ECE 661 Computer Vision

Homework 2

Vineeth Ravi

September 2, 2018

PU ID : 0030019456

EMAIL ID : ravi24@purdue.edu

1 Theory and Mathematical Background

In this section we describe the Mathematical Equations required and the steps required for the solution to Homework 2. We first, compute the Homography H matrix , which maps coordinates from the world plane to the image plane. Then, we map the coordinates of the world plane to the image plane , and compute the weighted average of pixel values in the image and insert it back into the world plane. This procedure is undertaken for performing tasks 1(a) , 1(b) and 2. The steps are described in detail in the following subsections.

1.1 Homography Estimation

The transformation of a point from 1 plane to another plane can be described by the mathematical equation:-

$$x' = Hx$$

where the matrix H is given by:-

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

Let $X' = [x', y', z']$ and $X = [x, y, z]^T$

Then we have the following set of equations :-

$$\begin{aligned} x'_1 &= h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \\ x'_2 &= h_{21}x_1 + h_{22}x_2 + h_{23}x_3 \\ x'_3 &= h_{31}x_1 + h_{32}x_2 + h_{33}x_3 \end{aligned}$$

The point in the physical world plane is given by (x , y) and a point in the physical image plane is given by (x' , y'). So, we have :-

$x = \frac{x_1}{x_3}$, $y = \frac{x_2}{x_3}$

$x' = \frac{x'_1}{x'_3}$, $y' = \frac{x'_2}{x'_3}$

Combining the mathematical equations above we get :-

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

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

We set $h_{33} = 1$, Since only the ratios matter in the matrix. Hence we are left with :-

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

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

We have 8 unknowns from 2 equations, So we require a minimum of 4 points to find all the unknowns. We write the equations in the form of a matrix :-

$$AH = b$$

$$\text{where A} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ \vdots & & & & & & & \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix}$$

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

$$\text{and b} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix}$$

Solving this matrix equation using pinv (pseudo-inverse) for the general case or (inverse) for the 4 points case we get

$$H = A^{-1}b.$$

Hence we have found the Homography matrix H between the 2 planes (the world plane and the image plane).

1.2 Weighted Average RGB Pixel Value

Once, we have found the Homography matrix, we use the Weighted average transformation of Pixels to compute the pixel values of the image plane and transfer it to the world plane. The metric used for the weighted average transformation is Euclidean Distance.

We employ this method because it gives more accurate results when compared to Bilinear interpolation and rounding off methods. We apply the transformation to a given coordinate in the world plane and obtain the coordinates in the image plane. If the coordinate are not integers, we computed the weighted average of the surrounding pixels, to get the pixel value using the Weighted Average Transform (Euclidean norm) and insert it back into the world plane.

For a given point p = (x',y') , the neighboring pixels are chosen as $p_1 : (x_1, y_1)$,$p_2 : (x_2, y_2)$, $p_3 : (x_1, y_2)$ and $p_4 : (x_2, y_2)$, where x_1 is the highest integer less than or equal to x', x_2 is the lowest integer greater than or equal to x', where y_1 is the highest integer is less than or equal to y' and y_2 is the lowest integer greater than or equal to y'. So, the value of the pixel at (x', y') is given by -

$$\text{Pixel Value} = \frac{d(p',p_1)f(p_1)+d(p',p_2)f(p_2)+d(p',p_3)f(p_3)+d(p',p_4)f(p_4)}{d(p',p_1)+d(p',p_2)+d(p',p_3)+d(p',p_4)}$$

where $d(p', p_i)$ is the euclidean distance between points p' and p_i, $i \in 1, 2, 3, 4$.

Hence, we have now computed the Pixel Value of the Image Plane , which is to be inserted back into the World Plane.

2 Tasks

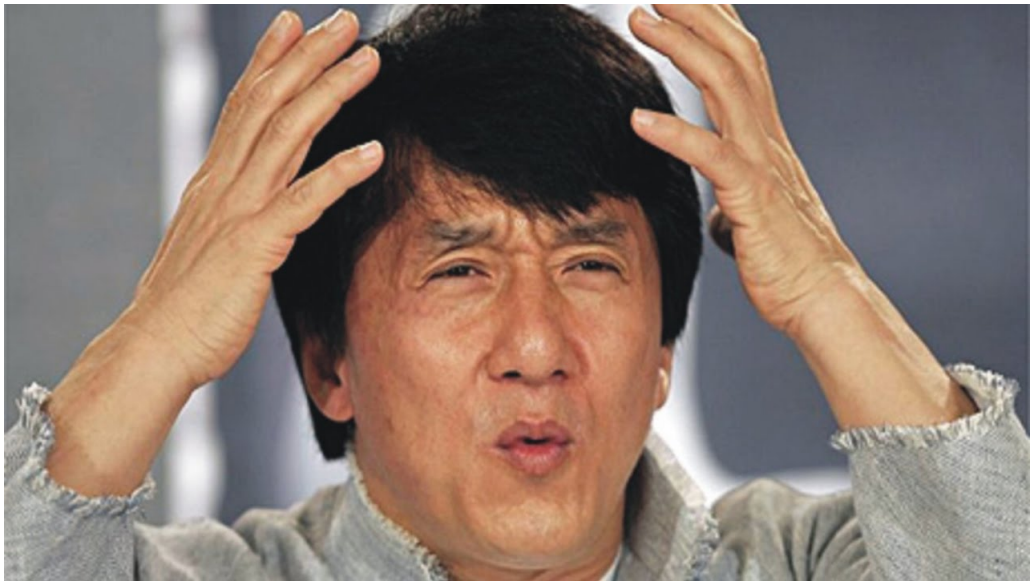
We now implement the above mathematical equations to obtain solutions for the Tasks 1(a), 1(b) and 2. We describe each of the tasks below, and the results as well as solutions for each task is included below :-

The input images for Tasks 1(a) and 1(b) are shown in figure 1 :-

(a) Image 1

(b) Image 2

(c) Image 3



(d) Jackie

Figure 1: Images given for Task 1

2.1 Task 1(a)

In this task, we need to project the image of Jackie onto the world planes of Image 1,2 and 3. We first obtain the boundary points of P,Q,R,S of each of the images using Irfan View Editor. In python, we first compute the Homography Matrix H, as described in section 1, between the images 1(d) and 1(a) ; 1(d) and 1(b); 1(d) and 1(c) , using the 4 boundary points P,Q,R,S and the image boundary of the Jackie image. Now we apply the Homography to the world plane image, to get the mapping onto the corresponding image plane. Now, we use the weighted average pixel value method described in section 1, to calculate the pixel values and insert it back into the world plane.

This method is used rather than applying the image plane transform into the world plane , because the latter would cause a lot of noise and black spots, Since the image plane cannot perfectly map with the reference plane, especially near the boundary regions.

Hence we have now projected the Jackie image 1(d) to the world planes 1(a), 1(b) and 1(c). The results obtained are shown below :-

(a) Jackie Output image1

(b) Jackie Output image2

(c) Jackie Output image3

Figure 2: Output images for Task 1(a)

2.2 Task 1(b)

Similar to the approach and logic we followed to complete Task 1(a) :-

In this task, we compute the Homographies between 1(a) and 1(b) ; 1(b) and 1(c). We then compute the resultant Homography matrix by multiplying the Homographies $H_{ba} * H_{cb}$. We do this, because of the same reason as in Task 1(a), to avoid noises and black spots inside the image boundary. We fill in the the pixel values of those coordinates in image3 which are not mapped within the image boundary of image1 with '0' or zeros so that they appear in black color. Again, here we used the Weighted Average method for getting pixel values, to get best results and reduce distortion. Finally the pixel values of 1(a) are inserted into 1(c).

The resultant image looks similar to image 1(c). The output is shown below :-

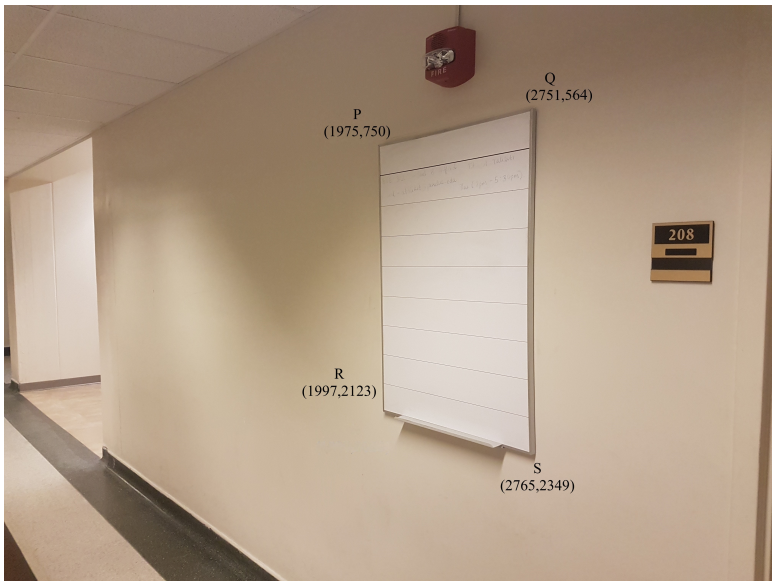
(a) Projection of Image 1 on Image 3

Figure 3: Output image for Task 1(b)

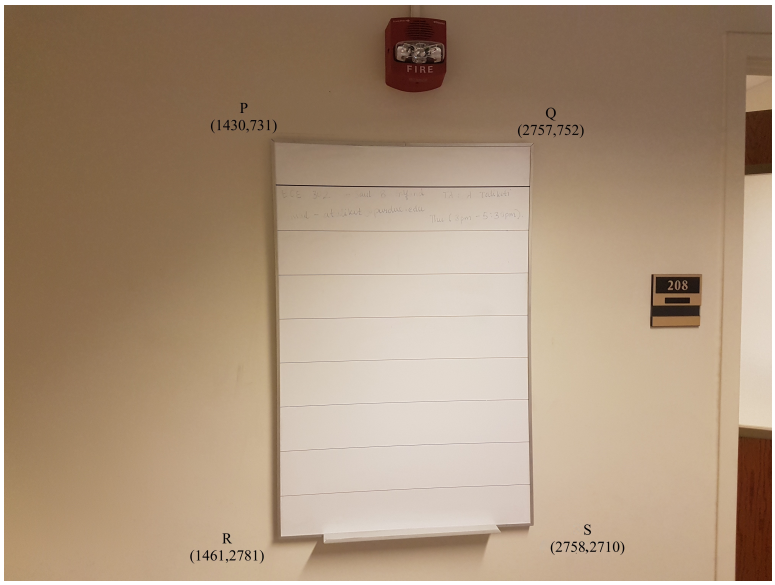
2.3 Task 2

For this task, all procedures are same as Task 1, The points chosen for computing the Homographies are mentioned in the images. Since only the input images are changed to custom taken images, the final results are directly shown below :-

The input images for Task2 are shown in figure 4 :-

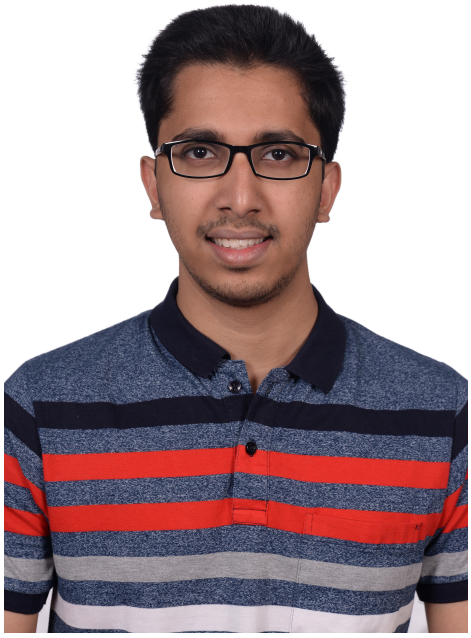


(a) Custom Image 1



(b) Custom Image 2

(c) Custom Image 3

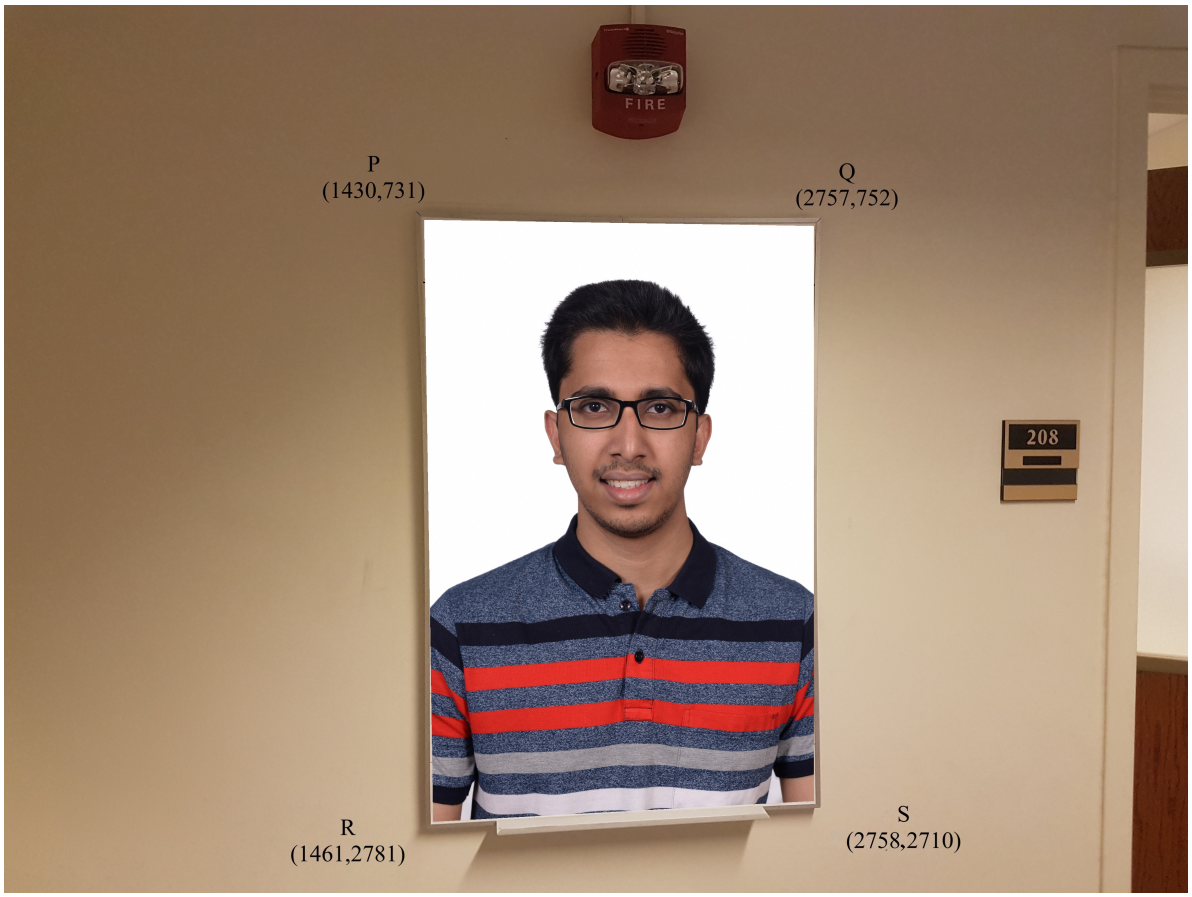


(d) Image of (me) Vineeth Ravi

Figure 4: Custom Images used for Task 2

These are the results for Task 2(a) :-

(a) Custom Output image1



(b) Custom Output image2

(c) Custom Output image3

Figure 5: Output images for Task 2(a)

This is the result for Task 2(b) :-

(a) Projection of Custom Image 1 on Custom Image 3

Figure 6: Output image for Task 2(b)

3 Python Code

The python code for Task 1(a) and 1(b) are shown below. Task 2 has the same code as Task 1, with different input images. So, it is not included again.

3.1 Task 1(a)

```
import numpy as np
import cv2
import math

def HomographyMatrix(src_img_pts,world_plane_pts):

    tb=[]
    ta=[]

    for i in range(0,len(src_img_pts)):

        tmp=[world_plane_pts[i][0],world_plane_pts[i][1],1,0,0,0,
        -src_img_pts[i][0]*world_plane_pts[i][0],-src_img_pts[i][0]*world_plane_pts[i][1]]
        ta.append(tmp)
        tmp=[0,0,0,world_plane_pts[i][0],world_plane_pts[i][1],1,
        -src_img_pts[i][1]*world_plane_pts[i][0],-src_img_pts[i][1]*world_plane_pts[i][1]]
        ta.append(tmp)
        A=np.asarray(ta)

        tmp=[src_img_pts[i][0],src_img_pts[i][1]]
        tb.append(tmp)
        tmp=np.asarray(tb)
        b=tmp.reshape(-1)


    tmp=np.dot(np.linalg.pinv(A),b.transpose())
    H=np.zeros((3,3))
    H[0]=tmp[0:3]
    H[1]=tmp[3:6]
    H[2][0:2]=tmp[6:8]
    H[2][2]=1

    return H

def WeightedAverageRGBPixelValue(pt, img):

    x1=int(math.floor(pt[0]))
    x2=int(math.ceil(pt[0]))
    y1=int(math.floor(pt[1]))
    y2=int(math.ceil(pt[1]))

    Wp=1/np.linalg.norm(np.array([pt[0]-x1,pt[1]-y1]))
    Wq=1/np.linalg.norm(np.array([pt[0]-x1,pt[1]-y2]))
    Wr=1/np.linalg.norm(np.array([pt[0]-x2,pt[1]-y1]))
    Ws=1/np.linalg.norm(np.array([pt[0]-x2,pt[1]-y2]))

    pixel_value = (Wp*img[y1][x1] + Wq*img[y2][x1] + Wr*img[y1][x2] + Ws*img[y2][x2])
    /(Wp+Wq+Wr+Ws)

    pixel_value
    return pixel_value

def ProjectionImage(H,src_img,world_plane_img):

    for i in range(0,np.shape(world_plane_img)[0]-1):
        for j in range(0,np.shape(world_plane_img)[1]-1):
```

```
            tmp=np.array([j,i,1])
            xp=np.array(np.dot(H,tmp))
            xp=xp/xp[2]
            if((xp[0]>0)and(xp[0]<src_img.shape[1]-1)and(xp[1]>0)
            and(xp[1]<src_img.shape[0]-1)):
                world_plane_img[i][j]=WeightedAverageRGBPixelValue(xp,src_img)



    output_img = world_plane_img
    return output_img

# Main Function Begins

image1=cv2.imread("1.jpg")
image2=cv2.imread("2.jpg")
image3=cv2.imread("3.jpg")
imageJ=cv2.imread("Jackie.jpg")

t1=[]
t2=[]
t3=[]
tJ=[]

tJ.append([0.0,0.0,1.0])
tJ.append([1279.0,0.0,1.0])
tJ.append([0.0,719.0,1.0])
tJ.append([1279.0,719.0,1.0])

pts_imageJ=np.asarray(tJ)

t1.append([1509.0,157.0,1.0])
t1.append([2971.0,727.0,1.0])
t1.append([1481.0,2257.0,1.0])
t1.append([3017.0,2061.0,1.0])

pts_image1=np.asarray(t1)

t2.append([1309.0,317.0,1.0])
t2.append([3021.0,593.0,1.0])
t2.append([1295.0,2027.0,1.0])
t2.append([3033.0,1899.0,1.0])

pts_image2=np.asarray(t2)

t3.append([913.0,728.0,1.0])
t3.append([2802.0,377.0,1.0])
t3.append([889.0,2093.0,1.0])
t3.append([2854.0,2238.0,1.0])

pts_image3=np.asarray(t3)


H=HomographyMatrix(pts_imageJ,pts_image1)
output=ProjectionImage(H,imageJ,image1)
cv2.imwrite('JackieOutput_image1.jpg',output)

H=HomographyMatrix(pts_imageJ,pts_image2)
output=ProjectionImage(H,imageJ,image2)
cv2.imwrite('JackieOutput_image2.jpg',output)

H=HomographyMatrix(pts_imageJ,pts_image3)
output=ProjectionImage(H,imageJ,image3)
cv2.imwrite('JackieOutput_image3.jpg',output)
```

3.2 Task 1(b)

```
import numpy as np
import cv2
import math

def HomographyMatrix(src_img_pts,world_plane_pts):

    tb=[]
    ta=[]

    for i in range(0,len(src_img_pts)):

        tmp=[world_plane_pts[i][0],world_plane_pts[i][1],1,0,0,0,
        -src_img_pts[i][0]*world_plane_pts[i][0],-src_img_pts[i][0]*world_plane_pts[i][1]]
        ta.append(tmp)
        tmp=[0,0,0,world_plane_pts[i][0],world_plane_pts[i][1],1,
        -src_img_pts[i][1]*world_plane_pts[i][0],-src_img_pts[i][1]*world_plane_pts[i][1]]
        ta.append(tmp)
        A=np.asarray(ta)

        tmp=[src_img_pts[i][0],src_img_pts[i][1]]
        tb.append(tmp)
        tmp=np.asarray(tb)
        b=tmp.reshape(-1)


    tmp=np.dot(np.linalg.pinv(A),b.transpose())
    H=np.zeros((3,3))
    H[0]=tmp[0:3]
    H[1]=tmp[3:6]
    H[2][0:2]=tmp[6:8]
    H[2][2]=1

    return H

def WeightedAverageRGBPixelValue(pt, img):

    x1=int(math.floor(pt[0]))
    x2=int(math.ceil(pt[0]))
    y1=int(math.floor(pt[1]))
    y2=int(math.ceil(pt[1]))

    Wp=1/np.linalg.norm(np.array([pt[0]-x1,pt[1]-y1]))
    Wq=1/np.linalg.norm(np.array([pt[0]-x1,pt[1]-y2]))
    Wr=1/np.linalg.norm(np.array([pt[0]-x2,pt[1]-y1]))
    Ws=1/np.linalg.norm(np.array([pt[0]-x2,pt[1]-y2]))

    pixel_value = (Wp*img[y1][x1] + Wq*img[y2][x1] + Wr*img[y1][x2] + Ws*img[y2][x2])
    /(Wp+Wq+Wr+Ws)

    pixel_value
    return pixel_value

def ProjectionImage(H,src_img,world_plane_img):

    for i in range(0,np.shape(world_plane_img)[0]-1):
        for j in range(0,np.shape(world_plane_img)[1]-1):
            tmp=np.array([j,i,1])
            xp=np.array(np.dot(H,tmp))
            xp=xp/xp[2]
            if((xp[0]>0)and(xp[0]<src_img.shape[1]-1)and(xp[1]>0)
            and(xp[1]<src_img.shape[0]-1)):
                world_plane_img[i][j]=WeightedAverageRGBPixelValue(xp,src_img)
```

```
            else:
                world_plane_img[i][j]=[0,0,0]



    output_img = world_plane_img
    return output_img

# Main Function Begins

image1=cv2.imread("1.jpg")
image2=cv2.imread("2.jpg")
image3=cv2.imread("3.jpg")

t1=[]
t2=[]
t3=[]

t1.append([1509.0,157.0,1.0])
t1.append([2971.0,727.0,1.0])
t1.append([1481.0,2257.0,1.0])
t1.append([3017.0,2061.0,1.0])

pts_image1=np.asarray(t1)

t2.append([1309.0,317.0,1.0])
t2.append([3021.0,593.0,1.0])
t2.append([1295.0,2027.0,1.0])
t2.append([3033.0,1899.0,1.0])

pts_image2=np.asarray(t2)

t3.append([913.0,728.0,1.0])
t3.append([2802.0,377.0,1.0])
t3.append([889.0,2093.0,1.0])
t3.append([2854.0,2238.0,1.0])

pts_image3=np.asarray(t3)

H1=HomographyMatrix(pts_image1,pts_image2)
H2=HomographyMatrix(pts_image2,pts_image3)
H=np.matmul(H1,H2)

output=ProjectionImage(H,image1,image3)
cv2.imwrite('Projection1on3.jpg',output)
```

THE END