COMPUTER VISION
PROJECT 2
SPRING 2018

PROF. GUIDO GERIG

SUBMITTED BY:
ACHYUT TIBREWALLA
Net ID: at3617

INTRODUCTION

In this project, I have implemented a program that transforms an image given an affine transformation (6 parameters), which includes rotation, translation, scaling and shear. All of the mentioned transformations were done separately also. The transformation is applied from the target image backwards to the source image, i.e. I step through each pixel of the new image, determine the position in the source image, and calculate the intensity at this pixel to be used for the target image. Two types of interpolations methods were used viz., Nearest neighbor and Bi-Linear.

A transformation can be determined based on a set of corresponding landmarks in a source and a target image. A minimum of 3 points with (x,y) coordinates is required, but more landmarks
result in a more stable solution by solving an over constrained linear equation system. I have implemented a module that gives pixel positions by clicking locations with the mouse and then used this module to create sets of corresponding pixel positions in a source and a target image. Then I solved the linear system equation to get the parameters for affine transformation.

The last part of the project constitutes detecting straight lines in an image by Hough transform by using the parametric form of the line.

1. Affine Image Transform

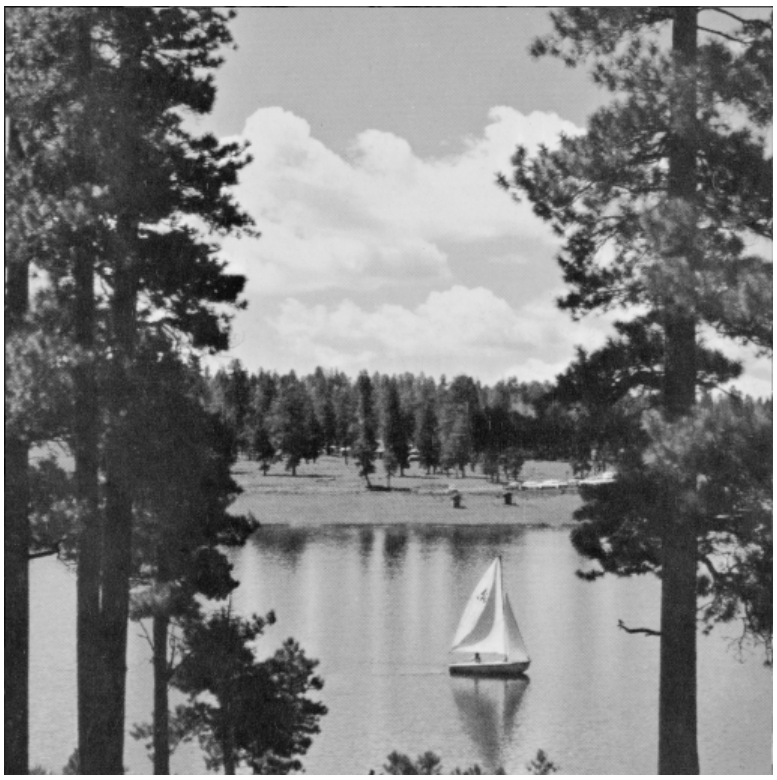
1.1 Resampling

In resampling, we took the input image and apply backward transform to get the pixel intensity in the target image. We use backward transformation because forward transform doesn't yield good result since we will not be able to find an intensity value for each pixel in the target image. For backward mapping, I have implemented two types of interpolation viz., Nearest Neighbor(NN) and Bilinear. For NN, I traverse through each pixel in the target image and find a corresponding point in the source image by taking the nearest neighbor of the point we calculated. For Bilinear, we take the weighted average of the intensity 4 neighboring pixels of the point. We take the inverse of the transformation matrix and multiply it with each pixel in the target image to get a point in the source image. For NN interpolation, the image will be pixelated and for Bilinear the image will appear smoothened. The results are shown and discussed below.

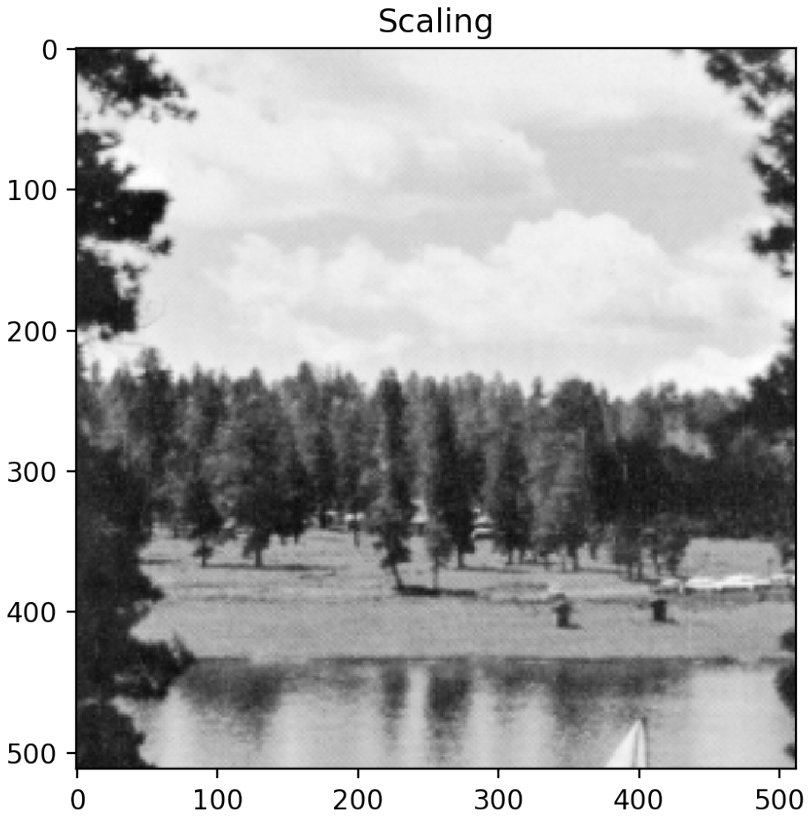
Algorithm:

1. Read the input image.
2. Initialize the transformation matrix according to the transformation to be applied.
3. Traverse through each pixel of the target image and multiply each index with the inverse of the transformation matrix to get an index position in the original image.
4. For NN interpolation, round the value of the index and take the corresponding intensity value.
5. For Bilinear interpolation, take the weighted average of the 4-neighboring pixel and calculate the corresponding intensity value.
6. Display the transformed image.
7. For the transformations, I shifted the axes center to the image center.

