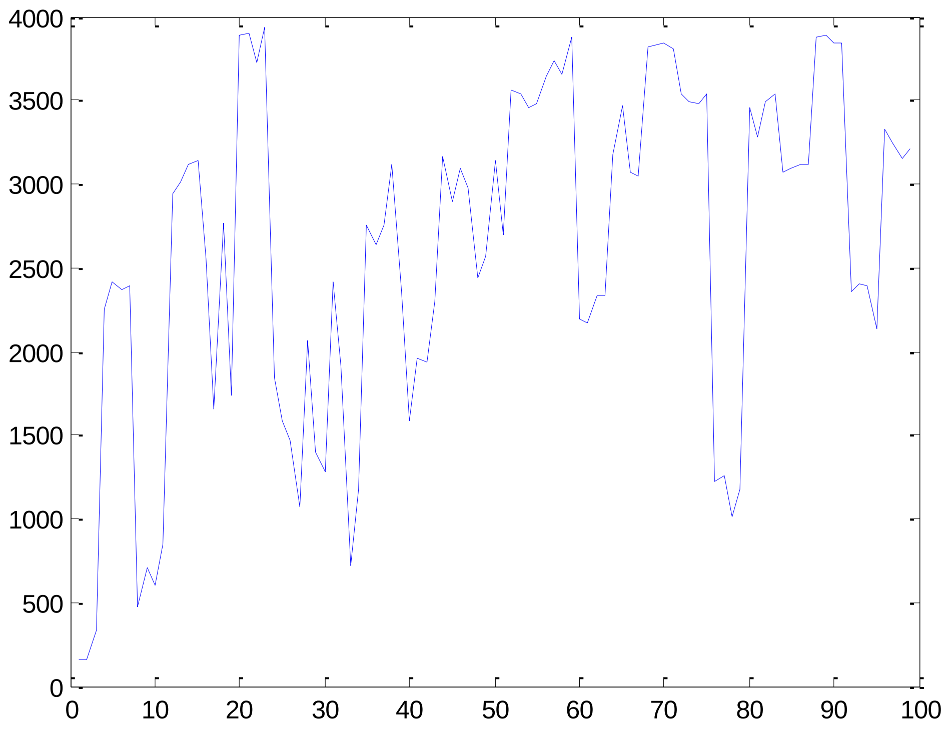
Exercise 5 – Object recognition

Code

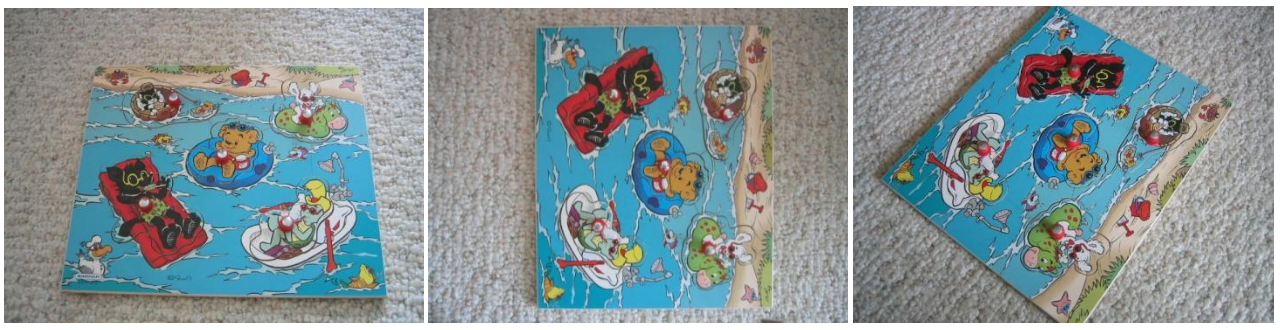
```
%% Load images

nIm = 100;
Images=[];
for cIm=0:(nIm-1),
    if(cIm<10)
        name=sprintf('ukbench0000%d.jpg',cIm);
    elseif(cIm<100)
        name=sprintf('ukbench000%d.jpg',cIm);
    end
    Images{cIm+1}=rgb2gray(imread(name));
end
%% Find keypoints
SIFTdescr=[];
for cIm=1:nIm,
    imwrite(Images{cIm},'HaaScript.pgm');
    [image, descrips, locs] = sift('HaaScript.pgm');
    SIFTdescr{cIm}=descrips;
end
%%
features = []
for cIm=1:nIm,
    features = [features; SIFTdescr{cIm}];
end

%% K-means
k=50;

[centers,mincenter,mindist,q2,quality] = kmeans2(features,k);

%% Image descriptors
imageDescriptors = [];
for cIm=1:nIm,
    imageDescriptors = [imageDescriptors; zeros(1, k)];
    for cSift=1:size(SIFTdescr{cIm}, 1),
        [m, argm] = min(sqrt(sum(((ones(k,1) * SIFTdescr{cIm}(cSift, :)) - 
centers).^2,  2)));
        imageDescriptors(cIm, argm) = imageDescriptors(cIm, argm) + 1;
    end
end

%% Test

testIm = 1;

X = ones(nIm - 1, 1) * imageDescriptors(testIm, :);
Y = imageDescriptors((1:nIm ~= testIm), :);
S = X+Y;
D = (X - Y).^2;
D((S == 0)) = 0;
S((S == 0)) = 1;
dist = sum(D ./ S, 2);
plot(dist)
```

