IMAGE TRANSFORMATION AND INTERPOLATION

Homework 1

410521308

資工四

黃書垣

Data:

letter_E.png

nature.png

First image: letter_E.png (source: http://clipart-library.com/clip-art/etsy-logo-transparent-png-6.htm)
Second image: nature.png (source: me)

Method:

1. Create an Empty image with size of output image so image will not get cropped.
2. Use transformation matrix to rotate image.

Transformation matrix:

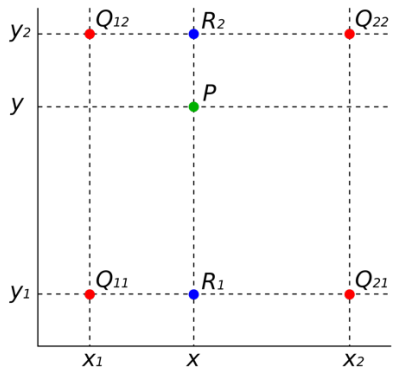
$$\mathbf{R} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

This matrix rotates the image in counterclockwise direction θ degrees.
For clockwise rotation I used -30 degrees.

3. Nearest neighbor interpolation
After rotation, values of new (x,y) coordinates may not be integer which would be a problem to create an image. We round up the (x,y) value to solve the problem.

4. Bilinear interpolation

Use values of the 4 nearest pixels, located in diagonal directions from a given pixel and takes a weighted average of these 4 pixels to arrive at its final value.



Code Explain

Rotation

```
#Define transform matrix for clockwise rotation 30 degrees
m = [[cos(radians(-30)), -sin(radians(-30))],
     [sin(radians(-30)), cos(radians(-30))]]
m = np.linalg.inv(m)
```

m is the transformation matrix and with -30 degrees it rotates the image clockwise 30 degrees.

Nearest neighbor interpolation

```
#Rotate image using nearest neighbor interpolation
def nearest_neighbor(img):
    #Construct blank output image
    outImg = np.zeros((outImgSize, outImgSize, 3), dtype=int)

    #rotate image
    for x in range(0, outImgSize):
        for y in range(0, outImgSize):
            #round new coordinates (nearest neighbor method)
            X = round(float(x*m[0][0] + (y-256)*m[0][1]))
            Y = round(float(x*m[1][0] + (y-256)*m[1][1]))

            #Check if X,Y is in img
            if X<512 and Y<512 and X>=0 and Y>=0:
                outImg[x][y] = img[X][Y]
```

x,y: coordinates for new image

X,Y: coordinates of original image

Use a double for loop to fill in the values for new image starting from (0,0). Find the value for new image (x,y) by getting the original images coordinates and assign the value from original image (X,Y) into new image (x,y). After calculation the coordinates may not be integer. We round the coordinates to find the coordinates we are looking for.

Bilinear interpolation

```
#Rotate image using bilinear interpolation.
def bilinear(img):
     #Construct blank output image
     outImg = np.zeros((outImgSize, outImgSize, 3), dtype=int)

     #rotate image
     for x in range(0, outImgSize):
          for y in range(0, outImgSize):
               X = float(x*m[0][0] + (y-256)*m[0][1])
               Y = float(x*m[1][0] + (y-256)*m[1][1])

               #Check if X,Y is in img
               if X<=511 and Y<=511 and X>=0 and Y>=0:
                    #Calculate new RGB level using bilinear method
                    X1, X2, Y1, Y2 = floor(X), ceil(X), floor(Y), ceil(Y)
                    R1 = (X2 - X) * img[X1][Y1] + (X-X1) * img[X2][Y1]
                    R2 = (X2 - X) * img[X1][Y2] + (X-X1) * img[X2][Y2]
                    outImg[x][y] = (Y2-Y) * R1 + (Y-Y1) * R2

     return outImg
```

The rotation part of Bilinear is almost the same as Nearest neighbor without the round part. Finding the value for new image (x,y) is different. We get the value with the following Equation:

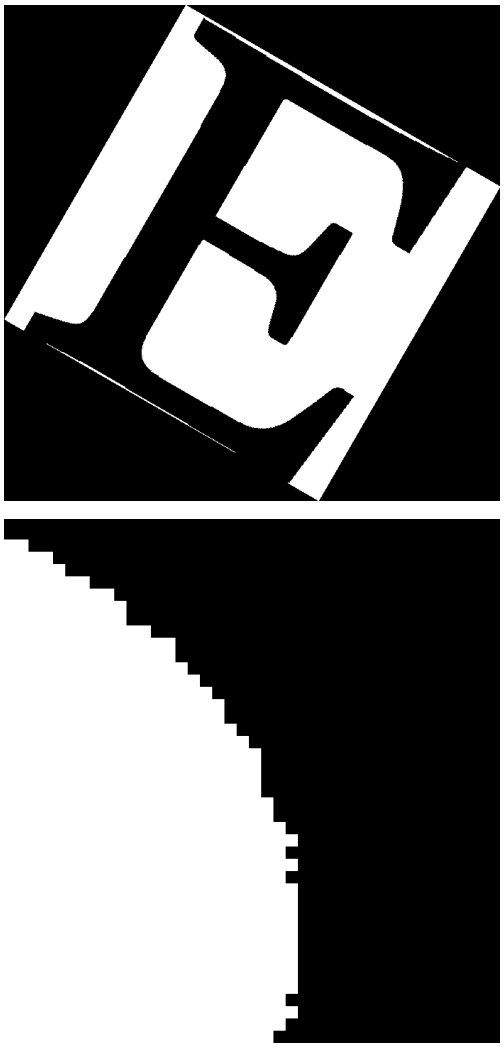
$$f(R_1) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \quad \text{where} \quad R_1 = (x, y_1),$$
$$f(R_2) \approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}) \quad \text{where} \quad R_2 = (x, y_2).$$

$$f(P) \approx \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2).$$

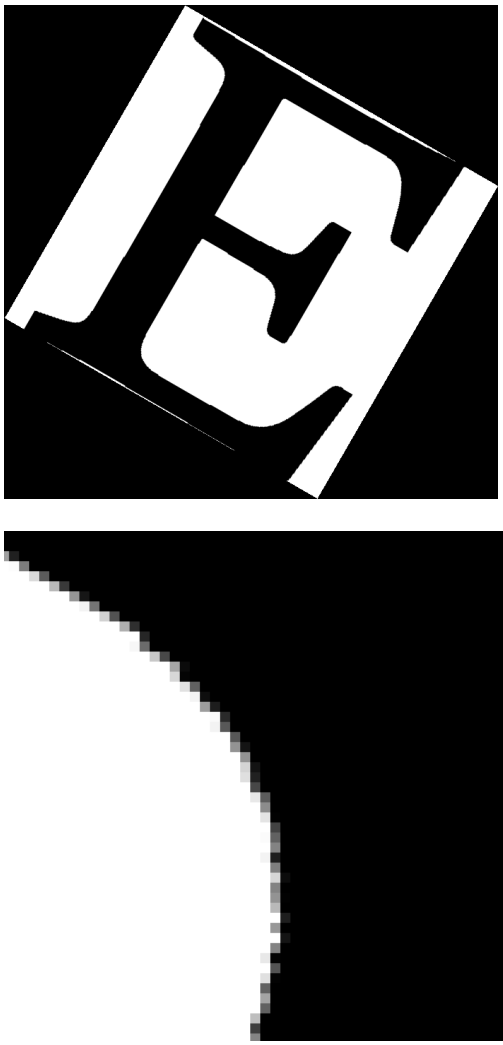
Result

letter_E.png

Nearest Neighbor

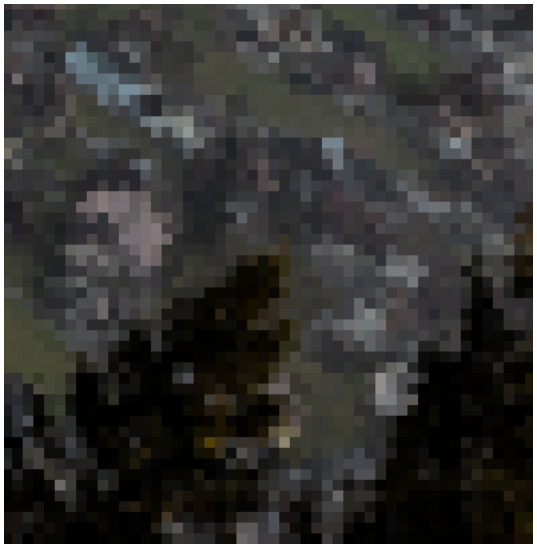


Bilinear



nature.png

Nearest neighbor



Bilinear

The results from nearest neighbor looks more pixelized. On the other hand, bilinear looks more smooth and visually appealing. I would choose to use bilinear interpolation for image rotation and enlargement most of the time. And use nearest neighbor if I’m doing pixel art.