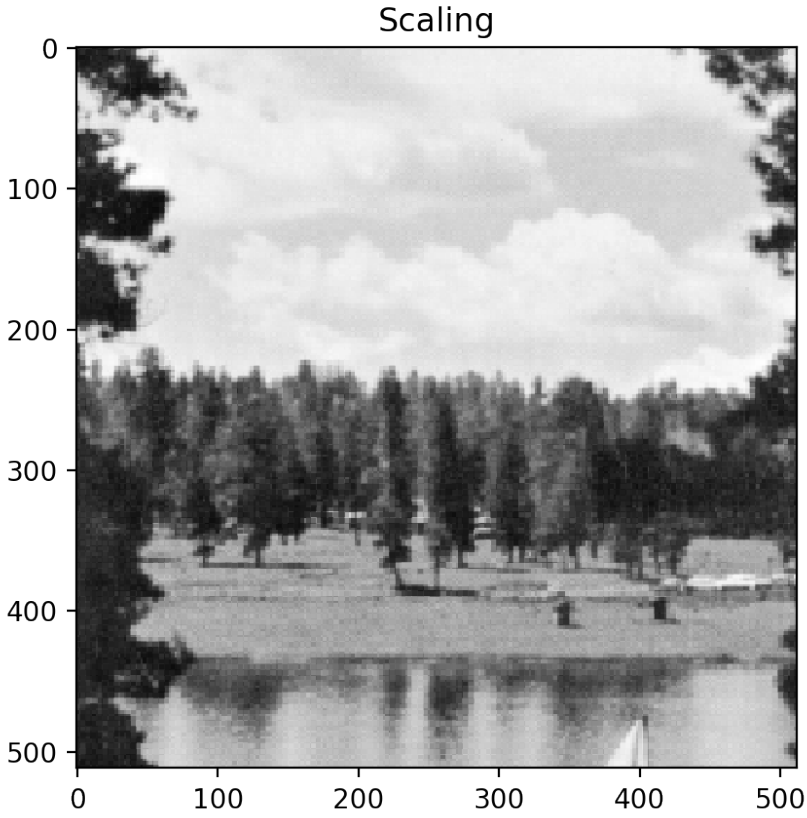
INPUT IMAGE:



Scaling: -

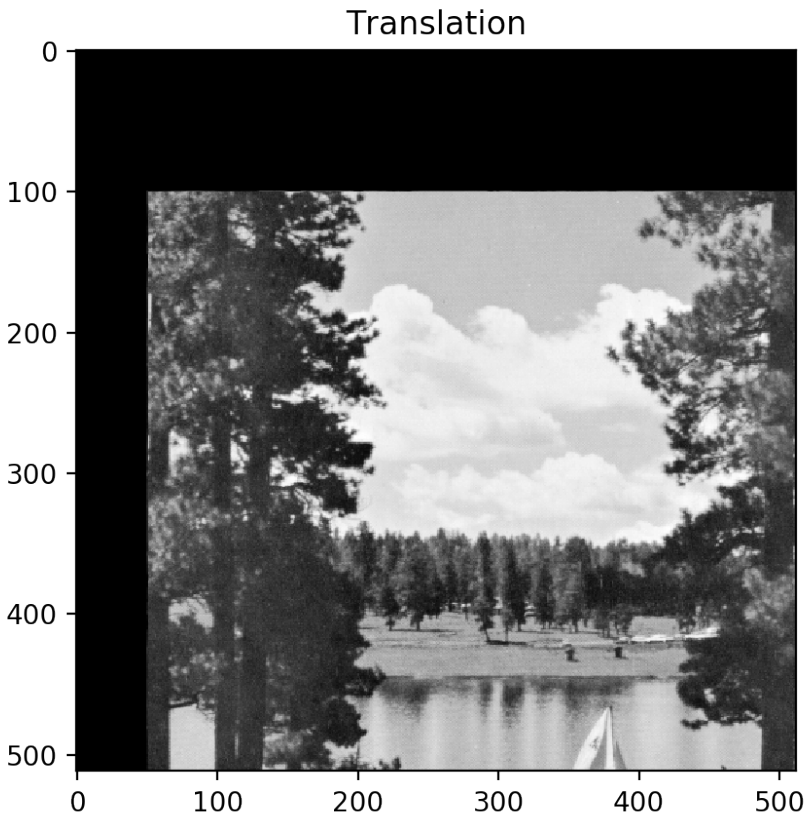


Scaling (Bilinear Interpolation)

