University ID : N17385760
Net ID : iv447

Project 2

1 Problem 1: Affine Image Transformation

1.1 Resampling :

Resampling can be implemented using two methods :

1. Nearest neighbor (NN): It takes the intensity from the pixel which is closest to the non-grid position of the transformation.

2. Bilinear Interpolation : Bilinear interpolation is extension of linear interpolation. In this we first perform of linear interpolation and then again in other direction. The interpolation as whole is quadratic not linear. Suppose we want to know the value of unknown function at point (x,y). It is assumed that we know the values at four points Q(11)= (x1,y1), Q(12) = (x1,y2), Q(21) = (x2,y1) and Q(22) = (x2,y2). We can first perform linear interpolation in x-direction and then in y-direction :

   f(x,y1) = ((x2-x)/(x2-x1))*f(Q(11)) + ((x-x1)/(x2-x1))*f(Q(21))
   f(x,y2) = ((x2-x)/(x2-x1))*f(Q(12)) + ((x-x1)/(x2-x1))*f(Q(22))

   f(x,y) = ((y2-y)/(y2-y1)) * f(x,y1) + ((y-y1)/(y2-y1)) * f(x,y2)

Translation, rotation, scaling, and shear transformation, using Nearest Neighbor and bilinear interpolation.

General Affine Transformation :
The general affine transformation can be defined with six parameters. There are six degrees of freedom for affine transformation :
θ : the rotation angle
Tx : the x component of the translation vector
Ty : the y component of the translation vector
Sx : the x component of the scaling vector
Sy : the y component of the scaling vector

shx : the x component of the shearing vector
shy : the y component of the shearing vector

Affine Transformation Matrix :
[a b x0 ; c d y0 ; 0 0 1];

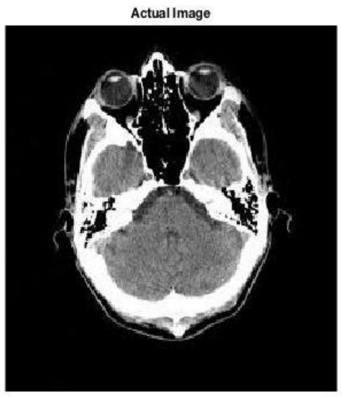
1. Translation using Nearest Neighbor Method and Bilinear Interpolation :

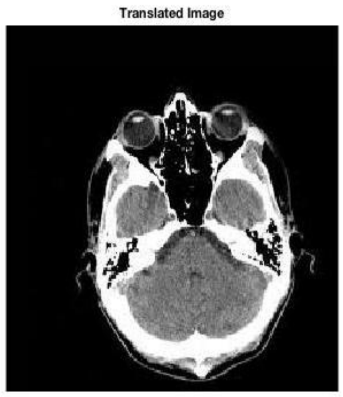
Affine Transformation Matrix for Translation :
[1   0   Tx ;   0   1   Ty ;   0   0   1];

Method Signature:
Translation(I, dx, dy , interpolation)
Where dx and dy are translation parameters and interpolation is the type of interpolation we want to apply.
1 : Nearest Neighbor Interpolation
2 : Bilinear Interpolation

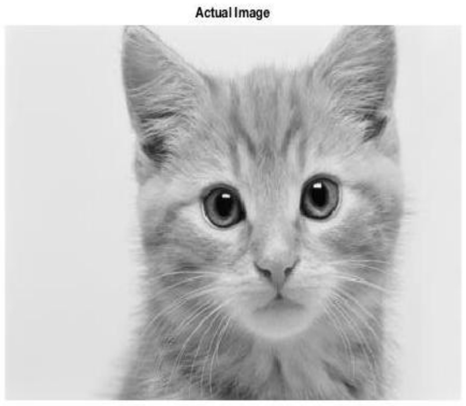
Load the image, convert it to gray scale image and then we apply translation transformation to it :

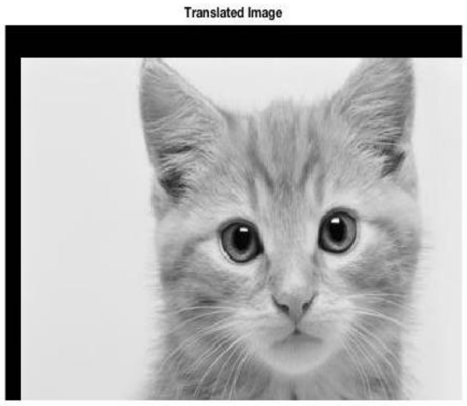
```
>> I = imread('../CTscan.jpg');
>> gray_image = rgb2gray(I);
>> i = Translation(gray_image,15, 25, 1);
```





```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = Translation(gray_image,15, 25, 2);
```





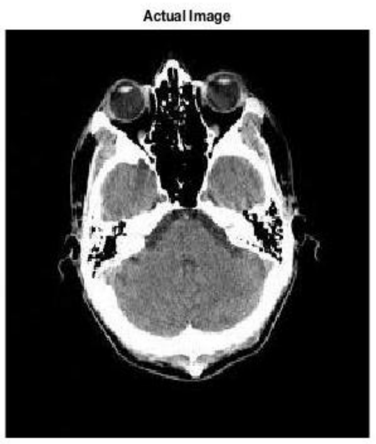
2. Rotation using Nearest Neighbor Method and Bilinear Interpolation :

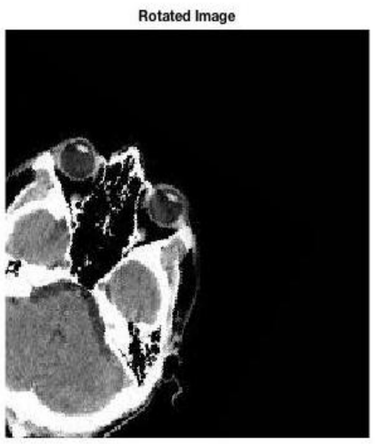
Affine Transformation Matrix for Rotation :
[cos(theta)   -sin(theta)   0 ;   sin(theta)   cos(theta)   0 ;   0   0   1];

Method Signature :
Rotation(InputImage, theta, interpolation)

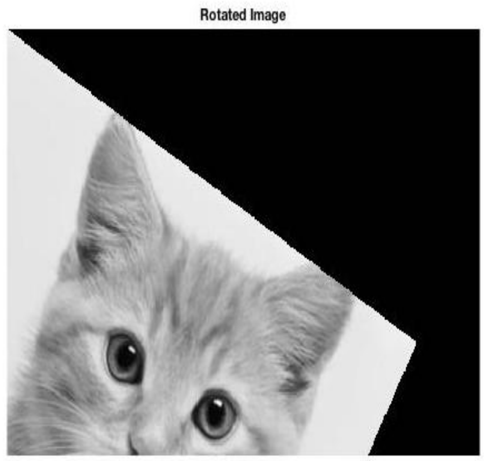
Where theta is the angle in radians for rotation and interpolation is the type of interpolation we want to apply.
1 : Nearest Neighbor Interpolation
2 : Bilinear Interpolation

```
>> I = imread('../CTscan.jpg');
>> gray_image = rgb2gray(I);
>> i = Rotation(gray_image,pi/6, 1);
```





```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = Rotation(gray_image,pi/6, 2);
```



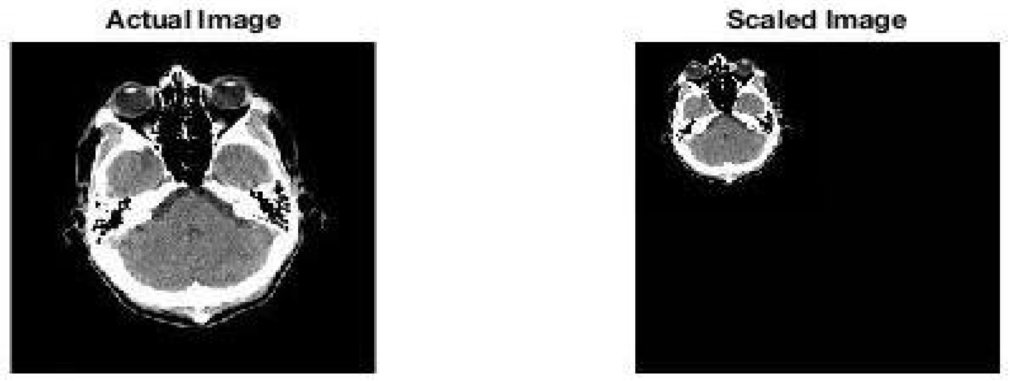
3. Scaling using Nearest Neighbor Method and Bilinear Interpolation :

