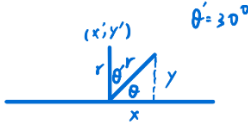


Image Processing HW1

Method

1. Rotation

Moving the center point to the middle of the picture to (0,0)



new point after rotation (x', y')

$$\begin{aligned} x' &= r \cos(\theta + \theta') \\ &= r (\cos\theta \cos\theta' - \sin\theta \sin\theta') \\ &= x \cos\theta' - y \sin\theta' \end{aligned} \quad \begin{aligned} y' &= r \sin(\theta + \theta') \\ &= r (\cos\theta \sin\theta' + \sin\theta \cos\theta') \\ &= x \sin\theta' + y \cos\theta' \end{aligned}$$

$\cos\theta = \frac{x}{r}, \sin\theta = \frac{y}{r}$

Code

```
def point_transform(img, x, y, degree):
    width, height = img.shape[:2]
    x = x - width / 2
    y = y - height / 2
    x1 = x * np.cos(degree / 180 * np.pi) - y * np.sin(degree / 180 * np.pi) + width / 2
    y1 = x * np.sin(degree / 180 * np.pi) + y * np.cos(degree / 180 * np.pi) + height / 2
    return x1, y1
```

2. Nearest neighbor

Round to the nearest neighbor ($f(x) = \text{int}(x+0.5)$)

3. Bilinear

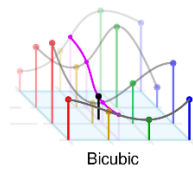
here $x_2 - x_1 = y_2 - y_1 = 1$

$$\begin{aligned} f(x, y_1) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}), \\ f(x, y_2) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22}). \end{aligned} \Rightarrow \begin{aligned} \text{let } x' &= x - x_1 \Rightarrow 1 - x' = x_2 - x \\ y' &= y - y_1 \Rightarrow 1 - y' = y_2 - y \\ f(x, y_1) &\approx (1 - x') f(Q_{11}) + x' f(Q_{21}) \\ f(x, y_2) &\approx (1 - x') f(Q_{12}) + x' f(Q_{22}) \\ f(x, y) &\approx (1 - y')(1 - x') f(Q_{11}) + (1 - y') x' f(Q_{21}) \\ &\quad + y'(1 - x') f(Q_{12}) + y' x' f(Q_{22}) \end{aligned}$$

Code

```
def get_point_value_bilinear(img, x, y):
    width, height = img.shape[:2]
    x1 = int(x)
    y1 = int(y)
    x2 = min(x1+1, width-1)
    y2 = min(y1+1, height-1)
    x = x - x1
    y = y - y1
    return (1-x)*(1-y)*img[x1, y1] + x*(1-y)*img[x2, y1] + (1-x)*y*img[x1, y2] + x*y*img[x2, y2]
```

4. Bicubic



$$f(p_0, p_1, p_2, p_3, x) = \left(-\frac{1}{2}p_0 + \frac{3}{2}p_1 - \frac{3}{2}p_2 + \frac{1}{2}p_3\right)x^3 + \left(p_0 - \frac{5}{2}p_1 + 2p_2 - \frac{1}{2}p_3\right)x^2 + \left(-\frac{1}{2}p_0 + \frac{1}{2}p_2\right)x + p_1$$

get the pink points by the fix) above,
then use the 4 points to get the real value

Code

```
def f_x_bicubic(x, p0, p1, p2, p3):  
    return (-p0/2 + 3*p1/2 - 3*p2/2 + p3/2)* x**3 + (p0 - 5*p1/2 + 2*p2 - p3/2)* x**2 + (-p0/2 + p2/2)* x + p1  
  
def get_point_value_bicubic(img, x, y):  
    width, height = img.shape[:2]  
    x0 = max(0, int(x-1))  
    x1 = int(x)  
    x2 = min(x1+1, width-1)  
    x3 = min(x1+2, width-1)  
    y0 = max(0, int(y-1))  
    y1 = int(y)  
    y2 = min(y1+1, height-1)  
    y3 = min(y1+2, height-1)  
    x = x-x1  
    y = y-y1  
    r0 = f_x_bicubic(x, img[x0, y0], img[x1, y0], img[x2, y0], img[x3, y0])  
    r1 = f_x_bicubic(x, img[x0, y1], img[x1, y1], img[x2, y1], img[x3, y1])  
    r2 = f_x_bicubic(x, img[x0, y2], img[x1, y2], img[x2, y2], img[x3, y2])  
    r3 = f_x_bicubic(x, img[x0, y3], img[x1, y3], img[x2, y3], img[x3, y3])  
    return np.clip(f_x_bicubic(y, r0, r1, r2, r3), [0,0,0], [255,255,255]).astype(np.uint8)
```

Result

(result also attached in code folder because they look similar after uploading to microsoft word)



original image



nearest neighbor



bilinear



bicubic



original image



nearest neighbor



bilinear



bicubic

Feedback

This homework is not as hard as I expected. The formulas can be derived easily, and the implementation is not too hard. There was one error took me some time to find when implementing, which is not converting `np.uint8` to float when doing bicubic interpolation. I knew I was wrong because the result looks weird, and I resolved it with the help of TA's clue in the HW announcement.