Hw3.

PAGE
DATE

Problem 1

1. line $\Rightarrow y = mx + b \Rightarrow mx - y + b = 0$.

multiply w to $mx - y + b = 0$.

$\rightarrow mxw - yw + bw = 0$.

$X = xw$, $Y = yw$.

Substitute e with X, Y.

$mX - Y + bw = 0$.

Convert to matrix version.

$$\begin{bmatrix} m & -1 & b \end{bmatrix} \begin{bmatrix} X \\ Y \\ w \end{bmatrix} = 0 \qquad L^T x' = 0$$

$$L = \begin{bmatrix} m \\ -1 \\ b \end{bmatrix} \quad x' = \begin{bmatrix} X \\ Y \\ w \end{bmatrix}$$

$$x' = \begin{bmatrix} X \\ Y \\ w \end{bmatrix} = \begin{bmatrix} xw \\ yw \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$\therefore$ points pass through the original line.

PAGE
DATE

Problem 1-2

(3,5) on the 2D plane, one with $z>0$ and one with $z<0$.

Let $z=2$ (one with $z>0$) and $z=-2$ (one with $z<0$)

If $z=2$, (3,5,1) can be written as (6,10,2). multiplying with constant.

If $z=-2$, (3,5,1) can be written as (-6,-10,-2).

So, $(x,y,z) \Rightarrow \begin{cases} (6,10,2) & z>0 \\ (-6,-10,-2) & z<0 \end{cases}$.

PAGE
DATE

Problem 1-3. Two homogeneous lines l_1, l_2.

Intersection point between the 2 lines is given by $a_1x + b_1y + c_1 = 0$, $a_2x + b_2y + c_2 = 0$

Hence, $\frac{x}{b_1c_2 - c_1b_2} = \frac{y}{c_1a_2 - a_1c_2} = \frac{1}{a_1b_2 - b_1a_2}$

$$x, y = \left(\frac{b_1c_2 - c_1b_2}{a_1b_2 - a_2b_1}, \frac{a_2c_1 - a_1c_2}{a_1b_2 - a_2b_1} \right)$$

The above equation can be written as

$L_1 = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \end{bmatrix}$ $L_2 = \begin{bmatrix} a_2 \\ b_2 \\ c_2 \end{bmatrix}$ in homogeneous space

$l_1 \times l_2$ is a vector and the vector is perpendicular to L_1 and L_2. and equal to 1.

$[x_1, y_1, z_1]^T = L_1 \times L_2$.

$L_1 \cdot [x_1, y_1, z_1]^T = 0$ and $L_2 \cdot [x_1, y_1, z_1]^T = 0$

and $\| [x_1, y_1, z_1] \| = 1$.

$a_1x_1 + b_1y_1 + c_1z_1 = 0$, $a_2x_1 + b_2y_1 + c_2z_1 = 0$

$x_1^2 + y_1^2 + z_1^2 = 1$.

Hence, $\begin{bmatrix} b_1c_2 - c_1b_2 \\ c_1a_2 - a_1c_2 \\ a_1b_2 - b_1a_2 \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix}$ ... (a)

Since $[x_1, y_1, z_1] = L_1 \times L_2$ in homogeneous space.

YANGJI

and candivide by z_1.

$$L_1 \times L_2 = \begin{bmatrix} \frac{b_1c_2 - c_1b_2}{a_1b_2 - b_1a_2} \\ \frac{c_1a_2 - a_1c_2}{a_1b_2 - b_1a_2} \\ 1 \end{bmatrix} \quad \ldots\ldots (b)$$

(a) and (b) can prove that homogeneous lines L_1 and L_2 intersection is given by $L_1 \times L_2$.

DATE

Problem #.14 $x+y-5=0 \Rightarrow L_1^T x=0 \Rightarrow L_1 = \begin{bmatrix} 1 \\ 1 \\ -5 \end{bmatrix}$

$4x-5y+\eta=0 \Rightarrow L_2^T x=0 \Rightarrow L_2 = \begin{bmatrix} 4 \\ -5 \\ \eta \end{bmatrix}$

$$L_1 \times L_2 = \begin{bmatrix} b_1c_2 - c_1b_2 \\ c_1a_2 - a_1c_2 \\ a_1b_2 - b_1a_2 \end{bmatrix}$$

and From the above L_1, L_2

$(a_1, b_1, c_1) = (1, 1, -5)$

$(a_2, b_2, c_2) = (4, -5, \eta)$

$$\Rightarrow \begin{bmatrix} 1\times\eta - (-5)\cdot(-5) \\ -5\cdot 4 - 1\cdot\eta \\ 1\times(-5) - 1\cdot 4 \end{bmatrix} = \begin{bmatrix} -18 \\ -2\eta \\ -9 \end{bmatrix}$$

and divide by -9.

$$\Rightarrow \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}$$ which is $L_1 \times L_2$

Hence $(2, 3)$ is the point of intersection in homogeneous space

PAGE
DATE

Problem 1.5 line with standard form given equation $ax+by+c=0$.

parallel line can be written as $ax+by+c_1=0$

From the previous problems, we can get $L_1 \times L_2 = \begin{bmatrix} bc_1 - cb \\ ca - ac_1 \\ ab - ab \end{bmatrix} = \begin{bmatrix} bc_1 - cb \\ ca - ac_1 \\ 0 \end{bmatrix}$

If $c_1 = 0$, $L_1 \times L_2 = \begin{bmatrix} -cb \\ ca \\ 0 \end{bmatrix} = \begin{bmatrix} b \\ -a \\ 0 \end{bmatrix}$

The point of intersection in homogeneous space $\Rightarrow \begin{bmatrix} b \\ -a \\ 0 \end{bmatrix}$.

~~For conver~~ In order to convert to the cartesian space, we can compute $l_1 \times l_2 = \begin{bmatrix} \frac{bc_1 - cb}{0} \\ \frac{ca - ac_1}{0} \\ 1 \end{bmatrix}$

Which is 2 parellel lines intersect at. (∞, ∞)

Therefore there is no intersection point

PAGE
DATE . .

Problem 1-6.

If the homogeneous line l goes through x_1, the line equation should be $L^T \cdot x_1 = 0$.

If the homogeneous line l goes through x_2, then line equation should be $L^T \cdot x_2 = 0$.

That means the relation between L and x_1 and x_2 are perpendicular.

(L is a vector perpendicular to x_1, x_2)

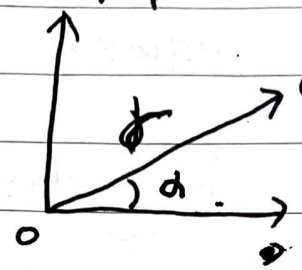
$L = x_1 \times x_2$.

Therefore, this is the justification. (Also, it is related to the cross product).

problem 2. 2D transformation.

point (x, y) in 2D plane.

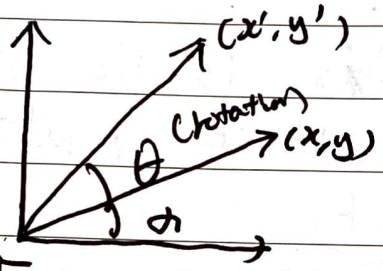
The Representation of graph looks like



so, $x = r\cos\alpha$

$y = r\sin\alpha$

After the rotation (θ), new point (x', y')



$x' = r\cos(\theta+\alpha)$

$y' = r\sin(\theta+\alpha)$

From the sin, cos Addition and Subtraction Formulas.

$x' = r(\cos\theta\cos\alpha - \sin\theta\sin\alpha)$

$y' = r(\sin\theta\cos\alpha + \cos\theta\sin\alpha)$

$x = r\cos\alpha \quad y = r\sin\alpha$

$x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

This can be represented as matrix as

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\therefore \begin{bmatrix} a' \\ b' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix}$$

PAGE
DATE

problem 2-2 $p_1 = (1,1)$ $p_2 = (2,1)$ $p_3 = (2,2)$ $p_4 = (1,2)$

This is related to the previous problem.

Given: rotation about p_2 through an angle of 45 degrees

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos 45 & -\sin 45 & -2\cos 45 + 1 \cdot \sin 45 + 2 \\ \sin 45 & \cos 45 & -2\sin 45 - 1\cos 45 + 1 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 2 - \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1 - \frac{3}{\sqrt{2}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

New vertices.

$p'_1 = (1,1,1)$ $p'_2 = (2,1,1)$ $p'_3 = (2,2,1)$

$p'_4 = (1,2,1)$

$$p'_1 = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 2 - \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1 - \frac{3}{\sqrt{2}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 - \frac{1}{\sqrt{2}} \\ 1 - \frac{1}{\sqrt{2}} \\ 1 \end{bmatrix} = \begin{bmatrix} 1.29 \\ 0.29 \\ 1 \end{bmatrix}$$

$$p'_2 = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 2 - \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1 - \frac{3}{\sqrt{2}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \\ 1 \end{bmatrix}$$

$$P_3' = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 2-\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1-\frac{2}{\sqrt{2}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1,29 \\ 1,71 \\ 1 \end{bmatrix}$$

$$P_4' = \begin{bmatrix} \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} & 2-\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & 1-\frac{3}{\sqrt{2}} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 0,59 \\ 1 \\ 1 \end{bmatrix}$$

$P_1' = (1,29, 0,29)$, $P_2' = (2,1)$, $P_3' = (1.29, 1,01)$
$P_4' = (0,59, 1)$

Problem 2.3

Given line $ax+by+c=0$.

1. Thanslate origin to the point.

$x=0$. $by+c=0$ $by=-c$ $y=-\frac{c}{b}$.

$(0, -\frac{c}{b})$

$x'=x-0$

$y'=y+\frac{c}{b}$ $\Rightarrow$ ① $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{c}{b} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

Then, rotate angle θ.

② $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

reflect across the x-axis.

$x'=x$

$y'=-y$.

③ $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$.

Then, rotate angle back θ.

④ $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

translate back to origin.

$x'=x+0$

$y'=y-\frac{c}{b}$

⑤ $\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{c}{b} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$

From ① ~ ⑤

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -\frac{c}{b} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & \frac{c}{b} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

then multiply all matrices. (homogeneous)

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos^2\theta - \sin^2\theta & 2\sin\theta\cos\theta & \frac{2c}{b}\sin\theta\cos\theta \\ 2\sin\theta\cos\theta & \sin^2\theta - \cos^2\theta & -\frac{x}{b}\cos^2\theta \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

$$\cos^2\theta = \frac{b^2}{a^2+b^2} \qquad \sin^2\theta = \frac{a^2}{a^2+b^2}.$$

$$\tan\theta = \frac{-a}{b} \qquad \sin\theta\cos\theta = \tan\theta\cos^2\theta$$

$$= \frac{-a}{b} \cdot \frac{b^2}{a^2+b^2} = \frac{-ab}{a^2+b^2}$$

When $b=0$.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & \frac{2c}{b} \times \frac{-ab}{a^2+b^2} \\ 0 & 1 & -\frac{2c}{b} \times \frac{b^2}{a^2+b^2} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} -1 & 0 & -\frac{2c}{a} \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}.$$

DATE

Problem 3.

Following Equations.

$x' = ax + by + t_x + \alpha x^2 + \beta y^2$,

$y' = cx + dy + t_y + \gamma x^2 + \theta y^2$.

We can set the H matrix to solve it.

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} H \end{bmatrix} \begin{bmatrix} x^2 \\ y^2 \\ xy \\ x \\ y \\ 1 \end{bmatrix}$$

$$\Downarrow$$

$$H = \begin{bmatrix} \alpha & \beta & 0 & a & b & t_x \\ \gamma & \theta & 0 & c & d & t_y \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Recall the system equation

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = H \begin{bmatrix} x^2 \\ y^2 \\ xy \\ x \\ y \\ 1 \end{bmatrix}.$$

Let P and q are homogeneous.

P and H is 3×6 matrix.

$P = H[0] \times [x^2, y^2, xy, x, y, 1]^T$

$H[2] \times [x^2, y^2, xy, x, y, 1]^T$

$q = H[1] \times [x^2, y^2, xy, x, y, 1]^T$

$H[2] \times [x^2, y^2, xy, x, y, 1]^T$.

PAGE
DATE

$h = [h_{11} \sim h_{3(}]^T$.
Vectorized form.
$a = [-x^2, -y^2, -xy, -x, -y, -1, 0, 0, 0, 0, 0, 0, px^2,$
$py^2, pxy, px, py, p]^T$.
$b = [0, 0, 0, 0, 0, 0, -x^2, -y^2, -xy, -x, -y, -1, qx^2$
$qy^2, qxy, qx, qy, q]^T$.

h can be solved by $a^T h = 0$, a_y

~~(However, In the given problem 'xy' i's not mentioned)~~

To solve the h (homogeneous matrix),
($h = [a, b, 0, a, b, t_x, r, 0, 0, c, d, t_y, 0, 0$
$0, 0, 0, 1]^T$.) which h has 10 unknowns
and we need at least ~~6~~ points.
Therefore, we need 5 pairs point
at least to decide the transform matrix

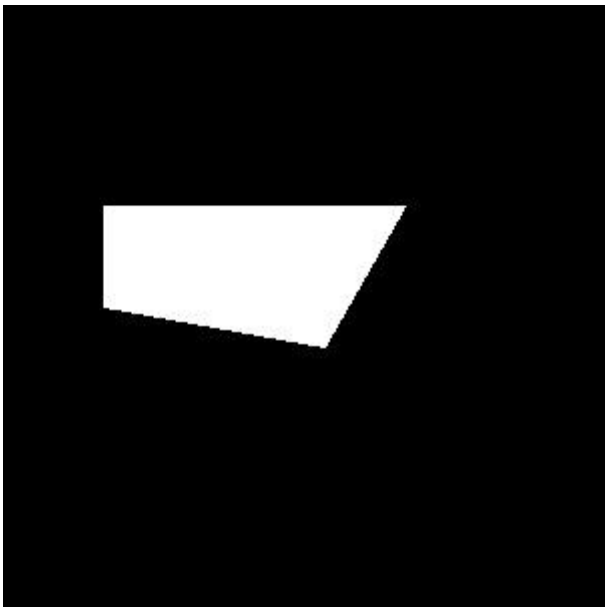
To get the best result, we can
use the SVD.

$u, s, vh = SVd(A)$

Problem 4: 2D Transformations on Images.

1.

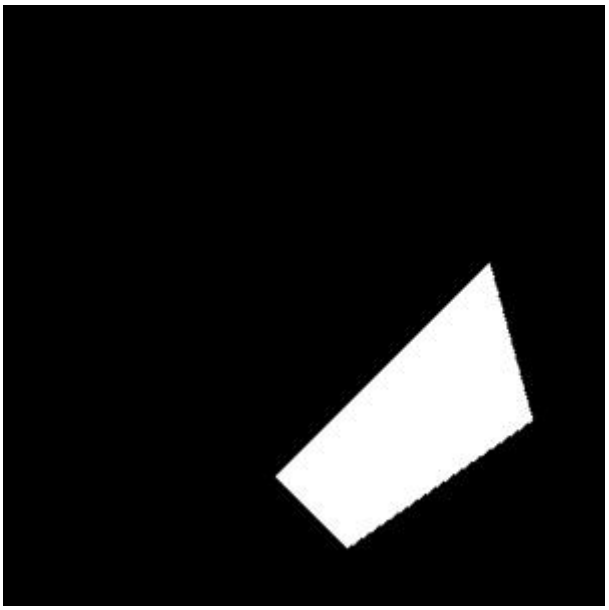
1. Original white irregular quadrilateral



2. Rotated white irregular quadrilateral.

   (1)Translate the image by (30,100)

   (2)Rotate the image by 45 degrees.

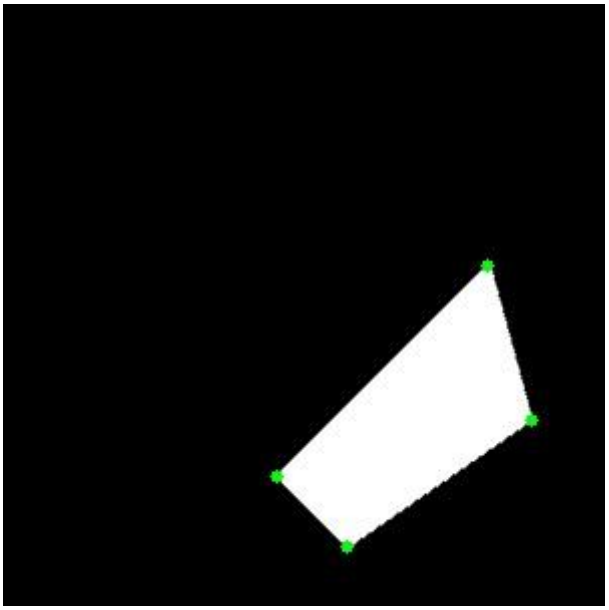


Code:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Construct the image _ 2D Transformations on Images.
def main():
    height, width = 300, 300 # construct an image of 300x300 pixels
    random = np.array([[50, 100], [200, 100], [160, 170], [50, 150]]) # random points for quadrilateral
    img = np.zeros((height, width, 3), dtype=np.uint8)
    quadrilateral = cv2.fillPoly(img, [random], (255,255,255))
    #show the result
    plt.imshow(cv2.cvtColor(quadrilateral, cv2.COLOR_BGR2RGB))
    plt.title('quadrilateral image')
    cv2.imwrite('data/original.jpg', quadrilateral)
    # (1) Translate the image by (30, 100)
    h, w = quadrilateral.shape[:2]
    # create translate matrix
    translate_matrix = np.float32([[1, 0, 30], [0, 1, 100]])
    # translated image with wrap affine transformation
    translated_img = cv2.warpAffine(quadrilateral, translate_matrix, (w, h))
    # Rotate the image around (150, 150)
    h, w = translated_img.shape[:2]
    rotation_matrix = cv2.getRotationMatrix2D((w / 2, h / 2), 45, 1)
    rotated_img = cv2.warpAffine(translated_img, rotation_matrix, (w, h))

    plt.imshow(cv2.cvtColor(rotated_img, cv2.COLOR_BGR2RGB))
    plt.title('rotate image')
    plt.show()
    cv2.imwrite('data/rotated.jpg', rotated_img)


if __name__ == '__main__':
    main()
```

2. Come up with a feature detector and descriptor to accurately match the four corners of the quadrilateral in the two pixels.

   Detected corners by rotated image.



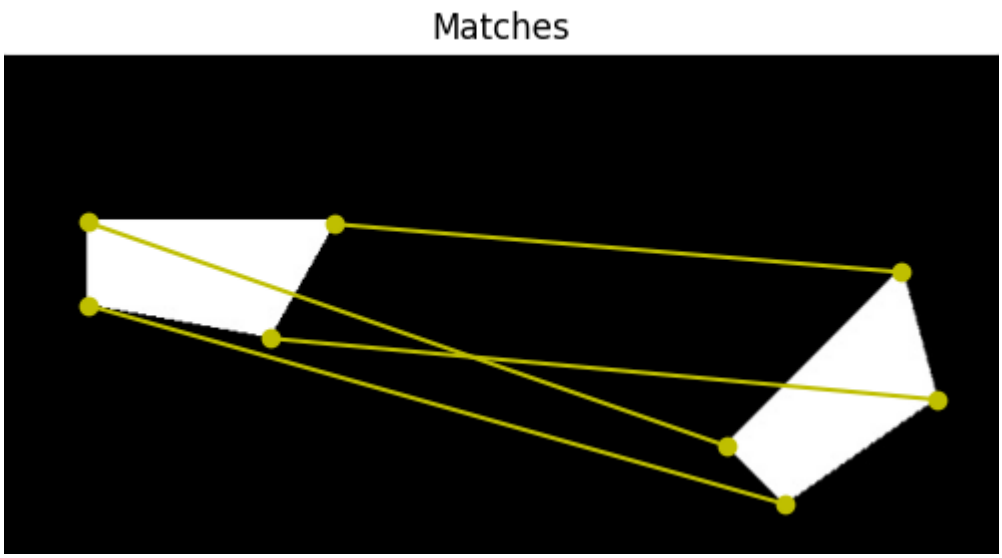
Code (feature detector):

```
class HarrisCorners:
    def __init__(self, orig_img, k, threshold):
        self.orig_img = orig_img
        self.img = cv2.cvtColor(orig_img, cv2.COLOR_BGR2GRAY)
        self.k = k
        self.threshold = threshold

    def find_corners(self):
        # Phase1: Use the sobel and find the filtered gradient
        gradient_x = cv2.Sobel(self.img, cv2.CV_64F, 1, 0)
        gradient_y = cv2.Sobel(self.img, cv2.CV_64F, 0, 1)
        # Phase2: calculate product of the gradients
        gradient_xx = gradient_x * gradient_x
        gradient_xy = gradient_x * gradient_y
        gradient_yy = gradient_y * gradient_y
        # Phase3: Apply Gaussian blurring
        gaussian_x = cv2.GaussianBlur(gradient_xx, (7,7), 0)
        gaussian_xy = cv2.GaussianBlur(gradient_xy, (7,7), 0)
        gaussian_yy = cv2.GaussianBlur(gradient_yy, (7,7), 0)
        # Compute all the gradient
        all = gaussian_x * gaussian_yy - gaussian_xy * gaussian_xy - self.k * (gaussian_x + gaussian_yy)**2

        # 5. threshold the corner response
        # set values to one if response is greater than certain threshold * max, else zero
        thresh_response = all > self.threshold * np.max(all)
        # find where corner responses are non zero
        thresh_indices = np.nonzero(thresh_response)
        thresh_response = []
        for i, j in zip(thresh_indices[0], thresh_indices[1]):
            th = [(i, j), all[i][j]]
            thresh_response.append(th)
        # reverse sort the threshold response values
        thresh_response = sorted(thresh_response,key=lambda x: x[1], reverse=True)
        # Non maximal suppression
        suppression = nms(thresh_response)

        return suppression

def main():
    # read original and rotated images.,
    quad_img = cv2.imread('data/original.jpg', 1)
```

```python
def main():
    # read original and rotated images.,
    quad_img = cv2.imread('data/original.jpg', 1)
    rotated_img = cv2.imread('data/rotated.jpg', 1)
    # compute harris corners from scratch
    original_harris = HarrisCorners(quad_img, k=0.01, threshold=0.2)

    rotated_harris = HarrisCorners(rotated_img, k=0.01, threshold=0.2)
    corners_img = np.zeros_like(quad_img)
    for index in original_harris.find_corners():
        cv2.circle(quad_img, (index[1], index[0]), 3, [0, 255, 0], -1)
    plt.figure()
    plt.imshow(corners_img, cmap='gray')
    plt.title('Detected corners')
    # show original image with detected corners
    plt.imshow(cv2.cvtColor(quad_img, cv2.COLOR_BGR2RGB))
    for index in rotated_harris.find_corners():
        cv2.circle(rotated_img, (index[1], index[0]), 3, [0, 255, 0], -1)
    plt.title('detected corners')
    cv2.imwrite('data/corners.jpg', rotated_img)
    plt.show()

if __name__ == '__main__':
    main()
```

Feature Descriptor

Output:



Output of Θ and tx, ty :

```
[[  0.648   0.672  34.019]
 [ -0.719   0.663 200.512]
 [ -0.     -0.     1.   ]]
-----theta-----
49
-----t_x-----
34
-----t_y-----
201
```

Code:

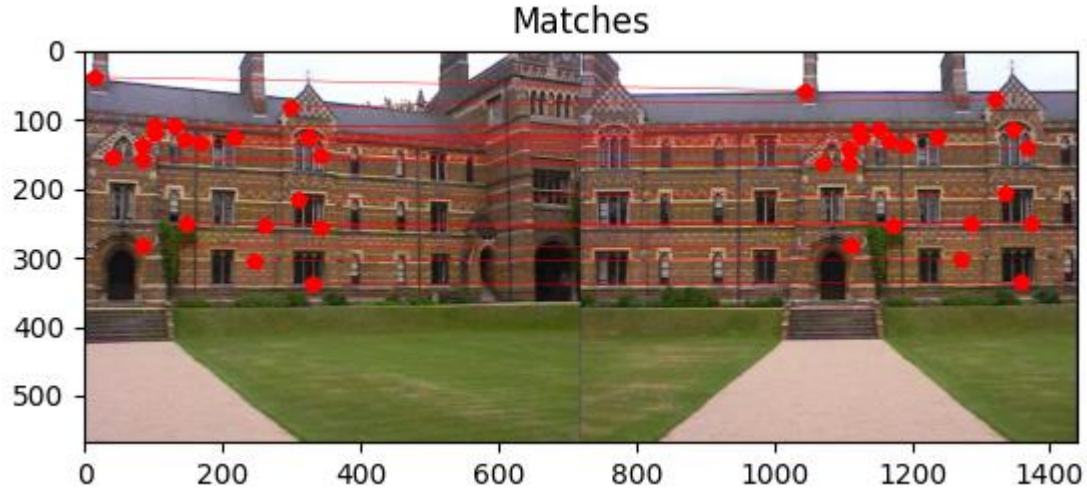
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from feature_detector import HarrisCorners
# Feature descriptor
def main():
    # read original and rotated images
    quad_img = cv2.imread('data/original.jpg', 1)
    rotated_img = cv2.imread('data/rotated.jpg', 1)
    # Featrue dectector from harris corner algo
    original_corner = HarrisCorners(quad_img,  k=0.01, threshold=0.2).find_corners()
    rotation_corners = HarrisCorners(rotated_img,  k=0.01, threshold=0.2).find_corners()
    # convert to grayscale
    img = cv2.cvtColor(quad_img, cv2.COLOR_BGR2GRAY)
    x, y = (10, 10)
    feature_des = []
    # feature descriptor add
    for i in range(len(original_corner)):
        patch = img[original_corner[i][0] - x:original_corner[i][0], original_corner[i][1] - y:original_corner[i][1]]
        #use the sobel cv2. function
        gradient_x = cv2.Sobel(patch, cv2.CV_64F, 1, 0)
        gradient_y = cv2.Sobel(patch, cv2.CV_64F, 0, 1)
        mag = (np.sqrt((gradient_x * gradient_x) + (gradient_y * gradient_y)).flatten())
        angle = (np.arctan2(gradient_y, gradient_x).flatten())
        feats = np.empty((9,0))
        for j in range(len(angle)):
            b = int(angle[j] // 20)
            feats[b] += mag[j]
        feature_des.append(sorted(feats))
        quad_feature_des = np.array(feature_des)

    img = cv2.cvtColor(rotated_img, cv2.COLOR_BGR2GRAY)
    feature_des = []
```

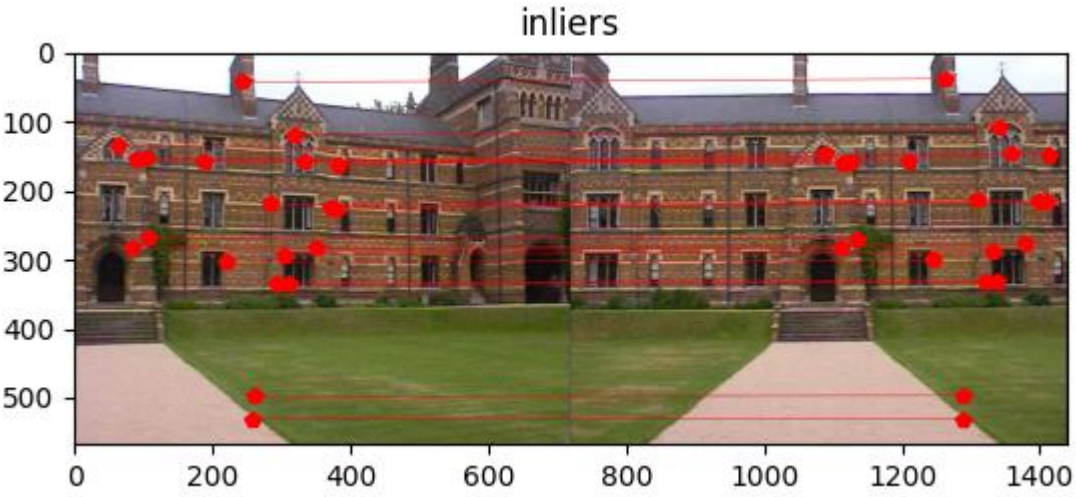
```
    img = cv2.cvtColor(rotated_img, cv2.COLOR_BGR2GRAY)
    feature_des = []
    for i in range(len(rotation_corners)):
        patch = img[rotation_corners[i][0] - x:rotation_corners[i][0], rotation_corners[i][1] - y:rotation_corners[i][1]]
        gradient_x = cv2.Sobel(patch, cv2.CV_64F, 1, 0)
        gradient_y = cv2.Sobel(patch, cv2.CV_64F, 0, 1)
        mag = (np.sqrt((gradient_x * gradient_x) + (gradient_y * gradient_y)).flatten())
        angle = (np.arctan2(gradient_y, gradient_x).flatten())

        feats = np.empty((9,0))
        for j in range(len(angle)):
            b = int(angle[j] // 20)
            feats[b] += mag[j]
        feature_des.append(sorted(feats))
        rot_feature_des = np.array(feature_des)

    matches = [[0, 0], [1, 1], [2, 2], [3, 3]]

    image = np.concatenate((quad_img, rotated_img), axis=1)

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.title('Matches')
    for f_match in matches:
        x_1, y_1 = original_corner[f_match[0]][::-1]
        x_2, y_2 = rotation_corners[f_match[1]][::-1]
        plt.plot(np.array([x_1, x_2+ 300]), np.array([y_1, y_2]), 'yo-')

    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.show()

    # linear_least_squares_optimization
    matrix_a = []
    for i in range(len(matches)):
        #matching points between original image and rotated image
        point = original_corner[matches[i][0]][::-1], rotation_corners[matches[i][1]][::-1]
        # Coordinates!
```

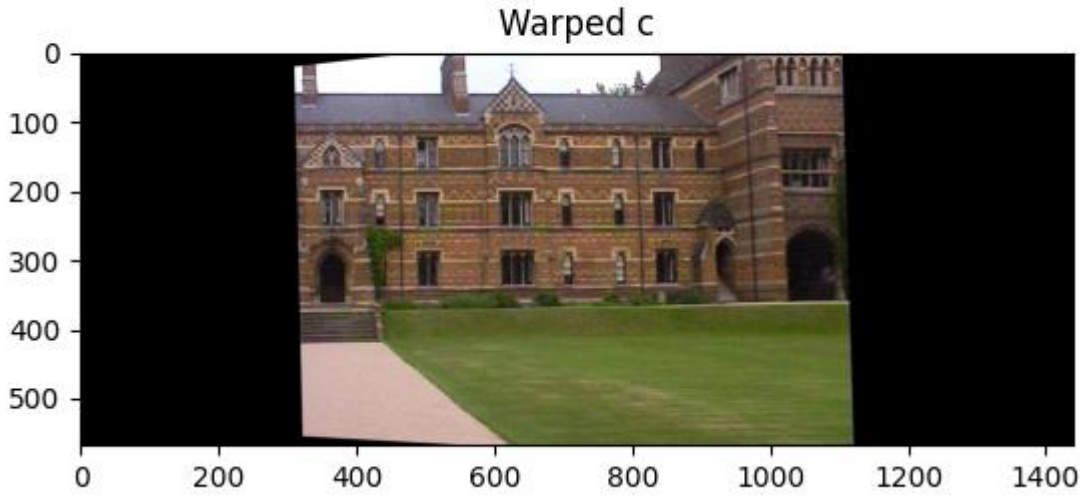
```
# linear_least_squares_optimization
matrix_a = []
for i in range(len(matches)):
    #matching points between original image and rotated image
    point = original_corner[matches[i][0]][::-1], rotation_corners[matches[i][1]][::-1]
    # Coordinates!
    (x_1, y_1), (x_2, y_2) = point[0], point[1]
    matrix_a.append([-x_1, -y_1, -1, 0, 0, 0, x_2 * x_1, x_2 * y_1, x_2]) #add linear equations in a matrix a
    matrix_a.append([0, 0, 0, -x_1, -y_1, -1, y_2 * x_1, y_2 * y_1, y_2])

#singular value decomposition
u, s, vh = np.linalg.svd(np.array(matrix_a), full_matrices=True)
print(u.shape, s.shape, vh.shape)
# get the final h_matrix
h_matrix = vh[-1].reshape(3, 3)

print(np.round(h_matrix,3))

theta = int(np.arccos(round(h_matrix[0][0], 3)) * 180 / np.pi)

t_x = int(round(h_matrix[0][2]))
t_y = int(round(h_matrix[1][2]))

print('-----theta-----')
# convert to degree
print(theta)

print('-----t_x-----')
print(t_x)

print('-----t_y-----')
print(t_y)
__name__ == '__main__':
main()
```

Problem 5. Image Stitching

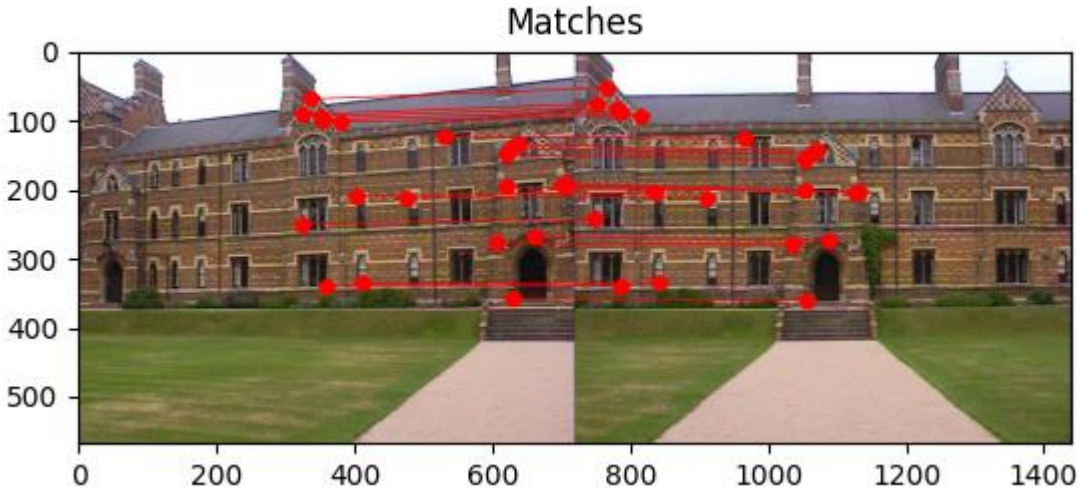
a. visualization of the set of tentative matches and the set of inliers for one image pair AND warped images for keble_b and keble_c

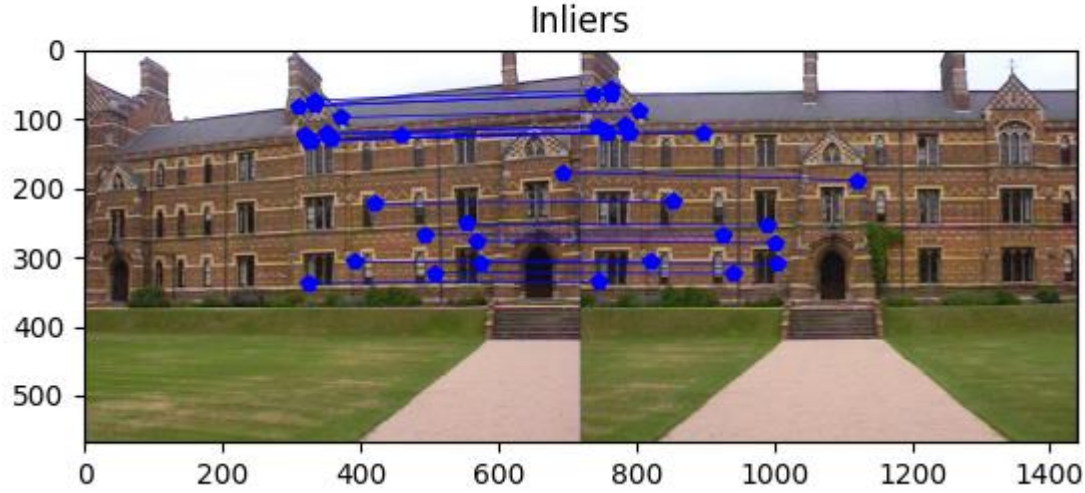


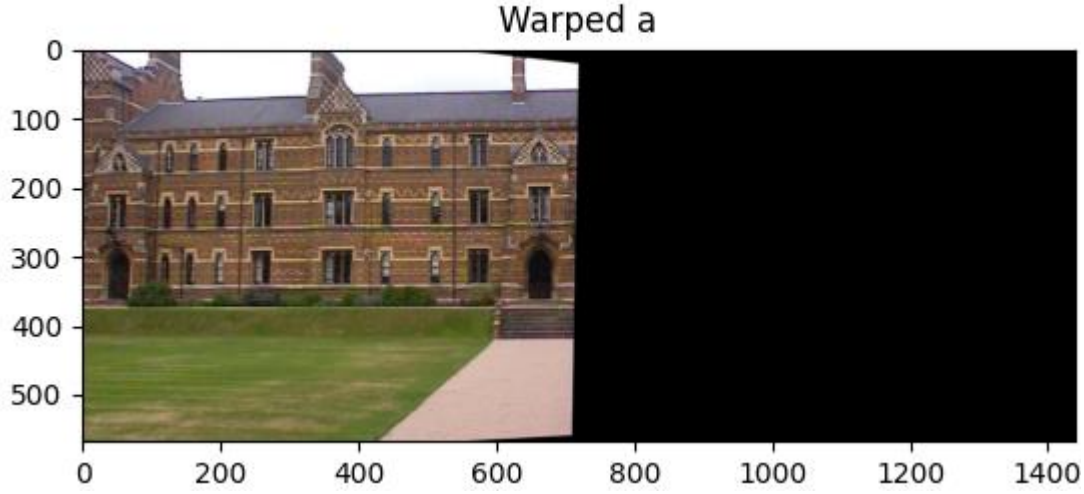




b. visualization of the set of tentative matches and the set of inliers for one image pair AND warped images for keble_a and keble_b







Final output(stitched output)

Code:

```python
import cv2
import numpy as np
import matplotlib.pyplot as plt
import random

def matching_key(kp_1, kp_2, des_1, des_2, threshold):
    # use euclidean distance to compute pairwise distances btw SIFT descriptors
    # performing "thresholding"
    print('matching start...')
    img_1_pts, img_2_pts = [], []
    for i in range(des_1.shape[0]):
        eucli_distance = []
        for j in range(des_2.shape[0]):
            # append euclidean distance with image 2 key points
            eucli_distance.append((np.linalg.norm(des_1[i] - des_2[j]), kp_2[j].pt))
        eucli_distance = sorted(eucli_distance, key=lambda x: x[0])
        # thresholding on the ratio between the first and the second
        # nearest neighbors.
        ratio = eucli_distance[0][0] / eucli_distance[1][0]
        if ratio < threshold:
            img_1_pts.append(list(kp_1[i].pt))
            img_2_pts.append(list(eucli_distance[0][1]))
    img_1_pts = np.array(img_1_pts)
    img_2_pts = np.array(img_2_pts)
    return img_1_pts, img_2_pts
'''
Robustly estimate homography using RANSAC_algo. Use a sample of 4-
points to compute each homography hypothesis. You will need to
write a function of the form:
H = computeH(im1 pts, im2 pts)
'''
```

```python
def compute_H(img1_pts, img2_pts):
    # img1_pts and im2_pts are 2xn matrices holding the (x,y)
    final = []
    for i in range(img1_pts.shape[1]):
        # need to set up a linear system of n equations( Ah = 0)
        (x_1, y_1), (x_2, y_2) = img1_pts[:, i], img2_pts[:, i]
        a = [-x_1, -y_1, -1, 0, 0, 0, x_2 * x_1, x_2 * y_1, x_2] #row1
        b = [0, 0, 0, -x_1, -y_1, -1, y_2 * x_1, y_2 * y_1, y_2] #row2
        final.append(a)
        final.append(b)
    # To get the better result, apply SVD(singular value decomposition)
    u, s, vh = np.linalg.svd(np.array(final))
    H = vh[-1].reshape(3,3)
    return H
```

```
'''
For the various RANSAC_algo parameters (number of iterations,inlier threshold),
'''
def RANSAC_algo(img_1_pts, img_2_pts, iterations, threshold):
    """
        This function is the implementation of Ransac algorithm
        :param img_1_pts: image 1 key points
        :param img_2_pts: image 2 key points
        :param iterations: How many iterations do you want?
        :param threshold: default = 0.5
        :return: best inliers
    """
    empty_inliers = []
    for i in range(iterations):
        # random sampling
        matchSample = np.random.randint(0, img_1_pts.shape[0], 4)
        pts1, pts2 = [], []
        for index in matchSample:
        #for i in range(0, 4):
            pts1.append(img_1_pts[index])
            pts2.append(img_2_pts[index])
        inliers = []
        for j in range(img_1_pts.shape[0]):
            error = np.append(img_1_pts[j], [1]).reshape(3,1)
            H_estimate = compute_H(np.transpose(pts1), np.transpose(pts2))
            output = np.matmul(H_estimate, error)
            output = output/ output[2][0]
            if np.linalg.norm(output[:2] - img_2_pts[j].reshape(2, 1)) < 5:
                inliers.append([img_1_pts[j], img_2_pts[j]])
        if len(inliers) > len(empty_inliers):
            # if the number of inliers is higher than previous iterations, update the best estimates
            empty_inliers = inliers
            empty_inliers = np.array(empty_inliers)
    return empty_inliers

def main():
    # read images
    img_a = cv2.imread('pictures/keble_a.jpg', 1)
    img_b = cv2.imread('pictures/keble_b.jpg', 1)
    img_c = cv2.imread('pictures/keble_c.jpg', 1)

    #Convert to image gray scale
    img1 = cv2.cvtColor(img_c, cv2.COLOR_BGR2GRAY)
    img2 = cv2.cvtColor(img_b, cv2.COLOR_BGR2GRAY)
    # phase 1: detect the local features(SIFT)
    sift = cv2.xfeatures2d.SIFT_create()
    # get the key points and descriptors from each images
    kp_1, des_1 = sift.detectAndCompute(img1, None)
    kp_2, des_2 = sift.detectAndCompute(img2, None)

    # get matches
    img_1_pts, img_2_pts = matching_key(kp_1, kp_2, des_1, des_2, 0.5)
    matches = np.array([[x, y] for x, y in zip(img_1_pts, img_2_pts)])

    ############ show match for image c and b #############
    img_comb = np.concatenate((img_c, img_b), axis=1)
    plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB), aspect='auto')
```

```
    # plot the matches only (not inliers)


    for i in np.random.randint(0, matches.shape[0], 20):
        x_1, y_1 = matches[i, 0, :]
        x_2, y_2 = matches[i, 1, :]
        plt.plot(np.array([x_1, x_2 + img_c.shape[1]]), np.array([y_1, y_2]), color='r', linewidth=0.3, marker='p')
    plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB))
    plt.show()
    ########### show the inliers for image c and b #########
    final_inliers = RANSAC_algo(img_1_pts, img_2_pts, 300, 0.5)
    img_comb = np.concatenate((img_c, img_b), axis=1)

    plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB), aspect='auto')


    for i in np.random.randint(0, final_inliers.shape[0], 20):
        x_1, y_1 = matches[i, 0, :]
        x_2, y_2 = matches[i, 1, :]
        plt.plot(np.array([x_1, x_2 + img_c.shape[1]]), np.array([y_1, y_2]), color='r', linewidth=0.3, marker='p')
    plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB))
    plt.show()
    print('inliers shown')

    # compute final homography
    H = compute_H(np.transpose(final_inliers[:, 0, :]), np.transpose(final_inliers[:, 1, :]))
```

```
# warp image c into reference frame of image b
warped_c = cv2.warpPerspective(img_c, H, (img_c.shape[1] + img_b.shape[1], img_c.shape[0]), flags=cv2.INTER_LINEAR)
plt.imshow(cv2.cvtColor(warped_c, cv2.COLOR_BGR2RGB))
plt.title('Warped c')
plt.show()
# stitch image b and c

warped_c[:, :500] = img_b[:, :500]

# compute final homography for image a and image b
# get matches
img_1_pts, img_2_pts = matching_key(kp_1, kp_2, des_1, des_2, 0.5)
matches = np.array([[x, y] for x, y in zip(img_1_pts, img_2_pts)])

############ show match for image a and b #############

img_comb = np.concatenate((img_a, img_b), axis=1)
w = img_a.shape[1]
plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB), aspect='auto')
# plot the matches only (not inliers)


for i in np.random.randint(0, matches.shape[0], 20):
    # for i in range(no_of_matches):
    x_1, y_1 = matches[i, 0, :]
    x_2, y_2 = matches[i, 1, :]
    plt.plot(np.array([x_1, x_2 + w]), np.array([y_1, y_2]), color='r', linewidth=0.3, marker='p')
plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB))
plt.show()

############ show the inliers for image a and b##########
final_inliers = RANSAC_algo(img_1_pts, img_2_pts, 300, 0.5)
img_comb = np.concatenate((img_a, img_b), axis=1)
w = img_a.shape[1]
plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB), aspect='auto')
```

```
    ########### show the inliers for image a and b#########
    final_inliers = RANSAC_algo(img_1_pts, img_2_pts, 300, 0.5)
    img_comb = np.concatenate((img_a, img_b), axis=1)
    w = img_a.shape[1]
    plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB), aspect='auto')


    for i in np.random.randint(0, final_inliers.shape[0], 20):
        x_1, y_1 = matches[i, 0, :]
        x_2, y_2 = matches[i, 1, :]
        plt.plot(np.array([x_1, x_2 + w]), np.array([y_1, y_2]), color='r', linewidth=0.3, marker='p')
    plt.imshow(cv2.cvtColor(img_comb, cv2.COLOR_BGR2RGB))
    plt.show()



    # compute final homography
    H_a_b = compute_H(np.transpose(final_inliers[:, 0, :]), np.transpose(final_inliers[:, 1, :]))
    H_a_b[0][2] = 0
    # warp a
    warped_a = cv2.warpPerspective(img_a, H_a_b, (img_a.shape[1] + img_b.shape[1], img_a.shape[0]), flags=cv2.INTER_LINEAR)
    plt.imshow(cv2.cvtColor(warped_a, cv2.COLOR_BGR2RGB))
    plt.title('Warped a')
    plt.show()
    # stitching image
    warped_a[:, 300:] = warped_c[:, :warped_a.shape[1] - 300]

    plt.imshow(cv2.cvtColor(warped_a, cv2.COLOR_BGR2RGB))
    plt.title('Stitched Image')
    plt.show()
    # save and write image
    cv2.imwrite('data/stitched_image.jpg', warped_a)

if __name__ == "__main__":
    main()
```