Q1.1

To begin this question I first start with a plane on which all points on this 3D plane will have the coordinates of (x,y, z = 0). Next I relate the 2D representation of the plane to the 3D plane using the equations below where P is a 3x3 matrix and Q is a 3x1 matrix.

$$x_1 = P_1 * Q \text{ and } x_2 = P_2 * Q$$

$$Q = {P^{-1}}_2 * x_2$$

$$\lambda x_1 = P_1 * ({P^{-1}}_2 * x_2)$$

$$\boxed{x_1 \equiv H x_2}$$

Hence, we can say that there exists some homography matrix H.

Q1.2

1. DOF = 8
2. 4 point pairs (because of 8 DOF)
3. $$\lambda \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}}_{\boxed{h}} \underbrace{\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}}_{\boxed{x_2'}}$$

$$\lambda = h_{31} * X + h_{32} * Y + h_{33}$$

$$\lambda * x' = h_{11} * X + h_{12} * Y + h_{13}$$

$$\lambda * y' = h_{21} * X + h_{22} * Y + h_{23}$$

$$x' = \frac{h_{11}*X+h_{12}*Y+h_{13}}{h_{31}*X+h_{32}*Y+h_{33}},\ y' = \frac{h_{21}*X+h_{22}*Y+h_{23}}{h_{31}*X+h_{32}*Y+h_{33}}$$

Re-assembling these equations to match the form $\boldsymbol{A_i h = 0}$, yields:

$$x'h_{31}X + h_{32}Yx' + h_{33}x' - h_{11}X - h_{12}Y - h_{13} = 0$$

$$y'h_{31}X + h_{32}Yy' + h_{33}y' - h_{21}X - h_{12}Y - h_{23} = 0$$

$$\begin{bmatrix} -X & -Y & -1 & 0 & 0 & 0 & x'X & Yx' & x' \\ 0 & 0 & 0 & -X & -Y & -1 & x'X & Yx' & y' \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix}$$

$$A_i = \begin{bmatrix} -X & -Y & -1 & 0 & 0 & 0 & x'X & Yx' & x' \\ 0 & 0 & 0 & -X & -Y & -1 & x'X & Yx' & y' \end{bmatrix}$$

4. Trivial solution is if all elements of h=0 then $A_i h = 0$. No the matrix A is not full rank because we don't have all of the values needed to find a unique solution for A. A will be an 8 by 9 matrix and we are multiplying by h (a 9X1 matrix) so we end up with 9 unknowns and only 8 equations, so we don't have any unique solutions to the equations but there any infinitely many solutions to the equations meaning that it does not have full rank. For the trivial solution of Ah where h = 0, there is a corresponding eigenvector and eigenvalue which both equal to zero. As the A gets larger the singular values should also grow and vice versa.

Q1.4.1

$$x_1 = K_1 * x_1 \text{ and } x_2 = K_2 R x_2$$

$$x_2 = K_2 R * \left(\frac{x_1}{K_1}\right) = K_2 R {K_1}^{-1} * x_1$$

$$x_2 \equiv H x_1$$

There exists a homography matrix H.

Q1.4.2

$$H^2 = (K_2 R {K_1}^{-1})^2$$

$$x_2 \equiv H^2 x_1$$

$$x_2 \equiv \underbrace{KR{K_1}^{-1} * KR{K_1}^{-1}}_{\boxed{2\theta}} x_1$$

$$x_2 \equiv KR^2K^{-1}x_1$$

*Where R is the rotate matrix = $\begin{bmatrix} cos\,\theta & -\,sin\,\theta & 0 \\ sin\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Q1.4.3

Planar homography would actually work really well when trying to map between images where there are many distinct features, however in images where there are not as many unique features this can cause error between matching corresponding points (i.e. building with many windows). It is also only sufficient for a point on a world plane (assuming z=0) but not other planes.

Q1.4.4

$$x = Px'$$

$$x' = x'_1 + \lambda(x'_2 - x'_1)$$

$$P * x' = P[x_1 + \lambda(x_2 - x_1)]$$

$$Px = Px_1 + \lambda(Px_2 - Px_1)$$

Let x' be equal to $\boldsymbol{Px}$, $\boldsymbol{x'_1 = Px_1}$, $\boldsymbol{x'_2 = Px_2}$

$Px = Px_1 + \lambda(Px_2 - Px_1)$ *substitute into equation to get

$$x' = x'_1 + \lambda(x'_2 - x'_1)$$

Line is maintained after applying projection P.

Q2.1.1

The Harris Corner detector finds the differences in intensity for a displacement of (u,v) is all directions.

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u,y+v) - I(x,y)]^2$$

It utilizes gaussians to control the blur and gaussians to control patch size, with the blurriness of the image being varied. We then compute the interestingness score (determine if a window can contain a corner or not). Whereas the FAST detector selects a pixel in the image (1) as an interest point or not, letting the intensity by Ip. A threshold value (t) is chosen, the pixel of interest is encircled and is evaluated to see if there is a set of n pixels in the circle (either brighter or darken than Ip @ t). The FAST method excludes large numbers of non-corners whereas the Harris detector will consider (to eliminate them) the entire window including corners, edges, and flat regions of the image. The FAST is computationally a lot faster than the Harris detector (5% vs 115%) but it does have drawbacks in that: the choice of pixels may not always be optimal, multiple features can be detected adjacent to one another, etc.

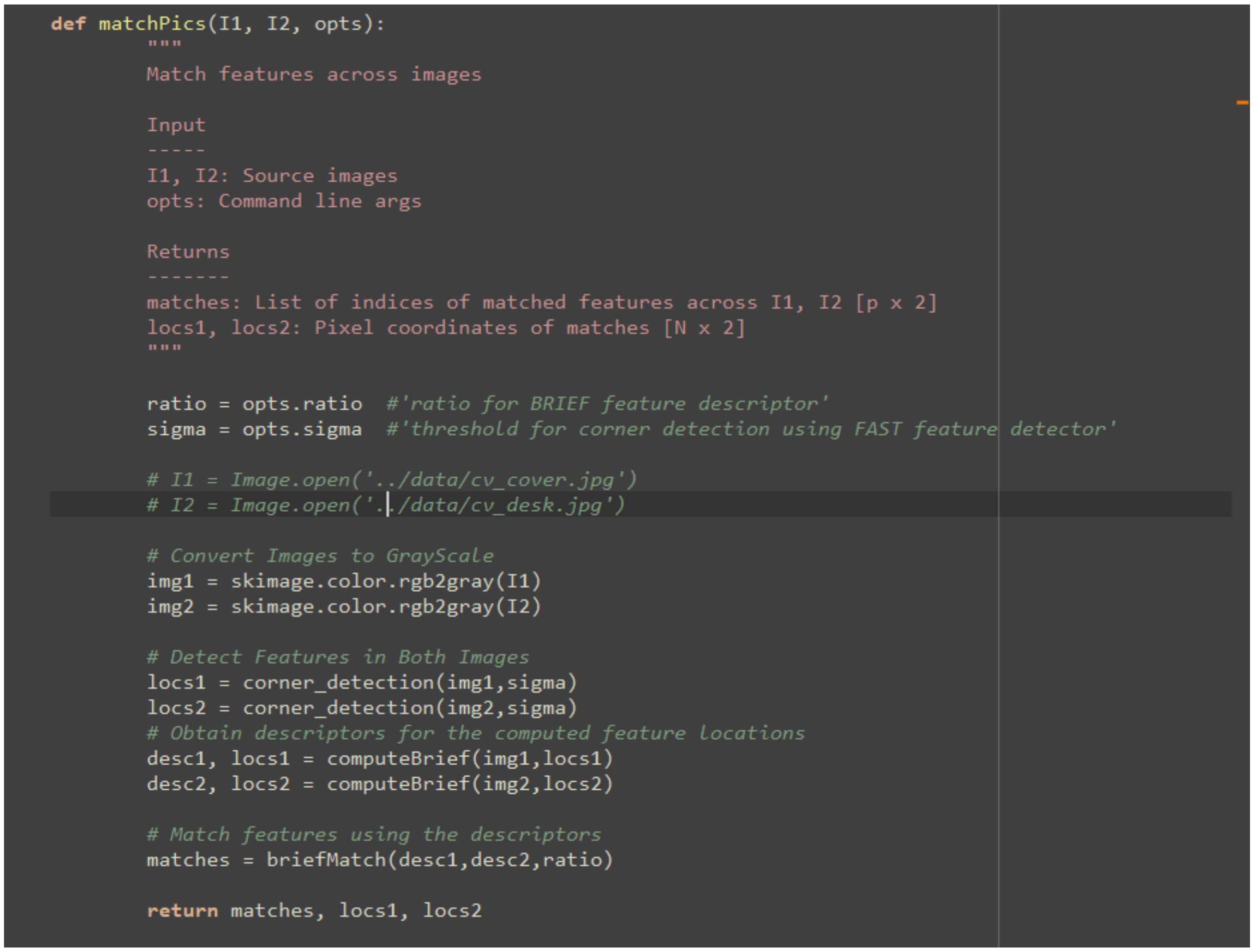
Q2.1.2

BRIEF provides a shortcut to finding binary strings directly without finding descriptors. Taking a smoothened image patch and selects a set of nd(x,y) location pair in a unique way. Intensity comparisons (of pixels) are done on these pairs returns a nd-dimensional bitstring. The claim here is that while filter banks use different filters (Gaussian, Laplace of gaussian etc.) to classify image features or smooth an image BRIEF descriptor use location pairs and compares the similarity between patches. Yes, Gaussian filter can be used as a descriptor.

