Solution to Problem Set 4

Assigned: March 23, 2006
Due: April 17, 2006

Problem 1 *(6.882) Belief Propagation for Segmentation*

In this problem you are asked to do foreground/background segmentation using belief propagation (BP). You will implement MMSE message passing (BP) in Markov Network which has the probabilistic model

$$p(X,Y) = \frac{1}{Z}\prod_i \Phi(x_i, y_i) \prod_{(i,j)\in E} \Psi(x_i, x_j), \tag{1}$$

where Y is the observed color image, and $X = \{x_i\}, x_i \in \{1, 0\}$ is foreground(1)/background(0) labeling. $\Phi(\cdot,\cdot)$ and $\Psi(\cdot,\cdot)$ are compatibility functions. Z is a normalization constant to make $p(X,Y)$ a pdf. We shall instruct you step by step on how to set the compatibility functions and how to do belief propagation.

Load `flower.bmp`, partial labeling of foreground `cfgmask.bmp` and partial labeling of background `cbgmask.bmp`. In each label image the white pixels indicate a valid sample. From the labeling and the image estimate the mean and covariance of the foreground and background pixels (RGB), denoted as $\mu_f, \mathbf{\Sigma}_f$ and $\mu_b, \mathbf{\Sigma}_b$. We have the likelihood

$$p(y_i|x_i = 1) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_f|^{1/2}} \exp\{-\frac{1}{2}(y_i - \mu_f)^T \mathbf{\Sigma}_f^{-1}(y_i - \mu_f)\} + \varepsilon, \tag{2}$$

and

$$p(y_i|x_i = 0) = \frac{1}{(2\pi)^{3/2}|\mathbf{\Sigma}_b|^{1/2}} \exp\{-\frac{1}{2}(y_i - \mu_b)^T \mathbf{\Sigma}_b^{-1}(y_i - \mu_b)\} + \varepsilon. \tag{3}$$

The term ε is used to capture the error of the Gaussian distribution. Set $\varepsilon = 0.01$ in this problem. Compatibility function Φ is defined similar to

posterior

$$\begin{cases} \Phi(y_i, x_i = 1) = \dfrac{p(y_i|x_i = 1)}{p(y_i|x_i = 1) + p(y_i|x_i = 0)} \\ \Phi(y_i, x_i = 0) = \dfrac{p(y_i|x_i = 0)}{p(y_i|x_i = 1) + p(y_i|x_i = 0)} \end{cases} \tag{4}$$

Compatibility function Ψ is defined as a symmetric matrix

$$\begin{bmatrix} 0.9 & 0.1 \\ 0.1 & 0.9 \end{bmatrix}. \tag{5}$$

In other words, the two neighboring nodes are very likely to have the same labeling.

In this homework we ask you to implement MMSE estimate for $\hat{x}_i$ by marginalizing over the other variables in the pdf. The message update rule is (for message from node i to j)

$$m_{ij}(x_j) \leftarrow c \sum_{x_i} \Psi(x_j, x_i)\Phi(x_i, y_i) \prod_{k \in N(i) \setminus j} m_{ki}(x_i). \tag{6}$$