Test

The distances between the first image descriptor and the other image descriptors are computed.

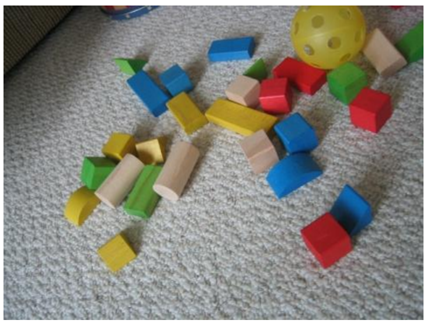
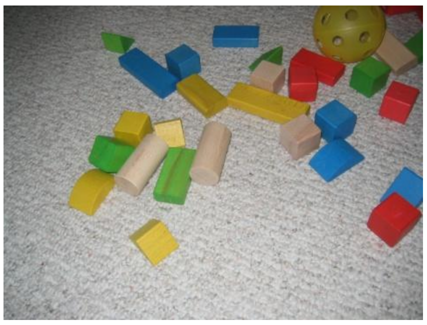
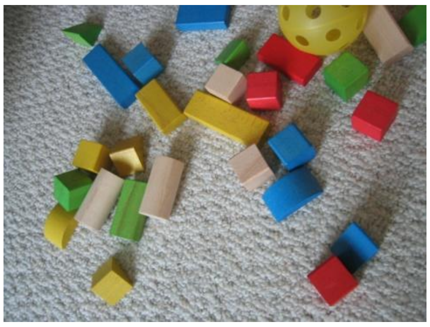
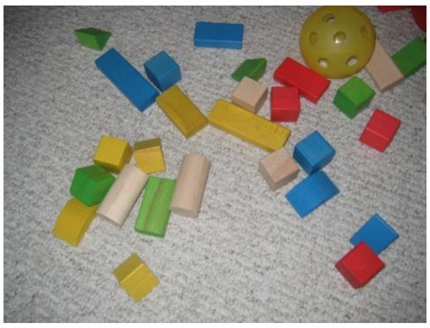


Images 1 to 3 show a very low distance.

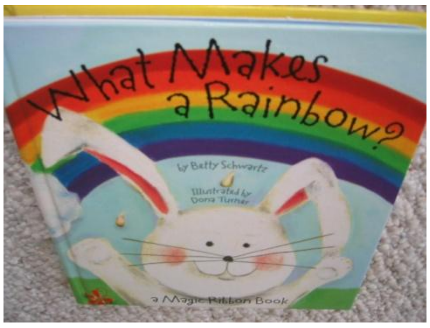


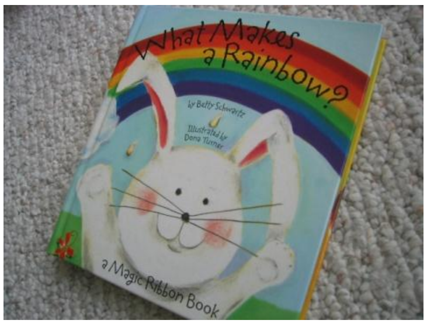
This is of course expected as these photos show the same scene as the test image from different viewpoints. This shows that SIFT features are indeed viewpoint independent.

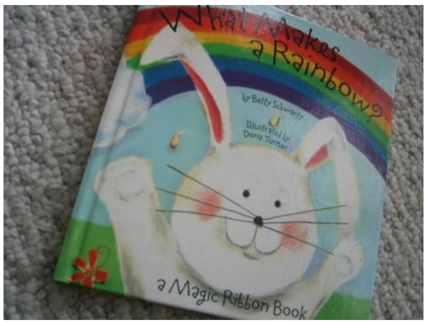
The same goes for images 8, to 11. However, we see there that this is because the same background is present in these photos as in the test photo; its particular texture must account for quite a few SIFT features.

Distances are relatively low for images 24 to 34. The reason is the same as before.