Q2.1.3

The Hamming distance is used for binary descriptors (such as BRIEF), it is a metric that will compare two binary strings of data. The BRIEF descriptors will be the binary strings of data we are trying to compare, and it will compute the error between the two and return the smallest one (smallest distance i.e. the closest pair of points). It is not dependent on the values of the binary strings (let’s call them xi and yi) but rather only if they are equal or not equal. The Nearest Neighbor functions as a form of proximity search in that the optimization problem is of finding the point given in a data set that is closest to a given point (or most similar to it), it calculates the distance between training points and samples points and is evaluated. The Hamming distance method is faster than the nearest neighbor method because the nearest neighbor method utilizes the Euclidian distance between points.

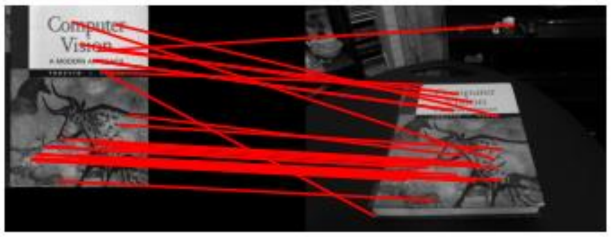
Q2.1.4



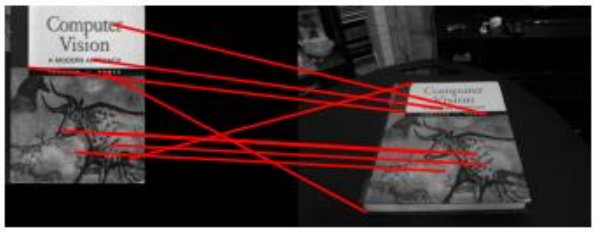
```
def matchPics(I1, I2, opts):
        """
        Match features across images

        Input
        -----
        I1, I2: Source images
        opts: Command line args

        Returns
        -------
        matches: List of indices of matched features across I1, I2 [p x 2]
        locs1, locs2: Pixel coordinates of matches [N x 2]
        """

        ratio = opts.ratio  #'ratio for BRIEF feature descriptor'
        sigma = opts.sigma  #'threshold for corner detection using FAST feature detector'

        # I1 = Image.open('../data/cv_cover.jpg')
        # I2 = Image.open('../data/cv_desk.jpg')

        # Convert Images to GrayScale
        img1 = skimage.color.rgb2gray(I1)
        img2 = skimage.color.rgb2gray(I2)

        # Detect Features in Both Images
        locs1 = corner_detection(img1,sigma)
        locs2 = corner_detection(img2,sigma)
        # Obtain descriptors for the computed feature locations
        desc1, locs1 = computeBrief(img1,locs1)
        desc2, locs2 = computeBrief(img2,locs2)

        # Match features using the descriptors
        matches = briefMatch(desc1,desc2,ratio)

        return matches, locs1, locs2
```

Q2.1.5

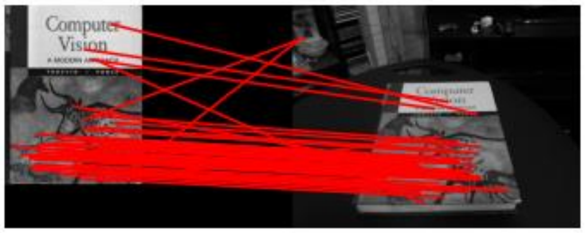
Sigma	Ratio
0.2	0.7
0.1	0.7
0.15	0.8
0.2	0.8
0.15	0.6



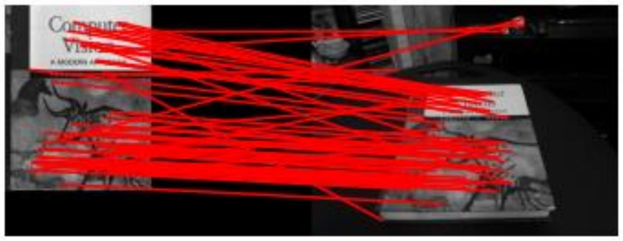
OG (0.15 – sigma and ratio -0.7)



