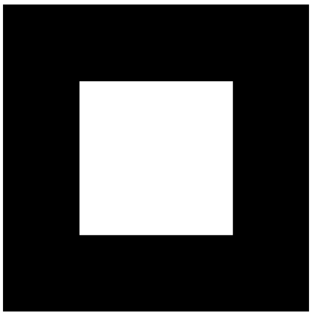
Homework No.5

Student ID: 41047902S Name: 鄭淮薰

Problem Statement

1. Create an image consisting of a white square with a black background, e.g.,



2. Rotate the image by 30 degrees. Use (a) rotation with neighbor interpolation, and (b) rotation with bilinear interpolation.

3. Compare the two results.

Results

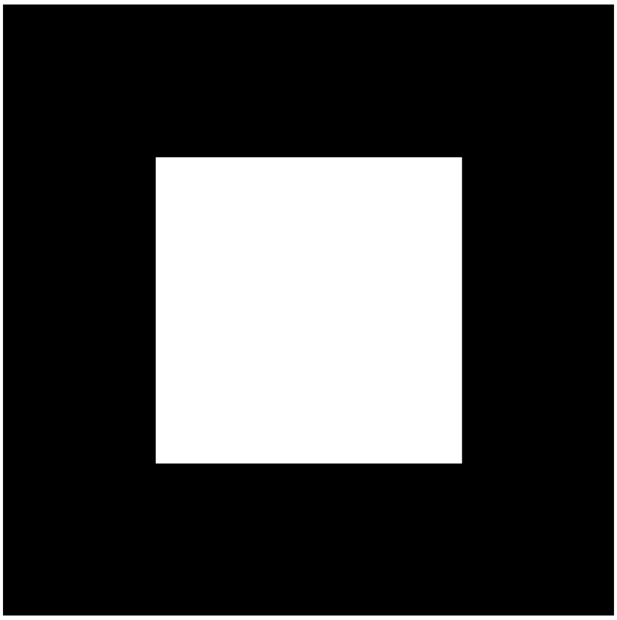
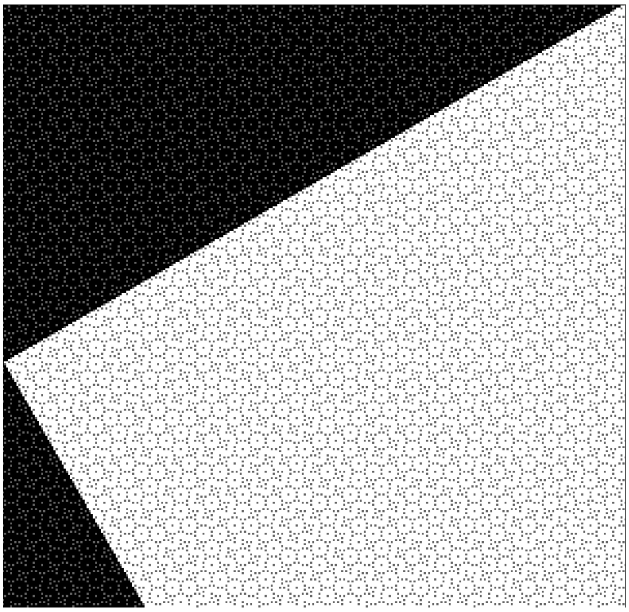
Fig1.1 Create Image	Fig1.2 Original Rotate Enlarge
	