However, distances are relatively low for images 76 to 79 while they seem very different from the test image. This is all the more surprising that the observed low distance remain for the whole series of these photos. Thus it is not just a singularity. This might be that the small area of floor is recognized as similar to the floor in the test image although this is not really noticeable to the naked eye.

Exercise 6 – Multiple View Geometry

Fundamental Matrix Estimation

We use the 8-point algorithm to estimate the fundamental matrix F.

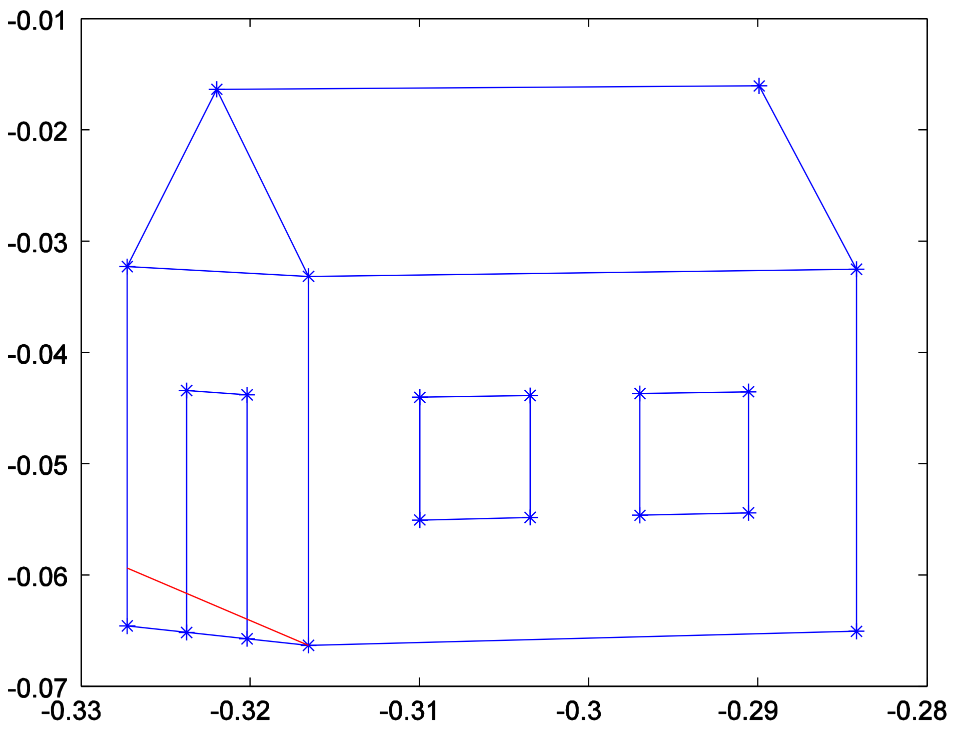
If (p_1, p_2) is a pair of corresponding points, $l_1 = Fp_1$ and $l_2 = F^T p_2$ are the epipolar lines passing through both points.

