COMPUTER VISION
ECE 661
HOMEWORK 2

Tanzeel U. Rehman

Email: *trehman@purdue.edu*

1. Solution:

Fundamentals and steps:

The homework requires estimation of the hymnographies between pairs of images and then use these hymnographies to warp the given and our own images. The fundamental of these two steps are given below.

1.1. Homography:

Given a point X in a planar scene and its corresponding pixel location X' in the image can be given by Eq. 1.

$$X' = HX \qquad \text{(Eq.1)}$$

Where, X and X' are homogeneous coordinates of the form $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$ and $\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix}$ for physical points (x, y), respectively, and (x', y') and H is non-singular homography matrix in homogeneous coordinates which can be represented by Eq. 2. Using Eq. 2 and Eq. 1, Eq. 2 can be re-written as Eq. 3, which can further be expended as Eq. 4. The physical coordinate of each pixel ($x' = x_1'/x_3'$, $y' = x_2'/x_3'$) can be obtained as in Eq. 5. Dividing both numerator and denominator by x_3 for both (x', y'), we get just the physical coordinates on both sides as given in Eq. 6. The Eq. 6 can be rearranged as Eq. 7.

$$H = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \qquad \text{(Eq.2)}$$

$$\begin{bmatrix} x_1' \\ x_2' \\ x_3' \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_1 \\ x_1 \end{bmatrix} \qquad \text{(Eq.3)}$$

$$x_1' = h_{11}x_1 + h_{12}x_2 + h_{13}x_3 \qquad \text{(Eq.4)}$$

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

$$x_3' = h_{31}x_1 + \ h_{32}x_2 + x_3$$

$$x' = \frac{x_1'}{x_3'} = \frac{h_{11}x_1 + \ h_{12}x_2 + h_{13}x_3}{h_{31}x_1 + \ h_{32}x_2 + x_3} \qquad \text{(Eq.5)}$$

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

$$x' = \frac{h_{11}x + \ h_{12}y + h_{13}}{h_{31}x_1 + \ h_{32}x_2 + 1} \qquad \text{(Eq.6)}$$

$$y' = \frac{x_2'}{x_3'} = \frac{h_{21}x + \ h_{22}y + h_{23}}{h_{31}x + \ h_{32}y + 1}$$

$$x' = h_{11}x + \ h_{12}y + h_{13} - \ h_{31}xx' - \ h_{32}x'y \qquad \text{(Eq.7)}$$

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

Now using the Eq. 7, we can compute the unknown parameters of homography matrix *H*. Since there are eight unknown parameters, therefore, we need atleast 8 equations to obtain the unique solution which can be obtained using atleast 4 point pair [(*x*, *y*), (*x*′, *y*′)] correspondences. Using these four point pair correspondences, we can write the Eq. 7 in a matrix format given as Eq. 8, which can simply be written as Eq. 8. The Eq. 8 can finally be solved by taking the inverse of matrix *A* (or through least squares in case of over-determined equation system) to compute 8 unknowns of homography matrix.

$$\begin{bmatrix} x_1' \\ y_1' \\ x_2' \\ y_2' \\ x_3' \\ y_3' \\ x_4' \\ y_4' \end{bmatrix} = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x_1x_1' & -y_1x_1' \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -x_1y_1' & -y_1y_1' \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x_2x_2' & -y_2x_2' \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_2' & -y_2y_2' \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x_3x_3' & -y_3x_3' \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -x_3y_3' & -y_3y_3' \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x_4x_4' & -y_4x_4' \\ 0 & 0 & 0 & x_4 & y_4 & 0 & -x_4y_4' & -y_4y_4' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{bmatrix} \quad \text{(Eq.8)}$$

$$Y = A\hat{X} \quad \text{(Eq.9)}$$

1.2. Computing RGB Values for each pixel location:

After computing the homography matrix, it can be applied to the homogeneous coordinates of point X to obtain the homogeneous coordinates of X'. These coordinates can be used to obtain the point coordinates in the physical space (x', y'), however, these values can be non-integral. For this homework, we rounded the non-integral values of (x', y') to the nearest integer. These rounded integers were then used for extracting the RGB values from the range image and were projected on the domain image.

2. Task 1:

2.1. Input Images:

Note: Definition of PSQR can be seen in the Fig 1b below. Outer corners of the painting were used for all the images in Task 1.1 and Task 1.2. The actual image coordinates for these points can be seen in the Table 1.

