ECE661: Homework 2

Rahul Deshmukh PUID: 0030004932

September 5, 2018

1 Logic and Methodology

For this Problem we can divide our work into two smaller problems:

a) Find Homography using 4 points

b) Mapping Image from Source to Destination

Finding Homography

For this sub-problem we have the equation $H * x = x'$;where H is the homography, x is the homogeneous coordinate of a point at the source image and x' is the homogeneous coordinate of a point at the destination image.

Now, we need to find the elements of the H,$3x3$ matrix. Also, as any other homography $k*H$ is equivalent to H and thus the information given by H is only in the ratios of the elements. Therefore, We can assume the $(3,3)$ index of H as 1, and will only need to find 8 other elements of H namely $h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}$ & h_{32}.

For any point $x = [x_1, y_1, 1]^T$ we have the equation:

$$H * x = x'$$

$$\Rightarrow \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x'_1 \\ x'_2 \\ x'_3 \end{bmatrix}$$

Thus we get:

$$h_{11}x_1 + h_{12}y_1 + h_{13} = x'_1$$

$$h_{21}x_1 + h_{22}y_1 + h_{23} = x'_2$$

$$h_{31}x_1 + h_{32}y_1 + 1 = x'_3$$

Now in the above equation we have the information of only the $\mathbb{R}^2$ coordinates. That is $x' = x'_1/x'_3$ and $y' = x'_2/x'_3$. Therefore, in order to make the right hand side as a vector of known quantities we need to divide the matrix equation by the scalar x'_3

Thus we get:

$$\frac{1}{(h_{31}x_1 + h_{32}y_1 + 1)} * \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = \begin{bmatrix} x'_1 \\ y'_1 \\ 1 \end{bmatrix}$$

Writing individual equation we get:

$$x'_1 = \frac{(h_{11}x_1 + h_{12}y_1 + h_{13})}{(h_{31}x_1 + h_{32}y_1 + 1)}$$

$$\Rightarrow h_{11}x_1 + h_{12}y_1 + h_{13} - h_{31}x_1x'_1 - h_{32}y_1x'_1 = x'_1$$

$$y'_1 = \frac{(h_{21}x_1 + h_{22}y_1 + h_{23})}{(h_{31}x_1 + h_{32}y_1 + 1)}$$

$$\Rightarrow h_{21}x_1 + h_{22}y_1 + h_{23} - h_{31}x_1y'_1 - h_{32}y_1y'_1 = y'_1$$

Note that, in the above equations we need to solve h_{ij}, therefore we will need atleast 4 more points in order to get atleast 8 equations to solve the 8 unknowns.

We can write out the system of equations in the form of $A\vec{x} = \vec{b}$. Also, if we order our vectors $\vec{x}$ and $\vec{b}$ in a particular order then we get a block Matrix for A, which will be suited in the future when we will have more than 4 points.

Let the four points in the source plane be denoted by $P = [P_1, P_2]^T$,$Q = [Q_1, Q_2]^T$,$R = [R_1, R_2]^T$ and $S = [S_1, S_2]^T$
and those in Destination Plane be denoted by $P' = [P'_1, P'_2]^T$,$Q' = [Q'_1, Q'_2]^T$,$R' = [R'_1, R'_2]^T$ and $S' = [S'_1, S'_2]^T$

Taking $\vec{x} = [h_{11}, h_{12}, h_{13}, h_{21}, h_{22}, h_{23}, h_{31}, h_{32}]^T$ and $\vec{b} = [P_1, Q_1, R_1, S_1, P_2, Q_2, R_2, S_2]^T$ we get the Matrix A as:

$$A = \begin{bmatrix} P_1 & P_2 & 1 & 0 & 0 & 0 & -P'_1P_1 & -P'_1P_2 \\ Q_1 & Q_2 & 1 & 0 & 0 & 0 & -Q'_1Q_1 & -Q'_1Q_2 \\ R_1 & R_2 & 1 & 0 & 0 & 0 & -R'_1R_1 & -R'_1R_2 \\ S_1 & S_2 & 1 & 0 & 0 & 0 & -S'_1S_1 & -S'_1S_2 \\ 0 & 0 & 0 & P_1 & P_2 & 1 & -P'_2P_1 & -P'_2P_2 \\ 0 & 0 & 0 & Q_1 & Q_2 & 1 & -Q'_2Q_1 & -Q'_2Q_2 \\ 0 & 0 & 0 & R_1 & R_2 & 1 & -R'_2R_1 & -R'_2R_2 \\ 0 & 0 & 0 & S_1 & S_2 & 1 & -S'_2S_1 & -S'_2S_2 \end{bmatrix}$$

Clearly the above matrix can be constructed using basic vectorization and if we have more points then those can be accommodated easily.

The function written out in Python for this task is:

```
def HMatrix(X,x):
    """returns a homography mat such that
    x=H*X and H[2,2]=1
    x is on destination Image
    X is on source Image
    x,X=np.array([[x1,y1],[x2,y2],[...],[...]])
    """
    n=np.shape(X)[0]
    m=np.shape(X)[1]
    A=np.zeros((2*n,8))
    b=np.reshape(x.T,n*m)

    block1=np.zeros((n,m+1))
    block1[:,-1]=np.ones((1,n))
    block1[:,:2]=X[:,:]
    A[:n,:3]=block1
    A[n:2*n,3:6]=block1

    block2=np.multiply(X.T,-1*x[:,0])
    block3=np.multiply(X.T,-1*x[:,1])

    A[:n,6:8]=block2.T
    A[n:2*n,6:8]=block3.T
```

```
h=np.linalg.solve(A,b)
h=np.concatenate([h,[1]])
H=np.reshape(h,(3,3))
return(H)
```

Mapping Image from Source to Destination

Now that we have calculated the homography, we now need to map the source image to the destination. This means that we need to assign a pixel value to any point in the destination plane which falls in our region of interest and the assigned pixel value will be obtained form the source image.

However, note that any point x' in the destination plane when transformed to the source plane, using $x = H^{-1}x'$, will not always have integer coordinates. As we have pixel values only at integer coordinates in any image, therefore we will need to interpolate the pixel value at x from its closest neighbors.

Also, the RGB pixel values will always be an integer in the range 0-255. Thus, the interpolated pixel value at x should conform to this format. Therefore, a interpolation scheme with sum of weights=1 will be best suited here.

Interpolation Scheme using L2 norm and squared L2 norm

For any point x, the four closest integer points can simply be constructed using a permutations of floor and ceiling of the coordinates of x. We are using Euclidean distance (or the L2 norm) of the point x to any of the closest point normalized by the sum of distances for all the four points as our interpolation weights.

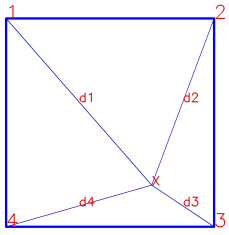

Figure 1: interpolation diagram

As shown from in Figure 1 we can define our interpolation scheme as:

$$c(x) = \frac{d_1 c(1) + d_2 c(2) + d_3 c(3) + d_4 c(4)}{d_1 + d_2 + d_3 + d_4}$$

where, $c(x)$ denotes pixel values at any point x and d_i denotes the distance of point i from the point x. Similarly for squared L2 norm we are just squaring the Euclidean distance

Interpolation Scheme using Bilinear Shape functions

Another way to obtain weights is to make use of Bilinear shape functions, a technique used in Finite Elements. The shape functions are defined using a parent element with orhogonal coordinates as ξ and η in the domain $-1 \leq \xi \leq 1$ and $-1 \leq \eta \leq 1$.

The corresponding shape function,N_i, at the point (ξ_i, η_i) is given by:

$$N_i(\xi, \eta) = \frac{(1 + \xi_i \xi) * (1 + \eta_i \eta)}{4} \qquad i = 1, 2, 3, 4$$

We can use these shape functions by converting our problem into parent domain, using a simple transformation of coordinates.