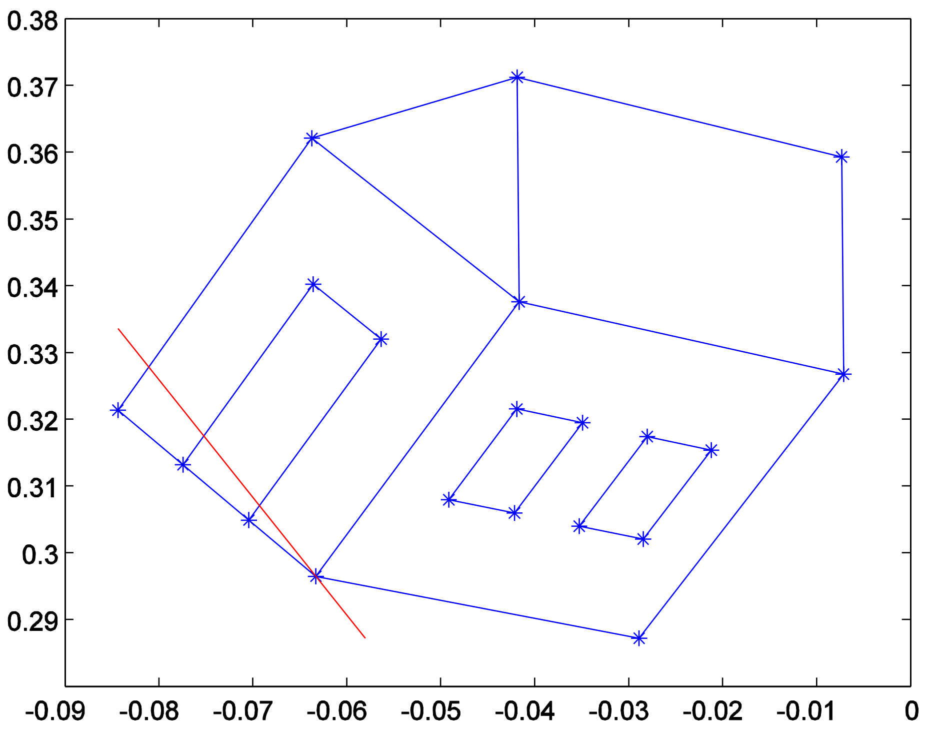
Code

```
nFrames=10; Draw=1;
[x,y,P,Lines]=Film(nFrames,Draw);

%% 8-point algo
```

```
cFrame1 = 1;
cFrame2 = 5;

B = [x(cFrame2,:)' .* x(cFrame1,:)', x(cFrame2,:)' .* y(cFrame1,:)',
x(cFrame2,:)', y(cFrame2,:)' .* x(cFrame1,:)', y(cFrame2,:)' .* y(cFrame1,:)',
y(cFrame2,:)', x(cFrame1,:)', y(cFrame1,:)', ones(size(x(cFrame2,:)'))];

[U,S,V] = svd(B,0);
F = reshape(V(:,9),3,3)';

l2 = F*[x(cFrame1,1); y(cFrame1,1); 1];
l1 = F'*[x(cFrame2,1); y(cFrame2,1); 1];


%% Draw epipolar line on first frame

drawline(l1, min(x(cFrame1,:)), max(x(cFrame1,:)), min(y(cFrame1,:)),
max(y(cFrame1,:)));

%% Draw epipolar line on second frame

drawline(l2, min(x(cFrame2,:)), max(x(cFrame2,:)), min(y(cFrame2,:)),
max(y(cFrame2,:)));
```

Test





RANSAC

At each iteration, the maximum number of iteration I is reevaluated following :

$$\epsilon = 1 - \frac{N_{inliers}}{N_{total}}$$

$$I = \frac{\log(1-p)}{\log(1-(1-\epsilon)^2)}$$

Where p is the amount of inliers expected to be found.

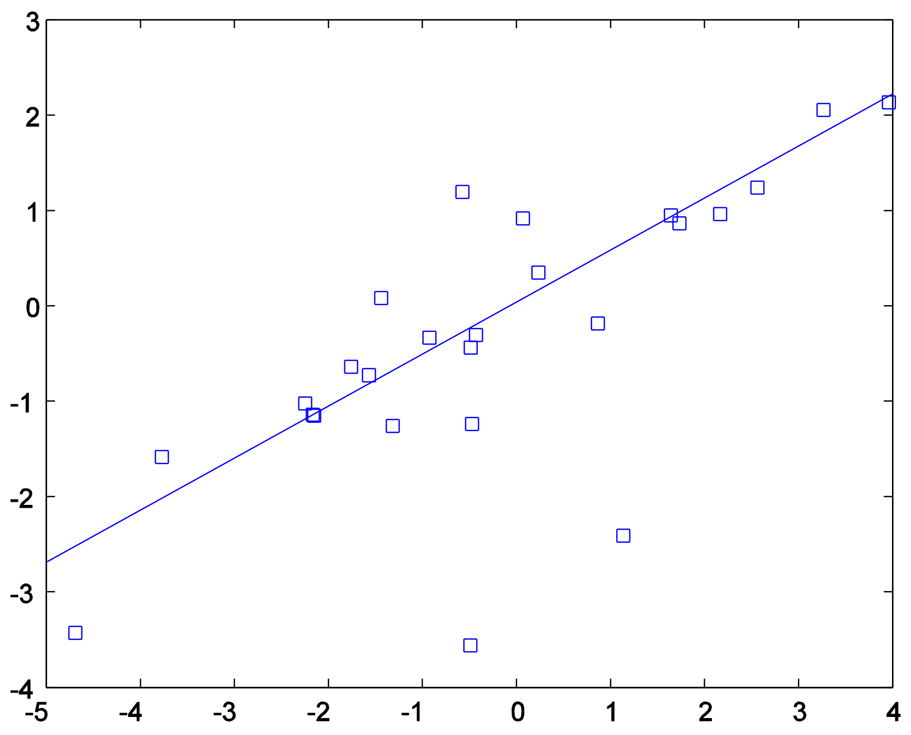
The final line is estimated using a linear regression from the inliers.

