ECE661: Computer Vision

Risi Jaiswal
rjaiswa@purdue.edu

September 10, 2020

1-A

As a part of task 1 there are 3 images of wall painting from different angles and we have to find homography between these images with given kitten image. Let pixels in painting images are given by $X_2 = (x, y)$ and $X_2' = (x', y')$ in kitten image. Let's assume homography is given by following matrix denoted by $\mathbf{H}$

$$\mathbf{H} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}$$

Since We are using homogeneous coordinate system and information lies in ratio so we can get away with one element of matrix. Without loss of generality $i = 1$. Since H is homography which will map points from X_3 to X_3' (where X_3 and X_3' are homogeneous representations of X_2 and X_2'), we can write

$$\implies \mathbf{H}X_3 = X_3' \tag{1}$$

$$\implies \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} x'' \\ y'' \\ z'' \end{bmatrix} \tag{2}$$

$$\implies x'' = ax + by + c, y'' = dx + ey + f \text{ and } gx + hy + 1 = z'' \tag{3}$$

$$\implies x' = \frac{x''}{z''} \text{ and } y' = \frac{y''}{z''} \tag{4}$$

$$\implies x' = \frac{ax + by + c}{gx + hy + 1} \text{ and } y' = \frac{dx + ey + f}{gx + hy + 1} \tag{5}$$

$$\implies x' = ax + by + c - gxx' - hx'y \text{ and } y' = dx + ey + f - gxy' - hyy' \tag{6}$$

As we can see to determine $\mathbf{H}$ completely we need at least 8 equations so we need to take 4 points. All these 8 equations can be written as a system of linear

equations in matrix as shown below

$$\begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \\ g \\ h \end{bmatrix} = \begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4 \\ y_4' \end{bmatrix}$$

By Solving above system of linear equations we get homography matrix. We apply these homography matrix to painting pixels and if resultant pixel is within ROI of kitten image we can replace pixels of painting with kitten. Since solution of system of linear equations involves small errors we get fractional pixel coordinates. I tried with just taking ceil of pixel coordinates and weighted average of 4 nearest neighbour where weights are characterized by $\mathbf{L^P}-$ norm as shown below.

$$p(x,y) = \frac{w_1 p(\lfloor x \rfloor, \lfloor y \rfloor) + w_2 p(\lfloor x \rfloor, \lceil y \rceil) + w_3 p(\lceil x \rceil, \lceil y \rceil) + w_4 p(\lceil x \rceil, \lfloor y \rfloor)}{w_1 + w_2 + w_3 + w4}$$

where w_i is given by (x' is one of the 4 nearest neighbour of x) :

$$w_i = \{(x - x')^p + (y - y')^p\}^{\frac{1}{p}}$$

Although I didn't see any noticeable difference between 3 techniques- just taking ceil , weighted average by $\mathbf{L^2}-$ norm and $\mathbf{L^1}-$ norm. It seems like it is because images are two large and 1-2 pixel values are not going to produce any noticeable difference. Results are attached below.

Figure 1: Projection of kitten on painting 1

Figure 2: Projection of kitten on painting 2

Figure 3: Projection of kitten on painting 3

1-B

In this part first I figured out homopgraphy between a and b & b and c as described in the previous section. When these two homography matrices got multiplied then due to linear transformation product matrix should be homography between a and c. Which got into result also when product homography was applied on A pixel coordinates, I got the image c kind of painting.

Figure 4: After applying homography H_{AC}

2-A

Task 2 was to do same procedure with our own images. Results are as below.