Fig1.3 NN Interpolation	Fig1.4 Bilinear Interpolation

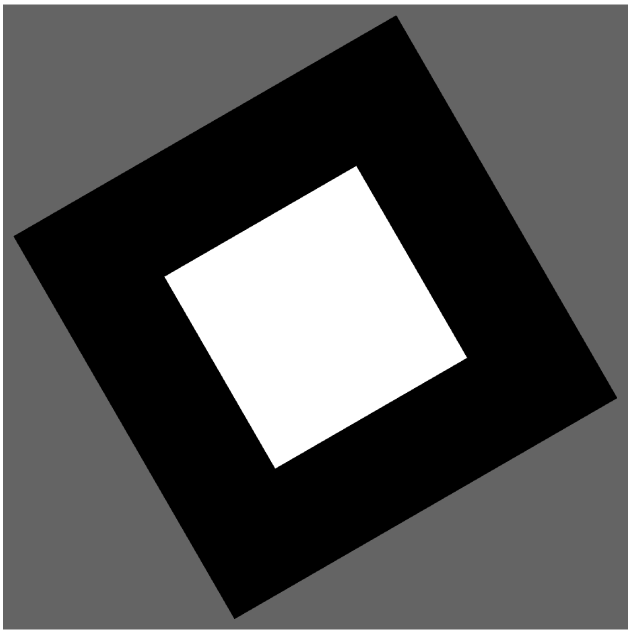
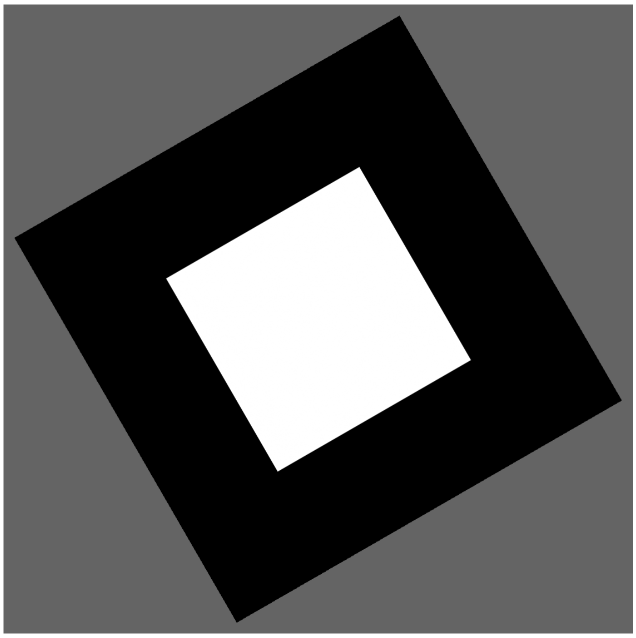

	

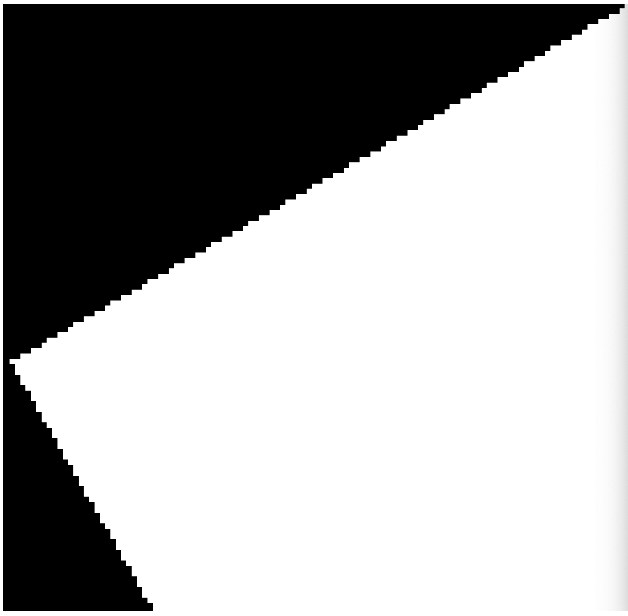
Fig1.5 NN Interpolation Enlarge	**Fig1.6 Bilinear Interpolation Enlarge**
	