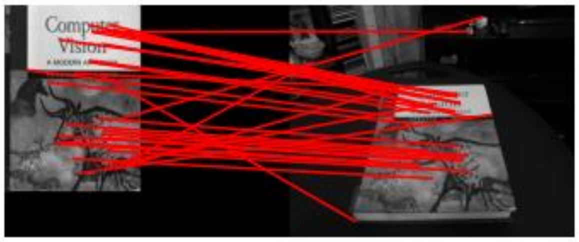
Sigma (0.2) and Ratio (0.7)



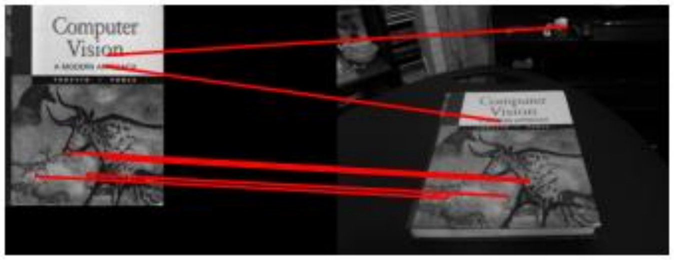
Sigma (0.1) and Ratio (0.7)



Sigma (0.15) and Ratio (0.8)



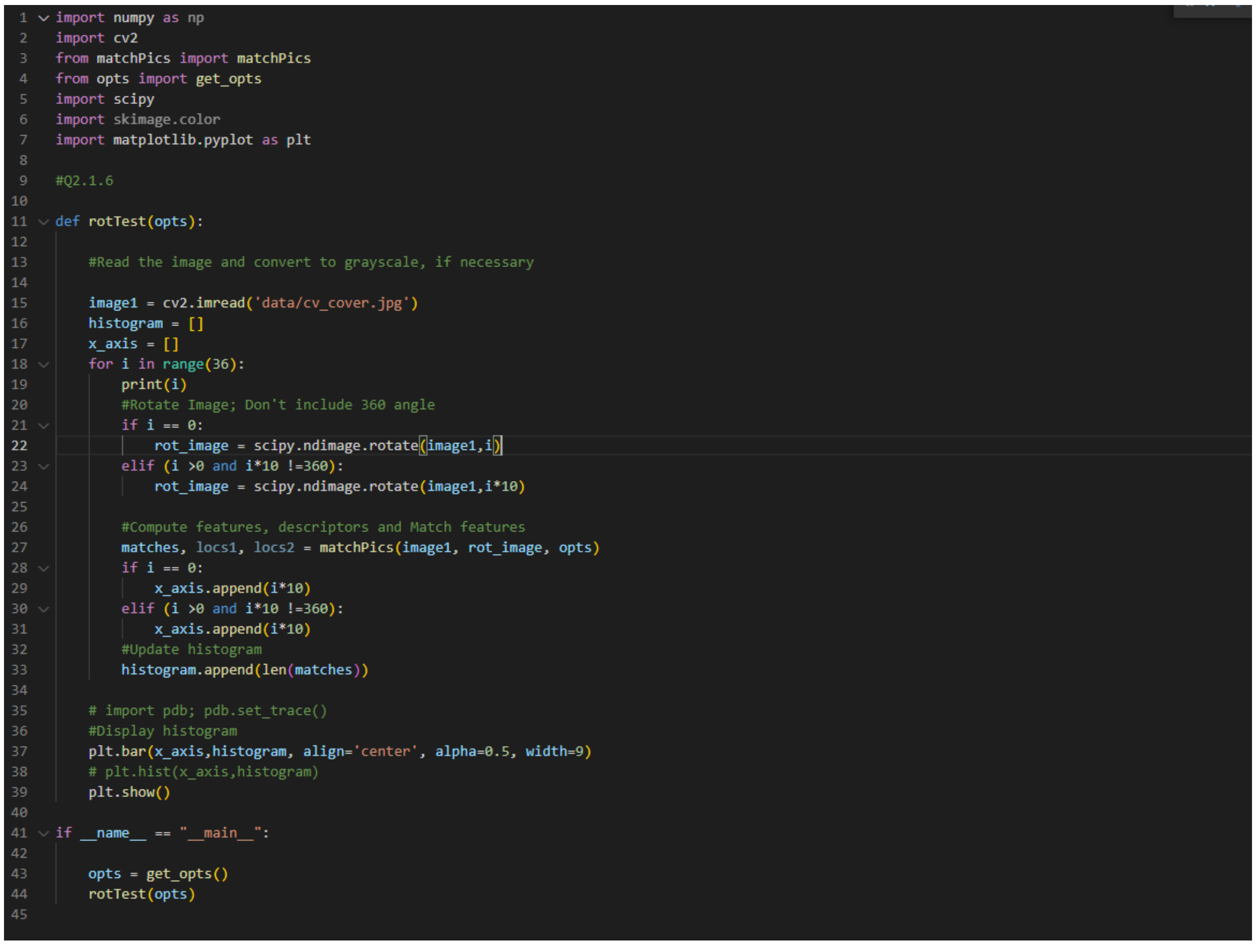
Sigma (0.2) and Ratio (0.8)