Figure 5: Projection of personality on Board 1

Figure 6: Projection of personality on Board 2

Figure 7: Projection of personality on Board 3

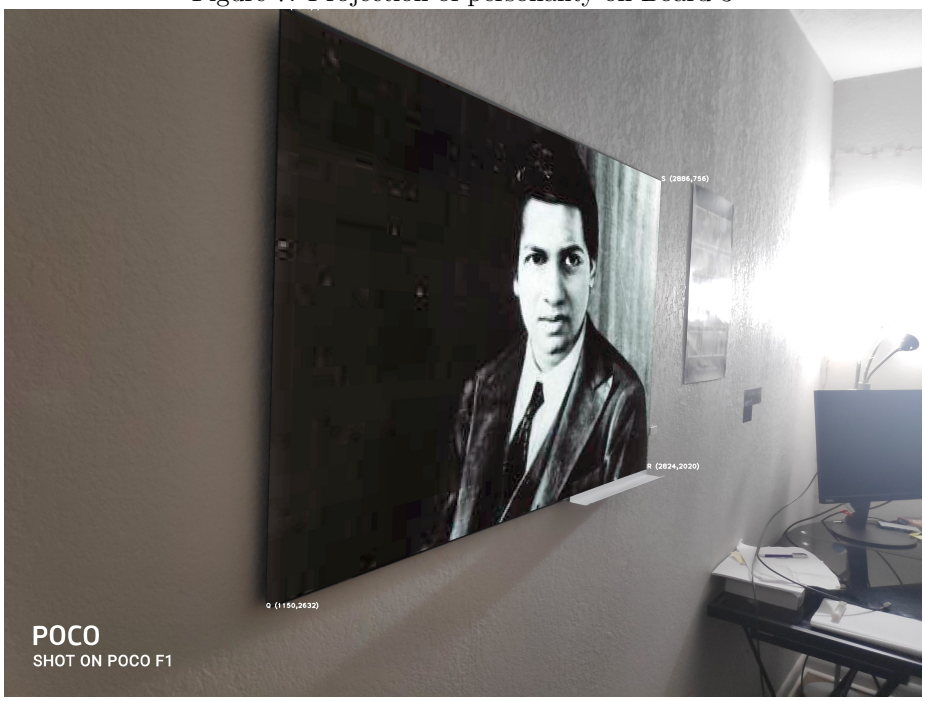

Figure 8: After applying homography H_{AC}

Source Code

Task1

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:55:44 2020

@author: risij
"""

import cv2
import numpy as np
import os
from math import ceil,floor
import pdb
np.set_printoptions(suppress=True)


class Homography:
    def __init__(self,frame_image,projected_image,F_points=None,P_points=None):
        '''
         Loading Images
        '''
        self.img_proj = cv2.imread(os.path.join(path,projected_image))
        self.img_frame = cv2.imread(os.path.join(path,frame_image))
        self.Width=self.img_proj.shape[1]
        self.Height=self.img_proj.shape[0]

        '''
        If ROI is not provided in constructor then using full image as ROI
        '''
        if(type(P_points)==np.ndarray):
            self.P_points=P_points
        else:
```

```
        self.P_points = np.array([[0,0],[0,self.Height],[self.Width,self.Height],[self.Width,0]])

    if(type(F_points)==np.ndarray):
        self.F_points=F_points
    else:
        self.F_points = np.array([[0,0],[0,self.img_proj.shape[0]],[self.img_proj.shape[1],self.img_proj.shape[0]],[self.img_proj.shape[1],0]])


    self.projected_image=projected_image
    self.frame_image=frame_image


def generate_homography(self):
    '''
    Creating Homography matrices for all 4 points P,Q,R,S and X_prime vector
    '''
    self.H = np.zeros((8,8))
    self.X_prime = np.zeros((8,1))

    for i in range(len(self.P_points)):
        self.H[2*i]   = np.array([self.F_points[i][0],self.F_points[i][1],1,0,0,0,-self.F_points[i][0]*self.P_points[i][0],-self.F_points[i][1]*self.P_points[i][0]])
        self.H[2*i+1] =np.array([0,0,0,self.F_points[i][0],self.F_points[i][1],1,-self.F_points[i][0]*self.P_points[i][1],-self.F_points[i][1]*self.P_points[i][1]])
        self.X_prime[2*i]=self.P_points[i][0]
        self.X_prime[2*i+1]=self.P_points[i][1]