Affine Transformation Matrix for Scaling :
[sx 0 0 ; 0 sy 0 ; 0 0 1];

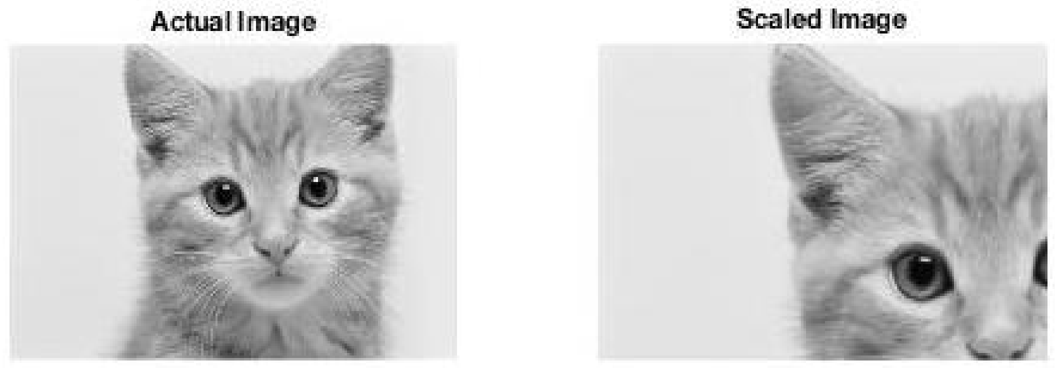
Method Signature :
ScaleImage(InputImage, sx,sy, interpolation)

Where sx and sy are the scaling factors and interpolation is the type of interpolation we want to apply.
1 : Nearest Neighbor Interpolation
2 : Bilinear Interpolation

```
>> I = imread('../CTscan.jpg');
>> gray_image = rgb2gray(I);
>> i = ScaleImage(gray_image, 0.5, 0.5, 1);
```



```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = ScaleImage(gray_image, 1.5, 1.5,  2);
```



4. Shear using Nearest Neighbor Method and Bilinear Interpolation :

Affine Transformation Matrix for Shear (Horizontal) :
[1   shx   0 ;   0   1   0 ;   0   0   1];

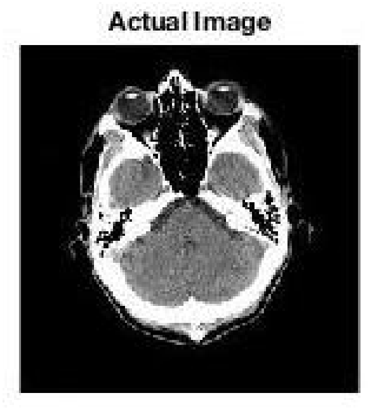
Affine Transformation Matrix for Shear (Vertical) :
[1   0   0 ;   shy   1   0 ;   0   0   1];

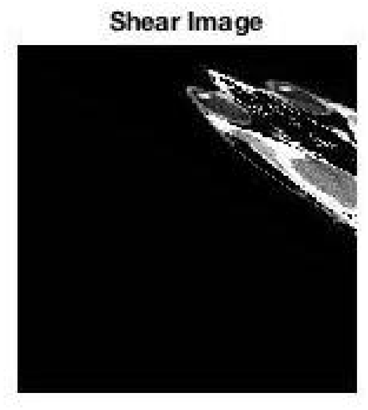
Method Signature :
shearX(InputImage, shx, interpolation);

shearY(InputImage, shy, interpolation);

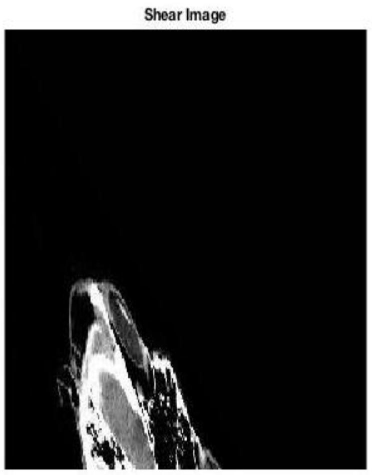
Where shx and shy are the shear factors and interpolation is the type of interpolation we want to apply.
1 : Nearest Neighbor Interpolation
2 : Bilinear Interpolation

```
>> I = imread('../CTscan.jpg');
>> gray_image = rgb2gray(I);
>> i = shearX(gray_image, 1.5 , 1);
```

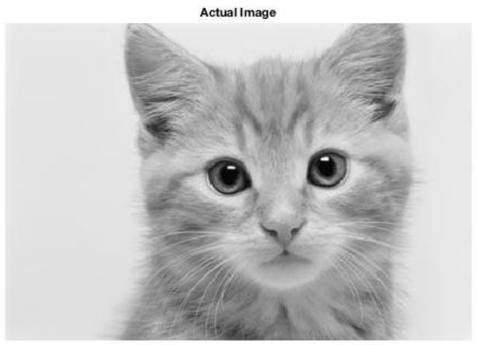


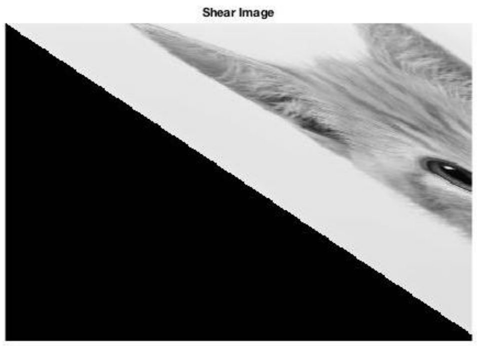


```
>> I = imread('../CTscan.jpg');
>> gray_image = rgb2gray(I);
>> i = shearY(gray_image, 1.5 , 1);
```

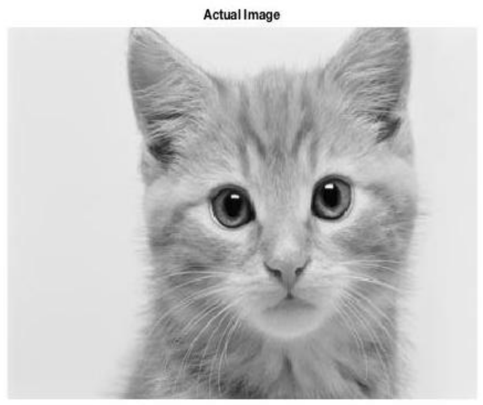


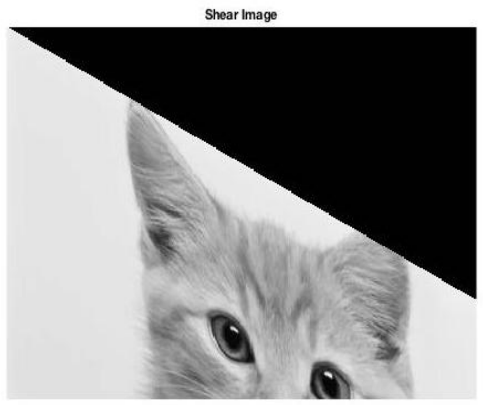
```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = shearX(gray_image, 1.5 , 2);
```





```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = shearY(gray_image, 0.5 , 2);
```