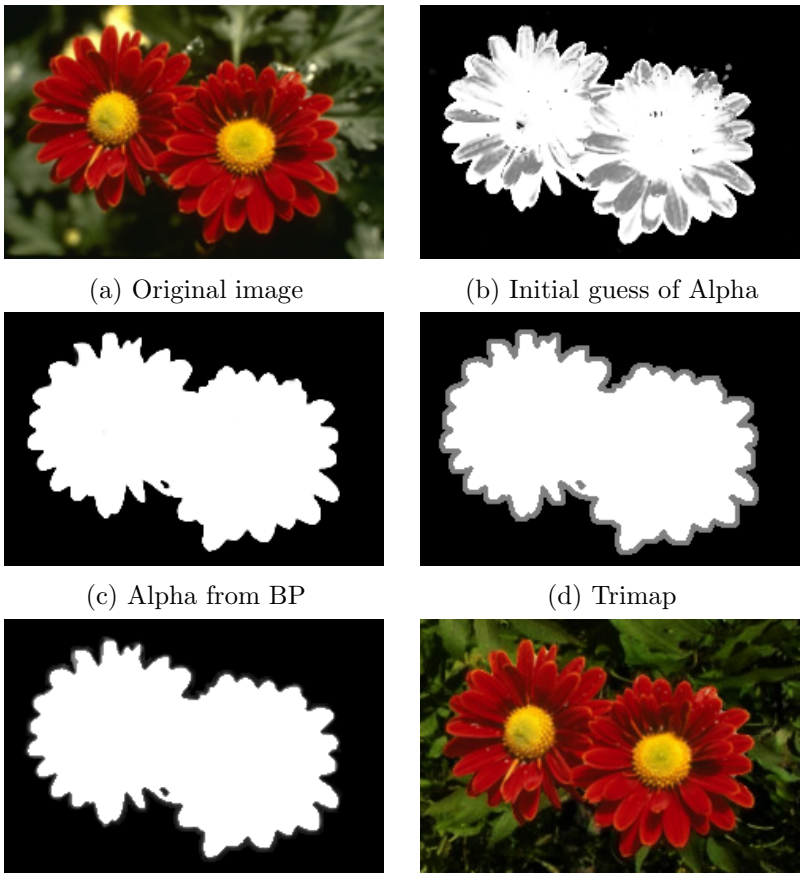
In the message update rule for MAP estimate $\sum_{x_i}$ is changed to $\max_{x_i}$ (not required for this problem). The belief, or the approximated marginal distribution is

$$b(x_i) = c\ \Phi(x_i, y_i) \prod_{j \in N(i)} m_{ji}(x_i), \tag{7}$$

which can be computed at the end of the message updating.

We assume that there are in general four neighbors for each pixel except for the boundaries. So there are four directional messages, M_{tb}, M_{bt}, M_{lr} and M_{rl}. You need to store them separately. In each step please allocate buffers for new messages M'_{tb}, M'_{bt}, M'_{lr} and M'_{rl} so that message passing is similar to filtering. In each step of iteration you can write a loop (slow!!) over each pixel for these messages, or write everything in matrix multiplication (faster!!). New message should also be normalized. Instead of directly updating message as $M_{tb} = M'_{tb}$, please use $M_{tb} = \alpha M'_{tb} + (1 - \alpha)M_{tb}$ to avoid flipping states. You may choose $\alpha = 0.7$.

(a) Implement MMSE estimate for X. Plot the mask of foreground pixels. Paste the foreground to `leaves.bmp` using the estimated alpha map from belief.



(a) Original image (b) Initial guess of Alpha

(c) Alpha from BP (d) Trimap

(e) Refined alpha matte (f) Pasting to a new background

Figure 1: Using belief propagation to segment foreground from background. Alpha matting is used for pasting.

(b) (Extra credit) To paste the foreground flower in a better way, you can use the alpha matting code we developed for problem set 2 so that the boundary pixels can be correctly handled. You may use `imerode` and `imdilate` functions to get a trimap.