Table 1: X and Y coordinates of the PSRQ and P′S′R′Q′ points for the images provided with homework

Points	Figure 1a	Figure 1b	Figure 1c	Figure 1d

	(x_i, y_i)	**(x_i, y_i)**	(x_i, y_i)	**(x_i', y_i')**
P	(233, 412)	(166, 515)	(74, 356)	(0, 0)
S	(1925, 202)	(1966, 631)	(1366, 110)	(1920, 0)
R	(1820, 1988)	(1969, 2096)	(1204, 2064)	(1920, 1125)
Q	(139, 1693)	(172, 2525)	(67, 1422)	(0, 1125)

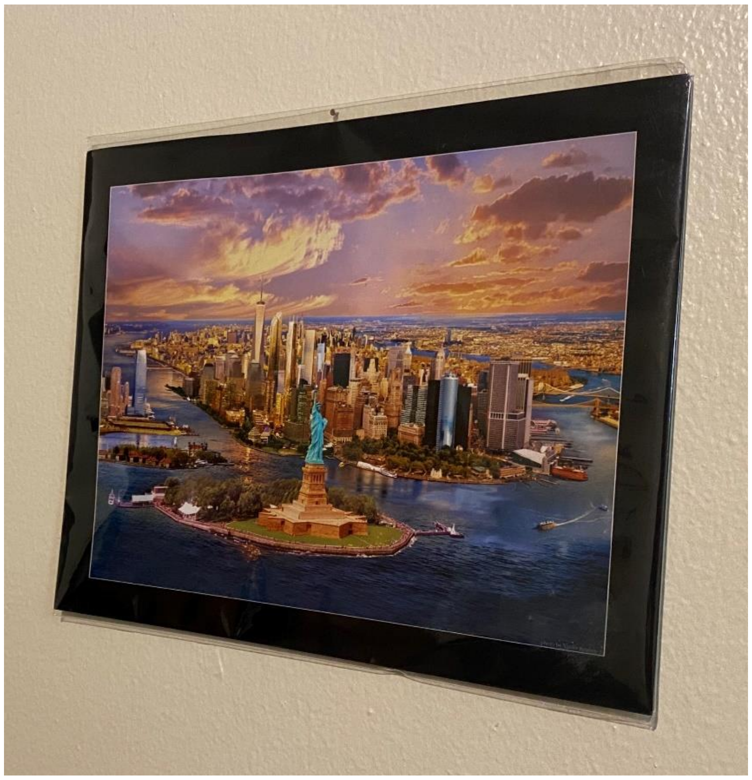

Fig 1a: Image of Painting1

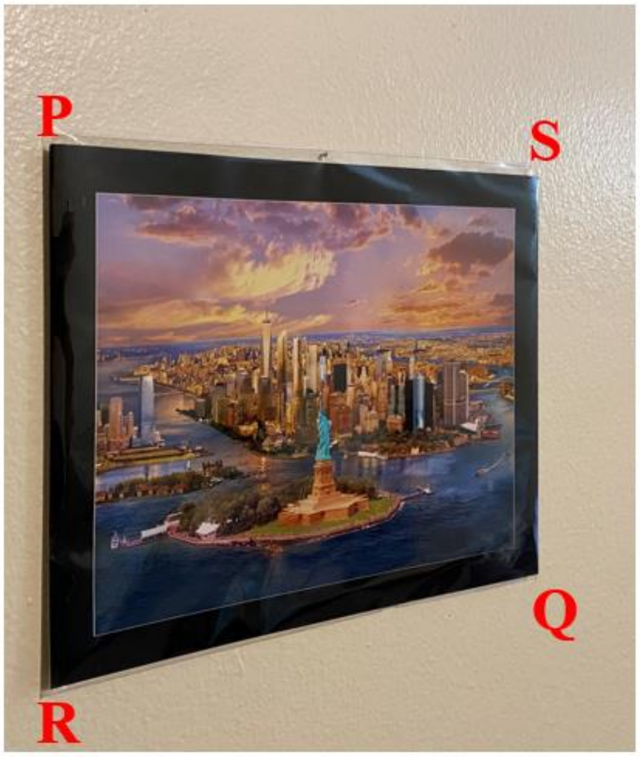

Fig 1b: Image of Painting2

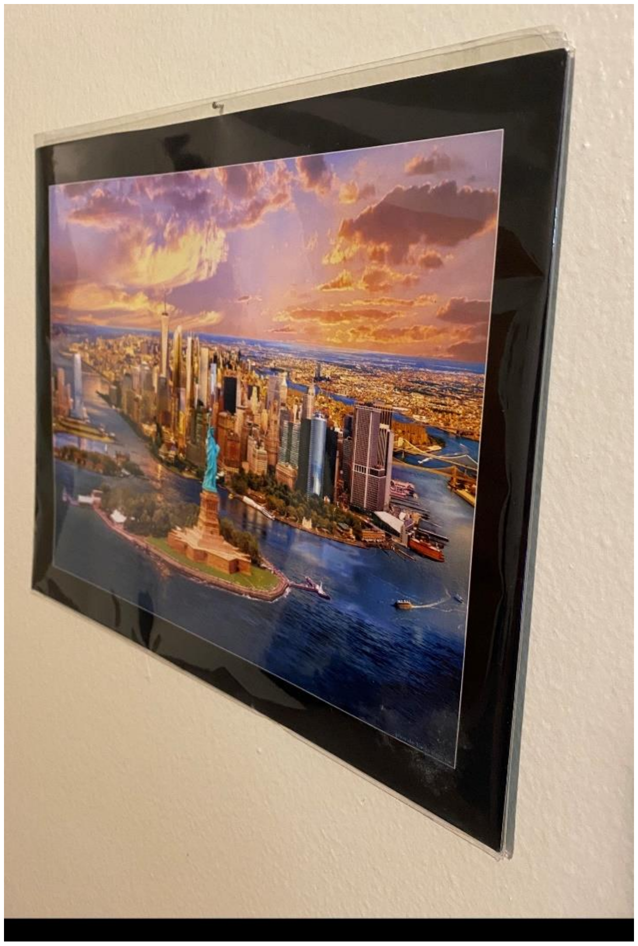

Fig 1c: Image of Painting3

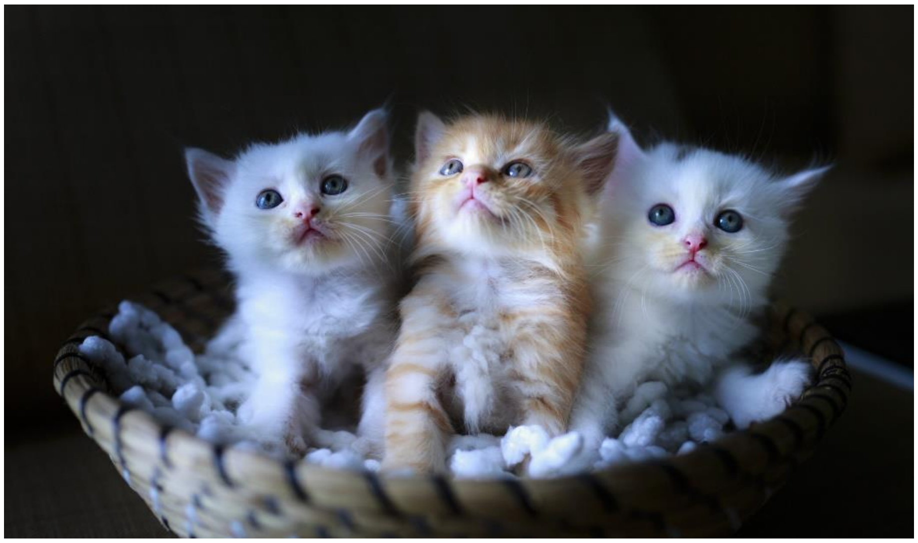

Fig 1d: Image of Kittens