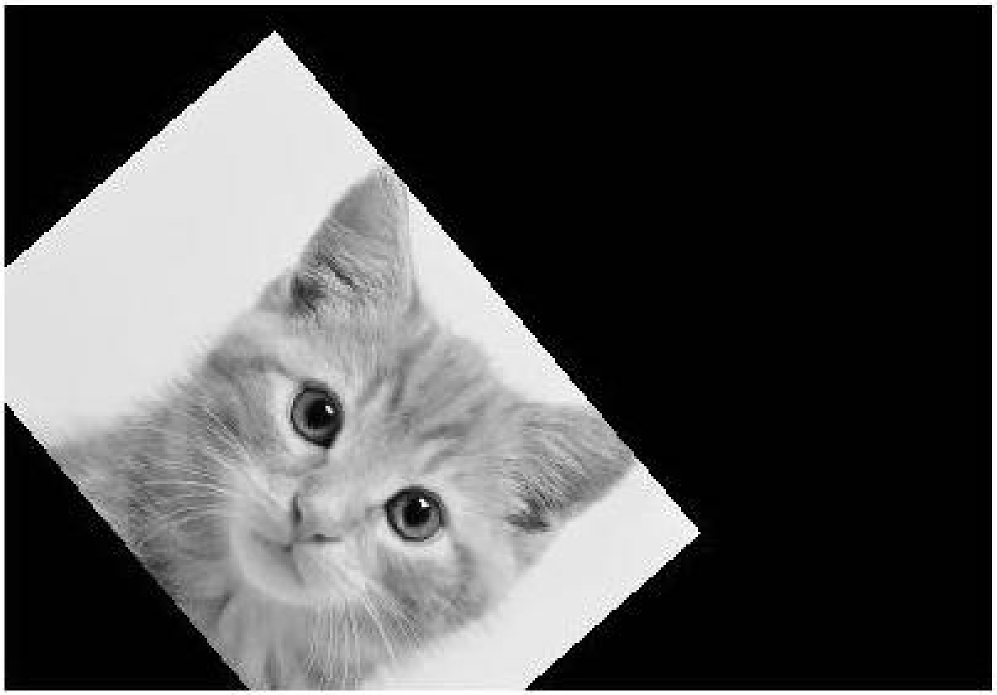


Affine Transformation with six parameters :

We can perform an affine transformation by applying values to all six parameters **.**

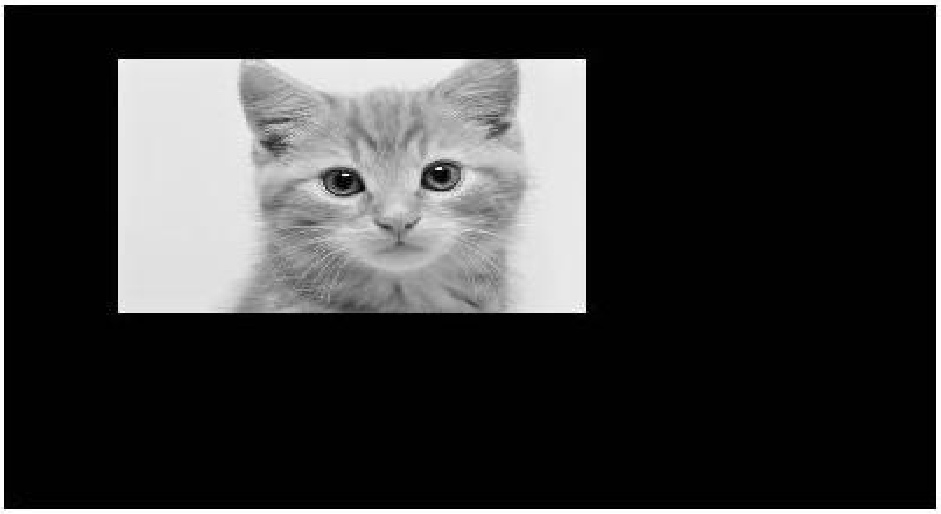
Example:
Using Nearest Neighbor interpolation. Here we are first rotating the image and then translating it.

```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = nnaffine(gray_image, cos(pi/6)*0.5, -sin(pi/6), sin(pi/6), cos(pi/6)*0.5, 110, 10);
>> imshow(uint8(i));
```



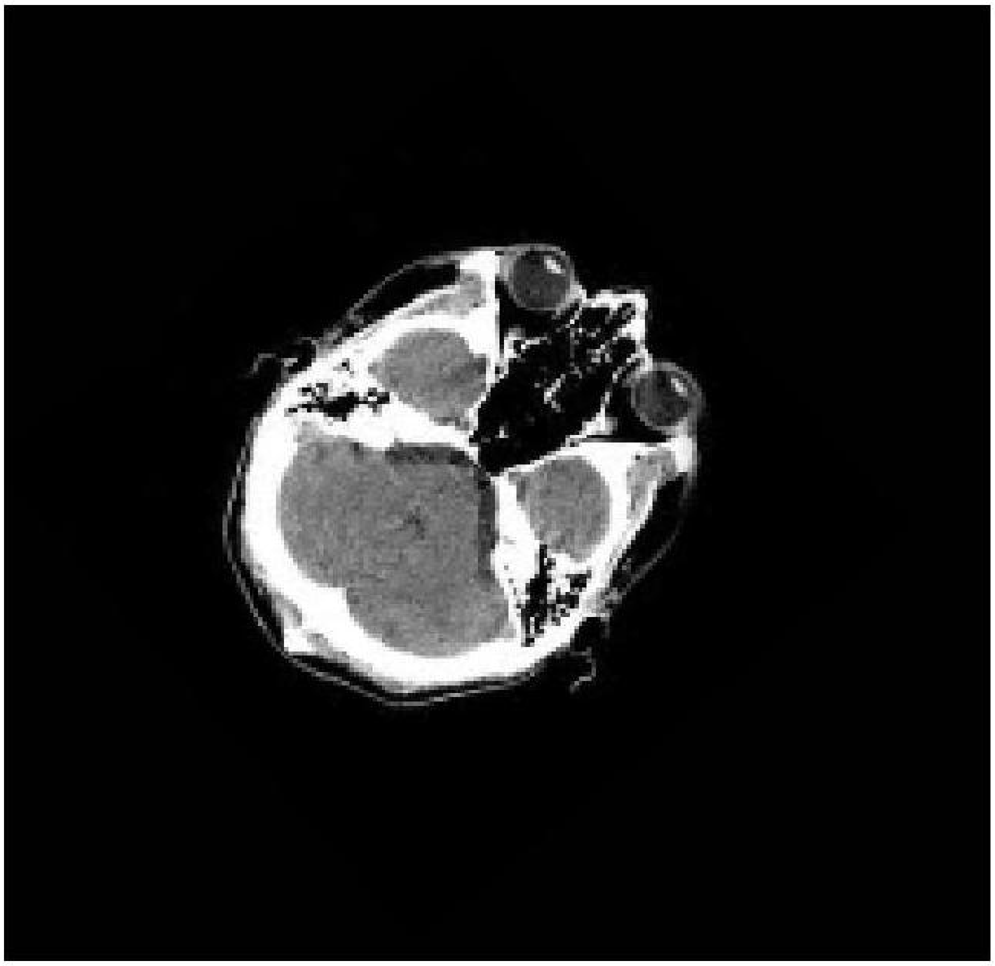
Using bilinear Interpolation. Here we are scaling and translating the image.

```
>> I = imread('../catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = blaffine(gray_image, 0.5 , 0, 0, 0.5, 50, 30);
>> imshow(uint8(i));
```



Other Example :

```
>> I = imread('../CTscan.jpg');
>> gray_image = rgb2gray(I);
>> i = blaffine(gray_image, cos(pi/6)*0.5, -sin(pi/6), sin(pi/6), cos(pi/6)*0.5, 110, 10);
>> imshow(uint8(i));
```



For the affine transformation, demonstrate the differences between nn and bilinear, best by large zooming of an image subregion). Provide a short description on what you did and what you see.

Applying nearest neighbor and bilinear interpolation and zooming the image for comparison :

```
>> I = imread('/Users/ishitaverma/ishitaverma/catImage.jpg');
>> gray_image = rgb2gray(I);
>> i = nnaffine(gray_image, cos(pi/6)*0.5, -sin(pi/6), sin(pi/6), cos(pi/6)*0.5, 110, 10);

>> T = imread('/Users/ishitaverma/ishitaverma/catImage.jpg');
>> gray_image = rgb2gray(T);
>> t = blaffine(gray_image, cos(pi/6)*0.5, -sin(pi/6), sin(pi/6), cos(pi/6)*0.5, 110, 10);
```

Zoomed image for nn :