Solution. See the following code.

```
im=imread('flower.bmp');
im=im2double(im);
```

```
%-------------------------------------------------------------------
% load mask of foreground and background color samples
%-------------------------------------------------------------------
cfgmask=imread('cfgmask.bmp');
if size(cfgmask,3)~=1
    cfgmask=rgb2gray(cfgmask);
end
cfgmask=im2double(cfgmask);
cfgindex=find(cfgmask>0);

cbgmask=imread('cbgmask.bmp');
if size(cbgmask,3)~=1
    cbgmask=rgb2gray(cbgmask);
end
cbgmask=im2double(cbgmask);
cbgindex=find(cbgmask>0);

[height,width,n]=size(im);
X=reshape(im,[height*width,n]);

cfgSample=X(cfgindex,:);
cbgSample=X(cbgindex,:);

%-------------------------------------------------------------------------
% compute the mean and covariance of the foreground and background samples
%-------------------------------------------------------------------------
cfgMean=mean(cfgSample);
cbgMean=mean(cbgSample);
cfginvCov=inv(cov(cfgSample));
cbginvCov=inv(cov(cbgSample));

%-------------------------------------------------------------------------
% compatibility from RGB color
%-------------------------------------------------------------------------
Y=X-repmat(cfgMean,[height*width,1]);
Z=Y*cfginvCov;
imprbfg=(2*pi)^(-1.5)*sqrt(det(cfginvCov))*exp(-sum(Z.*Y,2)/2);
imprbfg=reshape(imprbfg,[height,width])+1E-2;

Y=X-repmat(cbgMean,[height*width,1]);
Z=Y*cbginvCov;
imprbbg=(2*pi)^(-1.5)*sqrt(det(cbginvCov))*exp(-sum(Z.*Y,2)/2);
imprbbg=reshape(imprbbg,[height,width])+1E-2;

mask=imprbfg./(imprbfg+imprbbg);
```

```
C0(1,:,:)=mask;
C0(2,:,:)=1-mask;
figure;imshow(mask);
drawnow;

%--------------------------------------------------------------------------
% compatibility of connectivity
%--------------------------------------------------------------------------
E_h(1,:,:)=0.9*ones(height,width);
E_h(2,:,:)=1-E_h(1,:,:);
E_v=E_h;

%--------------------------------------------------------------------------
% Belief propagation
%--------------------------------------------------------------------------

% Initialize messages
Mtb=ones(2,height-1,width)/2;
Mbt=ones(2,height-1,width)/2;
Mlr=ones(2,height,width-1)/2;
Mrl=ones(2,height,width-1)/2;

Mtb1=Mtb;
Mbt1=Mbt;
Mlr1=Mlr;
Mrl1=Mrl;

nIterations=30;
alpha=0.7;
% message update
for k=1:nIterations