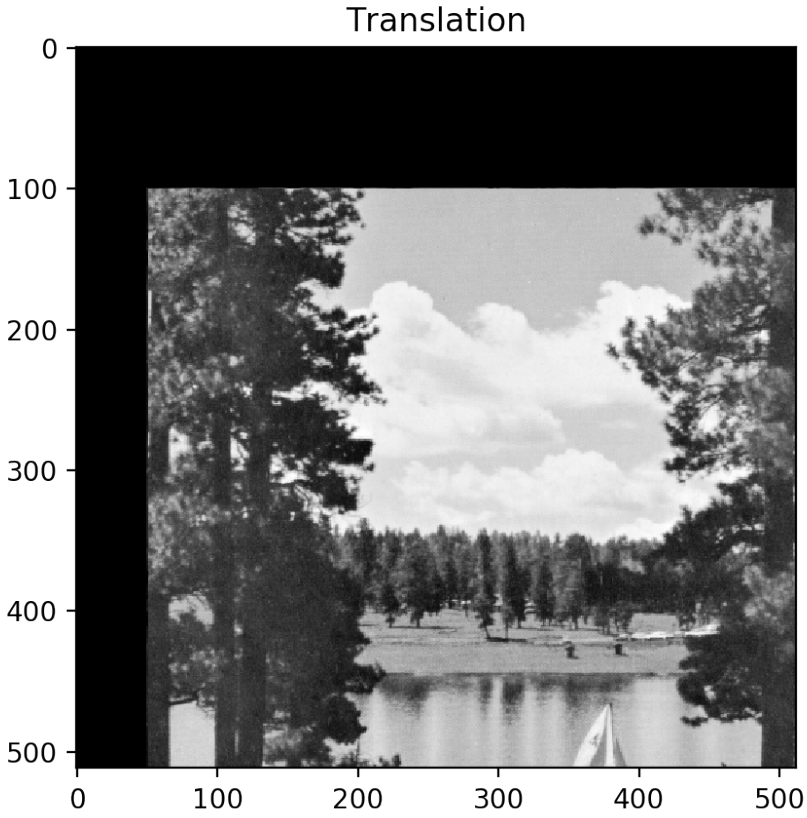


Scaling (NN Interpolation)

Translation:

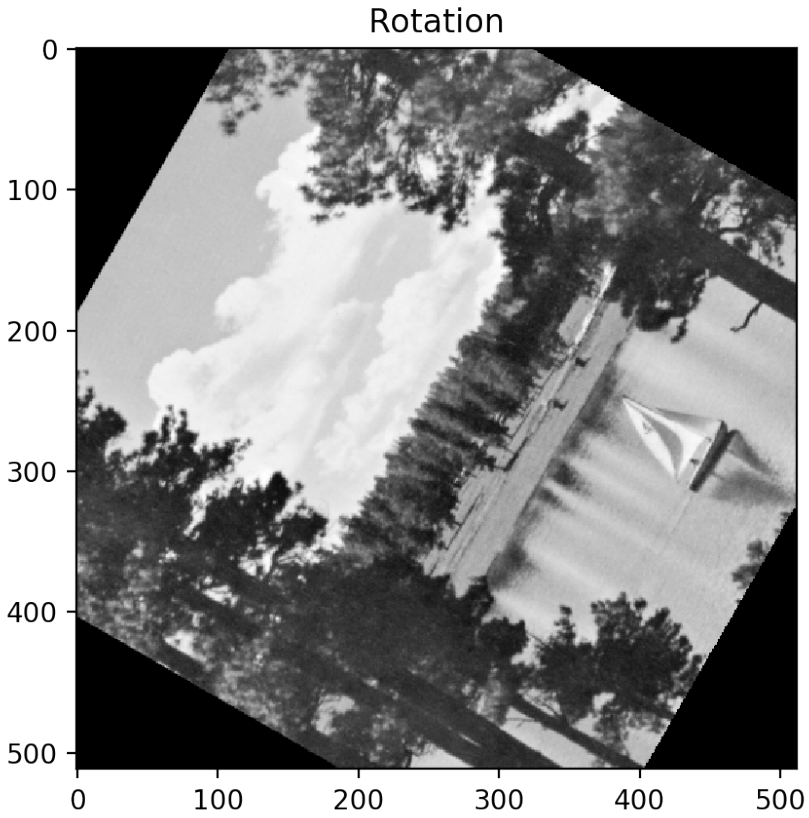


Translation (Bilinear)

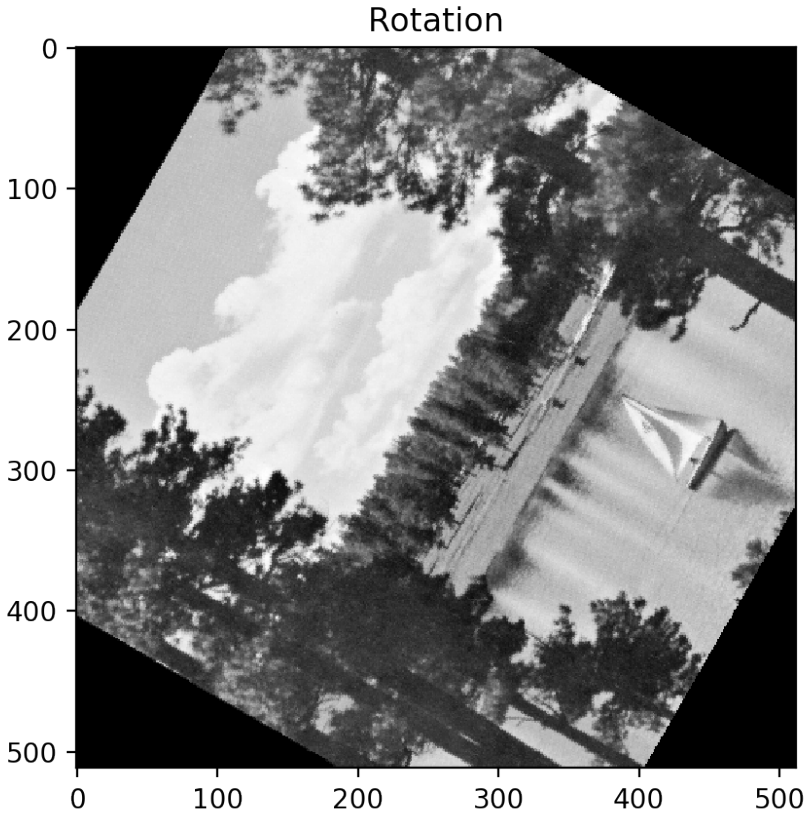


Translation (NN)