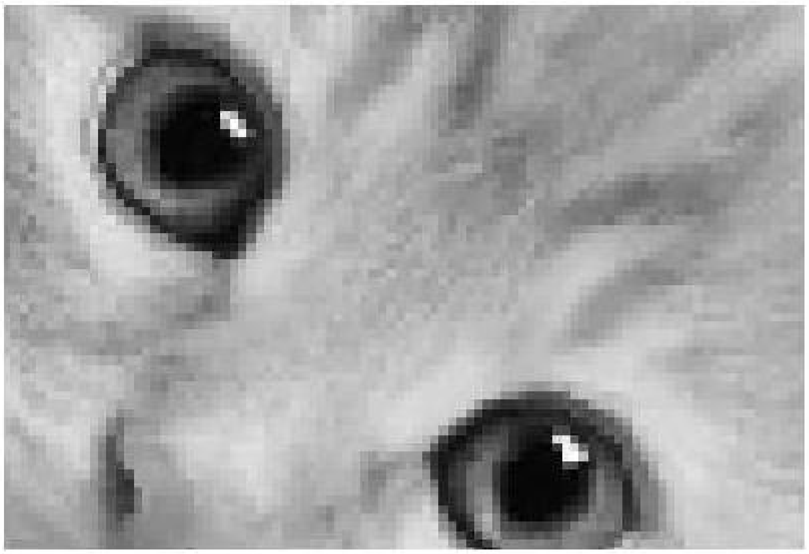


Figure 1

Zoomed image for bilinear interpolation :

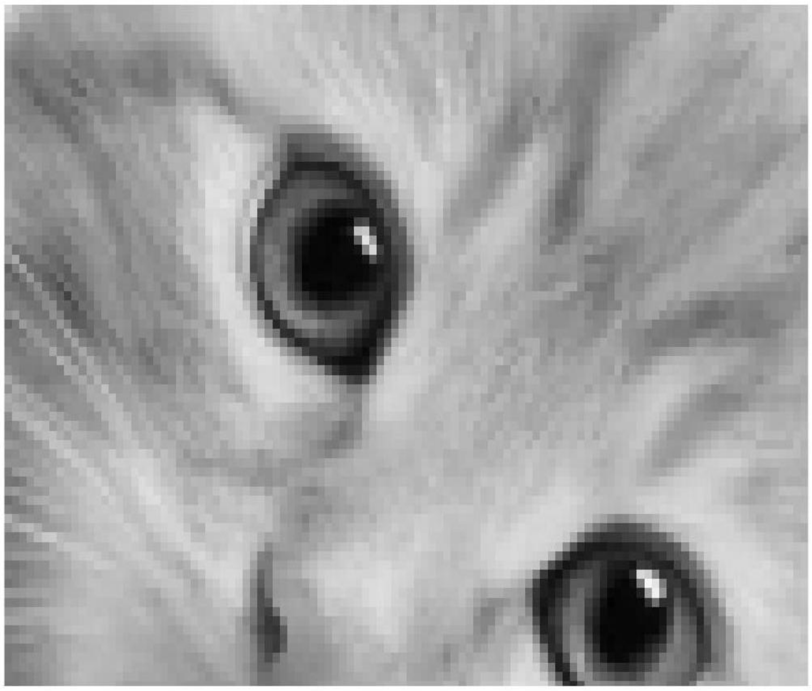


Figure 2

The nearest neighbor image is shown in Figure 1 and the bilinear image is shown in Figure 2. The area between the eyes of cat appears is the most distinguishable difference in these two images. If the zoom is enough, the changes are apparent. The one with bilinear interpolation is smoother than the nearest neighbor interpolation. The bilinear inter polant is better at preserving overall detail and continuity of an image.

1.2 Calculation of affine transform from landmarks

In this portion of the project we have to load two images (a source and target), and selects registration or landmark points highlighting similar points in the two images. The user can then select to solve for a linear transformation which will attempt to transform the source image into the target image. I have used matlab module that gives pixel positions by clicking locations with the mouse. I have selected 3 points from the source image and three corresponding points from the target image for transformation.

Steps :

1. Select the points from source and target image :

   This can be done in matlab using *getpts* library function.

2. We can use the points obtained from source and target images to create A and Y matrices and then solve for X to obtain six affine parameters.

   AX = Y

3. We can use gauss elimination to get the affine parameters.

4. Once we get affine parameters we can apply any (nn or bilinear) interpolation for transfroming source to target image.

Methods Used :

1. landmarkT(S, T, interpolation)

Where S is source, T is target and interpolation is the type of interpolation we want to apply.

1 : Nearest Neighbor Interpolation
2 : Bilinear Interpolation

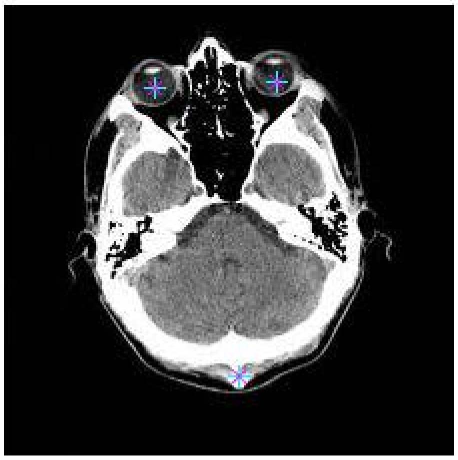
2. gauss_elimination(A,b)

Where A and b are the matrices used for calculation of unknown affine parameters.

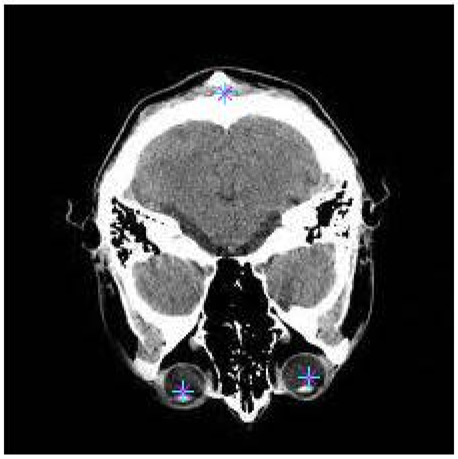
We will first select three points from source image and then corresponding three points from target image and then apply affine transformation using nearest neighbor interpolation method.

```
>> S = imread('../CTscan.jpg');
>> a = rgb2gray(S);
>> i1 = im2double(a);
>> T = imread('../CTscanR.jpg');
>> b = rgb2gray(T);
>> i2 = im2double(b);
>> o = landmarkT(i1,i2,1);
>> imshow(o);
```

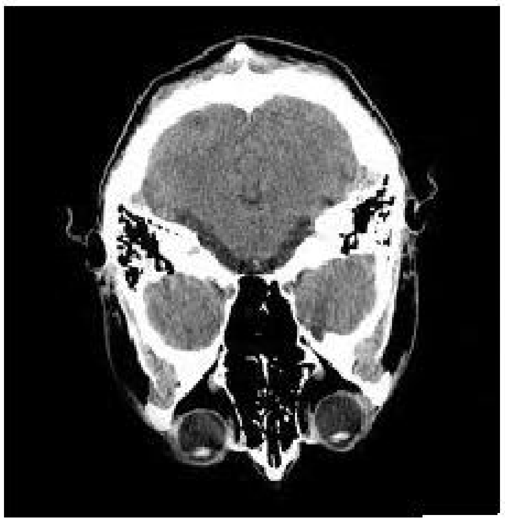
Source Image :



Target Image :



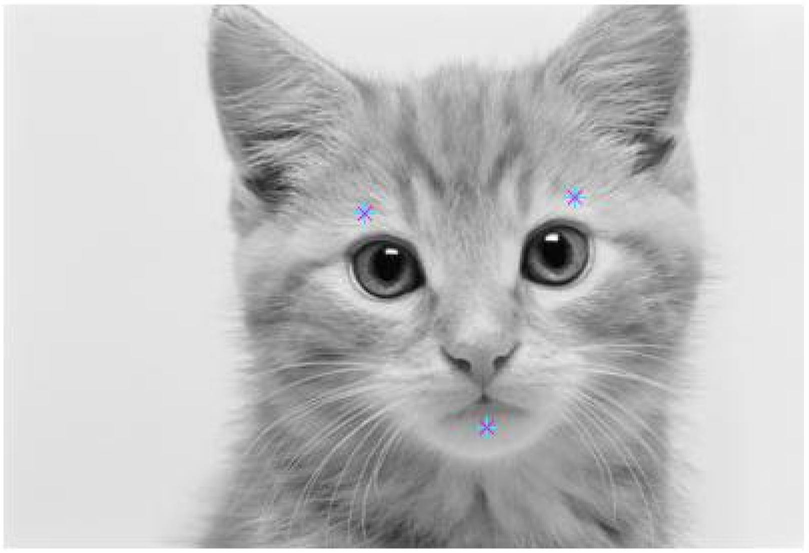
Final Transformed Image :