    disp(['Iteration ' num2str(k)]);
    % top to bottom
    foo=C0(:,1:end-1,:);
    foo(:,2:end,:)=foo(:,2:end,:).*Mtb(:,1:end-1,:);
    foo(:,:,2:end)=foo(:,:,2:end).*Mlr(:,1:end-1,:);
    foo(:,:,1:end-1)=foo(:,:,1:end-1).*Mrl(:,1:end-1,:);
    Mtb1(1,:,:)=E_v(1,1:end-1,:).*foo(1,:,:)+E_v(2,1:end-1,:).*foo(2,:,:);
    Mtb1(2,:,:)=E_v(2,1:end-1,:).*foo(1,:,:)+E_v(1,1:end-1,:).*foo(2,:,:);

    % bottom to top
    foo=C0(:,2:end,:);
    foo(:,1:end-1,:)=foo(:,1:end-1,:).*Mbt(:,2:end,:);
    foo(:,:,2:end)=foo(:,:,2:end).*Mlr(:,2:end,:);
    foo(:,:,1:end-1)=foo(:,:,1:end-1).*Mrl(:,2:end,:);
```

```
    Mbt1(1,:,:)=E_v(1,1:end-1,:).*foo(1,:,:)+E_v(2,1:end-1,:).*foo(2,:,:);
    Mbt1(2,:,:)=E_v(2,1:end-1,:).*foo(1,:,:)+E_v(1,1:end-1,:).*foo(2,:,:);

    % left to right
    foo=C0(:,:,1:end-1);
    foo(:,:,2:end)=foo(:,:,2:end).*Mlr(:,:,1:end-1);
    foo(:,2:end,:)=foo(:,2:end,:).*Mtb(:,:,1:end-1);
    foo(:,1:end-1,:)=foo(:,1:end-1,:).*Mbt(:,:,1:end-1);
    Mlr1(1,:,:)=E_h(1,:,1:end-1).*foo(1,:,:)+E_h(2,:,1:end-1).*foo(2,:,:);
    Mlr1(2,:,:)=E_h(2,:,1:end-1).*foo(1,:,:)+E_h(1,:,1:end-1).*foo(2,:,:);

    % right to left
    foo=C0(:,:,2:end);
    foo(:,:,1:end-1)=foo(:,:,1:end-1).*Mrl(:,:,2:end);
    foo(:,2:end,:)=foo(:,2:end,:).*Mtb(:,:,2:end);
    foo(:,1:end-1,:)=foo(:,1:end-1,:).*Mbt(:,:,2:end);
    Mrl(1,:,:)=E_h(1,:,1:end-1).*foo(1,:,:)+E_h(2,:,1:end-1).*foo(2,:,:);
    Mrl(2,:,:)=E_h(2,:,1:end-1).*foo(1,:,:)+E_h(1,:,1:end-1).*foo(2,:,:);

    % normalize new messages
    foo=repmat(sum(Mtb1,1),[2,1]);
    Mtb1=Mtb1./foo;
    foo=repmat(sum(Mbt1,1),[2,1]);
    Mbt1=Mbt1./foo;
    foo=repmat(sum(Mlr1,1),[2,1]);
    Mlr1=Mlr1./foo;
    foo=repmat(sum(Mrl1,1),[2,1]);
    Mrl1=Mrl1./foo;

    % update messsage
    Mtb=Mtb1*alpha+Mtb*(1-alpha);
    Mbt=Mbt1*alpha+Mbt*(1-alpha);
    Mlr=Mlr1*alpha+Mlr*(1-alpha);
    Mrl=Mrl1*alpha+Mrl*(1-alpha);
end

%-------------------------------------------------------------------------
% compute beliefs
%-------------------------------------------------------------------------
B=C0;
B(:,2:end,:)=B(:,2:end,:).*Mtb;
B(:,1:end-1,:)=B(:,1:end-1,:).*Mbt;
B(:,:,2:end)=B(:,:,2:end).*Mlr;
B(:,:,1:end-1)=B(:,:,1:end-1).*Mrl;
foo=repmat(sum(B,1),[2,1]);
```

```
B=B./foo;
alphamat=squeeze(B(1,:,:));
figure;imshow(alphamat);
```

