EE5175 - Lab 1

Geometric Transforms

Nikilesh B
EE17B112

1. Translate the given image (lena translate.png) by (tx = 3:75; ty = 4:3) pixels.

The given image is attached below:

We use the translation matrix known to us for performing this :

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} x_s \\ y_s \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

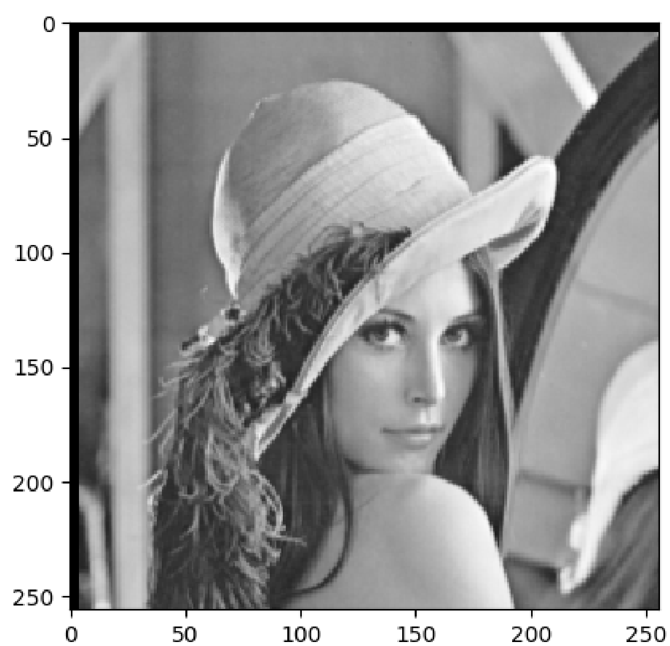
where x_t , y_t are target image coordinates
x_s, y_s are source image coordinates and
t_x, t_y are the given translation values.

We use Bilinear Interpolation on the given image and perform the translation needed to get a better quality resultant image.

The code for the function block alone is attached below:

```
def bilnr(src,tx, ty, xn, yn) :

    x = xn-(ty-1) ; y = yn-(tx-1)   #Mapping to the source grid
    xf = int(np.floor(x))  ; yf = int(np.floor(y))

    a = x-xf  ; b = y-yf      #distance from pixel

#Calculate intensity
    if xf >= (src.shape)[0]-1 or yf >= (src.shape)[1]-1 or xf<=0 or yf<= 0 :
      Ival = 0
    else :
      Ival = (1-a)*(1-b)*src[xf][yf] + (1-a)*(b)*src[xf][yf+1] +
(a)*(1-b)*src[xf+1][yf] + (a)*(b)*src[xf+1][yf+1]

    return Ival
```

The final image we get is attached below:



2.Rotate the given image (pisa rotate.png) about the image centre, so as to straighten the Pisa tower.

The given image is attached below:

We use the rotation matrix known to us for performing this :

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} \cos \Theta & \sin \Theta \\ -\sin \Theta & \cos \Theta \end{bmatrix} * \begin{bmatrix} x_s \\ y_s \end{bmatrix}$$

where x_t , y_t are target image coordinates
x_s, y_s are source image coordinates and

We find the angle Θ that needs to be rotated so that the tower is straight is

theta = -4* np.pi/180

Here also we use Bilinear Interpolation to get a better resultant image .