2.2. Outputs of Task 1.1

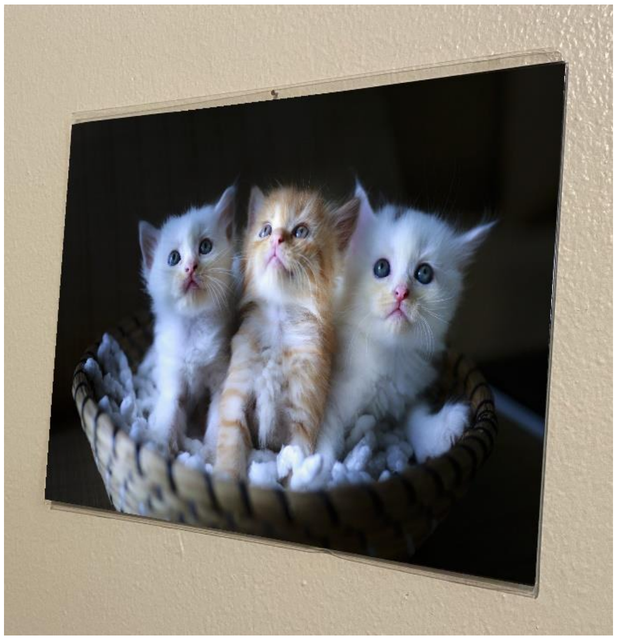

Fig 1: Results of Projecting Kittens on Figure 1a

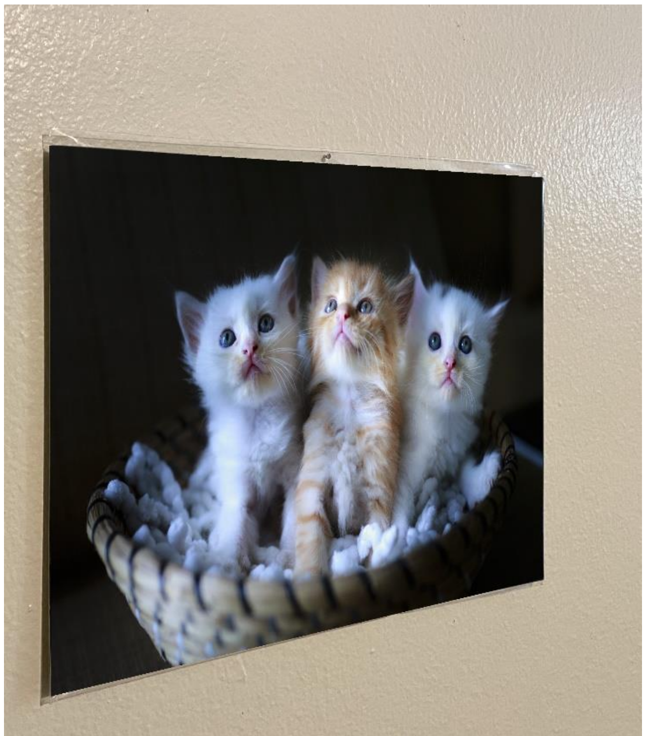

Fig 2: Results of Projecting Kittens on Figure 1b

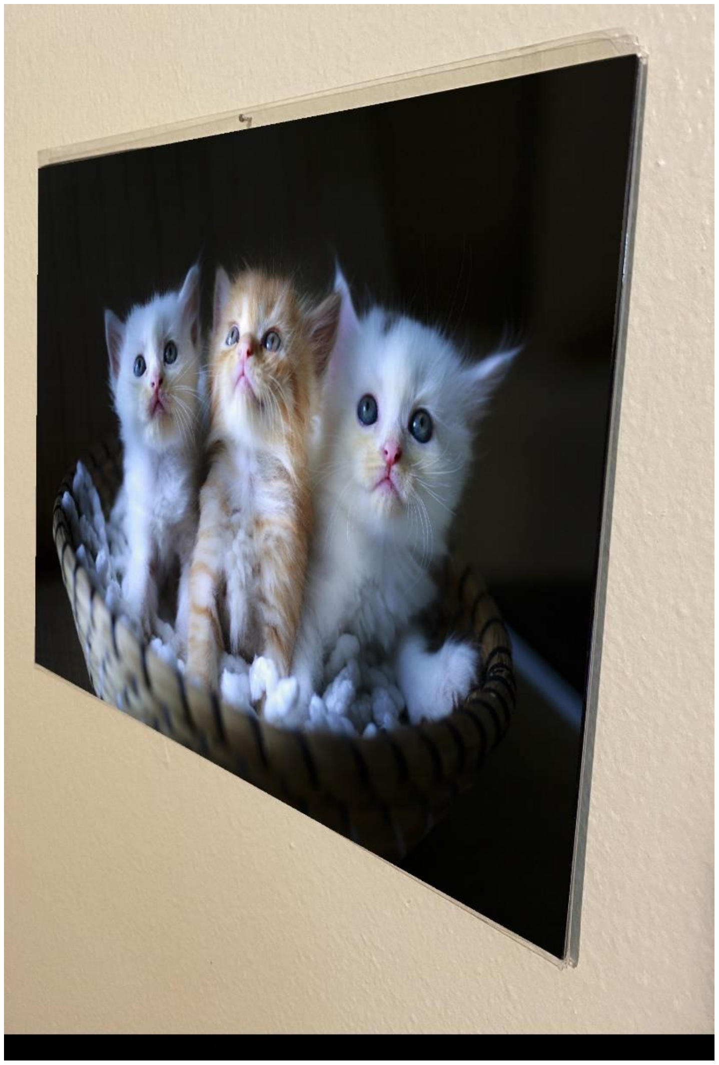

Fig 3: Results of Projecting Kittens on Figure 1c

2.3. Outputs of Task 1.2

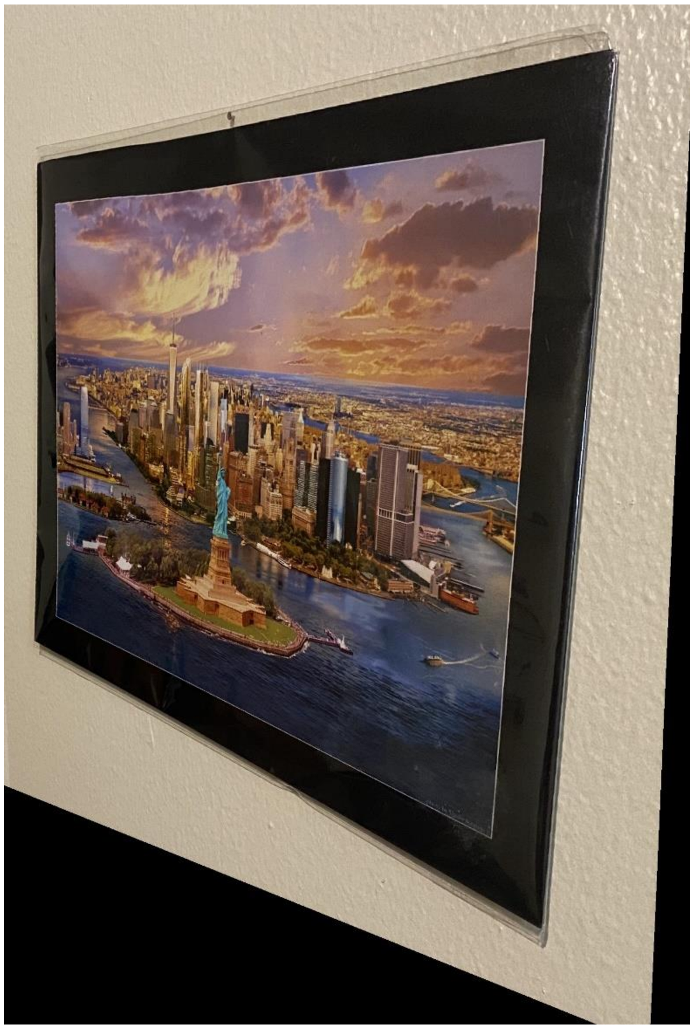

Fig 1: Results of Projecting Figure 1a on Figure 1c

3. Task 2:

3.1. Inputs of Task 2.1

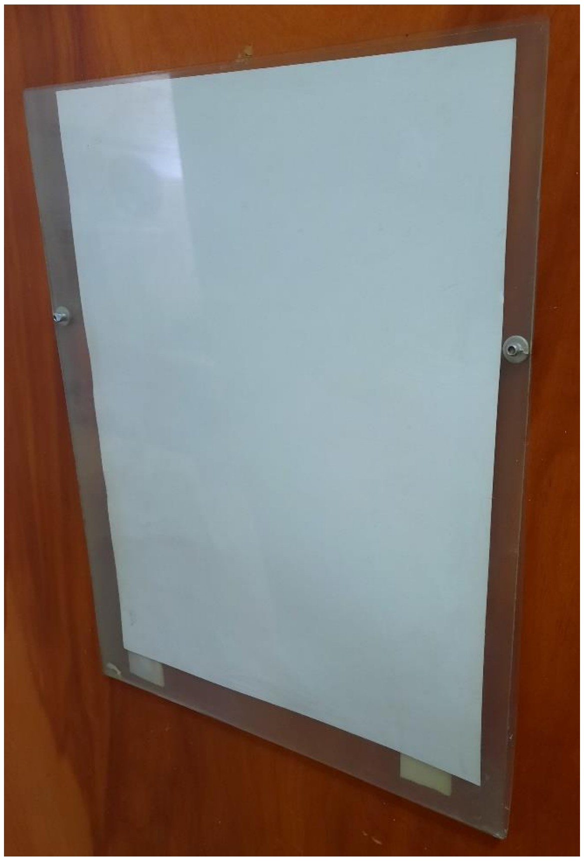
Fig 1a: Image of Firenotice1

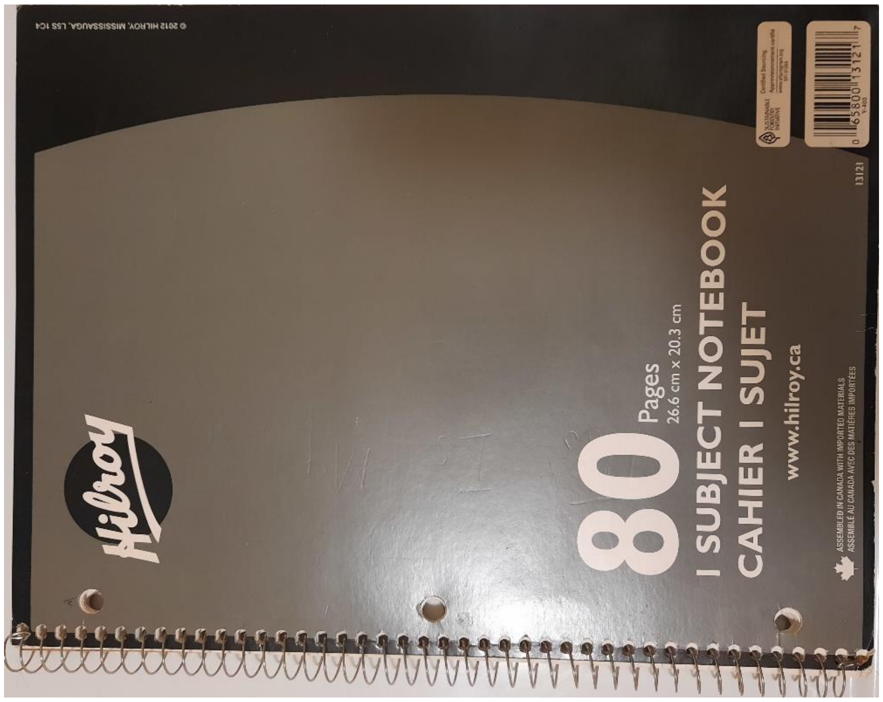

Fig 1d: Image of Notebook

Table 2: X and Y coordinates of the PSRQ and P′S′R′Q′ points for the images captured by myself

Points	Figure 1a	Figure 1b	Figure 1c	Figure 1d
	(x_i, y_i)	(x_i, y_i)	(x_i, y_i)	(x_i', y_i')
P	(396, 654)	(166, 515)	(74, 356)	(0, 0)
S	(3841, 250)	(1966, 631)	(1366, 110)	(6240, 0)
R	(3653, 5856)	(1969, 2096)	(1204, 2064)	(6240, 4944)
Q	(876, 4819)	(172, 2525)	(67, 1422)	(0, 4944)

3.2. Outputs of Task 2.2

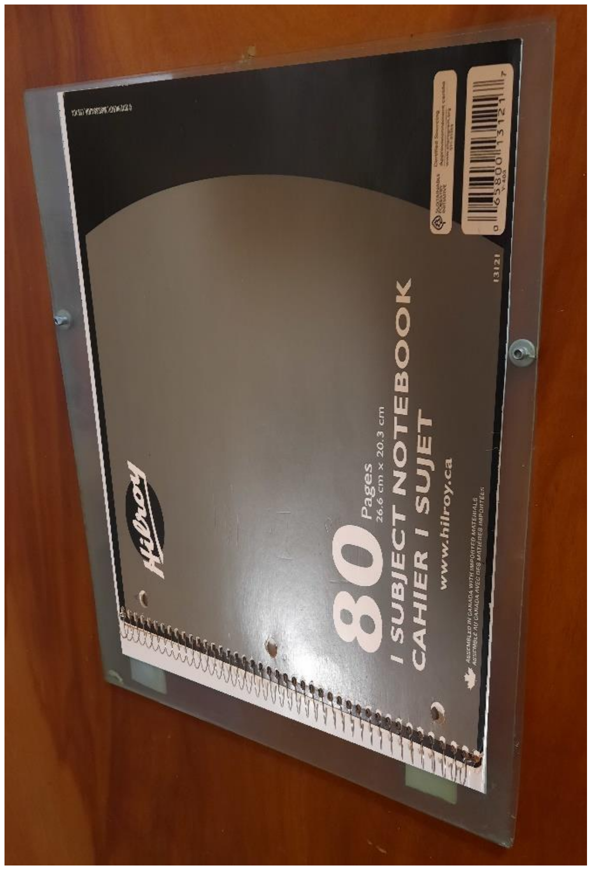

Fig 1: Results of Projecting Notebook on Figure 1a

4. Source Code:

4.1. Code for Task 1.1:

```
'''-------Main Code for Task-1.1---------'''
# Sequentially go over through 3 painting images and project the kittens

#Function for calling the Homography class and saving results
def Save(Domain_img,Range_image,Domain_pts,Range_pts,savefileint):
    # Initialize and call to the class responsible for computing and applying Homography
    homo=Homography(Range_image)
    H=homo.find_homography(Domain_pts,Range_pts)
    img_painted1=homo.transform_image(Domain_img,Domain_pts,H,1)
    savefilename="painting"+str(savefileint)+"_kittens.jpeg"
    cv2.imwrite(savefilename,img_painted1)


# Read all the painitng images (Domain Images)
painting_1 = cv2.imread('painting1.jpeg')
painting_2 = cv2.imread('painting2.jpeg')
painting_3 = cv2.imread('painting3.jpeg')
# Read the kitten image (Range Image)
image_kittens=cv2.imread('kittens.jpeg')

# Make 2D array of corner points of rectangles in different painitng images
PSRQ_Painting1 = np.array([[233,412],[1925,202],[1820,1988],[139,1693]])
PSRQ_Painting2 = np.array([[166,515],[1966,631],[1969,2096],[172,2525]])
PSRQ_Painting3 = np.array([[74,356],[1366,110],[1204,2064],[67,1422]])
# Make a 2D array of corner points of Kittens
PSRQ_kits = np.array([[0,0],[image_kittens.shape[1]-1,0],[image_kittens.shape[ 1]-1,
                image_kittens.shape[0]-1],[0,image_kittens.shape[0 ]-1 ]])
""
#Save the results for task 1.1
Save(painting_1,image_kittens,PSRQ_Painting1,PSRQ_kits,1)
Save(painting_2,image_kittens,PSRQ_Painting2,PSRQ_kits,2)
Save(painting_3,image_kittens,PSRQ_Painting3,PSRQ_kits,3)

""
```

4.2. Code for Task 1.2

```
'''-------Main Code for Task-1.2---------'''
painting_1 = cv2.imread('painting1.jpeg')
painting_2 = cv2.imread('painting2.jpeg')
painting_3 = cv2.imread('painting3.jpeg')

homo=Homography(painting_1)
H12= homo.find_homography (PSRQ_Painting2 , PSRQ_Painting1)
H23= homo.find_homography (PSRQ_Painting3 , PSRQ_Painting2)
H = np.dot(H12,H23)
Domian_pts_21=                    np.array([[0,0],[painting_3.shape[1]-1,0],[painting_3.shape[1]-
1,painting_3.shape[0]-1],[0,painting_3.shape[0 ]-1]])
img_painted1=homo.transform_image(painting_3,Domian_pts_21,H,2)
cv2.imwrite("Painting1_to_3.jpeg",img_painted1)
```