Please see results in Figure 1. □

Problem 2 *Efros and Leung Texture Synthesis for Inpainting*

In image editing you not only want to paste object, but also need to remove objects. In the previous problem we have learnt how to segment out the object. In this problem we shall learn how to fill in i.e. *inpaint* pixels behind the object.

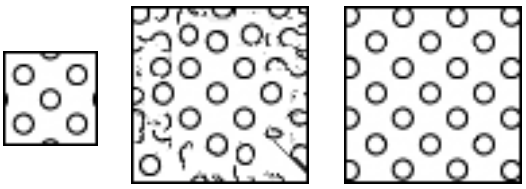
There are many criteria for inpainting. We ask you to implement texture synthesis-based approach which is particularly suitable for textured background. Read Efros and Leung's ICCV 1999 paper "Texture Synthesis by Non-parametric Sampling ". The pseudo code can be found at this link `http://gra phics.cs.cmu.edu/people/efros/research/NPS/alg.html`.

Write a MATLAB function `ImSyn=texturesyn_inpaint(im,mask,psize)` to synthesize a new image which has the same dimension as `im` (either grayscale or color). In `mask` the pixels marked as 1 are "search area", and those marked as 0 are "synthesis area". A `psize` x `psize` patch is used to find the best match for each pixel.

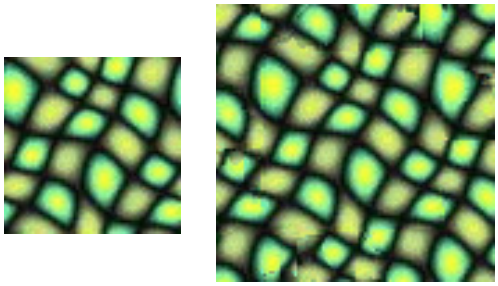
There are some tips for this MATLAB implementation. For each pixel you need to generate a template of both mask and pixel values. Fix the size of the template for each pixel. Treat both the mask and pixel values as zero for those outside image boundary.

(a) Load grayscale image `rings.bmp`. Put it at the center of a 64×64 image and grow texture. Try patch size 17 and 21, and see how the result varies. Then load color image `net.jpg` and grow it to a 100×100 image with patch size 23. It may take a while to run this code.

(b) (Extra credit) Load `hourse.bmp` and `'mask.bmp'`. Use the background pixels on the grass to remove the house. However, you do not have to use all the background pixels but only those close to the boundary. Run this code before you go to bed and see the result in the morning. You may also try your favorite way to accelerate searching.



(a) (b) psize=17 (c) psize=21



(d) (e) psize=23

(f) Input image

(g) Mask

(h) Gray pixels are search region

(i) Inpainting result

Figure 2: The result of texture synthesis and inpainting.

Solution. Here is the code for texture synthesis-based inpainting.

```
function ImSyn=texturesyn_inpaint(im,mask,wlength)
if mod(wlength,2)~=1
    wlength=wlength+1;
end
wsize=(wlength-1)/2;
[height,width,n]=size(im);
ImSyn=im;
mask0=mask;

template_mask0=zeros(wlength);
template_pixels0=zeros([wlength wlength n]);
template_gaussian=fspecial('gaussian',wlength,wsize/2);

searchlist=find(mask==1);
searchx=ceil(searchlist/height);
searchy=searchlist-(searchx-1)*height;
% make the matrix of the search pixels
N=length(searchlist);
M_mask=zeros(wlength^2,N);
M_pixels=zeros(wlength^2*n,N);
for k=1:N
    u=searchx(k);v=searchy(k);
    u1=max(u-wsize,1);u2=min(u+wsize,width);
    v1=max(v-wsize,1);v2=min(v+wsize,height);
    uu1=u1-u+wsize+1;uu2=u2-u+wsize+1;
    vv1=v1-v+wsize+1;vv2=v2-v+wsize+1;

    match_mask=template_mask0;
    match_mask(vv1:vv2,uu1:uu2)=mask0(v1:v2,u1:u2);
    M_mask(:,k)=match_mask(:);

    match_pixels=template_pixels0;
    match_pixels(vv1:vv2,uu1:uu2,:)=im(v1:v2,u1:u2,:);
    M_pixels(:,k)=match_pixels(:);
end
M_gaussian=repmat(template_gaussian(:),1,N);
M_mask=M_mask.*M_gaussian;
clear M_gaussian;

while 1
    newmask=imdilate(mask,strel('rectangle',[3 3]));
    PixelList=find(newmask==1 & mask==0);
    % if there is no pixel to fill, then break
```

```
if length(PixelList)==0
    break;
end
for i=1:length(PixelList)
    % get coordinate
    x=ceil(PixelList(i)/height);
    y=PixelList(i)-(x-1)*height;
    % get the bounding box of the pixel
    x1=max(x-wsize,1);x2=min(x+wsize,width);
    y1=max(y-wsize,1);y2=min(y+wsize,height);
    % compute the bounding box on the template
    xx1=x1-x+wsize+1;xx2=x2-x+wsize+1;
    yy1=y1-y+wsize+1;yy2=y2-y+wsize+1;

    template_mask=template_mask0;
    template_mask(yy1:yy2,xx1:xx2)=mask(y1:y2,x1:x2);

    template_pixesl=template_pixels0;
    template_pixels(yy1:yy2,xx1:xx2,:)=ImSyn(y1:y2,x1:x2,:);

    maxweight=template_mask.*template_gaussian;
    maxweight=sum(maxweight(:));

    % now find the best match
    T_mask=repmat(template_mask(:),1,N);
    T_pixels=repmat(template_pixels(:),1,N);
    overlap=T_mask.*M_mask;
    totalweight=sum(overlap,1)+eps;
    Diff=(T_pixels-M_pixels).^2;
    if n>1
        Diff=reshape(Diff,[wlength^2,n,N]);
        Diff=squeeze(sum(Diff,2));
    end
    errorlist=sum(Diff.*overlap,1)./totalweight*maxweight;
    hlist0=find(totalweight>=0.5*maxweight);
    errorlist=errorlist(hlist0);
    min_error=min(errorlist);
    hlist=find(errorlist<=min_error*1.04);
    % sampling
    IDX=hlist0(hlist(ceil(length(hlist)*rand)));
    xs=searchx(IDX);
    ys=searchy(IDX);
    ImSyn(y,x,:)=im(ys,xs,:);
end
mask(PixelList)=1;
```

```
end
```