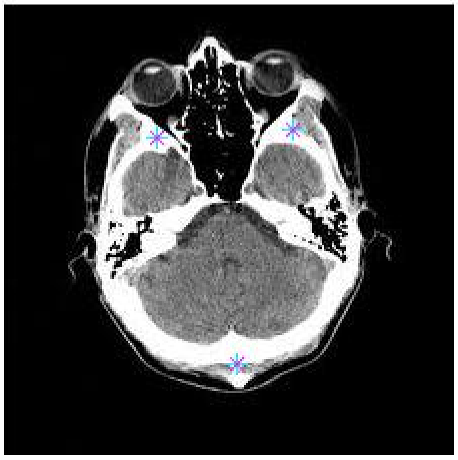
Other Example :

```
>> S = imread('../catImage.jpg');
>> a = rgb2gray(S);
>> i1 = im2double(a);
>> T = imread('../CTscan.jpg');
>> b = rgb2gray(T);
>> i2 = im2double(b);
>> o = landmarkT(i1,i2,2);
>> imshow(o);
```

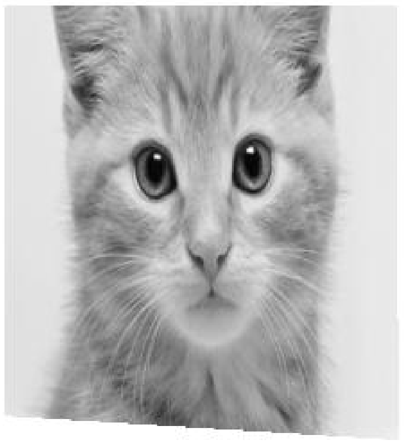
Source Image



Target Image



Final Translated Image :



Matlab Implementation of functions used for this portion of project :

1. Nearest Neighbor Interpolation: nnaffine takes the six affine parameter, creates affine transformation matrix and returns the transformed image back.

```
 function [OutputImage] = nnaffine( InputImage, a, b, c, d, x0, y0 )

 OutputImage = ones(size(InputImage,1),size(InputImage,2));

 T = [a b x0; c d y0; 0 0 1];
 for i = 1 : size(InputImage,1)
   for j = 1 : size(InputImage,2)
       point = [j i 1]';
       new_v = T\point;
       intensity = NearestNeighbor(InputImage,new_v);
       OutputImage(i,j) = intensity;
   end
 end
end
```

```
function [ value ] = NearestNeighbor(Image, Point)
nearest_point = round(Point);
value = 1;
if(nearest_point(2) > 0 && nearest_point(2) < size(Image,1) && nearest_point(1) > 0 &&
nearest_point(1) < size(Image,2))
   value = Image(nearest_point(2) , nearest_point(1));
end
end
```

2. Bilinear Interpolation : blaffine takes the six affine parameter, creates affine transformation matrix and returns the transformed image back.

```
function [OutputImage] = blaffine( InputImage, a, b, c, d, x0, y0 )

  OutputImage = ones(size(InputImage,1),size(InputImage,2));
  T = [a b x0; c d y0; 0 0 1];
  T
  for i = 1 : size(InputImage,1)
    for j = 1 : size(InputImage,2)
      point = [ j i 1]';
      point = T\point;
      point = [point(1)/point(3) point(2)/point(3)];
      OutputImage(i,j) = bilinear(InputImage, point);
    end
  end
end


function [value] = bilinear( I, point )
  value = 1;
  value1 = -1;
  value2 = -1;

  if(size(point,2) == 1 && size(point,1) == 2)
      point = point';
  end

  b1 = floor(point);
  b2 = b1 + [0 1];
```

```
b3 = b1 + [1 1];
b4 = b1 + [1 0];

t = point - b1;

if(b1(2) > 0 && b1(2) <= size(I,1) && b1(1) > 0 && b1(1) <= size(I,2))
  if(b4(1) <= size(I,2))
    value1 = (1-t(1))*I(b1(2),b1(1)) + t(1)*I(b4(2),b4(1));
  else
    value1 = I(b1(2),b1(1));
  end
else
  if(b4(2) > 0 && b4(2) <= size(I,1) && b4(1) > 0 && b4(1) <= size(I,2))
    value1 = I(b4(2),b4(1));
  end
end
if(b2(2) > 0 && b2(2) <= size(I,1) && b2(1) > 0 && b2(1) <= size(I,2))
  if(b3(1) <= size(I,2))
    value2 = (1-t(1))*I(b2(2),b2(1)) + t(1)*I(b3(2),b3(1));
  else
    value2 = I(b2(2),b2(1));
  end
else
  if(b3(2) > 0 && b3(2) <= size(I,1) && b3(1) > 0 && b3(1) <= size(I,2))
    value2 = I(b3(2),b3(1));
  end
end
if(value1 == -1 || value2 == -1)
  if(value1 == -1 && value2 == -1)
    value = 1;
  else
    if(value1 ~= -1 && value2 == -1)
      value = value1;
    else
        if(value1 == -1 && value2 ~= -1)
          value = value2;
        end
    end
  end
  else
    value = (1-t(2))*value1 + t(2)*value2;
  end
end
```

Functions for transformation , shearing , scaling and rotation :

Translation :

```
function [O] = Translation( I, dx, dy , interpolation)
   if (interpolation == 2)
       O = blaffine(I, 1, 0, 0, 1, dx, dy);
   else
       O = nnaffine(I, 1, 0, 0, 1, dx, dy);
   end

    subplot(2,2,1) , imshow(I) ;
    title("Actual Image");
    subplot(2,2,2) , imshow(uint8(O));
    title("Translated Image");
end
```

Rotation :

```
function [OutputImage] = Rotation( InputImage, theta, interpolation)
  if(interpolation == 2)
    OutputImage  = blaffine(InputImage, cos(theta), -sin(theta), sin(theta), cos(theta), 0,
0);
  else
    OutputImage = nnaffine(InputImage, cos(theta), -sin(theta), sin(theta), cos(theta), 0,
0);
  end

    subplot(2,2,1) , imshow(InputImage) ;
    title("Actual Image");
    subplot(2,2,2) , imshow(uint8(OutputImage));
    title("Rotated Image");
end
```

Scaling :

```
function [OutputImage] = ScaleImage( InputImage, cx, cy, interpolation)
  if(interpolation == 2)
    OutputImage = blaffine(InputImage, cx, 0, 0, cy, 0, 0);
  else
    OutputImage = nnaffine(InputImage, cx, 0, 0, cy, 0, 0);
  end

    subplot(2,2,1) , imshow(InputImage) ;
    title("Actual Image");
    subplot(2,2,2) , imshow(uint8(OutputImage));
    title("Scaled Image");
end
```

Shearing :

```
function [OutputImage] = shearX( InputImage, shx, interpolation)
  if(interpolation == 2)
    OutputImage = blaffine(InputImage, 1, shx, 0, 1, 0, 0);
 else
    OutputImage = nnaffine(InputImage, 1, shx, 0, 1, 0, 0);
  end

  subplot(2,2,1) , imshow(InputImage) ;
  title("Actual Image");
  subplot(2,2,2) , imshow(uint8(OutputImage));
  title("Shear Image");
end

function [OutputImage] = shearY(InputImage, shy, interpolation)
  if(interpolation == 2)
    OutputImage = blaffine(InputImage, 1, 0, shy, 1, 0, 0);
  else
    OutputImage = nnaffine(InputImage, 1, 0, shy, 1, 0, 0);
  end

  subplot(2,2,1) , imshow(InputImage) ;
  title("Actual Image");
  subplot(2,2,2) , imshow(uint8(OutputImage));
  title("Shear Image");
end
```

Translation using landmarks :

```
function [outputImage] = landmarkT( S, T, interpolation)

figure(121)
imshow(S);
[x,y] = getpts;
figure(122)
imshow(T)
[x2,y2] = getpts;

X = [ x(1),y(1),1,0,0,0;
      0,0,0,x(1),y(1),1;
      x(2),y(2),1,0,0,0;
      0,0,0,x(2),y(2),1;
      x(3),y(3),1,0,0,0;
      0,0,0,x(3),y(3),1]
X2=[x2(1),y2(1),x2(2),y2(2),x2(3),y2(3)];
p = gauss_elimination(X,X2);

if(interpolation == 2)
    outputImage = blaffine(S, p(1),p(2),p(4),p(5),p(3),p(6));
else
    outputImage = nnaffine(S, p(1),p(2),p(4),p(5),p(3),p(6));
end
end


function [x] = gauss_elimination(A,b)
n=length(b);

for p=1:n
    vec=[(1:p-1) n (p:n-1)];
    test=1;
    while A(p,p)==0
```

```
        if test==n
            error('Cannot invert matrix')
        end
        A=A(vec,:);
        b=b(vec);
        test=test+1;
    end
    b(p)=b(p)/A(p,p);
    A(p,:)=A(p,:)/A(p,p);
    for q=p+1:n
        b(q)=b(q)-A(q,p)*b(p);
        A(q,:)=A(q,:)-A(q,p)*A(p,:);
    end
end

x=zeros(n,1);
x(n)=b(n);
for p=n-1:-1:1
    x(p)=b(p);
    for q=p+1:n
        x(p)=x(p)-A(p,q)*x(q);
    end
end
```