4.3. Class for Homography computation and image warping

```
class Homography(object):

    def __init__(self,Range_img):
        """
        Class Responsible for computing and applying Homograpies.
        Inputs:
            Domain_pts: An n x 2 array containing coordinates of domian image points(Xi,Yi)
            range_point: An n x 2 array containing coordinates of range image points(Xi',Yi')
            Domain_img: Domain image
            Range_img: Range image

        """
        #self.Domain_pts=Domain_pts
        #self.Range_pts=Range_pts
        #self.Domain_img=Domain_img
        self.Range_img=Range_img
    def find_homography (self,Domain_pts,Range_pts):
        '''
        function for estimating the 3 x 3 Homography matrix
            Output: A 3 x 3 Homography matrix
        '''
        # Find num of points provided
        n = Domain_pts.shape[0]
        #Initialize A Design matrix having size of 2n x 8
        A = np.zeros((2*n,8))
        #Reshape the Range_pts to 1D vector of size 2*num_pts x 8
        y = Range_pts.reshape((2*n, 1))
        #Loop through all the points provided and stack them vertically, this will result in 2n x 8
Design matrix
```

```
        for i in range (n):
            A[i*2:i*2+2]=self.Get_A_matrix(Domain_pts[i],Range_pts[i])
        '''
        Compute the h vector (2n x 1) by using least sqaures solution. In case we have unique
        solution from 4 points, we can directly inverse the A design matrix. But for the
overdetermined
        system, we can use the least sqaures estimation.
        '''
        h=np.dot(np.linalg.inv(np.dot(A.T,A)),np.dot(A.T,y))
        # Add the last element as 1 in the h vector to obtain HC. Now h will be 2*n+1 x 1 vector.
        h = np.concatenate((h,1), axis=None)
        #Reshape the h vector to H homography matrix
        H = h.reshape((3,3))
        return H


    def Get_A_matrix(self,domain_point,range_point):
        '''
        function for generating a 2 x 8 design matrix needed to compute Homography
        Inputs:
            domain_point: Coordinates of a point in the domain image (x,y)
            range_point: Coordinates of corresponding point in the range image (x',y')
        Output: A 2 x 8 design matrix
        '''
        #Extract the x and y coordinates from a point pair
        x,y=domain_point[0], domain_point[1]
        xr,yr=range_point[0], range_point[1]
        #  Make A matrix
        A=np.array([[x,y,1,0,0,0,-x*xr,-y*xr],[0,0,0,x,y,1,-x*yr,-y*yr]])
        return A

    def transform_image(self,Domain_img,Domain_pts,Homography,task):
        '''
        function for applying the 3 x 3 Homography to the domian image to obtain new range pixel
        coordinates and then painting the range image onto the domian image using the new range
pixel coords.
        range
            Output: A new domian image with the range image projected on it.
        '''
        #Get the height and width of Domain Image
        height, width = Domain_img.shape[:2]
        if task==1:
            Domain_img_gen = Domain_img
        elif task==2:
            Domain_img_gen = np.zeros (Domain_img.shape,dtype='uint8')
        else:
            raise NotImplementedError('Task not implemented. Pick 1 or 2 for 1.1 and 1.2,
respectively')

        # Mask the PQRS region of the Domain image using fillpolygon tool, since it is not rectangle
        masked_domain =
cv2.fillPoly(np.zeros(Domain_img.shape[0:2],dtype='uint8'),[Domain_pts],255)
        # Traverse through each Domain image pixel by pixel   TODO: Vectorize the loops
        # Loop for controlling the columns of image
        for x in range(width):
            #loop for controlling the rows of image
            for y in range(height):
                #Check if we are inside the polygon defined by Domain_pts
                if (masked_domain[y,x]) > 0:
                    #Convert the domain pixel coordinates from the physical space to HC
                    Pix_domain = np.array([x,y,1])
                    #Apply the homogrpahy to the HC pixel coordinates to obtain the Range coordinates
X'=HX
                    Pix_range=Homography@ Pix_domain
                    #Convert from the range image pixel HC to physical space
                    Pix_range =Pix_range/Pix_range[2]
                    # Find the nearest neighbor by rounding to the nearest integer as pix coordinates are
ints
                    Pix_range = np.round(Pix_range).astype(np.int)
                    #Check if the new range coords computed from Homography are within bounds of
actual range image
                    #if(Pix_range[0] > 0 & Pix_range[1] > 0 & Pix_range[0] < self.Range_img.shape[1]
& Pix_range[1] < self.Range_img.shape[0]):
                    if(Pix_range[0] < self.Range_img.shape[1]) & (Pix_range[1] <
self.Range_img.shape[0]):
                      #Paint the Range image on the Domain image using new range coords
                      Domain_img_gen[y,x] = self.Range_img[Pix_range[1]][Pix_range[0]]
        return Domain_img_gen
```