Pleas see the results in Figure 2. □

Problem 3 *Image Morphing*

In this problem, you will implement image warping using radial basis function, and will then use it to perform morphing between two images. In image warping, a smooth displacement field is applied to deform an image. The displacement field that we will manipulate is a 2D to 2D function which, given a position in the **output** image, provides the original position in the **input** image. Note that this is the inverse of the displacement undergone by the image, because we need to look up for each new pixel the corresponding color in the old image.

In this problem, the field $f : \mathbb{R}^2 \mapsto \mathbb{R}^2$ will be sparsely specified by the user who will provide the before and after position of a sparse set of points as a set of pairs of 2D vectors, denoted as xi for the position before and yi for the positions after. The main task is, given these sparse correspondences, to interpolate a smooth displacement field f. For this, we will use radial basis functions, that is, smooth kernel functions centered on the sparse data points. We will use thin-plate RBFs that depend on the spatial distance to the data points (we will ignore the polynomial term sometimes included in RBFs). The basis function centered on data point x_i evaluated at an arbitrary point z is:

$$R(z, x_i) = \frac{1}{\sqrt{c + \|z - x_i\|^2}} \tag{8}$$

where $\|z - x_i\|$ is the distance between position z and data point x_i, and c is a parameter that controls the falloff of the function. set $c = 10$ at the beginning and experiment with it only once your code fully works.

Your main task is then to determine the coefficient (weight) of the basis function centered on each data point. These weights are found by solving a linear system that enforces interpolation. That is, the value of the RBF at each data point x_k should be the provided vector y_k, i.e.

$$\sum_i \alpha_i R(x_k, x_i) = y_k, \quad k = 1, \cdots, n \tag{9}$$

where $\alpha_i \in \mathbb{R}^2$.

Once you have computed an RBF model, you can warp the image by computing the color of each new pixel. Compute the displacement field at the location z by evaluating the RBF model.

$$f(z) = \sum_i \alpha_i R(z, x_i) \tag{10}$$

(You will notice a computational disadvantage of RBFs: each kernel needs to be evaluated, making the cost linear in the number of provided data points). Then perform a simple lookup and use the color of the displaced pixel $f(z)$ in the input image. You may use MATLAB function `interp2`.

(a) Load `chessboard.bmp`. Generate the the warped image according to the following specified displacement:

$$\begin{gathered}(32, 32) \mapsto (88, 58) \\ (224, 32) \mapsto (229, 82) \\ (224, 224) \mapsto (165, 212) \\ (32, 224) \mapsto (61, 178)\end{gathered}$$

You may change c and see how it affects the warp field.

(b) Load two face images `face1.bmp` and `face2.bmp`, and the labeled feature points `facepts1.mat` and `facepts2.mat`. Compute and display the forward (1 to 2) and backward (2 to 1) warp fields, and the corresponding warped images. (Hint: you can make debugging easier by replacing the face images by a grid picture of the same resolution, which will better reveal any problem.)