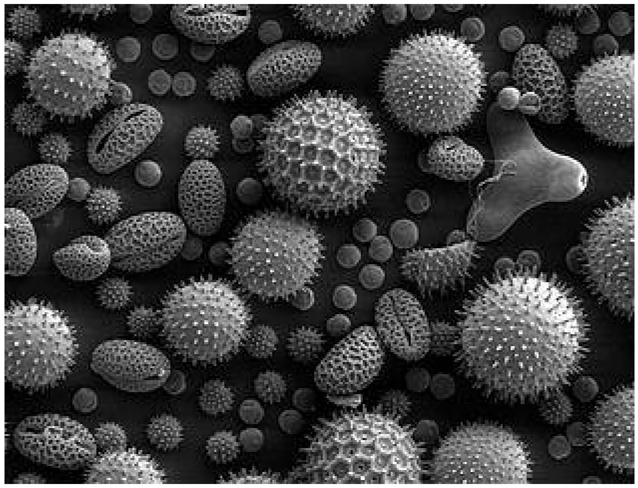
The bilinear function block is attached below:

```
#Bilinear function
def bilnr(src,theta, xn, yn) :
      #Mapping to src grid
    x = np.cos(theta)*xn - np.sin(theta)*yn
    y = np.sin(theta)*xn + np.cos(theta)*yn
    xf = int(np.floor(x)) + x_cen
    yf = int(np.floor(y)) + y_cen

    #distance from pixel
    a = x+x_cen-xf
    b = y+y_cen-yf

    #Calculate intensity
    if xf >= (src.shape)[0]-1 or yf >= (src.shape)[1]-1 or xf<=0 or yf<= 0 :
      Ival = 0
    else :
      Ival = (1-a)*(1-b)*src[xf][yf] + (1-a)*(b)*src[xf][yf+1] +
(a)*(1-b)*src[xf+1][yf] + (a)*(b)*src[xf+1][yf+1]
      #Ival = src[xf][yf]
    return Ival
```

The final image is attached below :

3. Scale the given image (cells scale.png) by 0:8 and 1:3 factors.

The given image is attached below:



We use the scaling matrix known to us for performing this :

$$\begin{matrix} x_t \\ y_t \end{matrix} = \begin{matrix} a & 0 \\ 0 & a \end{matrix} + \begin{matrix} x_s \\ y_s \end{matrix}$$

where x_t , y_t are target image coordinates
x_s, y_s are source image coordinates and
a is the given scaling factor value.

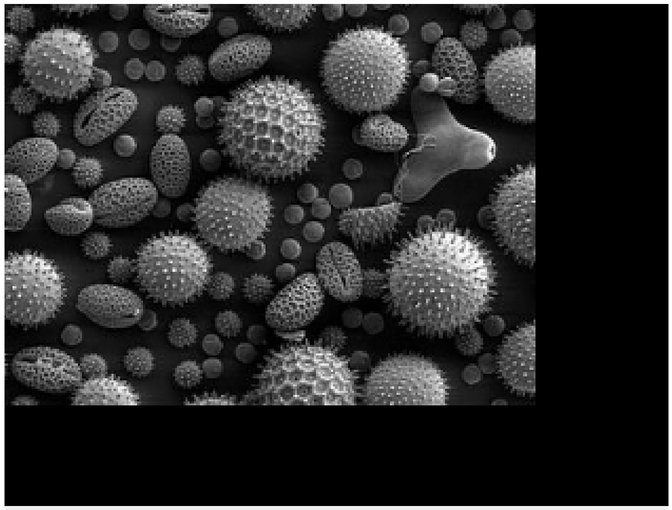
Here also we use Bilinear Interpolation to get a better quality of resultant image .

The bilinear function block is attached below:

```
      #Bilinear function
def bilnr(src, x_scl, y_scl, xn, yn) :
      #Mapping to src grid
    x = xn/x_scl
    y = yn/y_scl
    xf = int(np.floor(x)) + x_cen
    yf = int(np.floor(y)) + y_cen

    #distance from pixel
    a = x+ x_cen - xf
    b = y+ y_cen - yf

    #Calculate intensity
    if xf >= (src.shape)[0]-1 or yf >= (src.shape)[1]-1 or xf<=0 or yf<= 0
:
      Ival = 0
    else :
      Ival = (1-a)*(1-b)*src[xf][yf] + (1-a)*(b)*src[xf][yf+1] +
(a)*(1-b)*src[xf+1][yf] + (a)*(b)*src[xf+1][yf+1]
    return Ival
```

In our first case **a = 0.8** which is nothing but scaling down or zooming out.

The resultant image is attached below:



In our second case **a = 1.2** which is nothing but scaling up or zooming in.

The resultant image is attached below: