Medical Image Analysis
Third hand-in

Mariuta Nicolae (rqt629) September 22, 2015

Exercise 1

a. Mutual information

I have written a function for calculating the mutual information. I mostly used the algorithm described at the lecture for making the implementation.

```
function jointEntropy = my_mutual_information(I1, I2)
%replace all nans because otherwise some matlab functions will not work
I1(isnan(I1)) = 0;
I2(isnan(I2)) = 0;
%calculate the joint histogram
indrow = floor(double(I1(:))) + 1;
indcol = floor(double(I2(:))) + 1; %// Should be the same size as indrow
jointHistogram = accumarray([indrow indcol], 1);
jointProb = jointHistogram / numel(indrow);

%joint entropy
indNoZero = jointHistogram ~= 0;
jointProb1DNoZero = jointProb(indNoZero);
jointEntropy = -sum(jointProb1DNoZero.*log2(jointProb1DNoZero));

end
```

At the beginning I replace any NaN values I sometimes get from making the affine transformation with 0. After that I do basic calculation of the joint histogram and calculate the probability distribution.

In the next stage, I calculate the joint entropy based on the equation from Wikipedia page: https://en.wikipedia.org/wiki/Joint_entropy

In the exercise with affine transformations, I will also plot the results from this equation.

b. Sum-of-square difference

```
%function for calculating the sum of square difference
%just apply the formula
function d =  my_ssd(I1, I2)
%replace all nans because otherwise some matlab functions will not work
I1(isnan(I1)) = 0;
I2(isnan(I2)) = 0;

%calculation of ssd
d =sum((I1(:) - I2(:)).^2)/numel(I1);

end
```

For this exercise I also started by replacing all NaN values with 0 and then just written the equation that was also shown at the lecture:

$$\mathrm{d} = \frac{1}{\mathrm{N}}\sum(I_1 - I_2)^2$$

I will also plot results for this after making the affine transformation exercise.

The way I implemented both the calculation of mutual information and sum of square difference make the functions to also work on 3d images, so I will be able to use them at further exercises.

Exercise 2

a. Rotation of the image

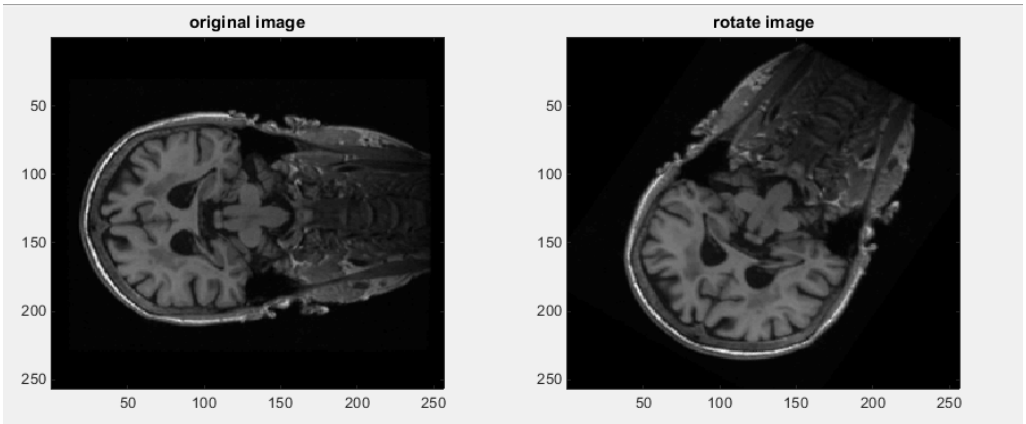
For the rotation of the image I started with making the 2d rotation by using the matrix:

$$R = \begin{pmatrix} cos\alpha & -sin\alpha \\ sin\alpha & cos\alpha \end{pmatrix}$$