(c) **Morphing**. Morphing interpolates seamlessly between two images by combining a warp of the initial and final images with linear interpolation of color values. Morphing performs spatial warping so that the provided features are displaced along time from their initial position to their final position. You will compute a 5-image morphing animation where t varies from 0 to 1 in steps of 0.2 (that is, the two inputs plus four intermediate images). For each intermediate image, first compute the interpolated position of each feature. Perform a spatial warp as above from the initial image to these intermediate positions. This will give you a first image. Similarly, perform a warp of the final image to these intermediate positions, whcih gives you a second image. Then

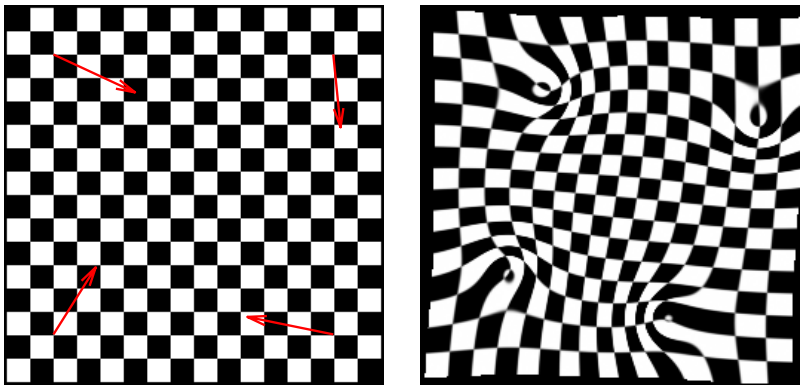


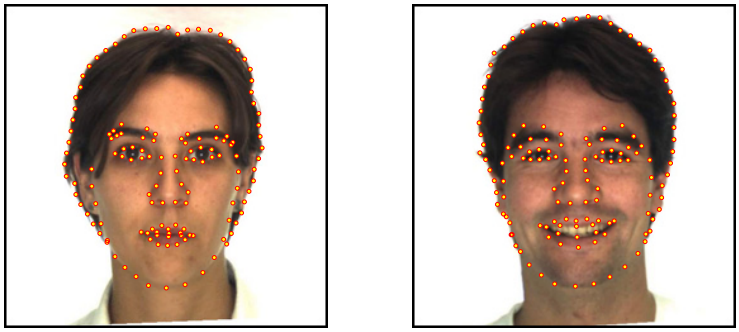
Figure 3: The chessboard is warped (right) according to the displacement of the four feature points (left) specified by the user.

simply blend the two images according to the value of time (where the influence of the initial image varies from 0 to 1 with as time progresses). Display the morphing sequence.

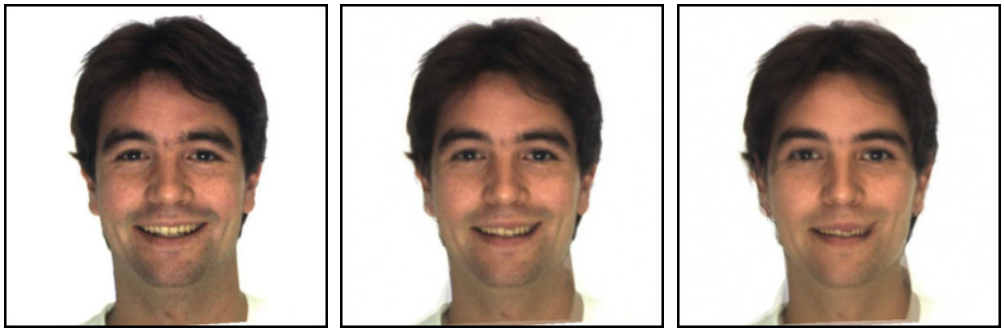
(c) (Extra credit) Perform high-quality resampling using elliptical Gaussians. Vary the color interpolation in a spatially-varying and non-linear manner. "Snap" the provided feature points to better match the local content (search the neighborhood to optimize the L2 difference between 3x3 windows of the two images).