Rotation

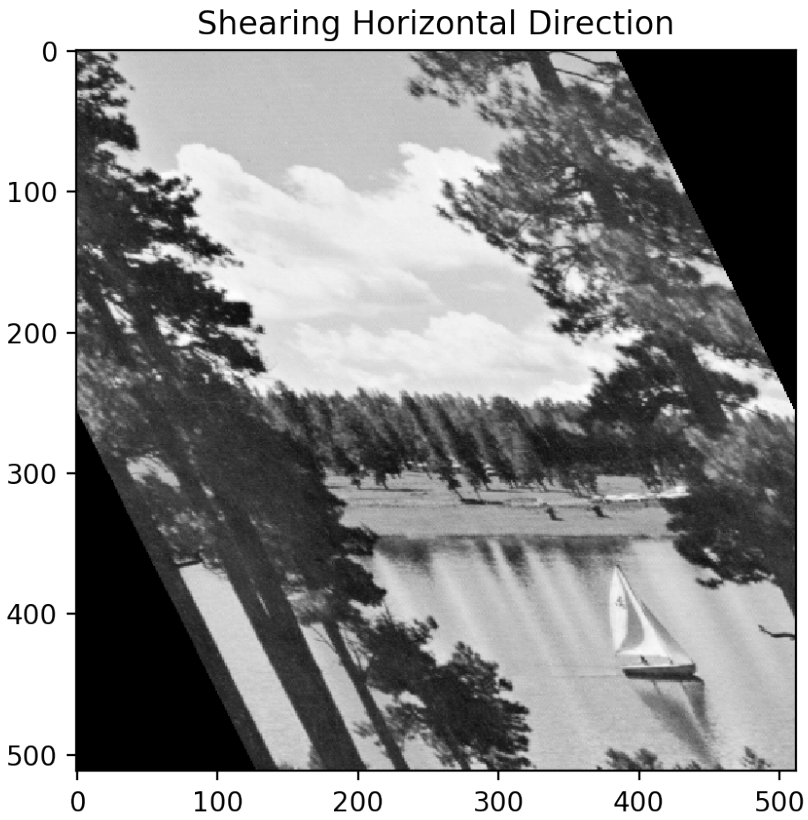


Rotation (Bilinear)

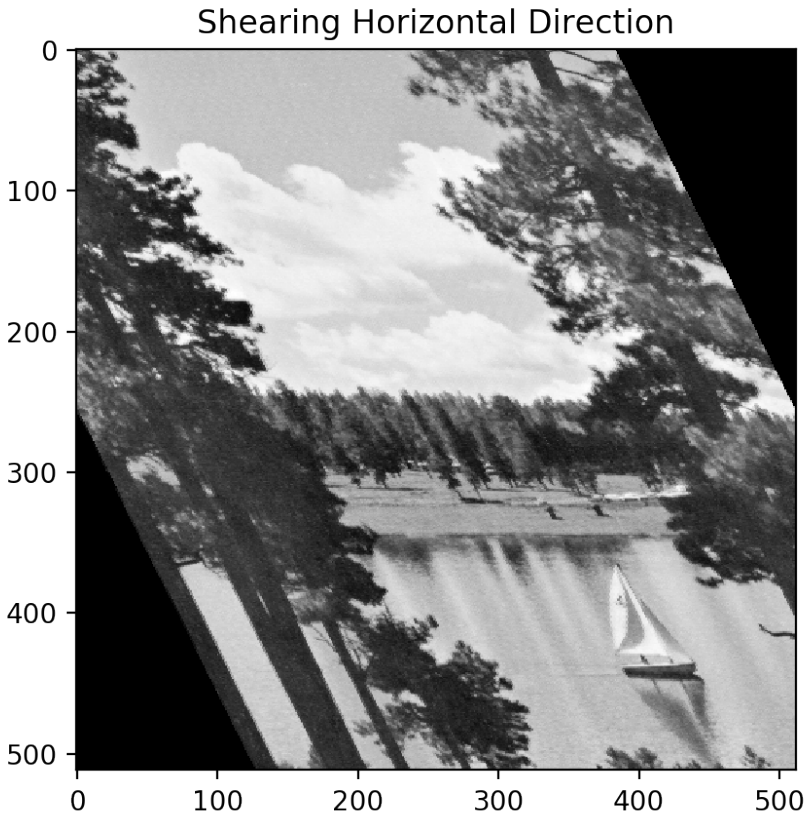


Rotation (NN)

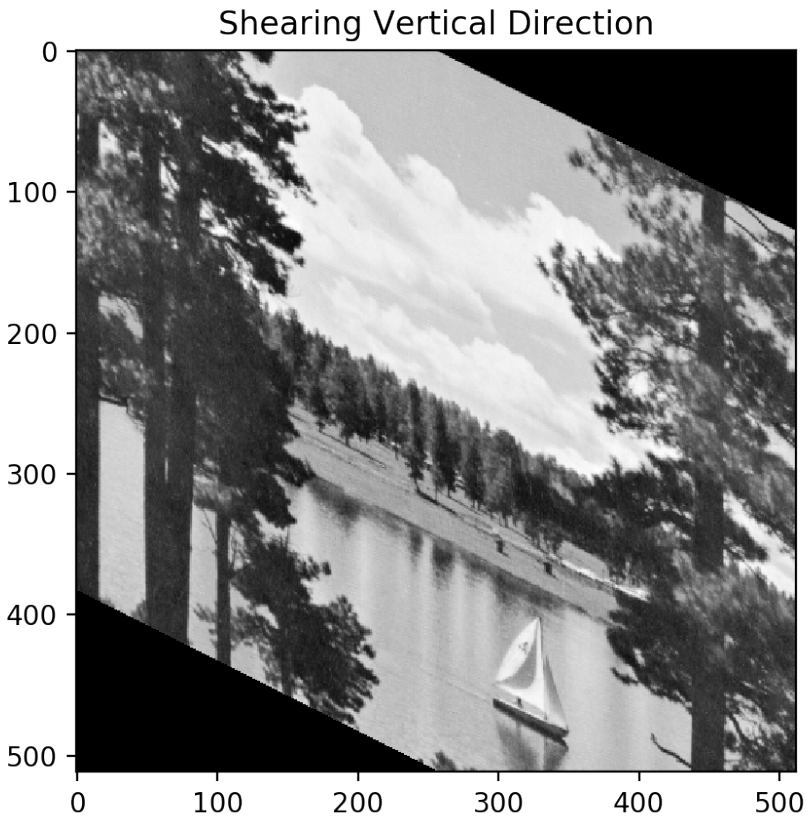
SHEARING:



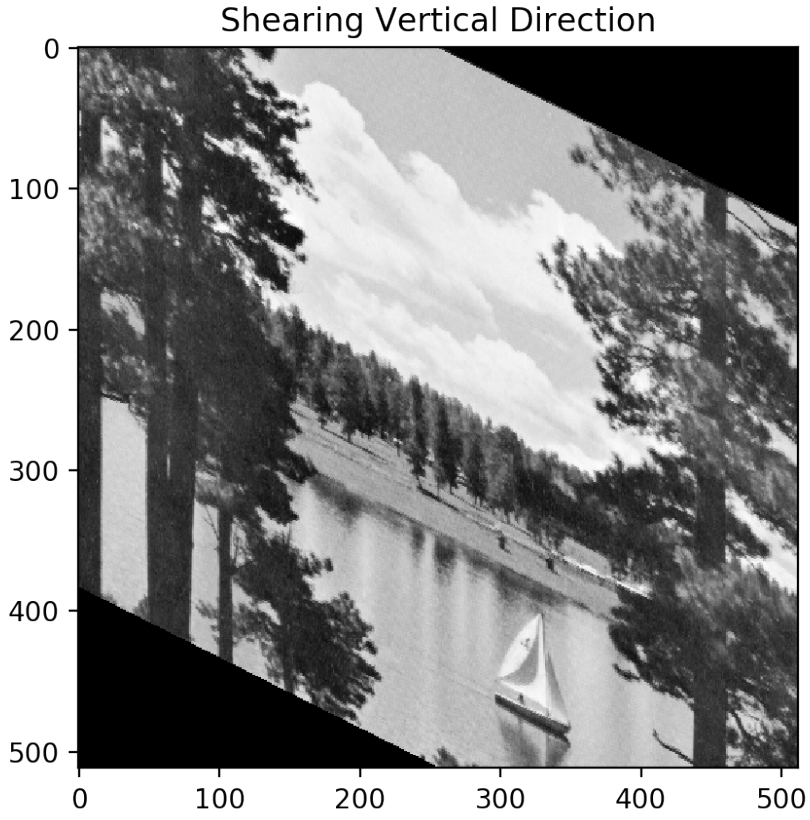
Shearing(Bilinear)


Shearing Horizontal Direction
0
100
200
300
400
500
0
100
200
300
400
500

Shearing(NN)



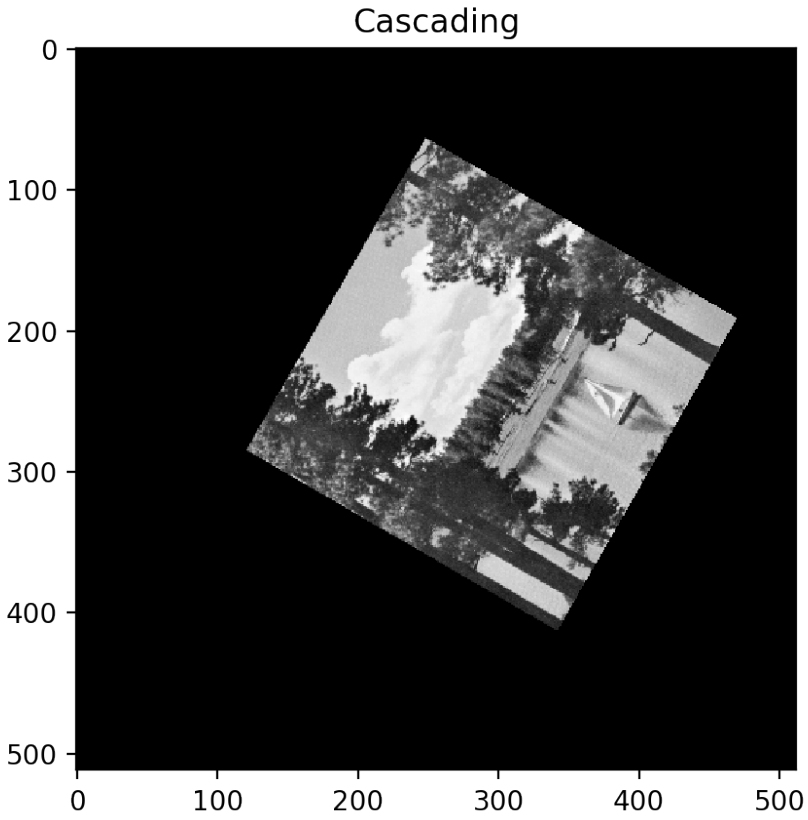
Shearing(Bilinear)


Shearing Vertical Direction
0
100
200
300
400
500
0
100
200
300
400
500

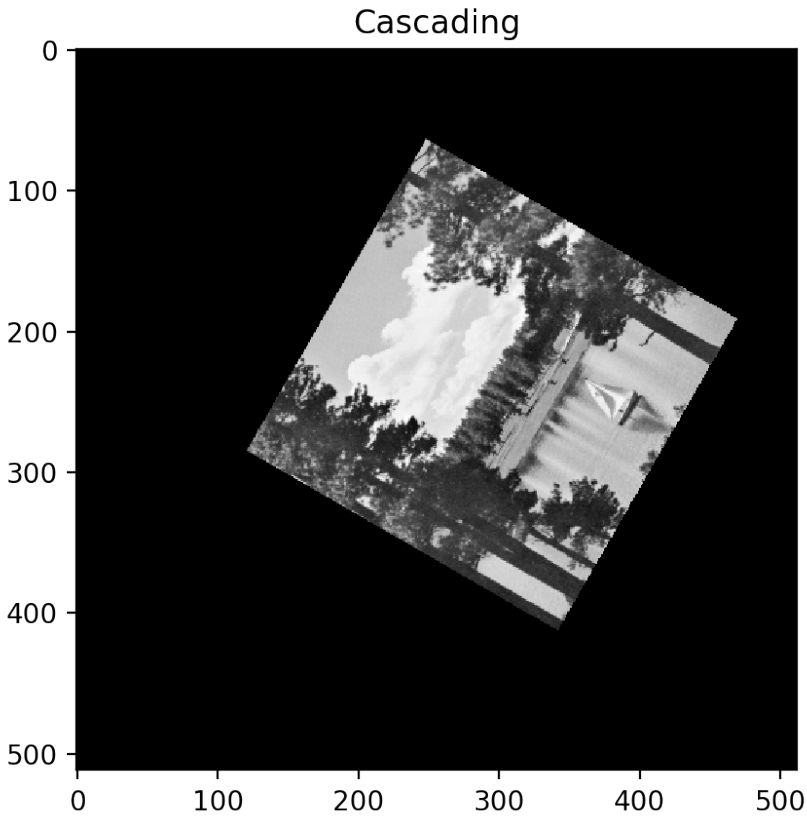
Shearing(NN)