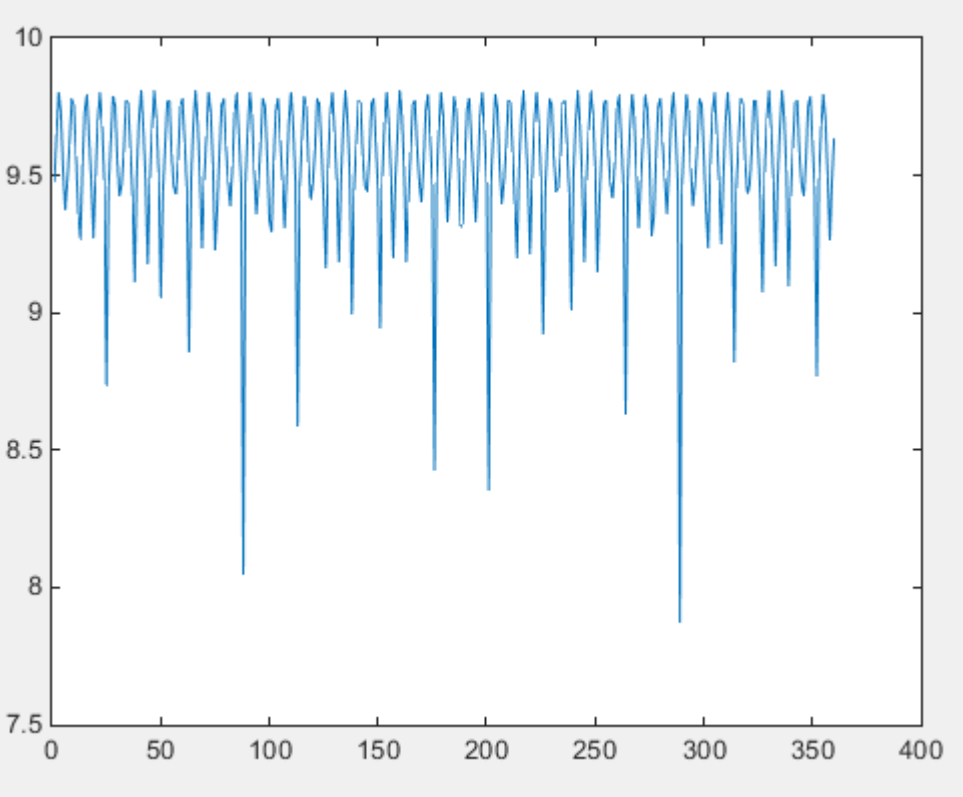
for making the rotation and calculate the new coordinates of the points. After this, I applied the interpolation to create the intensity values of the new pixels.

```
%my function for interpolation, takes as input the original 2d image and
angle
%in degrees
function Irotate =  my_rotate(I,pi)

%create the original grid
[Xgrid, Ygrid] = meshgrid(1:size(I,1),1:size(I,2));
pts = [Xgrid(:) Ygrid(:)] - size(I,1)/2;

%transform into radians
theta = 90*pi/180;

%calculate the new coordinates of the points
pts2 = ([cos(theta) -sin(theta) ; sin(theta) cos(theta)]*pts')'+size(I,1)/2;

%use interpolation to obtain the finale image
Irotate = interp2(I,reshape(pts2(:,1),256,256),reshape(pts2(:,2),256,256));

end
```

I obtain this result for using this function:



The image is properly rotated around the origin. By rotating this image with angles between 1-360 and calculating the mutual information using my function from exercise 1a, I obtain this plot:



It seems that at angle 360, the value of mutual information is almost the same as for angle 1 so it seems to work correctly my rotation and also the calculation of mutual information.