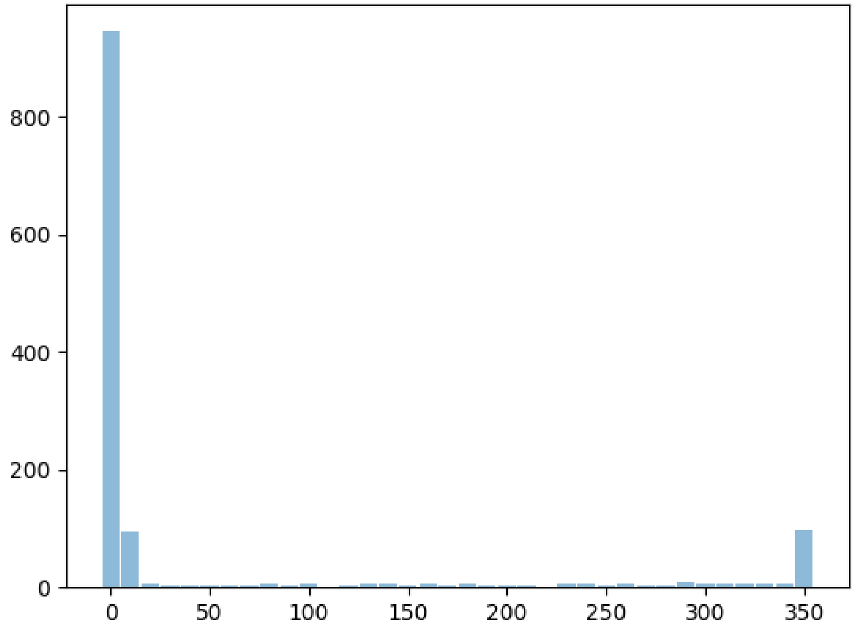
Sigma (0.15) and Ratio (0.6)

I varied the sigma and ratio parameters both separately and in one instance together, when the sigma value is lowered (0.1) there are more correspondences or matches found between the images but it seems to be slightly less accurate versus when the sigma value is raised (0.2) there are less matches found and the accuracy is minutely affected. When the ratio is varied, higher tends towards more matches between images and the accuracy overall seems to be much lower, and lower ratios tend to lower the number of matches and the accuracy. Varying the ratio and sigma together seems to produce mixed results in that it is hard to tell which parameter has more of an affect but the accuracy does not seem to have been that affected (it did not decrease by much) and the number of matches did increase as well.

Q2.1.6

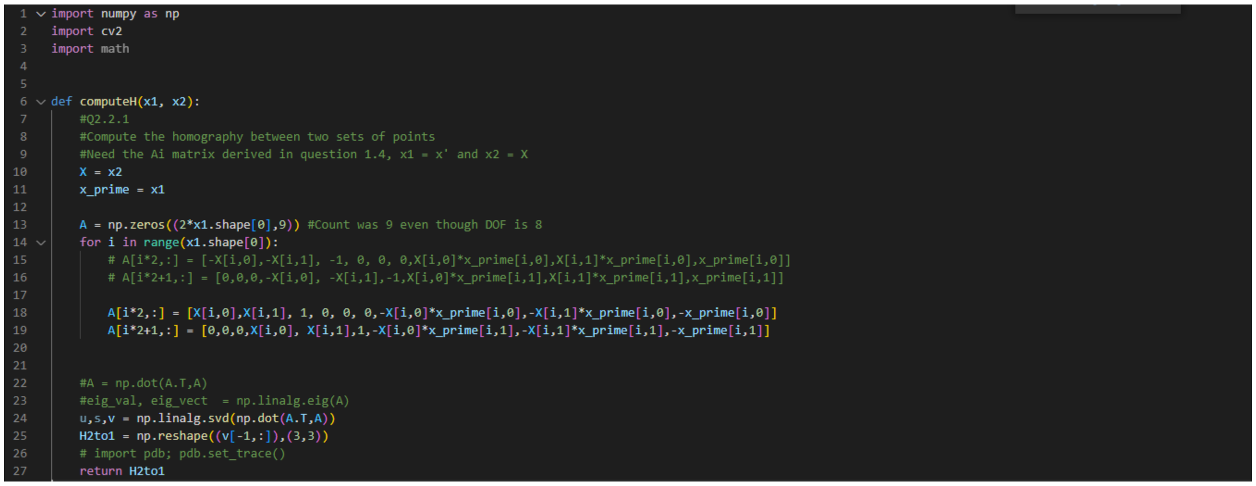


```
import numpy as np
import cv2
from matchPics import matchPics
from opts import get_opts
import scipy
import skimage.color
import matplotlib.pyplot as plt

#Q2.1.6

def rotTest(opts):

    #Read the image and convert to grayscale, if necessary

    image1 = cv2.imread('data/cv_cover.jpg')
    histogram = []
    x_axis = []
    for i in range(36):
        print(i)
        #Rotate Image; Don't include 360 angle
        if i == 0:
            rot_image = scipy.ndimage.rotate(image1,i)
        elif (i >0 and i*10 !=360):
            rot_image = scipy.ndimage.rotate(image1,i*10)

        #Computer features, descriptors and Match features
        matches, locs1, locs2 = matchPics(image1, rot_image, opts)
        if i == 0:
            x_axis.append(i*10)
        elif (i >0 and i*10 !=360):
            x_axis.append(i*10)
        #Update histogram
        histogram.append(len(matches))

    # import pdb; pdb.set_trace()
    #Display histogram
    plt.bar(x_axis,histogram, align='center', alpha=0.5, width=9)
    # plt.hist(x_axis,histogram)
    plt.show()

if __name__ == "__main__":

    opts = get_opts()
    rotTest(opts)
```