    '''
    Solving system of linear equations
    '''
    self.h = np.linalg.solve(self.H,self.X_prime)


    self.h = np.reshape(np.vstack((self.h,1)),(3,3))


def is_inside(self,x,y):
    if((x>0 and x<self.Width) and (y>0 and y<self.Height)):
```

```
        return True
    return False



def project_image(self,ext_h=None):
    '''
    iterating over frame and checking if coordinate is inside ROI of projected image
    then copying value from projected image other wise remaining frame pixel as it is
    '''
    for i in range(self.img_frame.shape[1]):
        for k in range(self.img_frame.shape[0]):
            coordinate_r3 = np.matmul(self.h,np.array([i,k,1]))
            coordinate_r2 = np.array([coordinate_r3[0],coordinate_r3[1]])/coordinate_r3[2]



            if(self.is_inside(coordinate_r2[0],coordinate_r2[1])):
                '''
                Taking weighted average of nearest neighbour
                '''
                try:
                    N1 = self.img_proj[floor(coordinate_r2[1]),floor(coordinate_r2[0])]
                    N2 = self.img_proj[floor(coordinate_r2[1]),ceil(coordinate_r2[0])]
                    N3 = self.img_proj[ceil(coordinate_r2[1]),floor(coordinate_r2[0])]
                    N4 = self.img_proj[ceil(coordinate_r2[1]),ceil(coordinate_r2[0])]
                    d1 = np.linalg.norm(np.array([floor(coordinate_r2[0]),floor(coordinate_r2[1])])-coordinate_r2,1)
                    d2 = np.linalg.norm(np.array([floor(coordinate_r2[0]),ceil(coordinate_r2[1])])-coordinate_r2,1)
                    d3 = np.linalg.norm(np.array([ceil(coordinate_r2[0]),floor(coordinate_r2[1])])-coordinate_r2,1)
                    d4 = np.linalg.norm(np.array([ceil(coordinate_r2[0]),ceil(coordinate_r2[1])])-coordinate_r2,1)
                    self.img_frame[k,i] = ((1/d1)*N1+(1/d2)*N2+(1/d3)*N3+(1/d4)*N4)/((1/d1)+(1/d2)+(1/d3)+(1/d4))
                    #self.img_frame[k,i] = self.img_proj[int(coordinate_r2[1]),int(coordinate_r2[0])]
                except IndexError:
                    #out of dimension mapping
                    pass
    '''
```

```
        Plotting points on frame image which were used in calculating homography
        '''
        cv2.putText(self.img_frame,'P ({0},{1})'.format(self.F_points[0][0],self.F_points[0][1]),(self.F_points[0][0],self.F_points[0][1]),1,2,(255,255,255),4)
        cv2.putText(self.img_frame,'Q ({0},{1})'.format(self.F_points[1][0],self.F_points[1][1]),(self.F_points[1][0],self.F_points[1][1]),1,2,(255,255,255),4)
        cv2.putText(self.img_frame,'R ({0},{1})'.format(self.F_points[2][0],self.F_points[2][1]),(self.F_points[2][0],self.F_points[2][1]),1,2,(255,255,255),4)
        cv2.putText(self.img_frame,'S ({0},{1})'.format(self.F_points[3][0],self.F_points[3][1]),(self.F_points[3][0],self.F_points[3][1]),1,2,(255,255,255),4)
        cv2.imwrite(self.projected_image.split(".")[0]+"_on_"+self.frame_image.split(".")[0]+"_result.jpeg",self.img_frame)



########################## Task 1 (i) #######################################
path = "hw2_Task1_Images"

'''
4 points of frame A
'''
F_points_A = np.array([[301,509],[240,1601],[1684,1822],[1772,351]])
homography = Homography("painting1.jpeg","kittens.jpeg",F_points=F_points_A)
homography.generate_homography()
homography.project_image()

'''
4 points of frame B
'''
F_points_B = np.array([[338,694],[332,2328],[1884,2004],[1888,750]])
homography = Homography("painting2.jpeg","kittens.jpeg",F_points=F_points_B)
homography.generate_homography()
homography.project_image()