Cascading:

In this cascading operation, I first multiply the matrices of individual transformation and this becomes our transformation matrix. The results are shown below:

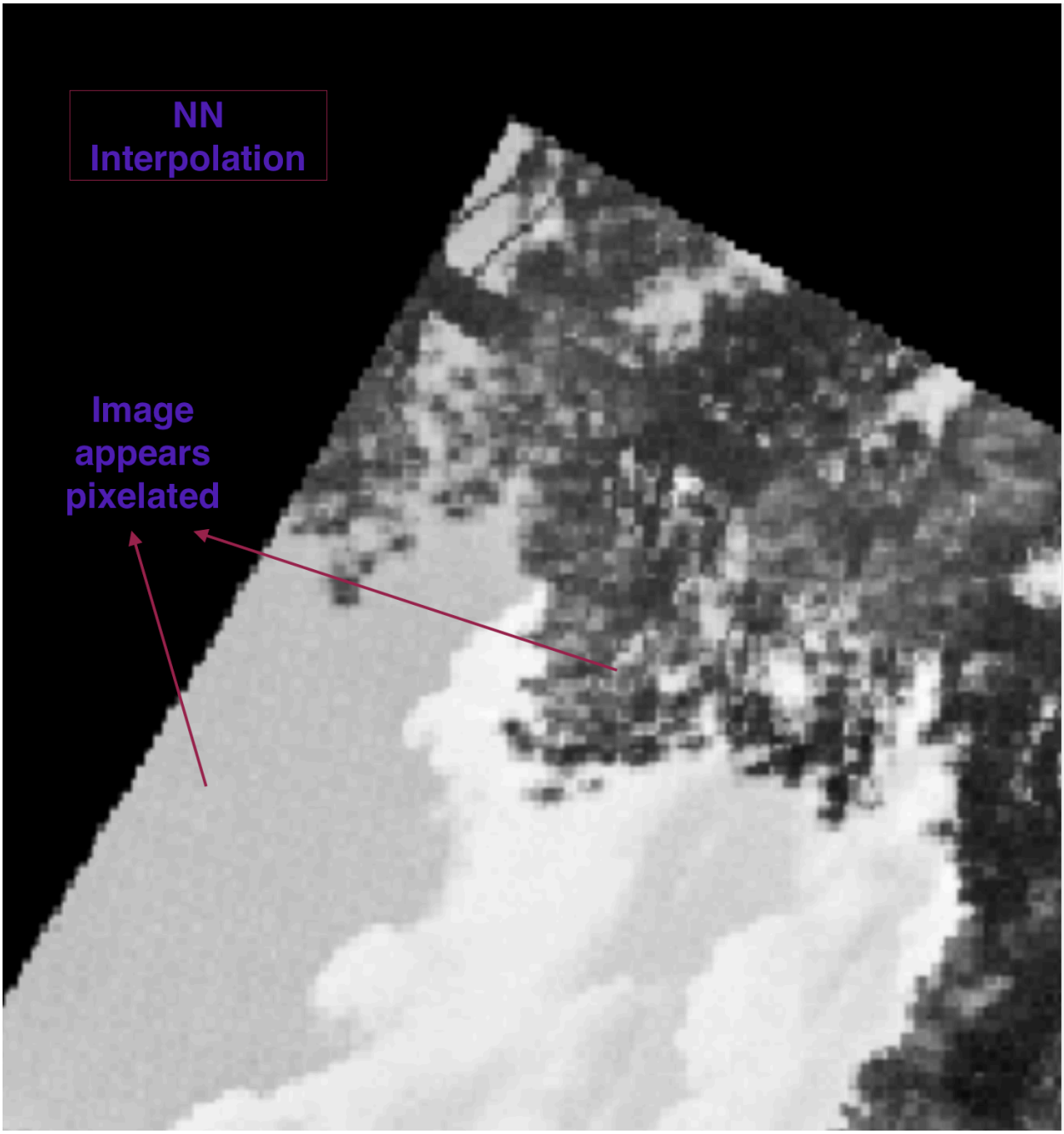


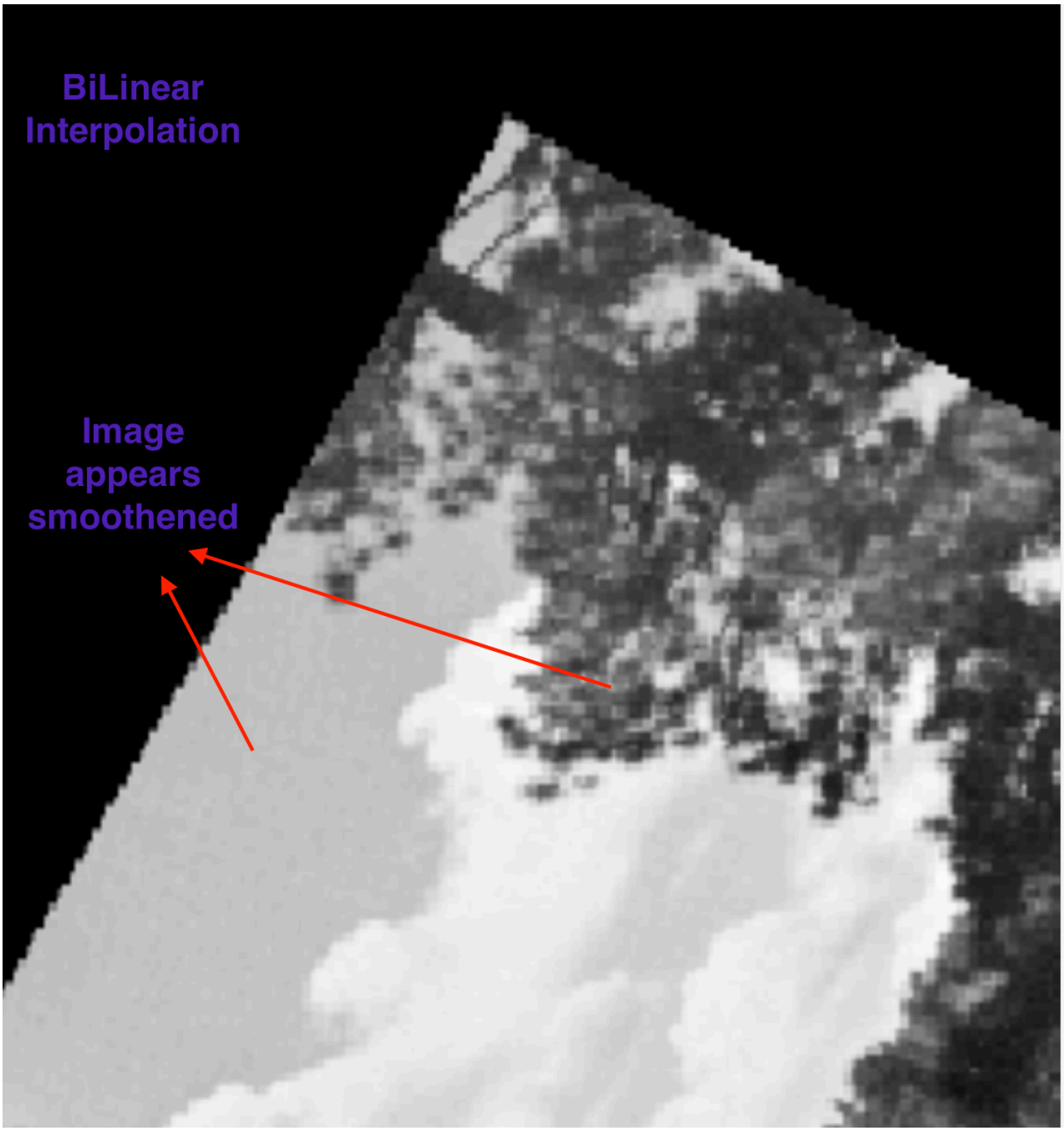
Cascading(NN)



Cascading(Bilinear)

For NN and Bilinear Interpolation, we can see the difference between the two interpolations. The NN interpolated image appears to be pixelated and Bilinear is smoothened. We can observe this by zooming in an image sub region.




BiLinear
Interpolation
Image
appears
smoothened

1.2 Affine Transform for Landmarks:

In this method, we choose the landmarks i.e., the corresponding points in source and target images and calculate our transformation matrices. We then transform our original image with the same transformation matrix and compare the two target images.

Algorithm:

1. Read the input image
2. Get the pixel locations from both the images by clicking on corresponding points with the mouse.
3. I setup 6 linear equations in terms of our 6 unknown values and then solve the linear equation to get the parameters for the affine transformation.