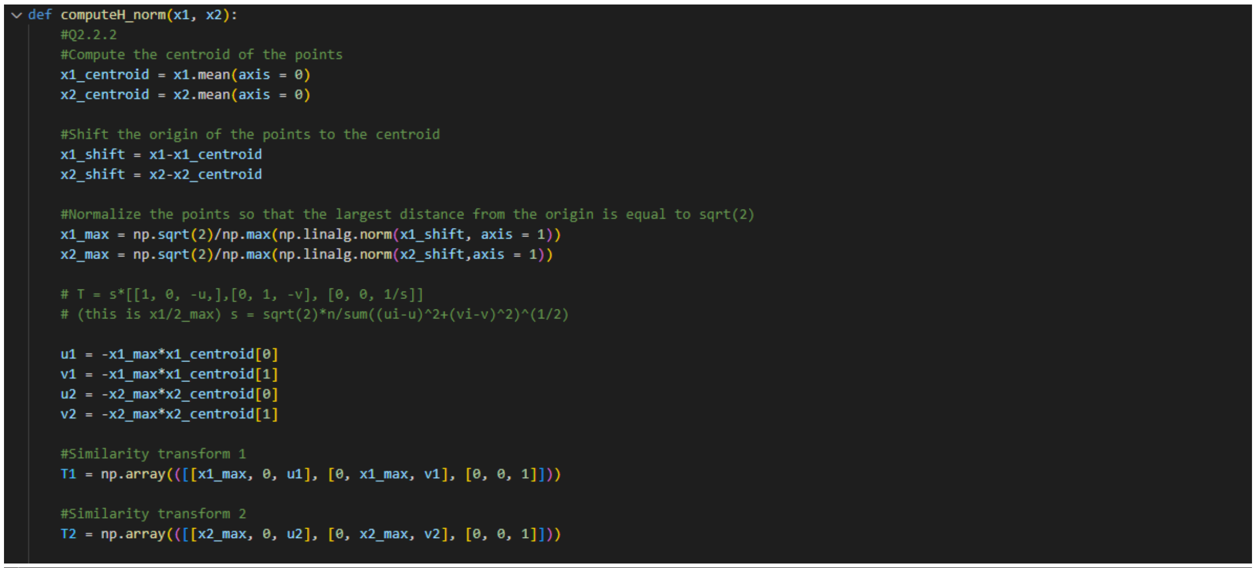
In the histogram shown above there are more matches at a rotation of 0 and 350 however in between these values there are not many matches that are found in the images. This could potentially be because as the image is rotated certain features are in a different orientation so this can lead to matches being missed, so the rotation and orientation of the image matters for the BRIEF matching process as it is not rotation invariant.

Q2.2.1

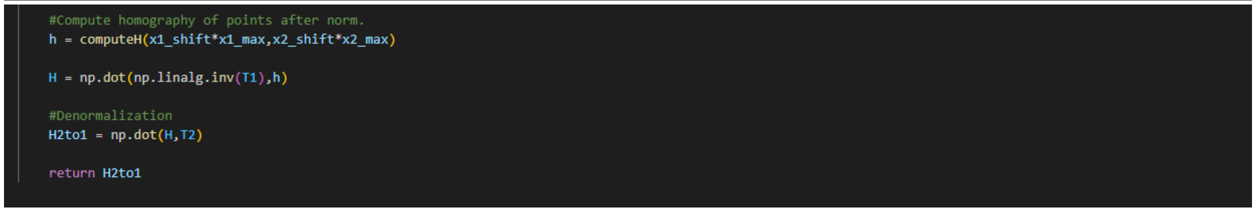


```python
import numpy as np
import cv2
import math


def computeH(x1, x2):
    #Q2.2.1
    #Compute the homography between two sets of points
    #Need the Ai matrix derived in question 1.4, x1 = x' and x2 = X
    X = x2
    x_prime = x1

    A = np.zeros((2*x1.shape[0],9)) #Count was 9 even though DOF is 8
    for i in range(x1.shape[0]):
        # A[i*2,:] = [-X[i,0],-X[i,1], -1, 0, 0, 0,X[i,0]*x_prime[i,0],X[i,1]*x_prime[i,0],x_prime[i,0]]
        # A[i*2+1,:] = [0,0,0,-X[i,0], -X[i,1],-1,X[i,0]*x_prime[i,1],X[i,1]*x_prime[i,1],x_prime[i,1]]

        A[i*2,:] = [X[i,0],X[i,1], 1, 0, 0, 0,-X[i,0]*x_prime[i,0],-X[i,1]*x_prime[i,0],-x_prime[i,0]]
        A[i*2+1,:] = [0,0,0,X[i,0], X[i,1],1,-X[i,0]*x_prime[i,1],-X[i,1]*x_prime[i,1],-x_prime[i,1]]


    #A = np.dot(A.T,A)
    #eig_val, eig_vect  = np.linalg.eig(A)
    u,s,v = np.linalg.svd(np.dot(A.T,A))
    H2to1 = np.reshape((v[-1,:]),(3,3))
    # import pdb; pdb.set_trace()
    return H2to1
```

Q2.2.2

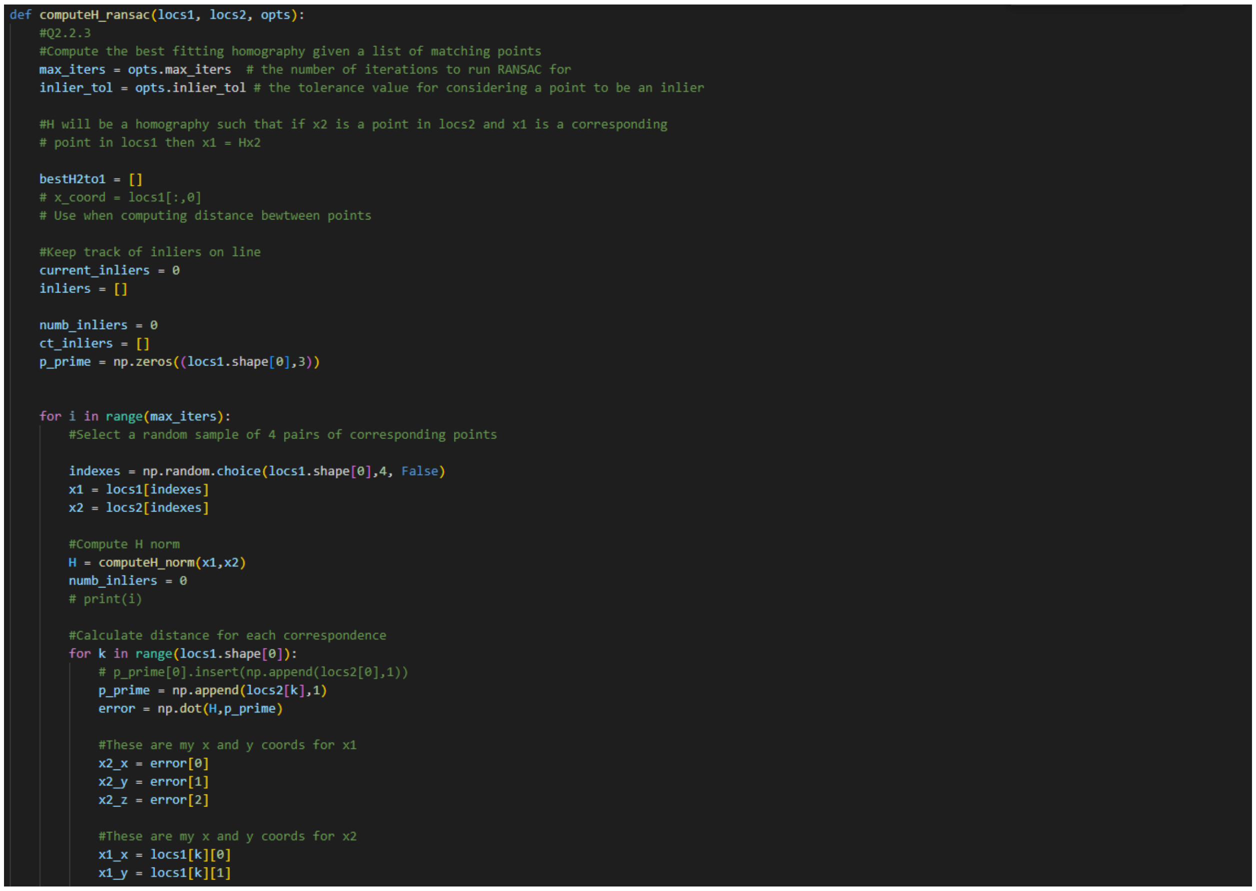


```
def computeH_norm(x1, x2):
    #Q2.2.2
    #Compute the centroid of the points
    x1_centroid = x1.mean(axis = 0)
    x2_centroid = x2.mean(axis = 0)

    #Shift the origin of the points to the centroid
    x1_shift = x1-x1_centroid
    x2_shift = x2-x2_centroid

    #Normalize the points so that the largest distance from the origin is equal to sqrt(2)
    x1_max = np.sqrt(2)/np.max(np.linalg.norm(x1_shift, axis = 1))
    x2_max = np.sqrt(2)/np.max(np.linalg.norm(x2_shift,axis = 1))

    # T = s*[[1, 0, -u,],[0, 1, -v], [0, 0, 1/s]]
    # (this is x1/2_max) s = sqrt(2)*n/sum((ui-u)^2+(vi-v)^2)^(1/2)

    u1 = -x1_max*x1_centroid[0]
    v1 = -x1_max*x1_centroid[1]
    u2 = -x2_max*x2_centroid[0]
    v2 = -x2_max*x2_centroid[1]

    #Similarity transform 1
    T1 = np.array(([[x1_max, 0, u1], [0, x1_max, v1], [0, 0, 1]]))

    #Similarity transform 2
    T2 = np.array(([[x2_max, 0, u2], [0, x2_max, v2], [0, 0, 1]]))
```