Problem 2: Hough Transform for straight lines without edge orientation

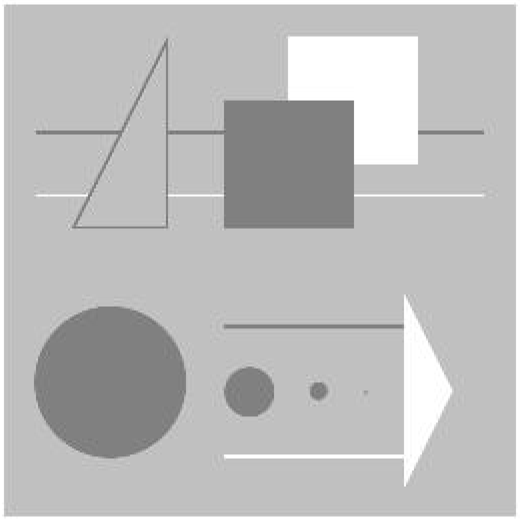
The **Hough transform** is a feature extraction technique used in **image** analysis, computer vision, and digital **image** processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure.

PreProcessing of Image for Hough Transform :

We will first pre process our image using Edge Detection function that we have already implemented in previous project. After that we will do the non max suppression and thresholding of the image to obtain the peak points.

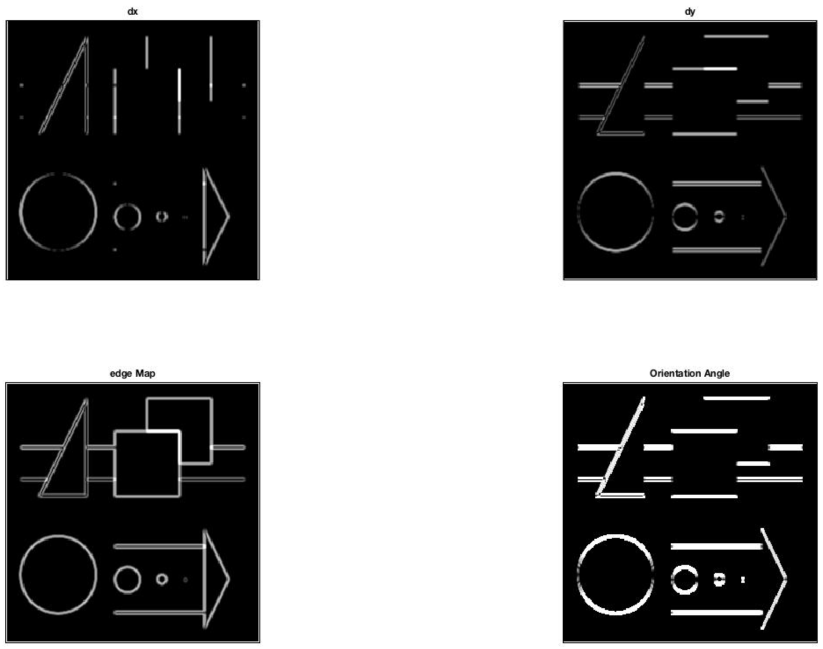
EdgeDetection of Image :

Original Image :



```
>> I = imread('../edges-lines-orig.png');
>> gray_image = rgb2gray(I);
>> w1 = [-1 0 1 ];
>> w2 = [-1 ; 0 ; 1];
>> [a,b] = edgeDetection(gray_image,w1,w2);
```

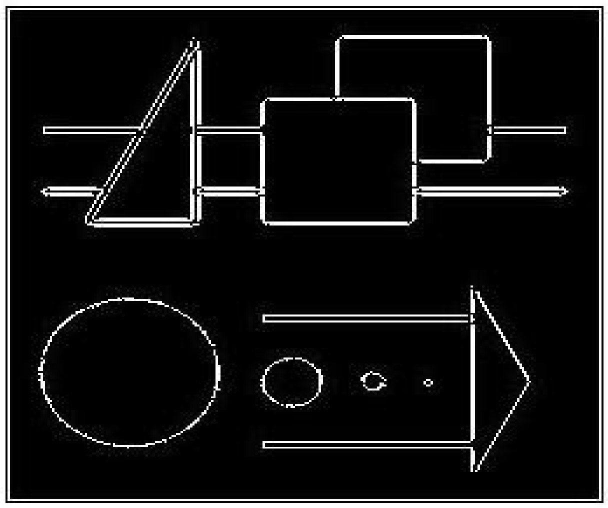
W1 and w2 are the horizontal and vertical filters. edgeDetection will return a : EdgeMap and b : OrientationMap for the Input Image.



Non Maximum Suppression of above obtained result :

```
>> nm = non_maxSuppression(a,b);
>> th = threshold(nm,0.95);
>> imshow(th,gray);
```

Thresholded Image :



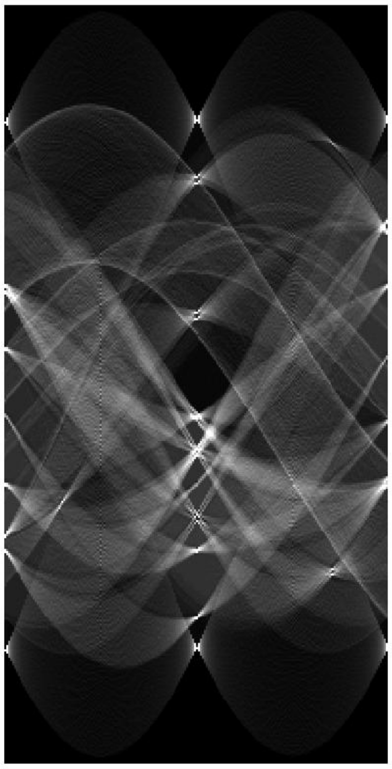
This thresholded image can be given as input to the Hough Transform function.

The **transform** is implemented by quantizing the **Hough** parameter space into finite intervals or **accumulator** cells. As the algorithm runs, each is transformed into a discretized curve and the **accumulator** cells which lie along this curve are incremented.

Hough Transform :

```
>> [x,y,z] = HoughTransform(th , gray_image , 50);
```

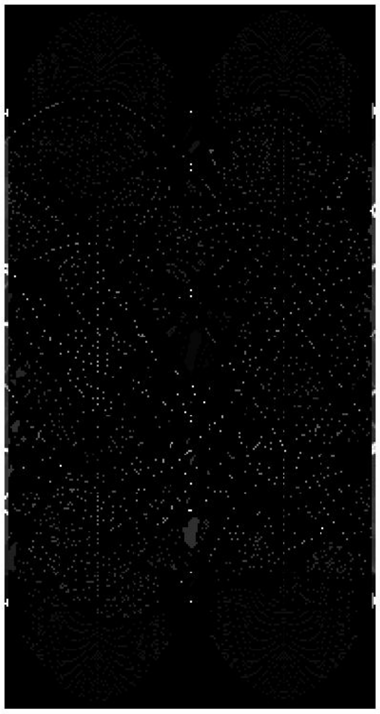
Hough Accumulator :



After calculation of the HT for our image points and accumulation of curves, we can detect the N largest maxima :

nonmaxsupRadial(I , n) : We are using this function to suppress all pixels which are not the maximum in a square of dimension n about itself. It makes use of helper function IsLocalMaxRadial.