'''
4 points of frame C
'''
F_points_C = np.array([[108,438],[120,1380],[1095,1848],[1215,303]])
homography = Homography("painting3.jpeg","kittens.jpeg",F_points=F_points_C)
homography.generate_homography()
```

```
homography.project_image()


'''
Plotting points over frames before projecting
'''

img = cv2.imread(os.path.join(path,"painting1.jpeg"))
cv2.putText(img,'P ({0},{1})'.format(F_points_A[0][0],F_points_A[0][1]),(F_points_A[0][0],F_points_A[0][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'Q ({0},{1})'.format(F_points_A[1][0],F_points_A[1][1]),(F_points_A[1][0],F_points_A[1][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'R ({0},{1})'.format(F_points_A[2][0],F_points_A[2][1]),(F_points_A[2][0],F_points_A[2][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'S ({0},{1})'.format(F_points_A[3][0],F_points_A[3][1]),(F_points_A[3][0],F_points_A[3][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.imwrite("painting1.jpeg",img)

img = cv2.imread(os.path.join(path,"painting2.jpeg"))
cv2.putText(img,'P ({0},{1})'.format(F_points_B[0][0],F_points_B[0][1]),(F_points_B[0][0],F_points_B[0][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'Q ({0},{1})'.format(F_points_B[1][0],F_points_B[1][1]),(F_points_B[1][0],F_points_B[1][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'R ({0},{1})'.format(F_points_B[2][0],F_points_B[2][1]),(F_points_B[2][0],F_points_B[2][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'S ({0},{1})'.format(F_points_B[3][0],F_points_B[3][1]),(F_points_B[3][0],F_points_B[3][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.imwrite("painting2.jpeg",img)

img = cv2.imread(os.path.join(path,"painting3.jpeg"))
cv2.putText(img,'P ({0},{1})'.format(F_points_C[0][0],F_points_C[0][1]),(F_points_C[0][0],F_points_C[0][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'Q ({0},{1})'.format(F_points_C[1][0],F_points_C[1][1]),(F_points_C[1][0],F_points_C[1][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'R ({0},{1})'.format(F_points_C[2][0],F_points_C[2][1]),(F_points_C[2][0],F_points_C[2][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(img,'S ({0},{1})'.format(F_points_C[3][0],F_points_C[3][1]),(F_points_C[3][0],F_points_C[3][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.imwrite("painting3.jpeg",img)

##########################Task 1 (ii) ########################################
homography_AB = Homography("painting2.jpeg","painting1.jpeg",F_points=F_points_B,P_points=F_points_A)
homography_AB.generate_homography()

homography_BC = Homography("painting3.jpeg","painting2.jpeg",F_points=F_points_C,P_points=F_points_B)
homography_BC.generate_homography()

'''
```

```
'''
Calculating A to C Homography by product of HAB and HBC
'''
homography_AC = np.matmul(homography_AB.h,homography_BC.h)


'''
Creating empty np array of size A
'''
FRAME = np.zeros((homography_AB.img_proj.shape[0],homography_AB.img_proj.shape[1],3))


'''
iterating over Image A then copying value from tranformed pixel locations into FRAME
'''
for i in range(homography_AB.img_proj.shape[1]):
    for k in range(homography_AB.img_proj.shape[0]):
        coordinate_r3 = np.matmul(homography_AC,np.array([i,k,1]))
        coordinate_r2 = np.array([coordinate_r3[0],coordinate_r3[1]])/coordinate_r3[2]
        try:
            FRAME[k,i] = homography_AB.img_proj[int(coordinate_r2[1]),int(coordinate_r2[0])]
        except IndexError:
            #out of dimension mapping
            pass