b. Translation of the image

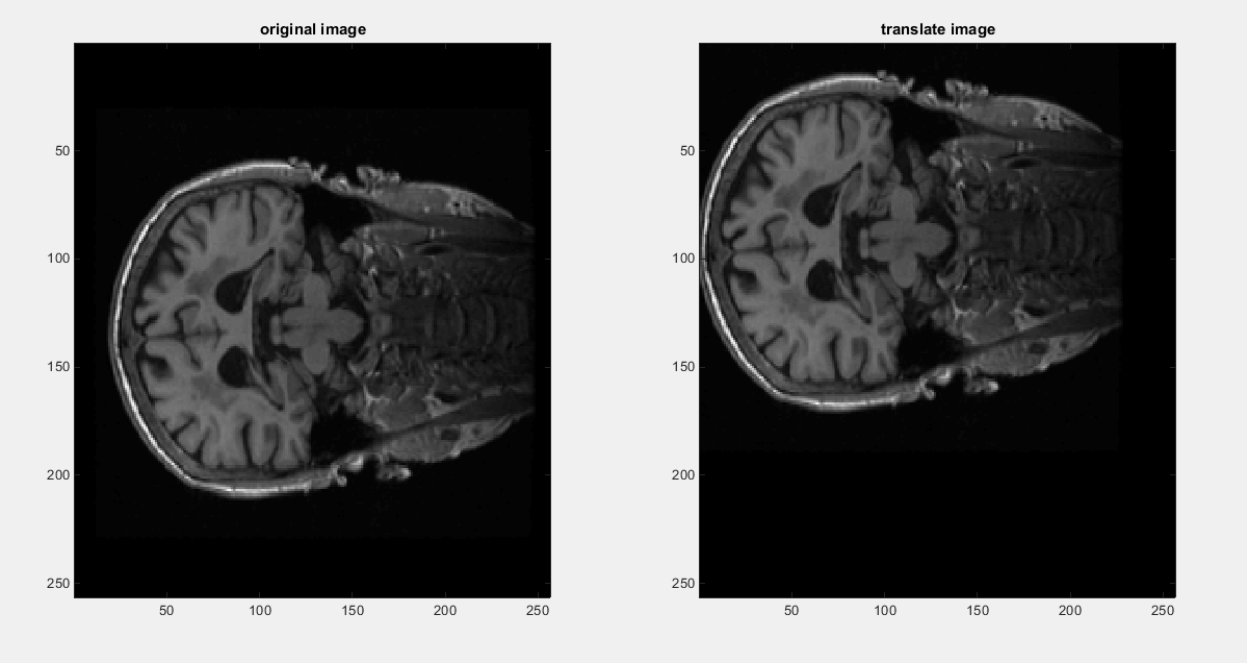
For this exercise I also started by making 2d translation, by using the matrix:

$$T = \begin{pmatrix} 1 & 0 & tx \\ 0 & 1 & ty \\ 0 & 0 & 1 \end{pmatrix}$$

and made the implementation very much like it was described at the lecture.

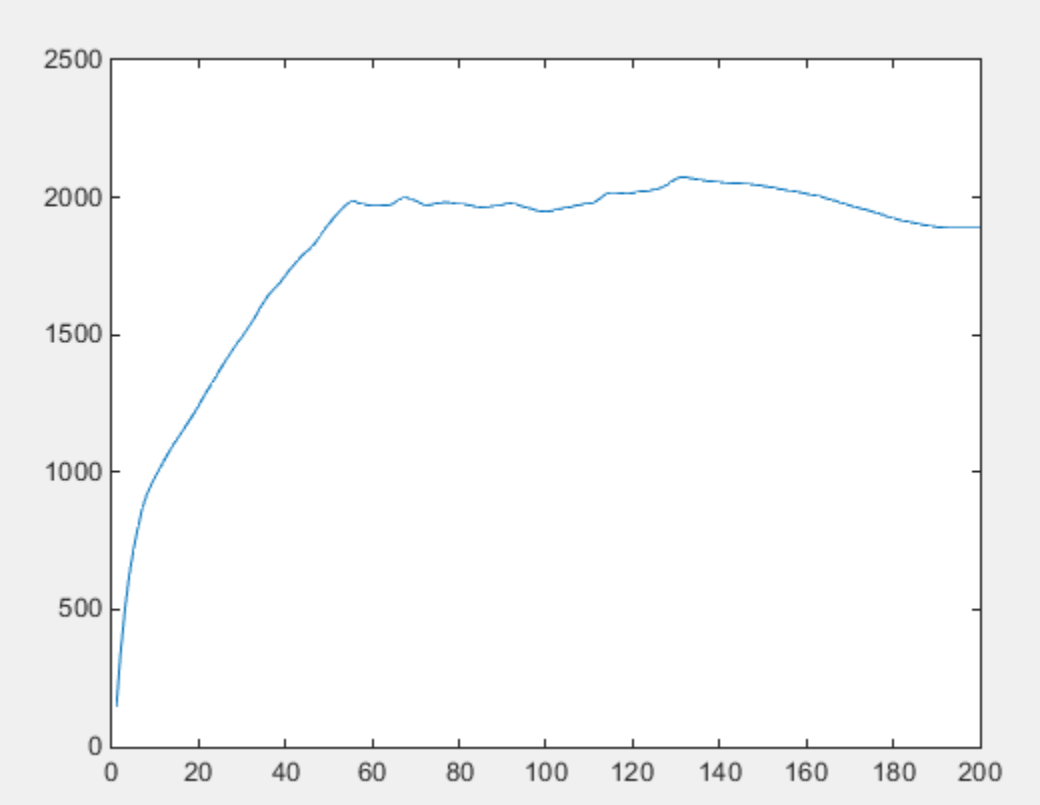
```
%my function for translation, takes as input the original 2d image and the
%number of pixels to move along x and y
function Itranslate =  my_translate(I,tx, ty)

%translation matrix
A = [1 0 tx ; 0 1 ty ; 0 0 1];


%create the original grid
[Xgrid, Ygrid] = meshgrid(1:size(I,1),1:size(I,2));
pts = [Xgrid(:) Ygrid(:)];

%calculate the new coordinates of the points
column = ones(size(pts,1),1);
pts = [pts column];
pts2 = (A * pts')';

%create the final image
Itranslate =
interp2(I,reshape(pts2(:,1),256,256),reshape(pts2(:,2),256,256));

end
```