Fig 1.1 為一黑底中間繪有白色方形的正方形圖片，Fig1.2 則將 Fig 1.1 逆時針旋轉 30° 後的圖片 (放大版，從中可明顯觀察到有許多缺洞及顏色不齊之處。而 Fig1.3 及 Fig4，則分別在旋轉時使用 Nearest-neighbor Interpolation 及 Bilinear Interpolation 來解決上述問題。為觀察方便，Fig1.5 與 Fig1.6 為這兩張的部分放大圖，從中可發現到使用 NN Interpolation 的圖在邊界處呈現鋸齒狀，而 Bininear Interpolation 的邊界較平滑完整，效果較前者更好。

Comment

在這項作業中，我先使用 cv2 模組來繪製一黑底中間白的正方形圖 (I)，接著製作兩張大小為正方形圖斜邊長大小的灰色空白圖 (I_1, I_2)，以避免原圖旋轉後超出邊界被裁減。

由於新圖 (I_1, I_2) 較舊圖 (I) 大，且我希望正方形以圖片中心做旋轉，所以除了在旋轉前後我都會將點做平移，整個流程為：

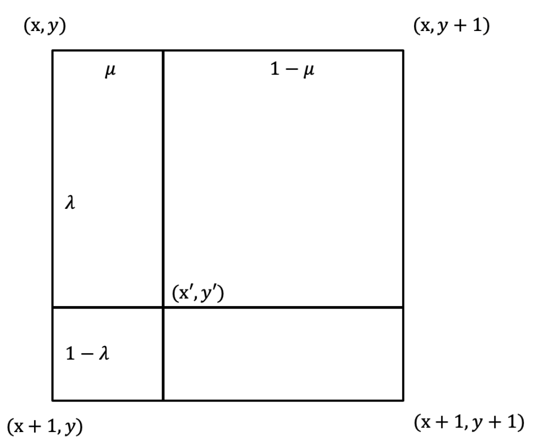
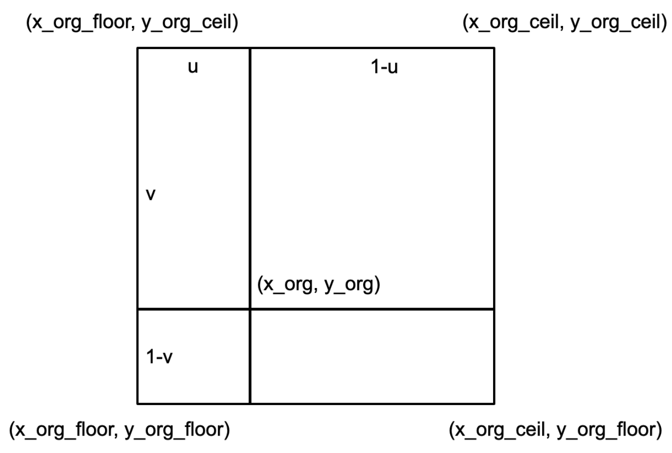
旋轉中心移至新圖 (I_1, I_2) 中心 -> 順時針旋轉 30° -> 旋轉中心移至舊圖 (I) 中心

此處有兩點需注意：

1. 第二步順時針旋轉是因為要使用 Inverse transformer 來解決 holes, truncation error 等問題。若想要最終圖為逆時針旋轉，則從新圖 (I_1, I_2) 找點時就需要以順時針的方法倒回去原圖 I 找點。
2. 在數學上 (題序中) 的轉移矩陣 (1) 是逆時針旋轉 ($\theta > 0$)，但由於圖座標中 y 軸向下為正向，與數學座標系不同，造成上述的旋轉矩陣改為順時針旋轉($\theta > 0$)。因此實作上我使用 (1) 而非 (2) 作為 inverse rotation matrix。

$$(1)\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix} \quad (2)\begin{bmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{bmatrix}$$