Code

```
%% Ransac
prob=0.99;
data = RanLine(10,15);
thres = 1;

data = [data; ones(1, size(data, 2))];
plot(data(1,:), data(2,:), 's')
hold on

converged = false;
maxScore=0;
best=[];
```

```
n = 0;
N = inf;
while N > n,
    perm = randperm(size(data, 2));
    m = ones(3);
    m(1:3,1:2) = data(:,perm(1:2));
    l = (m'\[0;0;1])';
    inliers = (abs(l*data)/norm(l) <= thres);
    if (sum(inliers) > maxScore),
        maxScore = sum(inliers);
        best = data(:, inliers);
    end

    eps = 1-(sum(inliers)/size(data, 2));
    N = log(1-prob)/log(1 - (1 - eps)^2);
    n = n+1;
end

l = polyfit(best(1,:), best(2,:), 1);
x=[floor(min(data(1,:))), ceil(max(data(1,:)))];
plot(x,polyval(l,x))
```

Test



3D Inference from a Plane

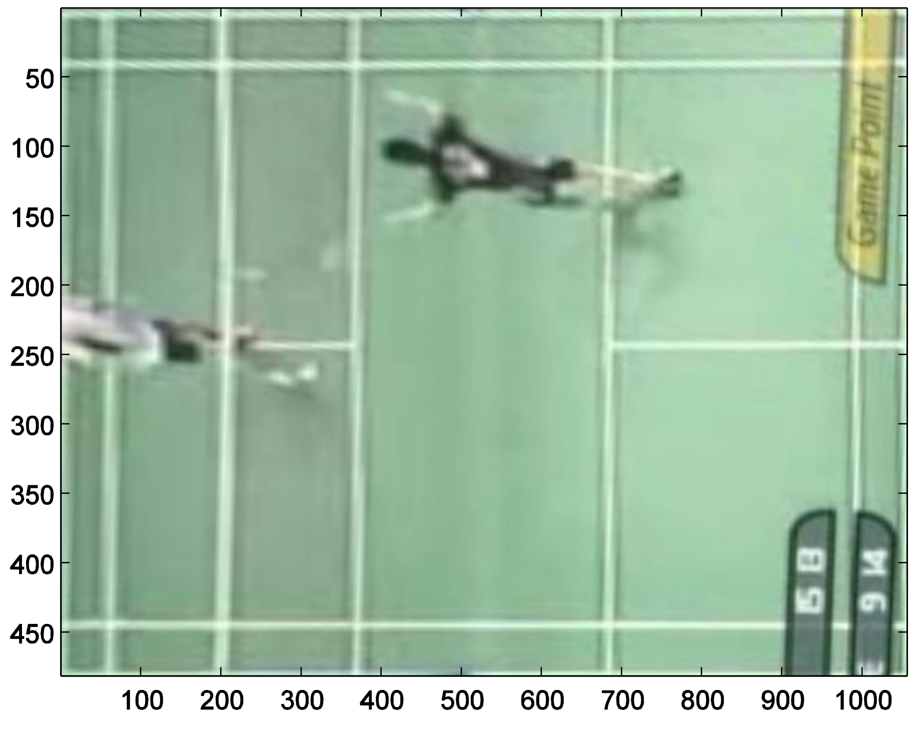
Code

```
%% 3d inference

image = imread('petergade.png');
imagesc(image)

Q = [0, 6.7+1.98+3.96+0.76, 0, 6.7+1.98+3.96+0.76;
    0, 0, 6.1, 6.1];

Q = [100 * Q; ones(1, size(Q,2))];


[Px,Py] = ginput(size(Q,2));
P = [Px'; Py'; ones(1, size(Q,2))];

%%

[Pn,Tp] = normalize(P);
[Qn,Tq] = normalize(Q);


M = [];
for i=1:size(P,2),
    M = [M; [zeros(1,3), -Qn(3,i)*(Pn(:,i)'), Qn(2,i)*(Pn(:,i)')];
[Qn(3,i)*(Pn(:,i)'), zeros(1,3), -Qn(1,i)*(Pn(:,i)')]; [-Qn(2,i)*(Pn(:,i)'),
Qn(1,i)*(Pn(:,i)'), zeros(1,3)]];
end

[U,S,V] = svd(M,0);
H = reshape(V(:,9),3,3);
H = Tp' * H * inv(Tq');

figure
Tr=maketform('projective', H);
WarpIm=imtransform(image,Tr,'YData',[0 Q(2,3)],'XData',[0 Q(1,2)]);
imagesc(WarpIm)

[x,y] = ginput(2);
```

Test

The furthest player is located at 3.34m from the back of the court and 3.17m from the right side; the other player is located 8.92m from the back and 1.75m from the right.