By making a translation using my function for the image at exercise 3, I obtain:



So the translation seems to work fine in also.

For the translation, I used the same image and translated in both directions with values 1-200 and plotted the sum of square difference:



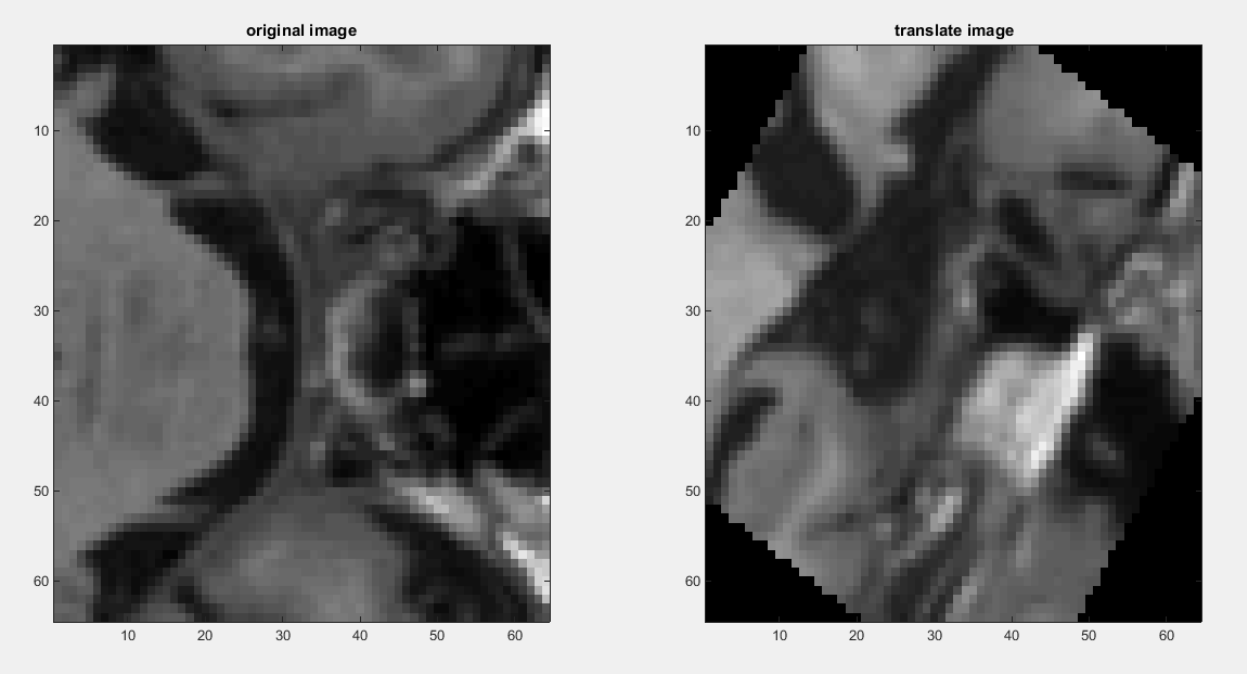
In this case, it also seems to work correctly, because the value of ssd is very low when the image is not translated much, but it gets high as it keeps translating.