Non Maximum Suppression of accumulation Map :



Thresholded Map :

The rho and theta values for threshold value of 50 :

Pairs :

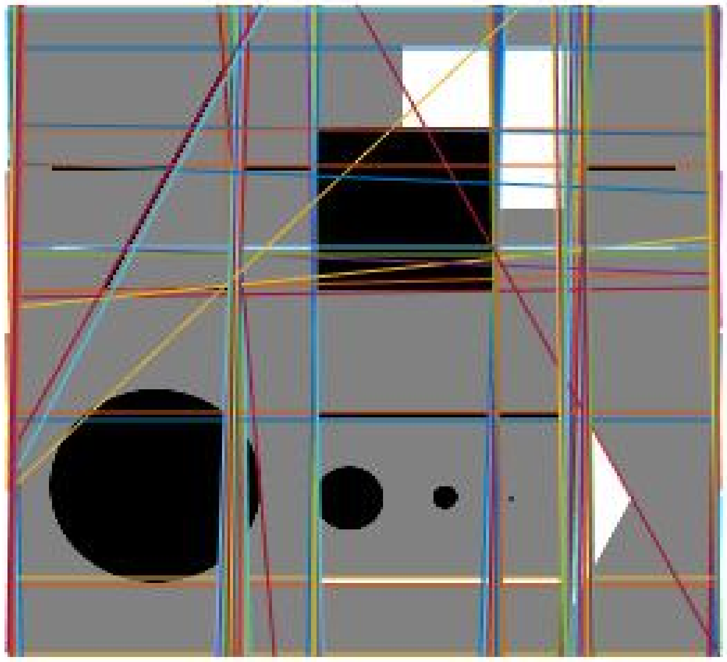
```
(-129,3.1259)
(-128,0.015708)
(-128,3.1259)
(-127,0.015708)
(-127,1.5708)
(-127,3.1259)
(-126,0)
(-126,0.015708)
(-126,3.1259)
(-125,0.015708)
(-100,1.5708)
(-97,1.5708)
(-79,3.1259)
(-78,3.1102)
(-78,3.1259)
(-77,3.1102)
(-76,3.0945)
(-76,3.1102)
(-75,3.1102)
(-75,3.1259)
```

(-74,3.1102)
(-74,3.1259)
(-73,3.1259)
(-48,0)
(-48,0.015708)
(-48,3.1259)
(-47,0)
(-47,0.015708)
(-47,3.1102)
(-47,3.1259)
(-46,3.1102)
(-45,0)
(-45,0.015708)
(-44,0)
(-44,0.015708)
(-43,0.015708)
(-42,0.07854)
(-35,1.5708)
(-32,1.5708)
(-18,0)
(-18,0.015708)
(-17,0)
(-17,0.015708)
(15,1.5865)
(18,3.1259)
(19,1.6022)
(23,1.6808)
(28,1.5237)
(30,1.5708)
(33,1.5708)
(45,3.1102)
(45,3.1259)
(46,3.1102)
(46,3.1259)
(47,0)
(47,0.015708)
(47,3.1102)
(47,3.1259)
(48,0)

```
(48,0.015708)
(48,2.3719)
(48,3.1102)
(48,3.1259)
(49,3.1102)
(56,0.47124)
(60,1.5237)
(65,1.5708)
(70,0.015708)
(71,0.015708)
(72,0)
(73,0)
(79,0)
(79,1.5551)
(79,1.5708)
(80,0)
(80,0.015708)
(81,0.015708)
(88,2.6861)
(91,2.6704)
(111,1.5708)
(124,3.1259)
(125,0.015708)
(125,3.1259)
(126,0.015708)
(126,1.5708)
(126,3.1259)
(127,0)
(127,0.015708)
(127,3.1259)
(128,0.015708)
```

We have calculated the corresponding N straight lines and will draw these lines back into the image space for comparison with the original image using overlay concept:

Final Result :

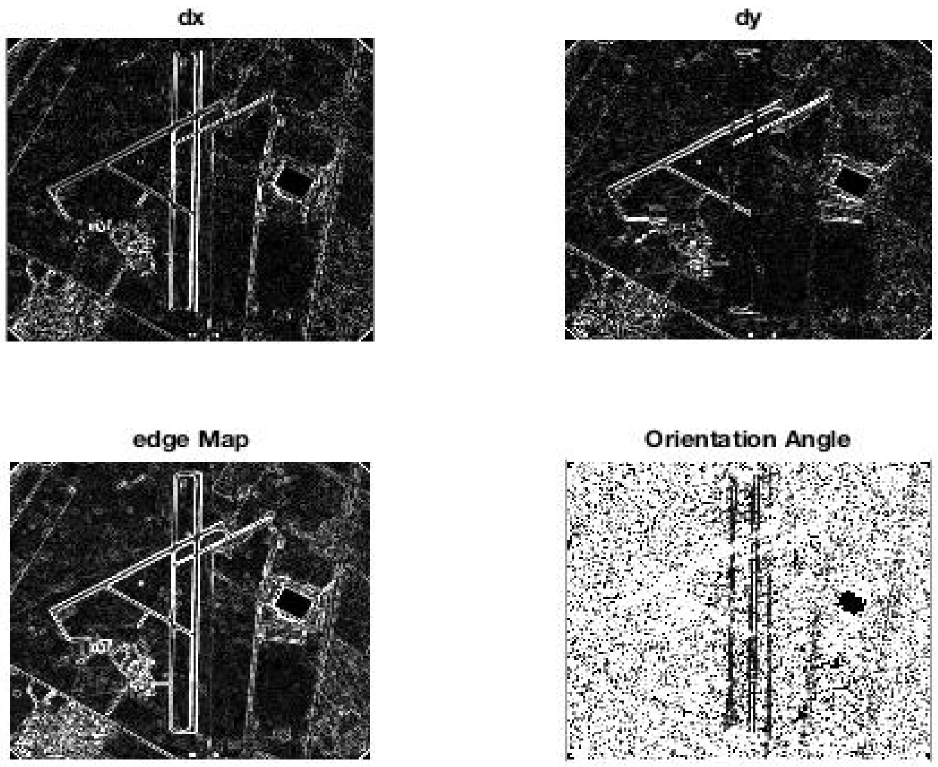


Hough Transform for Second Image (Airport) :

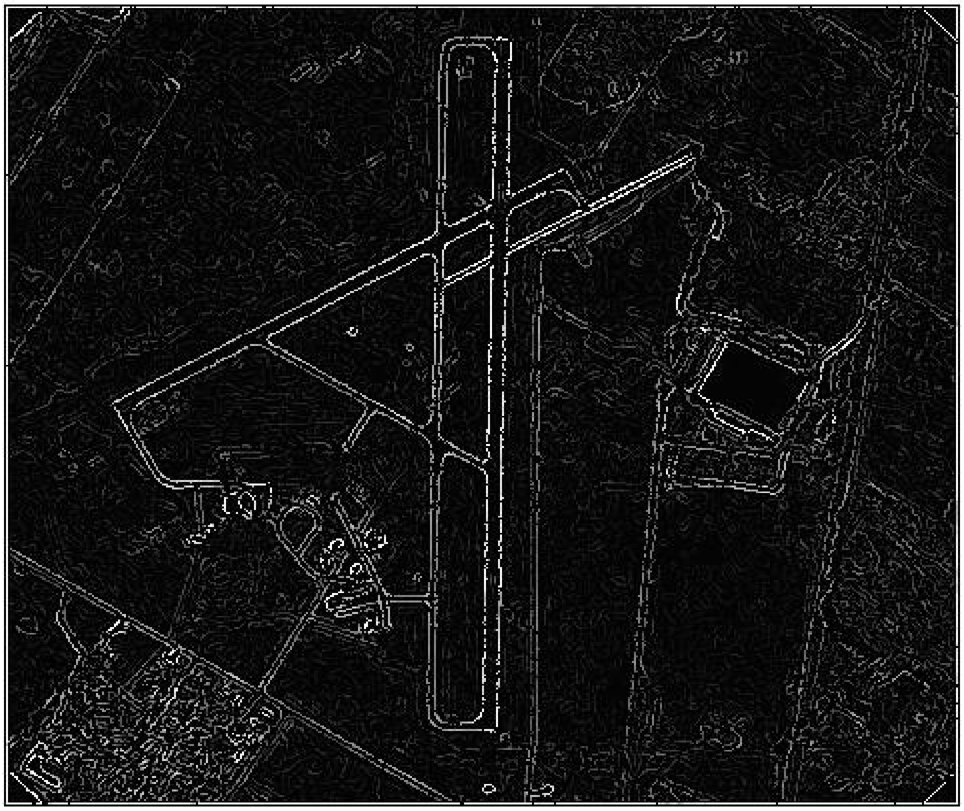
```
>> gray_image = imread('../mnn4-runway-Ohio.jpg');
>> w1 = [-1 0 1 ];
>> w2 = [-1 ; 0 ; 1];
>> [a,b] = edgeDetection(gray_image,w1,w2);
>> nm = non_maxSuppression(a,b);
>> th = threshold(nm,0.95);
>> imshow(th,gray);
```