YONEX
EUROSPORT
WORLD CHAMPS ANAHEIM
LIN 15 8
GADE 9 14
Game Point


50
100
150
200
250
300
350
400
450
100
200
300
400
500
600
700
800
900
1000
Game Point

Exercise 7 - Change Detection via MAF

MAF Components

In order to normalize the data, we subtract each layer of each image by its mean and divide it by its variance.

The MAF components are supposed to fulfill the following criteria:

$$\text{i) } \operatorname{Corr}\{Y_i(\boldsymbol{x}), Y_j(\boldsymbol{x})\} = 0,\ i \neq j,$$

$$\text{ii) } \operatorname{Corr}\{Y_i(\boldsymbol{x}), Y_i(\boldsymbol{x} + \boldsymbol{\Delta})\} = 1 - \tfrac{1}{2}\kappa_i,$$

These can be checked numerically.

The first criterion implies that the MAF components are orthogonal regarding the correlation.

Code

```
%% Load
Im87=freadenvit('thika87');
Im89=freadenvit('thika89');
[m,n,p] = size(Im87);

%% Normalize

Im87n = reshape(Im87, n*m, p);
Im87n = Im87n - ones(m*n, 1) * mean(Im87n);
C = cov(Im87n);
Im87n = Im87n * inv(sqrt(diag(diag(C))));
Im87n = reshape(Im87n, m, n, p);

Im89n = reshape(Im89, n*m, p);
Im89n = Im89n - ones(m*n, 1) * mean(Im89n);
C = cov(Im89n);
Im89n = Im89n * inv(sqrt(diag(diag(C))));
Im89n = reshape(Im89n, m, n, p);

%% Differences

I = Im87n - Im89;
Ih=I(2:end,1:end-1,:);
Iv=I(1:end-1,2:end,:);
Id=(Ih+Iv)/2;
I = I(1:end-1,1:end-1,:);

C = cov(reshape(I, (m-1)*(n-1), p));
Cd = cov(reshape(I-Id, (m-1)*(n-1), p));

[V, D] = eig(Cd, C);
```

```
[v, i] = sort(diag(D), 'descend');
V = V(:,i);

Y = reshape(reshape(I, (m-1)*(n-1), p) * V, m-1, n-1, p);
Yh=Y(2:end,1:end-1,:);
Yv=Y(1:end-1,2:end,:);
Yd=(Yh+Yv)/2;
Y = Y(1:end-1,1:end-1,:);

%% Check criteria
corr(reshape(Y, (m-2)*(n-2), p)) % Must be identity matrix
diag(corr(reshape(Y, (m-2)*(n-2), p), reshape(Yd, (m-2)*(n-2), p))) - (-(v/2)+1)
% must be equal to zero
```

Test

The correlation matrix of the MAF components is evaluated to:

```
ans =

    1.0000   -0.0003   -0.0004
   -0.0003    1.0000    0.0002
   -0.0004    0.0002    1.0000
```

This is acceptable since it is close to the identity

The difference between both member of the second equality is evaluated to:

```
ans =

   -0.0121
   -0.0020
   -0.0007
```

This is also acceptable as sufficiently small values.

Change detection

We apply a threshold on the magnitude of the MAF component corresponding to the highest autocorrelation between neighboring pixels.

A second option is to use a MRF where the cost for the label “no change” is equal to the previously mentioned magnitude and the cost of the label “change” is complementary.