c. Complete 3d affine transformation

For the purpose of making easier to implement the registration, I created a function that translates and rotates the 3d image by taking 6 parameters as input: translation on x, translation on y, translation on z, rotation on x, rotation on y and rotation on z. The implementation is almost the same, the only difference is the matrices that were used for the affine transformation, which I got from Wikipedia.

```
%my function to make rotation and translation where
%tx,ty,tz are the distances for translation
%Qx,Qy,Qz are the angles for rotations
function P =  my_3d_affine(I,tx,ty,tz,Qx,Qy,Qz)

%create the original grid
[Xgrid, Ygrid, Zgrid] = meshgrid(1:size(I,1),1:size(I,2),1:size(I,3));
pts = [Xgrid(:) Ygrid(:) Zgrid(:)] - size(I,1)/2;

%take sizes of the image for later use
dx = size(I,1);
dy = size(I,2);
dz = size(I,3);

%transform into radians
xtheta = 90*Qx/180;
ytheta = 90*Qy/180;
ztheta = 90*Qz/180;

%make rotation
rx = [1 0 0 ; 0 cos(xtheta) -sin(xtheta) ; 0 sin(xtheta) cos(xtheta)];
ry = [cos(ytheta) 0 sin(ytheta) ; 0 1 0 ; -sin(ytheta) 0 cos(ytheta)];
rz = [cos(ztheta) -sin(ztheta) 0 ; sin(ztheta) cos(ztheta) 0 ; 0 0 1];
rotate = rz*ry*rx;

pts_rotate = rotate * pts';

%make translation
row = ones(1, size(pts,1));
pts2 = [pts_rotate ; row];
translate = [1 0 0 tx ; 0 1 0 ty ; 0 0 1 tz ; 0 0 0 1];
pts_translate = (translate*pts2)' + size(I,1)/2 ;

%create the final image
P = interp3(I, reshape(pts_translate(:,1),dx,dy,dz),
reshape(pts_translate(:,2),dx,dy,dz), reshape(pts_translate(:,3),dx,dy,dz));
end
```

Because my computer could not handle the 256x256x256 size matrix multiplications and it took too long to make this transformation, I only took 64x64x64 block in the middle and used it to test the function and displayed the results:



Exercise 3: Rigid registration

I implemented the rigid registration as I understood it, but using the cost function with sum square difference. I started by calculating the gradients over x,y,z of the 3d image by using the Matlab function for calculating the gradient, because I had some issues with implementing my own, as I shown at the previous assignment.

After this step, I defined the rotation and translation matrices, together with their gradients and then calculated the gradient of sum of square difference function by using the equation shown in the lecture.

I used as much as possible whole matrix multiplication in Matlab because it runs a lot faster than using for loops.

In the end, I get scalar values for each of the 6 degrees of freedom that I had: rotation on x, rotation on y, rotation on z, translation on x, translation on y, translation on z. And I summed these values to obtain only one scalar that to be optimized. For optimization I used the Matlab function *fminunc* because it seemed easier to use.

I used the same block of size 64x64x64 from the whole image so I can process on my computer, and a transformation with parameters:

$$\delta x = 6, \delta y = 8, \delta z = 10, tx = 10, ty = 5, tz = 20$$

And after the optimization I obtained the values:

$$\delta x = 8.59, \delta y = 11.16, \delta z = 9.27, tx = 10, ty = 10, tz = 10$$

I think there is something wrong at computing the gradient because I keep getting only 10 for the translation values, and the values for rotation are also quite different from those of the original image.

A reason for this issue might be the fact that I took a part from the middle of the brain and some information is lost during the translation and rotation since valuable information goes out of the grid. If I could use the whole image, the part going out of the grid is just the black part surrounding the head.

But there is also high chance I did some mistake at calculating the gradient since it took me quite long time to understand how to calculate it.

Paper report: Global image registration using a symmetric block-matching approach

The article proposes a method for image registration that does not suffer so much from the directionality bias that can sometimes have huge effects. The method proposed is suitable for multimodal registration and that is robust to outliers in the input images. In comparison to asymmetric block-matching, this method is a lot more accurate and robust.