Solution. See the following code.

```
im1=im2double(imread('face1.bmp'));
im2=im2double(imread('face2.bmp'));

load facepts1.mat;
load facepts2.mat;

figure;imshow(imaddb(im1));hold on;
plot(facepts1(:,1),facepts1(:,2),'o','LineWidth',1,'MarkerSize',...
     4,'MarkerFaceColor','y','MarkerEdgeColor','r');
saveas(gcf,'face1.eps','psc2');

figure;imshow(imaddb(im2));hold on;
plot(facepts2(:,1),facepts2(:,2),'o','LineWidth',1,'MarkerSize',...
     4,'MarkerFaceColor','y','MarkerEdgeColor','r');
```



(a) Face 1 and its feature points (b) Face 2 and its feature points



(c) $\alpha = 0.0$ (d) $\alpha = 0.2$ (e) $\alpha = 0.4$

(f) $\alpha = 0.6$ (g) $\alpha = 0.8$ (h) $\alpha = 1.0$

Figure 4: Face morphing sequence of two faces from AR face database. Courtesy to Liang Lin's (MSRA) help on aligning the faces.

```
saveas(gcf,'face2.eps','psc2');

[height,width,n]=size(im1);
[xx,yy]=meshgrid(1:width,1:height);
npts=size(facepts1,1);

% set c
c=10;

% infer the coefficients for forward warping
X=repmat(facepts1(:,1),[1,npts]);
Y=repmat(facepts1(:,2),[1,npts]);
G0=1./sqrt((X'-X).^2+(Y'-Y).^2+c);
a=G0\(facepts2-facepts1);
XX=repmat(xx(:)',[npts,1]);
X0=repmat(facepts1(:,1),[1,height*width]);
YY=repmat(yy(:)',[npts,1]);
Y0=repmat(facepts1(:,2),[1,height*width]);
G=1./sqrt((XX-X0).^2+(YY-Y0).^2+c);
V_f(:,:,1)=reshape(sum(G.*repmat(a(:,1),[1,height*width]),1),[height,width]);
V_f(:,:,2)=reshape(sum(G.*repmat(a(:,2),[1,height*width]),1),[height,width]);

% infer the coefficients for backward warping
X=repmat(facepts2(:,1),[1,npts]);
Y=repmat(facepts2(:,2),[1,npts]);
G0=1./sqrt((X'-X).^2+(Y'-Y).^2+c);
a=G0\(facepts1-facepts2);
XX=repmat(xx(:)',[npts,1]);
X0=repmat(facepts2(:,1),[1,height*width]);
YY=repmat(yy(:)',[npts,1]);
Y0=repmat(facepts2(:,2),[1,height*width]);
G=1./sqrt((XX-X0).^2+(YY-Y0).^2+c);
V_b(:,:,1)=reshape(sum(G.*repmat(a(:,1),[1,height*width]),1),[height,width]);
V_b(:,:,2)=reshape(sum(G.*repmat(a(:,2),[1,height*width]),1),[height,width]);

n=6;
for k=1:n
    alpha=(k-1)/(n-1);
    for i=1:3
        Z1(:,:,i)=interp2(xx,yy,im2(:,:,i),V_f(:,:,1)*alpha...
                  +xx,V_f(:,:,2)*alpha+yy,'cubic');
    end
    for i=1:3
        Z2(:,:,i)=interp2(xx,yy,im1(:,:,i),V_b(:,:,1)*(1-alpha)...
                  +xx,V_b(:,:,2)*(1-alpha)+yy,'cubic');
```

```
    end
    Im=Z1*(1-alpha)+Z2*alpha;
    Im=Im(:);
    index=find(~(Im<=2 & Im>=-2));
    Im(index)=1;
    Im=reshape(Im,[height,width,3]);
    figure;imshow(imaddb(Im));
    saveas(gcf,['morphing_' num2str(k) '.eps'],'psc2');
end
```

See the results in Figure 4. □