最後在做 bilinear interpolation 的公式是根據題目及課堂講義中提供的公式推導的，對應的關係如下圖所示：

課本	實作
	

Source Code

```python
import cv2
import numpy as np

# Create an image consisting of a white square with a black background
src_img = np.zeros((800, 800, 3), np.uint8)
cv2.rectangle(src_img, (200, 200), (600, 600), (255, 255, 255), -1)
src_centerX, src_centerY = src_img.shape[1]/2, src_img.shape[0]/2

# create a new image to store the rotated image
src_rows, src_cols, _ = src_img.shape
rotated_size = int(np.sqrt(src_rows**2 + src_cols**2))
img_rotated_NN = np.zeros((rotated_size, rotated_size, 3), np.uint8)
img_rotated_NN.fill(100)
img_rotated_bilinear = img_rotated_NN.copy()
rotated_centerX, rotated_centerY = img_rotated_NN.shape[1]/2,
img_rotated_NN.shape[0]/2

# 將旋轉中心移至新圖片的中心
translation_matrix1 = np.array([
    [1, 0, -rotated_centerX],
    [0, 1, -rotated_centerY],
    [0, 0, 1]
])

# Inverse rotation matrix
theta = np.radians(30)
cos_theta, sin_theta = np.cos(theta), np.sin(theta)
inverse_rotation_matrix = np.array([
    [cos_theta, -sin_theta, 0],
    [sin_theta, cos_theta, 0],
    [0, 0, 1]
])

# 將旋轉中心移回原圖片的中心
translation_matrix2 = np.array([
    [1, 0, src_centerX],
    [0, 1, src_centerY],
    [0, 0, 1]
])
```

```
# Matrix multiplication
M = np.dot(translation_matrix2, np.dot(inverse_rotation_matrix,
translation_matrix1))

# Rotate image by 30 degrees with nearest neighbor interpolation
for y in range(rotated_size):
    for x in range(rotated_size):
        # transform the pixel coordinates to the original image coordinate
system
        x_org = M[0][0]*x + M[0][1]*y + M[0][2]
        y_org = M[1][0]*x + M[1][1]*y + M[1][2]

        # round the transformed coordinates to the nearest integer to get the
pixel value from the original image
        x_org_rounded, y_org_rounded = int(round(x_org)), int(round(y_org))

        # set the pixel value in the rotated image
        if x_org_rounded >= 0 and x_org_rounded < src_cols and y_org_rounded >=
0 and y_org_rounded < src_rows:
            img_rotated_NN[y][x] = src_img[y_org_rounded][x_org_rounded]


# Rotate image by 30 degrees with bilinear interpolation
for y in range(rotated_size):
    for x in range(rotated_size):

        # transform the pixel coordinates to the original image coordinate
system
        x_org = M[0][0]*x + M[0][1]*y + M[0][2]
        y_org = M[1][0]*x + M[1][1]*y + M[1][2]

        # get the pixel value from the original image
        x_org_floor, y_org_floor = int(np.floor(x_org)), int(np.floor(y_org))
        x_org_ceil, y_org_ceil = int(np.ceil(x_org)), int(np.ceil(y_org))

        # set the pixel value in the rotated image
        if x_org_floor >= 0 and x_org_ceil < src_img.shape[1] and y_org_floor
>= 0 and y_org_ceil < src_img.shape[0]:
            img_rotated_bilinear[y][x] = (x_org_ceil-x_org)*(y_org_ceil-
y_org)*src_img[y_org_floor][x_org_floor] + \
                                        (x_org_ceil-x_org)*(y_org-
y_org_floor)*src_img[y_org_ceil][x_org_floor] + \
                                        (x_org-x_org_floor)*(y_org_ceil-
y_org)*src_img[y_org_floor][x_org_ceil] + \
                                        (x_org-x_org_floor)*(y_org-
y_org_floor)*src_img[y_org_ceil][x_org_ceil]
```

```
# save the rotated images
cv2.imwrite('original.png', src_img)
cv2.imwrite('rotated_NN.png', img_rotated_NN)
cv2.imwrite('rotated_bilinear.png', img_rotated_bilinear)
```