The directionality in the image registration pipelines creates bias in subsequent analyses. There are several papers mentioned that describe the drawback of the asymmetric block-matching like the impact on the extraction of imaging biomarkers.

There have been several projects that tried to deal with this issue, but not many of them considered removing directionally bias in the case of global registration.

There are several differences in the proposed method: it is used block-matching to establish the spatial correspondences, where normalized cross correlation is used as a measure of similarity making it suitable for multimodal registration applications; joint backward and forward transformation parameters simultaneously calculated that removes the need to discretize transformed input images

into an average space. With this approach, the transformation is optimized with the native spaces of the input images.

The optimum transformation is estimated in an iterative method where the current estimated transformation is updated each iteration. Each iteration consists of two steps: first, the point correspondences between the two images are established using block matching. In the second step is to update the transformation parameters estimated through LTS regression. The block-matching correspondences for the symmetric approach are always established using the original image positions at every iteration. As a result, no matrix multiplication has to be done since the update is integrated into the LTS regression.

After the implementation, they evaluated the accuracy precision and robustness using several databases of MRI, PET, CT and the algorithm performed well for all tests.

Annexes with all the Matlab code

1. The main file

```
%%  rotate 2d image
clearvars;
%load image and set paramters
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(:,:,100));
theta = 90*pi/180;

%use my function for rotate
Irotate = my_rotate(I,10);

%show the results
subplot(1,2,1),imagesc(I),colormap('gray'),title('original image');
subplot(1,2,2),imagesc(Irotate),colormap('gray'),title('rotate image');

%% translate 2d image

%load image and set paramters
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(:,:,100));
tx = 20;
ty = 140;

%use my function for translate
Itranslate = my_translate(I,tx,ty);

%show the results
subplot(1,2,1),imagesc(I),colormap('gray'),title('original image');
```

```
subplot(1,2,2),imagesc(Itranslate),colormap('gray'),title('translate image');

%% Complete affine transformation including translation and rotation
clearvars;

B = my_load_mgh('orig_1.mgz');
%trim the image so I can obtain a matrix that I can process
Bpart = B(97:160,97:160,97:160);


Btransformed = my_3d_affine (Bpart, 1, 1, 1, 10, 10, 10);
Btransformed(isnan(Btransformed)) = 0;

%display a slice to observe the changes
I = squeeze(Bpart(:,32,:));
Iaffine = squeeze(Btransformed(:,32,:));

%show the results
subplot(1,2,1),imagesc(I),colormap('gray'),title('original image');
subplot(1,2,2),imagesc(Iaffine),colormap('gray'),title('translate image');


%% Mutual information
clearvars;
%load the image
B = my_load_mgh('orig_1.mgz');
I = squeeze(B(:,:,100));

%rotate image multiple times and calculate the rigid transformation
mutual_array = squeeze(zeros(360,1));
for i = 1 : 360
    I2 = my_rotate(I,i);
    mutual_array(i) = my_mutual_information(I,I2);
end

%plot the result
plot(1:360,mutual_array);



%% sum of square difference
clearvars;

B = my_load_mgh('orig_1.mgz');
I = squeeze(B(:,:,100));

%translate image multiple times and calculate the rigid transformation
ssd_array = squeeze(zeros(200,1));
for i = 1 : 200
    I2 = my_translate(I,i,i);
    ssd_array(i) = my_ssd(I,I2);
end

%plot the result
```

```
plot(1:200,ssd_array);


%% registration
%load the image
clearvars;
B = my_load_mgh('orig_1.mgz');
%trim the image so I can obtain a matrix that I can process
B1 = B(97:160,97:160,97:160);
B2 = my_3d_affine (B1, 5, 15, 3, 8, 8, 10);

%df = my_cost_function(0,0,0,0,0,0,B1,B2);


P = [10 10 10 10 10 10]
fdf = @(P)my_cost_function(P,B1,B2);
odf = fminunc(fdf,P);

disp(odf);
```