Interpolation Scheme using Round Up and Round Down

This method is computationally the least expensive as we are not finding any distances,which involves finding a square root, and niether involves any linear interpolation. In this method we just assign the pixel value at x as that of the point 1 in case of rounding down and point 3 in case of rounding up.(refer to Figure 1 for naming of points)

Further, The pixel value obtained from the above interpolation may have floats as value. Therefore, we need to take floor/ceiling of the values to convert them into integers.

The set of python functions which map the source image onto a destination image is:

```
#Interplation of colors from the source image
def InterpolatePixel(x,srcImg,distype):
    """
    Inputs:
        x= n.array([X,Y]) coordinates of a point
        srcImg=source image object a n,m,3 matrix
        distype= a string either L2 ,sqL2,BiLinear,RoundDown, RoundUp
    Output:
        cx= size(1,3) vector of pixel values with only integer values
    The function will interpolate the pixel values of the nearest neighbors and do a
        rounding
    to get integers
    """
    #finding the nearest interger neighbours
    p1=[int(np.floor(x[0])),int(np.floor(x[1]))]
    p2=[int(np.ceil(x[0])),int(np.floor(x[1]))]
    p3=[int(np.ceil(x[0])),int(np.ceil(x[1]))]
    p4=[int(np.floor(x[0])),int(np.ceil(x[1]))]
    #storing value of pixels at these points
    #x: col index & y:row index
    c1=srcImg[p1[1],p1[0],:]
    c2=srcImg[p1[1],p1[0],:]
    c3=srcImg[p1[1],p1[0],:]
    c4=srcImg[p1[1],p1[0],:]
    C=np.array([c1,c2,c3,c4])
    #storing weights for interpolation
    w=weights(np.array([p1,p2,p3,p4]),x,distype);#should return a list
    #interpolating
    fcx=np.dot(C.T,w)
    fcx=np.floor(fcx)
```

```
    cx=fcx.astype(int)
    return(cx)

def weights(pts,x,distype):
    """
    Function will give a vector of weights
    input:  pts = np.array([[X Y],
                    [X Y]...]
    x=np.array[X,Y]
    distype is a string with options L2 ,sqL2,BiLinear,RoundDown, RoundUp
    output:
        w=[w1,w2,w3,w4] a list
    """
    w=[]
    if distype=='L2':
        #use L2 norm for distance
        for i in range(np.shape(pts)[0]):
            w.append(np.linalg.norm(pts[i,:]-x))
        wsum=sum(w)
        w=w/wsum
        return(w)

    elif distype=='sqL2':
        #use squared L2 norm for distance
        for i in range(np.shape(pts)[0]):
            w.append((np.linalg.norm(pts[i,:]-x))**2)
        wsum=sum(w)
        w=w/wsum
        return(w)
    elif distype=='BiLinear':
        #using bilinear shape functions as weights
        xmin=pts[0,0] #the points were ordered in that specific manner
        xmax=pts[1,0]
        ymin=pts[0,1]
        ymax=pts[3,1]
        xi=2*((x[0]-xmin)/(xmax-xmin))-1 #parent element coordinates
        eta=2*((x[1]-ymin)/(ymax-ymin))-1
        w.append((1-xi)*(1-eta)/4.0) #shape functions
        w.append((1+xi)*(1-eta)/4.0)
        w.append((1+xi)*(1+eta)/4.0)
        w.append((1-xi)*(1+eta)/4.0)
        return(w)
    elif distype=='RoundDown':
        w=[1,0,0,0]
        return(w)
    elif distype='RoundUp':
        w=[0,0,1,0]
        return(w)

def mapImage(srcImg,srcPts,destImg,destPts,H,distype,path,name):
    """
    Input:
        srcImg & destImg: Image matrices
        srcpts& destpts =np.array([[X,Y],[X Y]...])
        H is a 3x3 matrix
    Output: Function will write out the merged image
    """
    resultImg=np.zeros(np.shape(destImg))
    invH=np.linalg.inv(H)

    xmin=min(srcPts[:,0])
    xmax=max(srcPts[:,0])
    ymin=min(srcPts[:,1])
    ymax=max(srcPts[:,1])
    # iterate over points in destImg
```

```
for i in range(np.shape(destImg)[0]):
    for j in range(np.shape(destImg)[1]):
        #i is row number and corresponds to y coordinate
        #j is col numer and corresponds to x coordinate
        x = np.dot(invH,[j,i,1])
        x=np.array([x[0]/x[-1],x[1]/x[-1]])
        #now we have a non-integer point in src Img
        #check if the point lies insisde the window in the source image
        if x[0]>xmin and x[0]<xmax and x[1]>ymin and x[1]<ymax:
            #find new pixel value
            cx = InterpolatePixel(x,srcImg,distype)
            #assign new pixel value to destImg
            resultImg[i,j,:]=cx
        else:
            resultImg[i,j,:]=destImg[i,j,:]

cv2.imwrite(path+name+'.jpg',resultImg)
return()
```

Methodology for Task 1(b) of homework

For this task we can make use of the functions we have created earlier. We can easily find the homographies and find their product. When mapping the source image, we don't have a destination image, for this a blank canvas (all black) was made of the same size as that of the source image.

Coordinates of picked points

Task 1	**Task 2**
$img1P = [1516, 170]$	$img1P = [3016, 1471]$
$img1Q = [2953, 720]$	$img1Q = [3320, 1500]$
$img1R = [1491, 2237]$	$img1R = [3040, 1941]$
$img1S = [2998, 2048]$	$img1S = [3364, 1946]$
$img2P = [1323, 330]$	$img2P = [2969, 1603]$
$img2Q = [3012, 615]$	$img2Q = [3355, 1616]$
$img2R = [1301, 2012]$	$img2R = [2980, 2097]$
$img2S = [3031, 1899]$	$img2S = [3382, 2105]$
$img3P = [920, 730]$	$img3P = [2637, 1475]$
$img3Q = [2800, 380]$	$img3Q = [2966, 1412]$
$img3R = [899, 2091]$	$img3R = [2639, 1943]$
$img3S = [2851, 2231]$	$img3S = [2980, 1909]$
$img4P = [190, 3]$	$img4P = [0, 0]$
$img4Q = [190 + 856, 3]$	$img4Q = [np.shape(img4)[1] - 1, 0]$
$img4R = [190, 3 + 717]$	$img4R = [0, np.shape(img4)[0] - 1]$
$img4S = [190 + 856, 3 + 717]$	$img4S = [np.shape(img4)[1] - 1, np.shape(img4)[0] - 1]$

2 Given Images with manually picked Points

2.1 Images of Task 1

Figure 2: image 1(a)

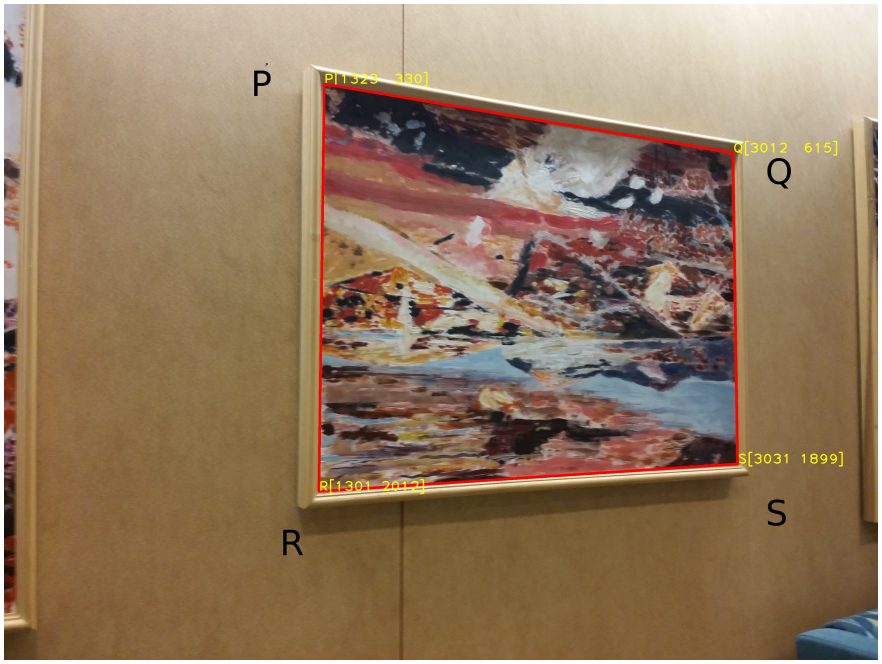

Figure 3: image 1(b)

Figure 4: image 1(c)

Figure 5: image 1(d)

2.2 Images of Task 2

Figure 6: image 2(a)

Figure 7: image 2(b)

Figure 8: image 2(c)

Figure 9: image 2(d)

3 Results

3.1 Results of Task 1

3.1.1 Results 1(a)

Figure 10: image 1(d) mapped onto 1(a)

Figure 11: image 1(d) mapped onto 1(b)

Figure 12: image 1(d) mapped onto 1(c)

3.1.2 Results 1(b)

Figure 13: image 1(a) mapped using homography H12*H23

3.2 Results of Task 2

3.2.1 Results 2(a)

Figure 14: image 2(d) mapped onto 2(a)

Figure 15: image 2(d) mapped onto 2(b)

Figure 16: image 2(d) mapped onto 2(c)

3.2.2 Results 2(b)

Figure 17: image 2(a) mapped using homography H12*H23

4 Complete Code

```
#author: rahul deshmukh
#import libraries
import cv2
import numpy as  np

path='./pics2/'
#read images
img1=cv2.imread(path+'1.jpg')
img2=cv2.imread(path+'2.jpg')
img3=cv2.imread(path+'3.jpg')
img4=cv2.imread(path+'BoilerMakerSpecial.png')

#defining the manually picked coordinates
#img1
img1P=[3016,1471]
img1Q=[3320,1500]
img1R=[3040,1941]
img1S=[3364,1946]
#img2
img2P=[2969,1603]
img2Q=[3355,1616]
img2R=[2980,2097]
img2S=[3382,2105]
#img3
img3P=[2637,1475]
img3Q=[2966,1412]
img3R=[2639,1943]
img3S=[2980,1909]

#for full image
img4P=[0,0]
img4Q=[np.shape(img4)[1]-1,0]
img4R=[0,np.shape(img4)[0]-1]
img4S=[np.shape(img4)[1]-1,np.shape(img4)[0]-1]

distype='L2'
pts1=np.array([img1P,img1Q,img1S,img1R])
pts2=np.array([img2P,img2Q,img2S,img2R])
pts3=np.array([img3P,img3Q,img3S,img3R])
pts4=np.array([img4P,img4Q,img4S,img4R])
#----------------task a-------------------------#
H41=HMatrix(pts4,pts1)
mapImage(img4,pts4,img1,pts1,H41,distype,path,'result41'+distype)
H42=HMatrix(pts4,pts2)
mapImage(img4,pts4,img2,pts2,H42,distype,path,'result42'+distype)
H43=HMatrix(pts4,pts3)
mapImage(img4,pts4,img3,pts3,H43,distype,path,'result43'+distype)
#----------------task b-------------------------#
H12=HMatrix(pts1,pts2)
H13=HMatrix(pts2,pts3)
H_calc=np.dot(H12,H13)

img1P=[0,0]
img1Q=[np.shape(img1)[1]-1,0]
img1R=[0,np.shape(img1)[0]-1]
img1S=[np.shape(img1)[1]-1,np.shape(img1)[0]-1]
pts1=np.array([img1P,img1Q,img1S,img1R])
destImg=np.zeros(np.shape(img3))

mapImage(img1,pts1,destImg,pts1,H_calc,distype,path,'task1b')
```