ECE 66100 Computer Vision **Name:** *Nikita Ravi (ravi30)*
Homework 3
09/17/2022

1 Theoretical Background

1.1 Point to Point Correspondence

To remove the distortions from the image, we adopted a technique called Point to Point Correspondence. This technique involves a transformation of a homogeneous point in the domain $\vec{x}$ to a homogeneous point in the range $\vec{x'}$ by a homography matrix H where H represents a linear mapping from one homogeneous coordinate to another in different planes of the same dimension. $\vec{x'}$ is computed by multiplying H with $\vec{x}$. The matrix H can be expressed as

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}$$

Therefore, using the relation mentioned above, the homogeneous point in the range is given by:

$$x'_1 = h_{11}x_1 + h_{12}x_2 + h_{13}x_3$$

$$x'_2 = h_{21}x_1 + h_{22}x_2 + h_{23}x_3$$

$$x'_3 = h_{31}x_1 + h_{32}x_2 + h_{33}x_3$$

To obtain the coordinates in the range on the real plane, we have to divide the first two coordinates of the homogeneous coordinate by the third coordinate, thus giving

$$x' = \frac{x'_1}{x'_3} = \frac{h_{11}x_1 + h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

$$y' = \frac{x'_2}{x'_3} = \frac{h_{21}x_1 + h_{22}x_2 + h_{23}x_3}{h_{31}x_1 + h_{32}x_2 + h_{33}x_3}$$

We can simplify the x' and y' coordinate to also include the actual coordinates in the domain on the real plane by dividing the RHS by x_3 as shown below

$$x' = \frac{h_{11}x + h_{12}y + h_{13}}{h_{31}x + h_{32}y + h_{33}}$$

$$y' = \frac{h_{21}x + h_{22}y + h_{23}}{h_{31}x + h_{32}y + h_{33}}$$

We make the above x' and y' equations into a system of linear equations:

$$h_{11}x + h_{12}y + h_{13} - h_{31}xx' - h_{32}yx' - h_{33}x' = 0$$

$$h_{21}x + h_{22}y + h_{23} - h_{31}xy' - h_{32}yy' - h_{33}y' = 0$$

Because there are nine unknowns, we will have eight system of linear equations representing four different homogeneous points. This can be represented in the matrix form $A\vec{h} = \vec{b}$

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}$$

and therefore, $\vec{h}$ is calculated by doing the following computation: $A^{-1}\vec{b}$. We then rearrange $\vec{h}$ into the 3×3 matrix H shown in the previous page, where $h_{33} = 1$. We can now map between the world plane and the image plane using this homography.

1.2 Two-Step Method

The two-step method involves first removing the projective distortion using the vanishing line method and then removing the affine distortion using the cosine method.

1.2.1 Removing the Projective Distortion

Removing the projective distortion from an image involves mapping the vanishing lines in the image back to parallel lines in the world plane. This is achieved by mapping the vanishing line back to the line at infinity. To derive the homography matrix to remove the projective distortion, we will require two distinct pair of parallel lines in the world plane. This is because, we know that when an image has projective distortion, the parallel lines that exist in the real world will have been transformed into two lines appearing to converge at a certain vanishing point. We require two such vanishing points to compute the vanishing line which is necessary to construct the homography matrix. If the HC format of the vanishing line is $\begin{bmatrix} l_1 & l_2 & l_3 \end{bmatrix}^T$, then the homography matrix H is

$$H_p = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ l_1 & l_2 & l_3 \end{bmatrix}$$

We then use H_p on the image to remove the projective distortion

1.2.2 Removing the Affine Distortion

Now that we restored the parallelism features of the image, we now need to figure out a way to restore all the perpendicular angles that existed in the image in the world plane. In other words, even though the parallel lines are restored, the angles are not restored so there exists an affine distortion. The distortion is removed via the cosine method:

$$cos(\theta) = \frac{\vec{l}^T \mathbf{C}_\infty^* \vec{m}}{\sqrt{(\vec{l}^T \mathbf{C}_\infty^* \vec{l})(\vec{m}^T \mathbf{C}_\infty^* \vec{m})}}$$

The above expression is set equal to zero since θ is set to 90 degrees so the expression is further simplified to

$$\vec{l}^T \mathbf{C}_\infty^* \vec{m} = 0$$

In order to estimage what the affine transformation homography H_a is from the above simplified expression, we substitute $H_a^T \vec{l'}$ into $\vec{l}$ and $H_a^T \vec{m'}$ into $\vec{m}$, thus giving

$$\vec{l'^T} H_a \mathbf{C}_\infty^* H_a^T \vec{m'} = 0$$

where $H_a \mathbf{C}_\infty^* H_a^T = \begin{bmatrix} AA^T & 0 \\ 0 & 0 \end{bmatrix}$.

We denote AA^T as matrix $S = \begin{bmatrix} s_{11} & s_{12} \\ s_{12} & s_{22} \end{bmatrix}$. The component s_{22} is set to one because we want to preserve the ratios. From this we get the linear equation

$$s_{11} m'_1 l'_1 + s_{12}(l'_1 m'_2 + l'_2 m'_1) + s_{22} l'_2 m'_2 = 0$$

Because there are only two unknowns (s_{11} and s_{12}) in the above expression, we require only two distinct pairs of orthogonal lines to solve for S. Once we have S, we can determine what A by single value decomposition (SVD) operation: $S = AA^T = VD^2V^T$ where $A = VDV^T$

Once we compute A, we can set up the affine transformation homography as

$$H_a = \begin{bmatrix} A & 0 \\ 0 & 1 \end{bmatrix}$$

This transformation homography is used to restore the angles in the image after the projective distortion is removed.

1.3 One-Step Method

The one-step method removes both the projective and affine distortions in one go.

Let **C'*** be a projection of the dual degenerate conic $\mathbf{C}_\infty^*$, then the one-step method uses a transformation homography H that maps **C'*** back to $\mathbf{C}_\infty^*$, thus removing both distortions.

The projection of the dual degenerate conic can be expressed as follows

$$\mathbf{C'}^* = H_a \mathbf{C}_\infty^* H_a^T = \begin{bmatrix} a & \frac{b}{2} & \frac{d}{2} \\ \frac{b}{2} & c & \frac{e}{2} \\ \frac{d}{2} & \frac{e}{2} & f \end{bmatrix}$$

f is set to one to maintain the ratios, so there are a total of five unknowns. As a result, we require five distinct orthogonal lines to solve this matrix. Similar to the two-step method, we find the singular value decomposition (SVD) of **C'*** to determine $\mathbf{AA}^T$. Once we have A, the vector $\vec{v}$ is calculated using the following relation:

$$\mathbf{A}\vec{v} = \begin{bmatrix} \frac{d}{2} \\ \frac{e}{2} \end{bmatrix}$$

The transformation homography is given by

$$\mathbf{H} = \begin{bmatrix} A & 0 \\ v^T & 1 \end{bmatrix}$$

This transformation homography is used to restore both the parallel lines and the angles in the image.

2 The Inputs and Outputs

Figure 1: Building Input

Figure 2: Nighthawks Input

Figure 3: Pikachu Input

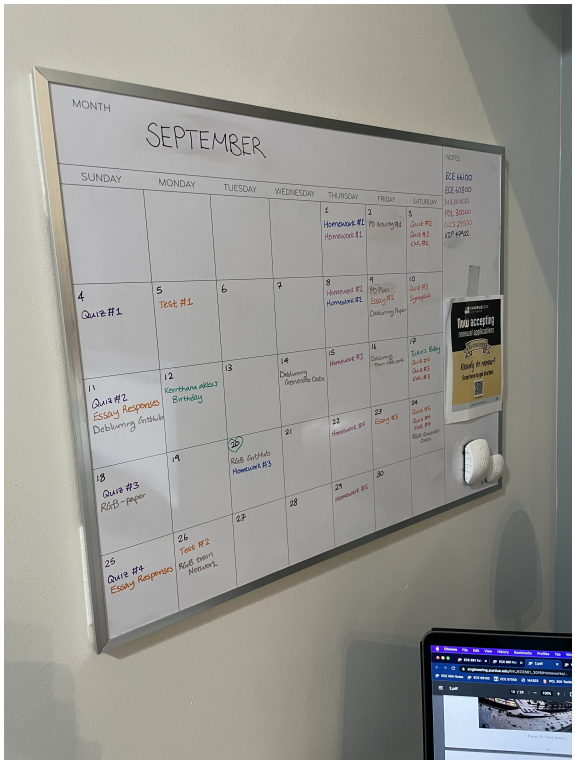

Figure 4: Calendar Input

Figure 5: Building Point-to-Point

Figure 6: Nighthawks Point-to-Point

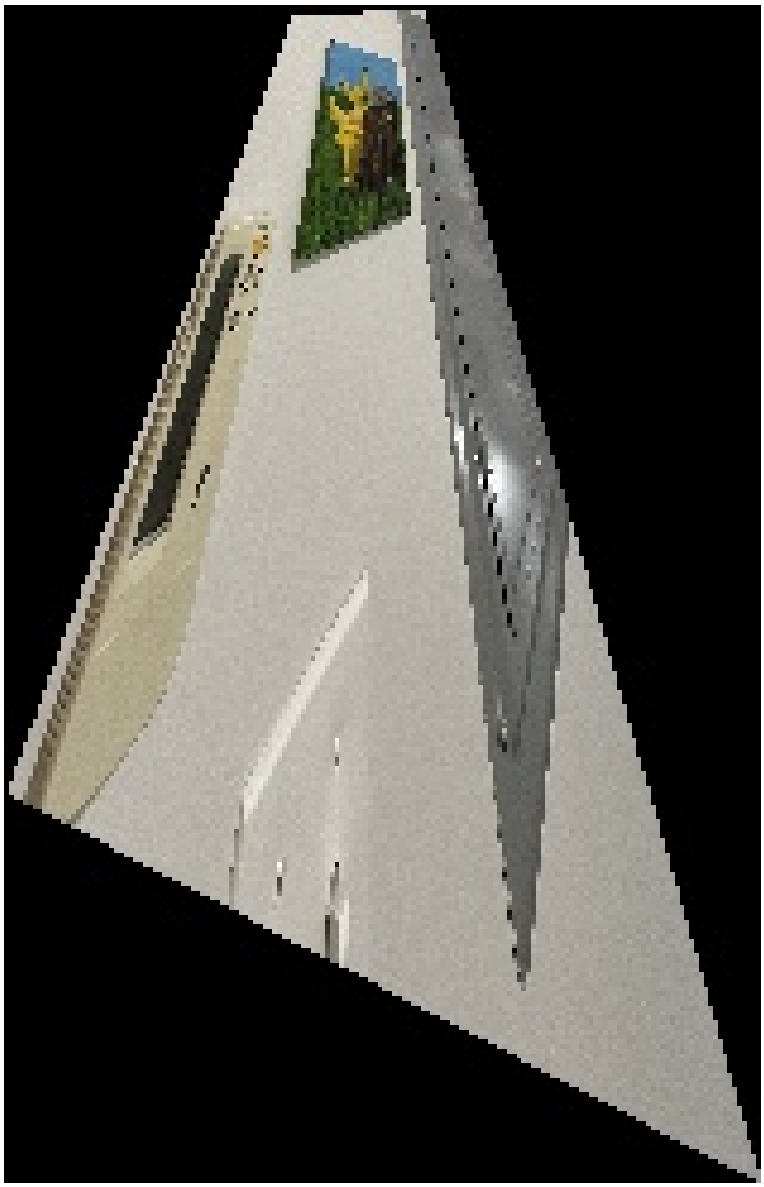

Figure 7: Pikachu Point-to-Point

Figure 8: Calendar Point-to-Point

Figure 9: Building Two-Step

Figure 10: Nighthawks Two-Step

Figure 11: Pikachu Two-Step

Figure 12: Calendar Two-Step

Figure 13: Building One-Step

Figure 14: Nighthawks One-Step

Figure 15: Pikachu One-Step

Figure 16: Calendar One-Step

1 Observations

From conducting the point-to-point correspondence method, the two-step method, and the one-step method, I came to the conclusion that the two-step method is the efficient choice when it comes to getting robust and quick results. The two-step method was more optimized in terms of run-time the one-step and point-to-point correspondence methods. The one-step method also requires five accurate pairs of orthogonal lines or otherwise you'd end up getting the outputs shown in the previous pages. Whereas, with the two-step method, you'd only need two pairs of orthogonal lines and you'd still end up getting decent results.

2 The Code

```
# Name: Nikita Ravi
# Class: ECE 66100
# Homework #3
# Deadline: 09/20/2022

# Import Modules
import numpy as np
import cv2
import math

def displayimage(image):
    cv2.imshow("window", image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    quit()

def computehomography(distorted, undistorted):
    n = distorted.shape[1]
    A = np.zeros((2*n, 2*n))
    b = np.zeros((2*n, 1))
    H = np.zeros((3,3))

    for idx in range(n):
        A[2*idx] = [undistorted[0][idx], undistorted[1][idx], 1, 0, 0, 0, (-
    undistorted[0][idx] * distorted[0][idx]), (-undistorted[1][idx] *
    distorted[0][idx])]
        A[2*idx + 1] = [0, 0, 0, undistorted[0][idx], undistorted[1][idx], 1,
    (-undistorted[0][idx] * distorted[1][idx]), (-undistorted[1][idx] *
    distorted[1][idx])]
        b[2*idx] = distorted[0][idx]
        b[2*idx + 1] = distorted[1][idx]

    h = np.matmul(np.linalg.inv(A), b)

```

```
    row = 0
    for idx in range(0, len(h), 3):
        spliced = h[idx:idx+3]
        if(len(spliced) == 3):
            H[row] = spliced.T
        else:
            H[row] = np.append(spliced, [1])
        row += 1

    return H

def createblankimage(H, height, width):
    corners = np.array([[0,0,1], [0,width,1], [height, width, 1], [height
    ,0,1]])
    cornerPrime = np.matmul(H, corners.T).T
    for idx in range(len(cornerPrime)):
        cornerPrime[idx] = cornerPrime[idx] / cornerPrime[idx][2]

    minimum, maximum = cornerPrime.min(axis = 0), cornerPrime.max(axis = 0)
    ymin, ymax, xmin, xmax = int(math.floor(minimum[0])), int(math.ceil(
   maximum[0])), int(math.floor(minimum[1])), int(math.ceil(maximum[1]))

    blank = np.zeros((ymax - ymin, xmax - xmin, 3))
    return blank, ymin, xmin

def transform(undistorted, blank, H, ymin, xmin, path):
    undistortedheight, undistortedwidth, undistortedchannel = undistorted.
   shape
    blankheight, blankwidth, blankchannel = blank.shape

    for i in range(blankheight):
        for j in range(blankwidth):
            xPrime = np.array((i+ymin,j+xmin,1),dtype = 'l')
            X = np.matmul(np.linalg.inv(H), xPrime.T)
            X /= X[-1]
            X = X.astype(int)

            if X[0]¡undistortedheight and X[0]¿=0 and X[1]¡undistortedwidth
   and X[1]¿=0:
                blank[i][j] = undistorted[X[0]][X[1]]

    cv2.imwrite(path, blank)

def pointtopointcorrespondence(undistortedimg, undistortedpoints,
   distortedpoints, path):
    H = computehomography(undistortedpoints, distortedpoints)
    height, width, channel = undistortedimg.shape
    blank, ymin, xmin = createblankimage(H, height, width)
    transform(undistortedimg, blank, H, ymin, xmin, path)

def calculatelinesfrompoints(pt1, pt2):
    line = np.cross(pt1, pt2)
    return line

```

```
def calculatepointfromlines(l1, l2):
    point = np.cross(l1, l2)
    return point

def homographytoremoveprojectivedistortion(P,Q,R,S):
    l1 = calculatelinesfrompoints(P, S) #Line PS
    l2 = calculatelinesfrompoints(Q, R) #Line QR
    vanishingPoint1 = calculatepointfromlines(l1, l2)
    vanishingPoint1 = vanishingPoint1 / vanishingPoint1[-1]

    l3 = calculatelinesfrompoints(P, Q) # Line PQ
    l4 = calculatelinesfrompoints(S, R) # Line SR
    vanishingPoint2 = calculatepointfromlines(l3, l4)
    vanishingPoint2 = vanishingPoint2 / vanishingPoint2[-1]

    vanishingLine = calculatelinesfrompoints(vanishingPoint1,
   vanishingPoint2)
    vanishingLine = vanishingLine / vanishingLine[-1]

    H = np.zeros((3,3))
    H[0][0] = H[1][1] = 1
    H[2] = vanishingLine

    return H

def homographytoremoveaffinedistortion(P,Q,R,S):
    """
    P ------- S
    -         -
    -         -
    -         -
    -         -
    Q ------- R
    """

    lpq = calculatelinesfrompoints(P, Q) # Line PQ
    lpq = lpq / lpq[-1]

    lsr = calculatelinesfrompoints(S, R) # Line SR
    lsr = lsr / lsr[-1]

    lps = calculatelinesfrompoints(P, S) # Line PS
    lps = lps / lps[-1]

    lqr = calculatelinesfrompoints(Q, R) # Line QR
    lqr = lqr / lqr[-1]

    A = np.zeros((2, 2))
    A[0][0] = lpq[0] * lqr[0]
    A[0][1] = lpq[0] * lqr[1] + lpq[1] * lqr[0]
    A[1][0] = lps[0] * lsr[0]
    A[1][1] = lps[0] * lsr[1] + lps[1] * lsr[0]

    b = np.zeros((2,1))
```

```
    b[0] = -lpq[1] * lqr[1]
    b[1] = -lps[1] * lsr[1]

    s11s12 = np.matmul(np.linalg.inv(A), b)
    S = np.array([[s11s12[0], s11s12[1]], [s11s12[1], 1]], dtype=float)

    u, s, vh = np.linalg.svd(S) # S = V * D 2  * V.T

    D = np.sqrt(np.diag(s))
    A = np.matmul(np.matmul(u, D), u.T)

    # H = np.zeros((3,3))
    H = np.append(A,[[0,0]], axis=0)
    H = np.append(H,[[0],[0],[1]], axis=1)
    return H

def twostepmethod(image, X, path):
    P,Q,R,S = X[0], X[1], X[2], X[3]
    removeprojectivehomography = homographytoremoveprojectivedistortion(P,Q,R,S)
    imagewithonlyaffine = np.matmul(removeprojectivehomography, X.T).T

    P,Q,R,S = imagewithonlyaffine[0], imagewithonlyaffine[1], imagewithonlyaffine[2], imagewithonlyaffine[3]
    removeaffinehomography = homographytoremoveaffinedistortion(P,Q,R,S)

    H = np.matmul(np.linalg.inv(removeaffinehomography), removeprojectivehomography)
    height, width, channel = image.shape
    blank, ymin, xmin = createblankimage(H, height, width)
    transform(image, blank, H, ymin, xmin, path)

def homographytoremoveprojectiveandaffine(P,Q,R,S, Psq, Qsq, Rsq, Ssq):
    """
    P ------- S
    |         |
    |         |
    |         |
    |         |
    Q ------- R
    """

    lpq = np.cross(P, Q)
    lpq = lpq / lpq[-1]

    lsr = np.cross(S, R)
    lsr = lsr / lsr[-1]

    lps = np.cross(P, S)
    lps = lps / lps[-1]

    lqr = np.cross(Q, R)
    lqr = lqr / lqr[-1]
```

```
    lprsq = np.cross(Psq, Rsq)
    lprsq = lprsq / lprsq[-1]

    lsqsq = np.cross(Qsq,Ssq)
    lsqsq = lsqsq / lsqsq[-1]

    A = np.zeros((5,5))
    b = np.zeros((5,1))

    # PQ and QR
    A[0][0] = lpq[0] * lqr[0]
    A[0][1] = (lpq[0] * lqr[1] + lpq[1] * lqr[0]) / 2
    A[0][2] = lpq[1] * lqr[1]
    A[0][3] =(lpq[0] * lqr[2]+ lpq[2] * lqr[0]) / 2
    A[0][4] = (lpq[2] * lqr[1] + lpq[1] * lqr[2]) / 2
    b[0] = -lpq[2] * lqr[2]

    # PQ and PS
    A[1][0] = lpq[0] * lps[0]
    A[1][1] = (lpq[0] * lps[1] + lpq[1] * lps[0]) / 2
    A[1][2] = lpq[1] * lps[1]
    A[1][3] = (lpq[2] * lps[0] + lpq[0] * lps[2]) / 2
    A[1][4] = (lpq[2] * lps[1] + lpq[1] * lps[2]) / 2
    b[1] = -lpq[2] * lps[2]

    # SR and PS
    A[2][0] = lsr[0] * lps[0]
    A[2][1] = (lsr[0] * lps[1] + lsr[1] * lps[0]) / 2
    A[2][2] = lsr[1] * lps[1]
    A[2][3] = (lsr[2] * lps[0] + lsr[0] * lps[2]) / 2
    A[2][4] = (lsr[2] * lps[1] + lsr[1] * lps[2]) / 2
    b[2] = -lsr[2] * lps[2]

    # SR and QR
    A[3][0] = lsr[0] * lqr[0]
    A[3][1] = (lsr[0] * lqr[1] + lsr[1] * lqr[0]) / 2
    A[3][2] = lsr[1] * lqr[1]
    A[3][3] = (lsr[2] * lqr[0] + lsr[0] * lqr[2]) / 2
    A[3][4] = (lsr[2] * lqr[1] + lsr[1] * lqr[2]) / 2
    b[3] = -lsr[2] * lqr[2]

    # PRsq and SQsq
    A[4][0] = lprsq[0] * lsqsq[0]
    A[4][1] = (lprsq[0] * lsqsq[1] + lprsq[1] * lsqsq[0]) / 2
    A[4][2] = lprsq[1] * lsqsq[1]
    A[4][3] = (lprsq[2] * lsqsq[0] + lprsq[0] * lsqsq[2]) / 2
    A[4][4] = (lprsq[2] * lsqsq[1] + lprsq[1] * lsqsq[2]) / 2
    b[4] = -lprsq[2] * lsqsq[2]

    stemp = np.matmul(np.linalg.inv(A), b)
    stemp = stemp / np.max(stemp)

    S = np.zeros((2,2))
```

```
    S[0][0] = stemp[0]
    S[0][1] = S[1][0] = stemp[1] / 2
    S[1][1] = stemp[2]

    u, s, vh = np.linalg.svd(S)

    D = np.sqrt(np.diag(s))
    A = np.matmul(np.matmul(u, D), u.T)

    temp = np.array([stemp[3]/2, stemp[4]/2])
    v = np.matmul(np.linalg.inv(A), temp)

    H = np.zeros((3,3))
    H[0][0] = A[0][0]
    H[0][1] = A[0][1]
    H[1][0] = A[1][0]
    H[1][1] = A[1][1]
    H[2][0] = v[0]
    H[2][1] = v[1]
    H[2][2] = 1
    return H

def onestepmethod(image, X, Xsq, path):
    P,Q,R,S = X[0], X[1], X[2], X[3]
    Psq, Qsq, Rsq, Ssq = Xsq[0], Xsq[1], Xsq[2], Xsq[3]
    H = homographytoremoveprojectiveandaffine(P,Q,R,S, Psq, Qsq, Rsq,
    Ssq)

    height, width, channel = image.shape
    blank, ymin, xmin = createblankimage(H, height, width)
    transform(image, blank, H, ymin, xmin, path)

def getimages(path, image):
    path += image
    img = cv2.imread(path)

    return img

def getwidthheightcoordinates(image):
    coor = np.array([])
    if(image == "building"):
        coor = np.array([[0,0,1], [0,900,1], [300, 900, 1], [300, 0, 1]])
    elif(image == "nighthawks"):
        coor = np.array([[0,0,1], [0,85,1], [150,85,1], [150,0,1]])
    elif(image == "pikachu"):
        coor = np.array([[0,0,1], [0,25,1], [25,25,1], [25,0,1]])
    elif(image == "calendar"):
        coor = np.array([[0,0,1], [0,42,1], [53,42,1], [53,0,1]])

    return coor

def getpqrscoordinates(image):
    pqrs = np.array([])
    if(image == "building"):
```

```
        pqrs = np.array([[243,204,1], [236,370,1], [296,375,1], [299,218,1]])
    elif(image == "nighthawks"):
        pqrs = np.array([[77,182,1], [77,656,1], [807,620,1], [815,208,1]])
    elif(image == "pikachu"):
        pqrs = np.array([[1071,596,1], [1108,2724,1], [2271,2432,1],
    [2312,984,1]])
    elif(image == "calendar"):
        pqrs = np.array([[161,342,1], [506,3471,1], [2659,2596,1],
    [2677,769,1]])

    return pqrs

def getsquarepqrscoordinates(image):
    pqrs = np.array([])

    if(image == "building"):
        pqrs = np.array([[243,204,1], [239,281,1], [296,291,1], [299,218,1]])
    elif(image == "nighthawks"):
        pqrs = np.array([[1095,271,1], [1095,598,1], [1408,628,1],
    [1400,236,1]])
    elif(image == "pikachu"):
        pqrs = np.array([[1071,596,1], [1108,2724,1], [2271,2432,1],
    [2312,984,1]])
    elif(image == "calendar"):
        pqrs = np.array([[748,1006,1], [789,1485,1], [1140,1480,1],
    [1103,1025,1]])

    return pqrs

if __name__ == "__main__":
    building = getimages(r"./hw3images/", "building.jpg")
    buildingworldcoordinates = getwidthheightcoordinates("building")
    buildingPQRScoordinates = getpqrscoordinates("building")
    buildingsquarepqrscoordinates = getsquarepqrscoordinates("building")

    nighthawks = getimages(r"./hw3images/", "nighthawks.jpg")
    nighthawksworldcoordinates = getwidthheightcoordinates("nighthawks")
    nighthawksPQRScoordinates = getpqrscoordinates("nighthawks")
    nighthawkssquarepqrscoordinates = getsquarepqrscoordinates("
    nighthawks")

    pikachu = getimages(r"./hw3images/", "pikachu.jpg")
    pikachuworldcoordinates = getwidthheightcoordinates("pikachu")
    pikachuPQRScoordinates = getpqrscoordinates("pikachu")
    pikachusquarepqrscoordinates = getsquarepqrscoordinates("pikachu")

    calendar = getimages(r"./hw3images/", "calendar.jpg")
    calendarworldcoordinates = getwidthheightcoordinates("calendar")
    calendarPQRScoordinates = getpqrscoordinates("calendar")
    calendarsquarepqrscoordinates = getsquarepqrscoordinates("calendar")

    # Task 1.1a
    pointtopointcorrespondence(building, buildingworldcoordinates.T,
    buildingPQRScoordinates.T, r"./Task 1a/buildingtransformed.jpg")
```

```
pointtopointcorrespondence(nighthawks, nighthawksworldcoordinates.T,
nighthawksPQRScoordinates.T, r"./Task 1a/nighthawkstransformed.jpg")

# Task 1.1b
## Two-Step
twostepmethod(building, buildingPQRScoordinates, r"./Task 1b/
buildingtransformed.jpg")
twostepmethod(nighthawks, nighthawksPQRScoordinates, r"./Task 1b/
nighthawkstransformed.jpg")

## One-Step
onestepmethod(building, buildingPQRScoordinates,
buildingsquarepqrscoordinates, r"./Task 1c/buildingtransformed.jpg")
onestepmethod(nighthawks, nighthawksPQRScoordinates,
nighthawkssquarepqrscoordinates, r"./Task 1c/nighthawkstransformed.jpg
")

# Task 2.1a
pointtopointcorrespondence(pikachu, pikachuworldcoordinates.T,
pikachuPQRScoordinates.T, r"./Task 2a/pikachutransformed.jpg")
pointtopointcorrespondence(calendar, calendarworldcoordinates.T,
calendarPQRScoordinates.T, r"./Task 2a/calendartransformed.jpg")

# Task 2.1b
## Two-Step
twostepmethod(pikachu, pikachuPQRScoordinates, r"./Task 2b/
pikachutransformed.jpg")
twostepmethod(calendar, calendarPQRScoordinates, r"./Task 2b/
calendartransformed.jpg")

## One-Step
onestepmethod(pikachu, pikachuPQRScoordinates,
pikachusquarepqrscoordinates, r"./Task 2c/pikachutransformed.jpg")
onestepmethod(calendar, calendarPQRScoordinates,
calendarsquarepqrscoordinates, r"./Task 2c/calendartransformed.jpg")
```

Listing 1: The Code