2. The function for affine transform in 3d

```
%my function to make rotation and translation where
%tx,ty,tz are the distances for translation
%Qx,Qy,Qz are the angles for rotations
function P =  my_3d_affine(I,tx,ty,tz,Qx,Qy,Qz)

%create the original grid
[Xgrid, Ygrid, Zgrid] = meshgrid(1:size(I,1),1:size(I,2),1:size(I,3));
pts = [Xgrid(:) Ygrid(:) Zgrid(:)] - size(I,1)/2;

%take sizes of the image for later use
dx = size(I,1);
dy = size(I,2);
dz = size(I,3);

%transform into radians
xtheta = 90*Qx/180;
ytheta = 90*Qy/180;
ztheta = 90*Qz/180;

%make rotation
rx = [1 0 0 ; 0 cos(xtheta) -sin(xtheta) ; 0 sin(xtheta) cos(xtheta)];
ry = [cos(ytheta) 0 sin(ytheta) ; 0 1 0 ; -sin(ytheta) 0 cos(ytheta)];
rz = [cos(ztheta) -sin(ztheta) 0 ; sin(ztheta) cos(ztheta) 0 ; 0 0 1];
rotate = rz*ry*rx;

pts_rotate = rotate * pts';

%make translation
row = ones(1, size(pts,1));
pts2 = [pts_rotate ; row];
```

```
translate = [1 0 0 tx ; 0 1 0 ty ; 0 0 1 tz ; 0 0 0 1];
pts_translate = (translate*pts2)' + size(I,1)/2 ;

%create the final image
P = interp3(I, reshape(pts_translate(:,1),dx,dy,dz),
reshape(pts_translate(:,2),dx,dy,dz), reshape(pts_translate(:,3),dx,dy,dz));
end
```

3. The cost function

```
%my cost function; returns the ssd and derivative of ssd
function df =  my_cost_function(P, I1, I2)
xtheta = P(1);
ytheta = P(2);
ztheta = P(3);
tx = P(4);
ty = P(5);
tz = P(6);


I1(isnan(I1)) = 0;
I2(isnan(I2)) = 0;

%calculate gradients of I2
[gx,gy,gz] = gradient(I2);
G = [gx(:) gy(:) gz(:)];

%affine transformation matrices:
rx = [1 0 0 0 ; 0 cos(xtheta) -sin(xtheta) 0 ; 0 -sin(xtheta) cos(xtheta) 0 ;
0 0 0 1];
ry = [cos(ytheta ) 0 sin(ytheta) 0 ; 0 1 0 0 ; -sin(ytheta) 0 cos(ytheta) 0 ;
0 0 0 1];
rz = [cos(ztheta) sin(ztheta) 0 0 ; -sin(ztheta) cos(ztheta) 0 0 ; 0 0 1 0 ;
0 0 0 1];
t = [1 0 0 tx ; 0 1 0 ty ; 0 0 1 tz ; 0 0 0 1];
Tx = [1 0 0 tx ; 0 1 0 0 ; 0 0 1 0 ; 0 0 0 1];
Ty = [1 0 0 0 ; 0 1 0 ty ; 0 0 1 0 ; 0 0 0 1];
Tz = [1 0 0 0 ; 0 1 0 0 ; 0 0 1 tz ; 0 0 0 1];
%gradients for transformation matrices:
drx = [0 0 0 0 ; 0 -sin(xtheta) -cos(xtheta) 0 ; 0 -cos(xtheta) -sin(xtheta)
0 ; 0 0 0 1];
dry = [-sin(ytheta) 0 cos(ytheta) 0 ; 0 1 0 0 ; -cos(ytheta) 0 -sin(ytheta) 0
; 0 0 0 1];
drz = [-sin(ztheta) cos(ztheta) 0 0 ; -cos(ztheta) -sin(ztheta) 0 0 ; 0 0 1 0
; 0 0 0 1];
dtx = [ 0 0 0 1 ; 0 0 0 0 ; 0 0 0 0 ; 0 0 0 0 ];
dty = [ 0 0 0 0 ; 0 0 0 1 ; 0 0 0 0 ; 0 0 0 0 ];
dtz = [ 0 0 0 0 ; 0 0 0 0 ; 0 0 0 1 ; 0 0 0 0 ];

%gradient of ssd over the 6 degrees of freedom
diff = I1-I2;
diff = diff(:);
d = diff*sum(G);
```

```
row = ones(1, size(d,1))';
disp(size(d));
disp(size(row));
d = [d  row];

dQrx = (1/numel(I2))*sum(sum((-2)*drx*ry*rz*t*d',2));
dQry = (1/numel(I2))*sum(sum((-2)*rx*dry*rz*t*d',2));
dQrz = (1/numel(I2))*sum(sum((-2)*rx*ry*drz*t*d',2));
dQtx = (1/numel(I2))*sum(sum((-2)*rx*ry*rz*dtx*d',2));
dQty = (1/numel(I2))*sum(sum((-2)*rx*ry*rz*dty*d',2));
dQtz = (1/numel(I2))*sum(sum((-2)*rx*ry*rz*dtz*d',2));

%matrix with all gradients
df = dQrx + dQry + dQrz + dQtx + dQty + dQtz;

end
```