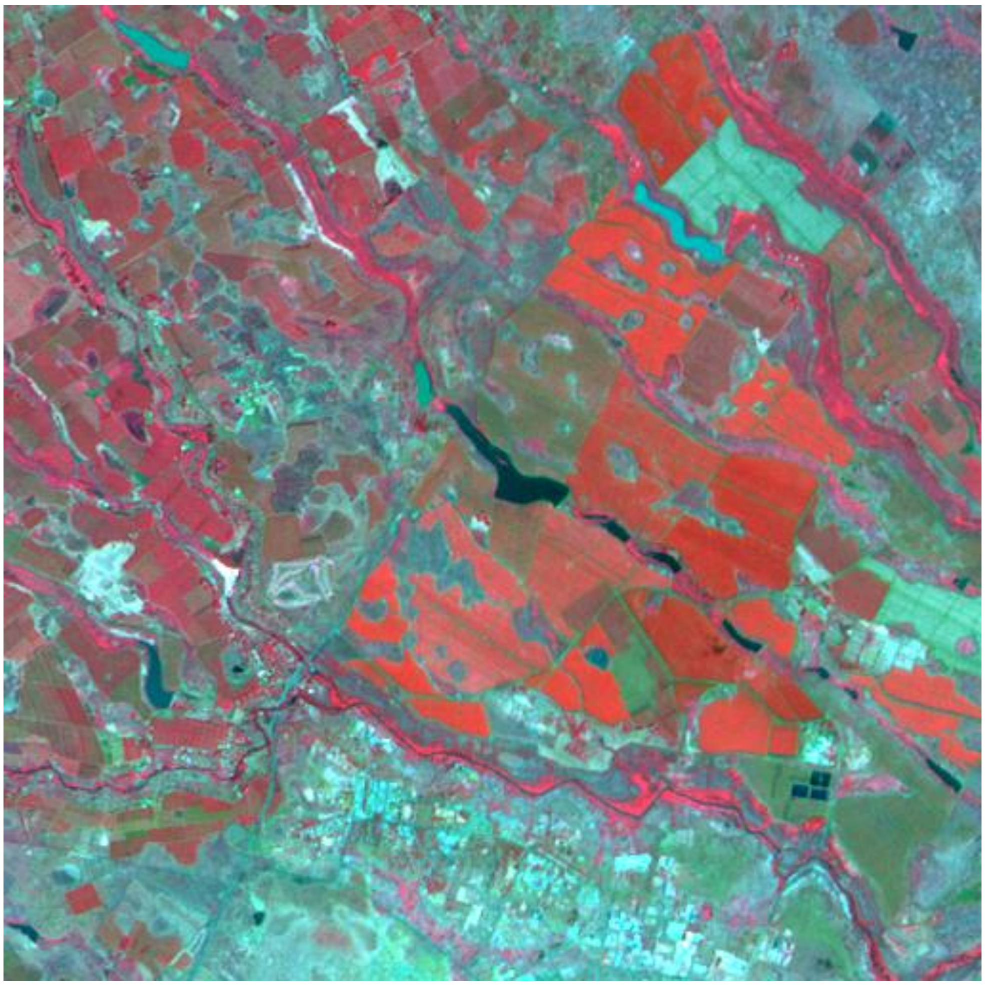
Code

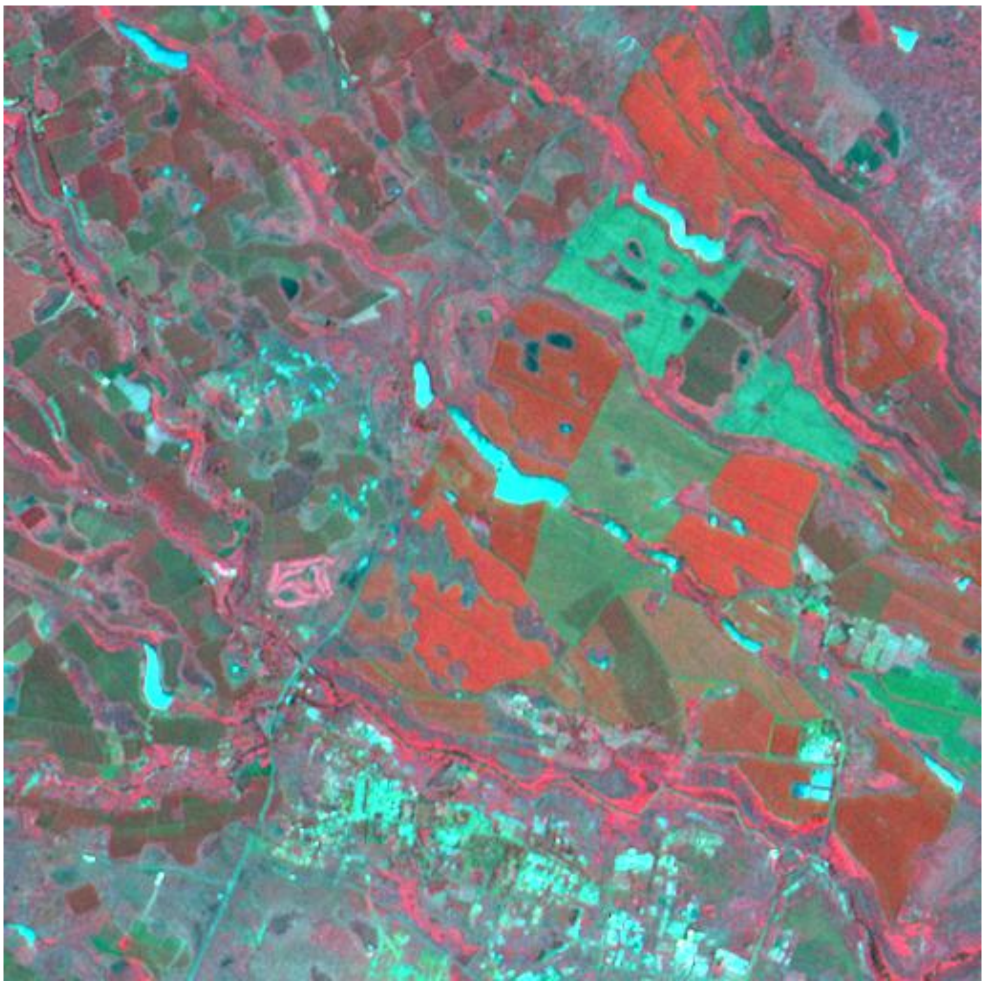
```
%% Threshold

thres = 0.5;

diff=(abs(Y(:,:,3)) / max(max(abs(Y(:,:,3)))) > thres);


%%
figure
```

```
imshowrgb(Im87, [3 2 1], 3);
figure
imshowrgb(Im89, [3 2 1], 3);
figure
colormap('gray');
imagesc(diff);

%% MRF

[H, W] = size(Y(:,:,3));
v = reshape(abs(Y(:,:,3)), H*W, 1);

cost(:,2) = max(v) - v;
cost(:,1) = v;

%%

beta = 3;

b = cost(:,2);
a = cost(:,1);

sourceCost = max(0, b - a);
sinkCost = max(0, a - b);

indices = (1:H*W)';

% Vertical cliques
xV = indices(mod(indices, H) ~= 0);
yV = xV + 1;

% Horizontal cliques
xH = indices((indices + H) <= W*H);
yH = xH + H;

% Solution
x = [xV; xH];
y = [yV; yH];

a = -beta * ones(size(x));
b = zeros(size(x));
c = zeros(size(x));
d = -beta * ones(size(x));

sinkCost(y) = sinkCost(y) + c - d;
sourceCost(x) = sourceCost(x) + c - a;
directCost = b + c - a - d;
reverseCost = zeros(size(x));

TerminalWeights = [indices, sourceCost, sinkCost];
EdgeWeights=[x, y, directCost, reverseCost];
```