cv2.putText(FRAME,'P ({0},{1})'.format(F_points_A[0][0],F_points_A[0][1]),(F_points_A[0][0],F_points_A[0][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(FRAME,'Q ({0},{1})'.format(F_points_A[1][0],F_points_A[1][1]),(F_points_A[1][0],F_points_A[1][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(FRAME,'R ({0},{1})'.format(F_points_A[2][0],F_points_A[2][1]),(F_points_A[2][0],F_points_A[2][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(FRAME,'S ({0},{1})'.format(F_points_A[3][0],F_points_A[3][1]),(F_points_A[3][0],F_points_A[3][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)

cv2.imwrite("combined_homography_on_A.jpeg",FRAME)
```

Task2

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep  9 15:55:44 2020

@author: risij
"""

import cv2
import numpy as np
import os
from math import ceil,floor
import pdb
np.set_printoptions(suppress=True)


class Homography:
    def __init__(self,frame_image,projected_image,F_points=None,P_points=None):
        '''
         Loading Images
        '''
        self.img_proj = cv2.imread(os.path.join(path,projected_image))
        self.img_frame = cv2.imread(os.path.join(path,frame_image))
        self.Width=self.img_proj.shape[1]
        self.Height=self.img_proj.shape[0]

        '''
        If ROI is not provided in constructor then using full image as ROI
        '''
        if(type(P_points)==np.ndarray):
            self.P_points=P_points
        else:
            self.P_points = np.array([[0,0],[0,self.Height],[self.Width,self.Height],[self.Width,0]])

        if(type(F_points)==np.ndarray):
```

```
        self.F_points=F_points
    else:
        self.F_points = np.array([[0,0],[0,self.img_proj.shape[0]],[self.img_proj.shape[1],self.img_proj.shape[0]],[self.img_proj.shape[1],0]])


    self.projected_image=projected_image
    self.frame_image=frame_image


def generate_homography(self):
    '''
    Creating Homography matrices for all 4 points P,Q,R,S and X_prime vector
    '''
    self.H = np.zeros((8,8))
    self.X_prime = np.zeros((8,1))

    for i in range(len(self.P_points)):
        self.H[2*i]   = np.array([self.F_points[i][0],self.F_points[i][1],1,0,0,0,-self.F_points[i][0]*self.P_points[i][0],-self.F_points[i][1]*self.P_points[i][0]])
        self.H[2*i+1] =np.array([0,0,0,self.F_points[i][0],self.F_points[i][1],1,-self.F_points[i][0]*self.P_points[i][1],-self.F_points[i][1]*self.P_points[i][1]])
        self.X_prime[2*i]=self.P_points[i][0]
        self.X_prime[2*i+1]=self.P_points[i][1]

    '''
    Solving system of linear equations
    '''
    self.h = np.linalg.solve(self.H,self.X_prime)


    self.h = np.reshape(np.vstack((self.h,1)),(3,3))


def is_inside(self,x,y):
    if((x>0 and x<self.Width) and (y>0 and y<self.Height)):
        return True
    return False
```

```
def project_image(self,ext_h=None):
    '''
    iterating over frame and checking if coordinate is inside ROI of projected image
    then copying value from projected image other wise remaining frame pixel as it is
    '''
    for i in range(self.img_frame.shape[1]):
        for k in range(self.img_frame.shape[0]):
            coordinate_r3 = np.matmul(self.h,np.array([i,k,1]))
            coordinate_r2 = np.array([coordinate_r3[0],coordinate_r3[1]])/coordinate_r3[2]



            if(self.is_inside(coordinate_r2[0],coordinate_r2[1])):
                '''
                Taking weighted average of nearest neighbour
                '''
                try:
                    N1 = self.img_proj[floor(coordinate_r2[1]),floor(coordinate_r2[0])]