4. My interpolation function

```
%my function for interpolation. Input parameters: original image, original
%grid and the new points
function Interpolate = my_interpolation(I, pts, ptsShift)
% the interpolate image
Interpolate = zeros(size(I));

for i = 1:size(pts,1)
   xi = pts(i,1);yi = pts(i,2);
   x = ptsShift(i,1);y = ptsShift(i,2);

   %calculate weights for pixels in original image:
    w11 = ((xi+1) - x)*((yi+1) - y);
    w21 = (x - xi)*((yi+1) - y);
    w12 = ((xi+1) - x)*(y - yi);
    w22 = (x - xi)*(i - yi);

    %calculate interpolation at point (xi,yi) in the new grid
    Interpolate(xi,yi) = I(xi,yi)*w11 + I(xi+1,yi)*w21 +  I(xi,yi+1)*w12 +
I(xi+1,yi+1)*w22/(((xi+1)-xi)*((yi+1)-yi));
end

end
```

5. Function for calculating the mutual information

```
function jointEntropy = my_mutual_information(I1, I2)
%replace all nans because otherwise some matlab functions will not work
```

```
I1(isnan(I1)) = 0;
I2(isnan(I2)) = 0;
%calculate the joint histogram
indrow = floor(double(I1(:))) + 1;
indcol = floor(double(I2(:))) + 1; %// Should be the same size as indrow
jointHistogram = accumarray([indrow indcol], 1);
jointProb = jointHistogram / numel(indrow);

%joint entropy
indNoZero = jointHistogram ~= 0;
jointProb1DNoZero = jointProb(indNoZero);
jointEntropy = -sum(jointProb1DNoZero.*log2(jointProb1DNoZero));

end
```

6. The function for rotate in 2d

```
%my function for interpolation, takes as input the original 2d image and
angle
%in degrees
function Irotate =  my_rotate(I,pi)

%create the original grid
[Xgrid, Ygrid] = meshgrid(1:size(I,1),1:size(I,2));
pts = [Xgrid(:) Ygrid(:)] - size(I,1)/2;

%transform into radians
theta = 90*pi/180;

%calculate the new coordinates of the points
pts2 = ([cos(theta) -sin(theta) ; sin(theta) cos(theta)]*pts')'+size(I,1)/2;

%use interpolation to obtain the finale image
Irotate = interp2(I,reshape(pts2(:,1),256,256),reshape(pts2(:,2),256,256));

end
```

7. The function calculating sum-squared-difference

```
%function for calculating the sum of square difference
%just apply the formula
function d =  my_ssd(I1, I2)
%replace all nans because otherwise some matlab functions will not work
I1(isnan(I1)) = 0;
I2(isnan(I2)) = 0;

%calculation of ssd
d =sum((I1(:) - I2(:)).^2)/numel(I1);

end
```

8. The function for translation in 2 d

```
%my function for translation, takes as input the original 2d image and the
%number of pixels to move along x and y
function Itranslate =  my_translate(I,tx, ty)

%translation matrix
A = [1 0 tx ; 0 1 ty ; 0 0 1];


%create the original grid
[Xgrid, Ygrid] = meshgrid(1:size(I,1),1:size(I,2));
pts = [Xgrid(:) Ygrid(:)];

%calculate the new coordinates of the points
column = ones(size(pts,1),1);
pts = [pts column];
pts2 = (A * pts')';

%create the final image
Itranslate =
interp2(I,reshape(pts2(:,1),256,256),reshape(pts2(:,2),256,256));

end
```