4. Use the above obtained transformation matrix on the original image to get the new transformed image.
5. Check both the target images.

Linear Equation Matrix (for 3 points):

Solution Method

We've used this technique several times now. We set up 6 linear equations in terms of our 6 unknown values. In this case, we know the coordinates before and after the mapping, and we wish to solve for the entries in our Affine transform matrix.

This gives the following solution:

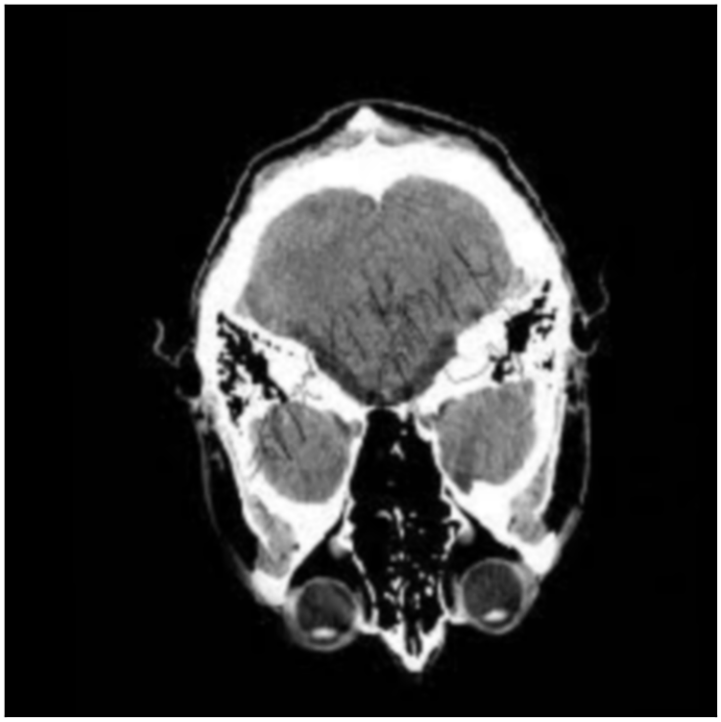
$$\mathbf{X}^{-1}\mathbf{x}' = \mathbf{a}$$

$$\underbrace{\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \end{bmatrix}}_{\mathbf{x}'} = \underbrace{\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_1 & y_1 & 1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_2 & y_2 & 1 \\ x_3 & y_3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_3 & y_3 & 1 \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{21} \\ a_{22} \\ a_{23} \end{bmatrix}}_{\mathbf{a}}$$

Original Image: This is the original image.