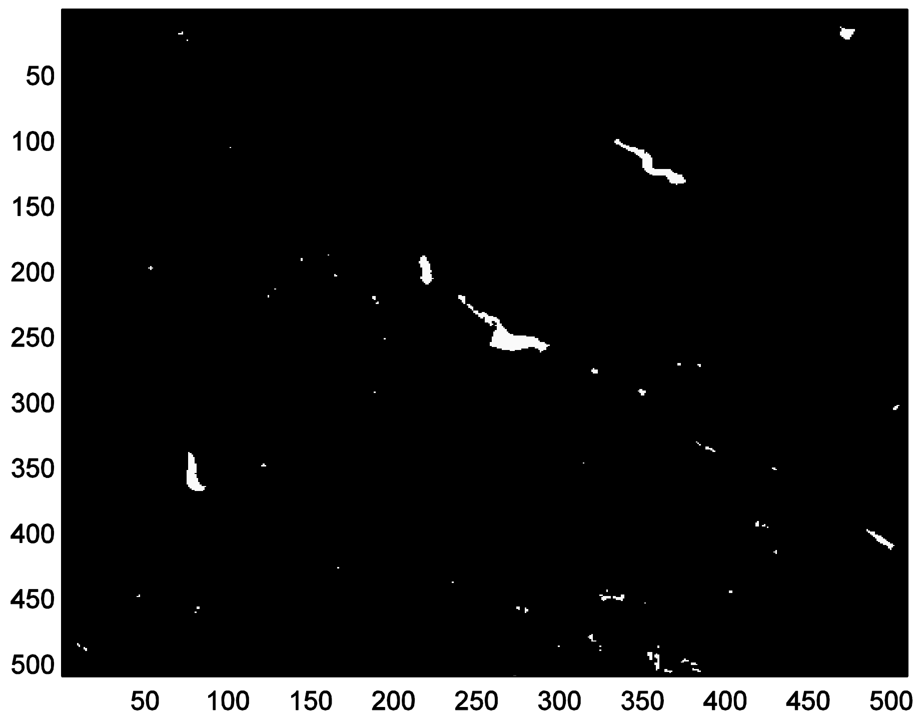
```
[Cut,Flow]=GraphCutMex(H*W,TerminalWeights,EdgeWeights);

result = ones(H*W, 1);
result(Cut', :) = 0;
result = reshape(result, H, W);


%%
figure
imshowrgb(Im87, [3 2 1], 3);
figure
imshowrgb(Im89, [3 2 1], 3);
figure
colormap('gray');
imagesc(result);
```

Tests





Result obtained with the threshold method:



Result obtained with the MRF.