Question 5 CS663 assignment 1

For every part, I have shown my code, displayed the images, and slightly explained the code wherever necessary. My answers are as below.

Part a-

Image 1-

Image 2-

Code for part 1-

```
im1=imread('goi1.jpg');
im2=imread('goi2_downsampled.jpg');
imshow(im1);
imshow(im2);
```

I just read both the images and displayed them here.

Part b-

Code for part 2-

```
for i=1:12,
    imshow(im1);
    [x1(i), y1(i)] = ginput(1);
    imshow(im2);
    [x2(i), y2(i)] = ginput(1);
end;
one1= ones(1,12);
init= [x1; y1; one1];
final= [x2; y2; one1];
A= final/init;
```

```
A =

    1.0356    0.0274   27.8680
   -0.0185    1.0143   21.5316
   -0.0000   -0.0000    1.0000

>>
```

One1 is the 1D array of ones I used, as the matrix transform operation has [x,y,1] in it for every coordinate. In matrix form y=A/B means y= A(B^-1)

Part c-

Image 1

image 2

image 3(transformed)

Code for part 3-

```
im3= im2;
for r = 1:size(im2, 1)    % for number of rows of the image
   for c = 1:size(im2, 2)    % for number of columns of the image
      k1= inv(A)* [r; c; 1];
      k2= round(k1);
      indic= 1;
      if((k2(1)<=0 | k2(1)>size(im1, 1)))
         indic=0;
      end
      if((k2(2)<=0 | k2(2)>size(im1, 2)))
         indic=0;
      end
      if(indic==1)
         im3(r,c)= im1(k2(1),k2(2));
      else
         im3(r,c)= 0;
      end
      % increment counter loop

   end

end
imshow(im3);
```

What I did here was I said im3=im2 just to initalise an image with the same dimensions as image 2. Then I iterated through every pixel of that image and back calculated the corresponding pixel in image 1 by nearest neighbour method, and gave that pixel the same intensity as the one in image 1. Nearest neighbour means that the coordinates of the corresponding point we got in image 1 are just rounded off to the closest integral x and y point, using the round function.

Part d-

Image 1

image 2

image 3 (transformed)

Code for part 4-

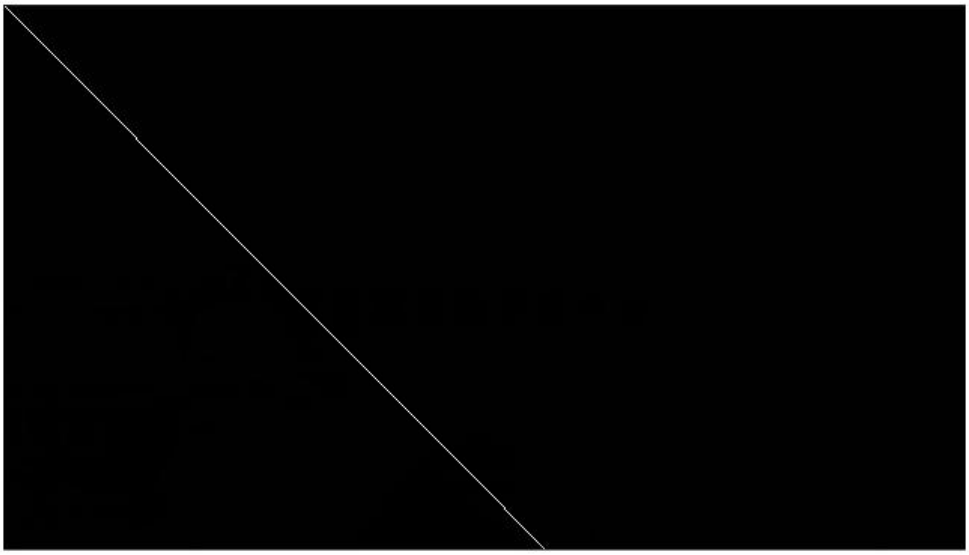
```
im4= im2;
for r = 1:size(im2, 1)    % for number of rows of the image
   for c = 1:size(im2, 2)    % for number of columns of the image
      k1= inv(A)* [r; c; 1];
      k2= floor(k1);
      k3= ceil(k1);
      indic= 1;
      if((k2(1)<=0 | k3(1)>size(im1, 1)))
         indic=0;
      end
      if((k2(2)<=0 | k3(2)>size(im1, 2)))
         indic=0;
      end
      if(indic==1)
         im4(r,c)=    round(((im1(k2(1),k2(2))*(k3(1)-k1(1))*(k3(2)-k1(2)))    +    (im1(k2(1),k3(2))*(k3(1)-
k1(1))*(k1(2)-k2(2)))    +    (im1(k3(1),k2(2))*(k1(1)-k2(1))*(k3(2)-k1(2)))    +    (im1(k3(1),k3(2))*(k1(1)-
k2(1))*(k1(2)-k2(2))))    ) ;
      else
         im4(r,c)= 0;
      end
   end
end

imshow(im4);
```

What I did here was again I did im4=im2 just to initalise an image with the same dimensions as image 2. Then I iterated through every pixel of that image and back calculated the corresponding pixel in image 1 by bilinear interpolation method, and gave that pixel the same intensity as the one in image 1. In bilinear interpolation, I got the intensity of the im4 point by adding the intensity of every one of the 4 points in image 1 multiplied by the area of the square opposite to that point. K3 contains x and y rounded up, k2 contains them rounded down. Hence in the multiplication, I have multiplied by differences of k2 and k3 in such a way, that the area will always be positive without using any absolute value function.

Part e-

If the first n point we chose were collinear,
The affine matrix would have a non zero nullity, and hence be rank deficient. Thus it will transform 2D pictures into a line or point.
Also we cannot use the pre image method as A is a singular matrix, its inverse won't exist. Thus we have to multiply A with image 1 to get a corresponding index in image 2.
We then use the same intensity in image 1 as the corresponding point in image 2.



code-

```
im1=imread('goi1.jpg');
im2=imread('goi2_downsampled.jpg');
for i=1:12,
    x1(i) = i;
    y1(i) = i;
    x2(i) = i+1;
    y2(i) = i+1;
end;
one1= ones(1,12);
init= [x1; y1; one1];
final= [x2; y2; one1];
A= final/init;
im3=im2/255 ;
for r = 1:size(im1, 1)    % for number of rows of the image
    for c = 1:size(im1, 2)    % for number of columns of the image
        k1= A* [r; c; 1];
        k2= round(k1);
        indic= 1;
        if((k2(1)<=0 | k2(1)>size(im1, 1)))
            indic=0;
        end
        if((k2(2)<=0 | k2(2)>size(im1, 2)))
            indic=0;
        end
        if(indic==1)
            im3(k2(1),k2(2))= im1(r,c);
        else
            im3(k2(1),k2(2))= 0;
        end
    end
end
imshow(im3);
```

In the above code I just initialised an image of same size as im2, but with all black squares. Then I transformed that image accordingly to get the above.

Here we could not calculate the corresponding pixel in im1 from im2, as A is not invertible if collinear points are chosen. Hence I did a straightforward calculation of corresponding im3 pixel for every pixel in im1, and gave corresponding pixels the same intensity.

Note:- The only reason image2 and the image I got by transformation aren't exactly similar are because I didn't choose the corner points of the image properly as it was tough to find the corresponding points on the other image, and I realised this only after finishing the report. But the logic I have implemented is correct.