Original Image :

Edge Detection :



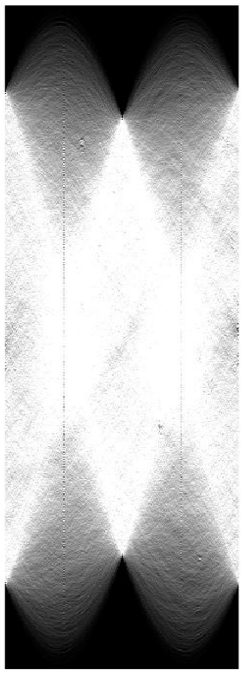
Thresholded Image :



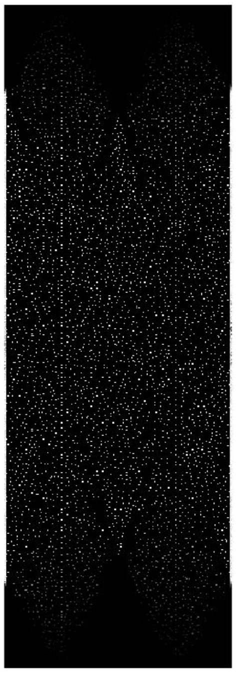
Hough Transform :

```
>> [x,y,z] = HoughTransform(th , gray_image , 130);
```

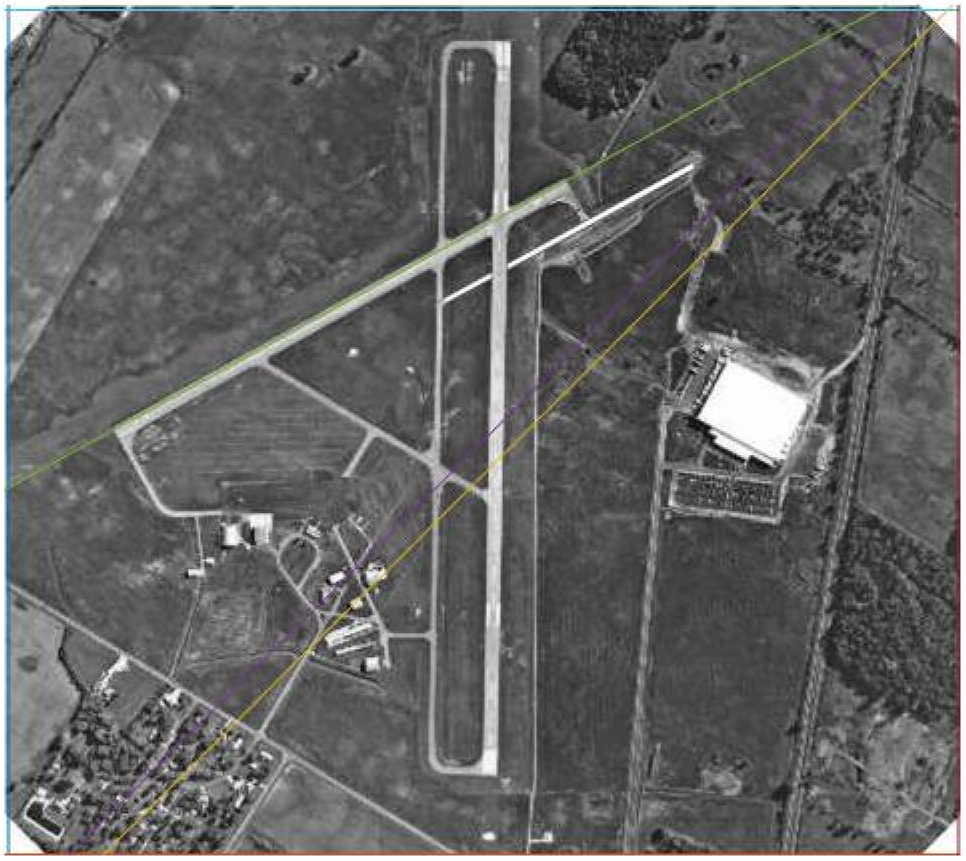
Hough Accumulator :



Non Maximum Suppression of accumulation Map :



Final Image :



Matlab Implementation for hough Transform :

```
function [Pout,P,Th_Image] = HoughTransform( EdgeMap , InputImage ,thres)
    maxRho = round(sqrt((size(EdgeMap,1)/2)^2 + (size(EdgeMap,2)/2)^2) + 1);
    minRho = -maxRho;
    P = zeros(maxRho-minRho,200);
    cntrX = size(EdgeMap,2) / 2;
    cntrY = size(EdgeMap,1) / 2;

    for i =1:size(EdgeMap,1)
        for j =1:size(EdgeMap,2)
            if(EdgeMap(i,j) > 1)
                x = j - cntrX;
                y = size(EdgeMap,1) - i - cntrY;
                for tCount = 0:199
                    rho = floor(x*cos(tCount*pi/200) + y*sin(tCount*pi/200));
                    rho = rho - minRho + 1;
                    P(rho,tCount+1) = P(rho,tCount+1) + 1;
                end
            end
        end
    end

    mnP = min(min(P));
    mxP = max(max(P));
    Pout = zeros(size(P,1),size(P,2));

    for i=1:size(P,1)
        for j=1:size(P,2)
            Pout(i,j) = (255/(mxP-mnP+1))*(P(i,j)-mnP+1);
        end
    end

    P = nonmaxsupRadial(P,5);
    Th_Image = threshold(P,thres);

    hold on;
    imshow(InputImage,[min(min(InputImage)) max(max(InputImage))]);
    hold on;
    for row=1:size(Th_Image,1)
        for col = 1:size(Th_Image,2)
            if(Th_Image(row,col) > 1)
                theta = (col-1)*pi/200;
                rho = row - 1 + minRho;
                x = -cntrX:1:cntrX;
                disp(strcat('(', num2str(rho), ',', num2str(theta), ')'));
                if(abs(sin(theta)) < eps)
                    x=ones(1,2*cntrY+1)*rho;
                    y=-cntrY:1:cntrY;
                    x = x + cntrX;
                    y = size(EdgeMap,1) - y - cntrY;
                else
```

```
                y = -x*cot(theta) + rho/sin(theta);
                x = x + cntrX;
                y = size(EdgeMap,1) - y - cntrY;
            end
            hold on;
            plot(x,y);
            hold on;
        end
    end
  end
  axis([0 2*cntrX 0 2*cntrY]);
end
```

Function for non maximum Suppression :

```
function output = nonmaxsupRadial( I , n )
  O = zeros(size(I,1),size(I,2));

  %edge cases preserved
  for i=1:size(I,1)
    for j=1:size(I,2)
        O(i,j) = I(i,j);
    end
  end

  for row=1+floor(n/2):size(I,1)-floor(n/2)
      for col=1+floor(n/2):size(I,2)-floor(n/2)
          if(IsLocalMaxRadial(I(row-floor(n/2):row+floor(n/2),col-floor(n/2):col+floor(n/2)),
n))
            O(row,col) = I(row,col);
          else
            O(row,col) = 0;
          end
      end
  end

  mnO = min(min(O));
  mxO = max(max(O));
  output = zeros(size(O,1),size(O,2));
```

```
    for i=1:size(O,1)
        for j=1:size(O,2)
            output(i,j) = (255/(mxO-mnO+1))*(O(i,j)-mnO+1);
        end
    end
end
```

Helper function IsLocalMaxRadial :

```
function tf = IsLocalMaxRadial( p, size )

maxValue = max(max(p));
tf = (p(floor(size/2)+1,floor(size/2)+1) == maxValue);

end
```

Threshold Function :

```
function threshold_image = threshold( InputImage, threshold)
  OutputImage = ones(size(InputImage,1),size(InputImage,2));

  for i = 1 : size(InputImage,1)
    for j = 1 : size(InputImage,2)
        if(InputImage(i,j) >= threshold)
            OutputImage(i,j) = InputImage(i,j);
        end
    end
  end
max_thresh = max(OutputImage(:));
threshold_image = uint8((double(OutputImage) ./ max_thresh) .* 255);
end
```