COMPUTER VISION

PROJECT 3

Part -1

The OpenCV warpPerspective function was used in the following way in the first part:

```
cv2.warpPerspective(image, transformationMatrix, (width, height),
flags=cv2.INTER_LINEAR+cv2.WARP_INVERSE_MAP, borderMode=cv2.BORDER_CONSTANT )
```

The transformation matrix was calculated as:

```
transformationMatrix= np.float32([
        [c, 0 , -c * u0],
        [0, c, -c * v0],
        [-a, -b, f + a * u0 + b * v0]
])
```

This was done so because according to the OpenCV documentation :

$$\texttt{dst}(x,y) = \texttt{src}\left(\frac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}}, \frac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}}\right)$$

From the equation given in the project documentation :

$$X = \frac{c(u-u0)}{f-a(u-u0)-b(v-v0)} \qquad Y = \frac{c(v-v0)}{f-a(u-u0)-b(v-v0)}$$

The above two equations can be written as:

$$X = \frac{cu-cu0}{-au-bv+f+au0+bv0} \qquad Y = \frac{cv-cv0}{-au-bv+f+au0+bv0}$$

The above two equations can be compared with the equation given in the OpenCV documentation and the above transformation matrix is obtained. The above equations transform Image coordinates to World Coordinates, however we have to do the inverse so the flag cv2.WARP_INVERSE_MAP is used. Further, we have used the cv2.INTER_LINEAR flag so that the interpolation is bilinear. We further use the border mode cv2.BORDER_CONSTANT so that out of range values are replaced by a constant and by default the constant is 0.

Part -2

For the second part the same function and the same transformation matrix is used. However, to make sure that the left side of the image is closer than the right side u0 is width/2 and v0 is height/2 and a is c/1000 and b is 0. To make sure that the bottom side is closer than the top side of the image a is 0 and b is -c/1000. The following output images were obtained with the command line ‘python mynetid2.py test.jpg 1’ :