                    N2 = self.img_proj[floor(coordinate_r2[1]),ceil(coordinate_r2[0])]
                    N3 = self.img_proj[ceil(coordinate_r2[1]),floor(coordinate_r2[0])]
                    N4 = self.img_proj[ceil(coordinate_r2[1]),ceil(coordinate_r2[0])]
                    d1 = np.linalg.norm(np.array([floor(coordinate_r2[0]),floor(coordinate_r2[1])])-coordinate_r2,1)
                    d2 = np.linalg.norm(np.array([floor(coordinate_r2[0]),ceil(coordinate_r2[1])])-coordinate_r2,1)
                    d3 = np.linalg.norm(np.array([ceil(coordinate_r2[0]),floor(coordinate_r2[1])])-coordinate_r2,1)
                    d4 = np.linalg.norm(np.array([ceil(coordinate_r2[0]),ceil(coordinate_r2[1])])-coordinate_r2,1)
                    self.img_frame[k,i] = ((1/d1)*N1+(1/d2)*N2+(1/d3)*N3+(1/d4)*N4)/((1/d1)+(1/d2)+(1/d3)+(1/d4))
                    #self.img_frame[k,i] = self.img_proj[int(coordinate_r2[1]),int(coordinate_r2[0])]
                except IndexError:
                    #out of dimension mapping
                    pass
    '''
    Plotting points on frame image which were used in calculating homography
    '''
    cv2.putText(self.img_frame,'P ({0},{1})'.format(self.F_points[0][0],self.F_points[0][1]),(self.F_points[0][0],self.F_points[0][1]),1,2,(255,255,255),4)
```

```
        cv2.putText(self.img_frame,'Q ({0},{1})'.format(self.F_points[1][0],self.F_points[1][1]),(self.F_points[1][0],self.F_points[1][1]),1,2,(255,255,255),4)
        cv2.putText(self.img_frame,'R ({0},{1})'.format(self.F_points[2][0],self.F_points[2][1]),(self.F_points[2][0],self.F_points[2][1]),1,2,(255,255,255),4)
        cv2.putText(self.img_frame,'S ({0},{1})'.format(self.F_points[3][0],self.F_points[3][1]),(self.F_points[3][0],self.F_points[3][1]),1,2,(255,255,255),4)
        cv2.imwrite(self.projected_image.split(".")[0]+"_on_"+self.frame_image.split(".")[0]+"_result.jpeg",self.img_frame)



##########################Task 2 (ii) ########################################
homography_AB = Homography("board2.jpg","board1.jpg",F_points=F_points_B,P_points=F_points_A)
homography_AB.generate_homography()

homography_BC = Homography("board3.jpg","board2.jpg",F_points=F_points_C,P_points=F_points_B)
homography_BC.generate_homography()

homography_AC = np.matmul(homography_AB.h,homography_BC.h)

FRAME = np.zeros((homography_AB.img_proj.shape[0],homography_AB.img_proj.shape[1],3))


'''
iterating over Image A then copying value from tranformed pixel locations into FRAME
'''
for i in range(homography_AB.img_proj.shape[1]):
    for k in range(homography_AB.img_proj.shape[0]):
        coordinate_r3 = np.matmul(homography_AC,np.array([i,k,1]))
        coordinate_r2 = np.array([coordinate_r3[0],coordinate_r3[1]])/coordinate_r3[2]
        try:
            FRAME[k,i] = homography_AB.img_proj[int(coordinate_r2[1]),int(coordinate_r2[0])]
        except IndexError:
            #out of dimension mapping
            pass


cv2.putText(FRAME,'P ({0},{1})'.format(F_points_A[0][0],F_points_A[0][1]),(F_points_A[0][0],F_points_A[0][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(FRAME,'Q ({0},{1})'.format(F_points_A[1][0],F_points_A[1][1]),(F_points_A[1][0],F_points_A[1][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
```

```
cv2.putText(FRAME,'R ({0},{1})'.format(F_points_A[2][0],F_points_A[2][1]),(F_points_A[2][0],F_points_A[2][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.putText(FRAME,'S ({0},{1})'.format(F_points_A[3][0],F_points_A[3][1]),(F_points_A[3][0],F_points_A[3][1]),cv2.FONT_HERSHEY_PLAIN,2,(255,255,255),thickness=4)
cv2.imwrite("board_combined_homography_on_A.jpg",FRAME)
```