Target Image: This is the inverse of the original image and is used to calculate the transformation matrix.

Transformed Image: This is the image I obtained by transforming the original image with the transformation matrix calculated with landmark positions from the original and target image. We see that the transformation matrix calculated from the landmark position gives us the same target image.



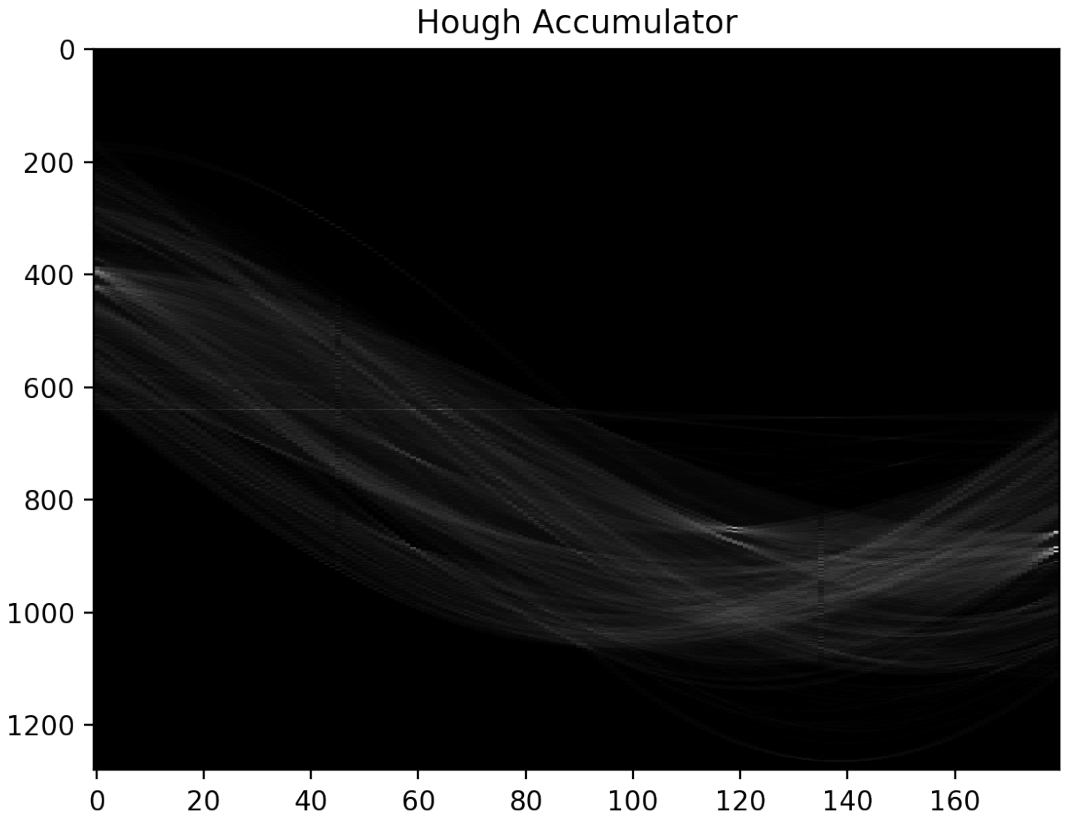
2. Hough Transform

Algorithm:

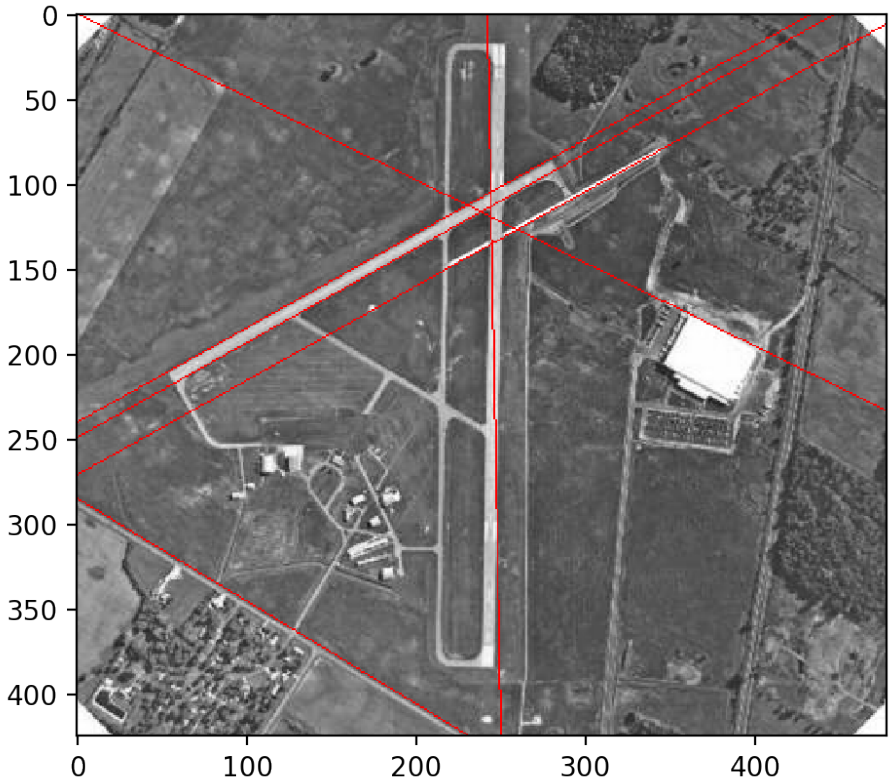
1. Read the input image
2. Apply edge detection and thresholding to get a subset of edge candidate to be processed.
3. Create an accumulator array for accumulating the votes in the Hough space for the input image.
4. Apply non-maxima suppression to the Hough space i.e., keeps the cell values intensity that have larger votes than all neighboring cells.
5. Choose N maximum peaks.
6. Take the index of the N maximum peaks and calculate the corresponding rectangular coordinates.
7. Draw the detected straight lines back onto the original image.
8. Display the image.

Original Image:

Hough Accumulator:



Detected Straight lines:



Found Runway and Roads