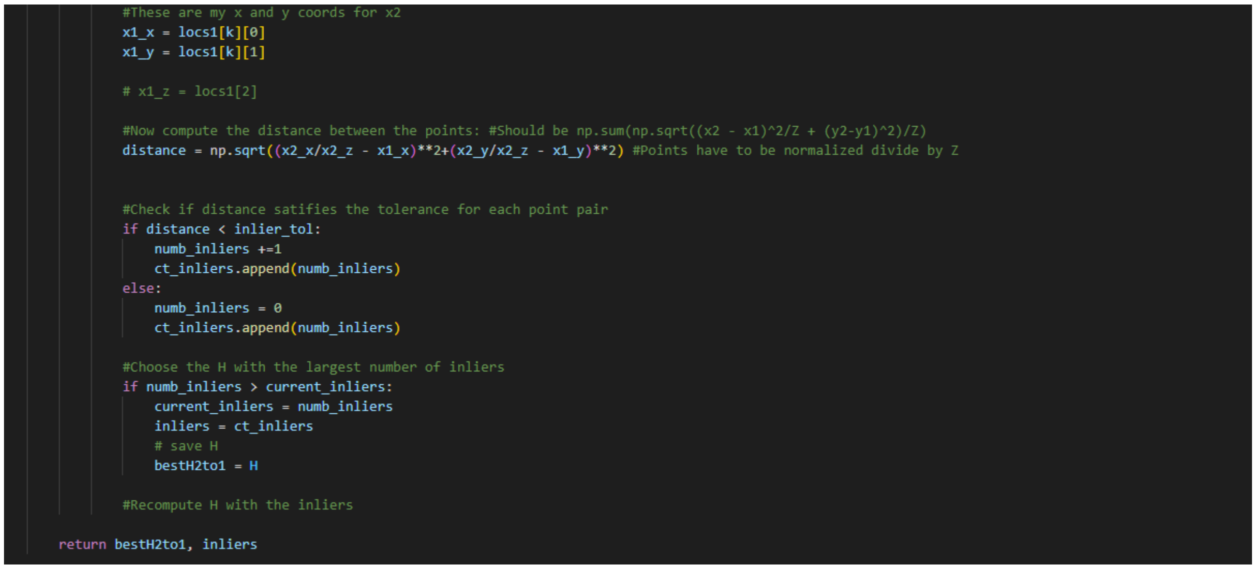


```
    #Compute homography of points after norm.
    h = computeH(x1_shift*x1_max,x2_shift*x2_max)

    H = np.dot(np.linalg.inv(T1),h)

    #Denormalization
    H2to1 = np.dot(H,T2)

    return H2to1
```

Q2.2.3

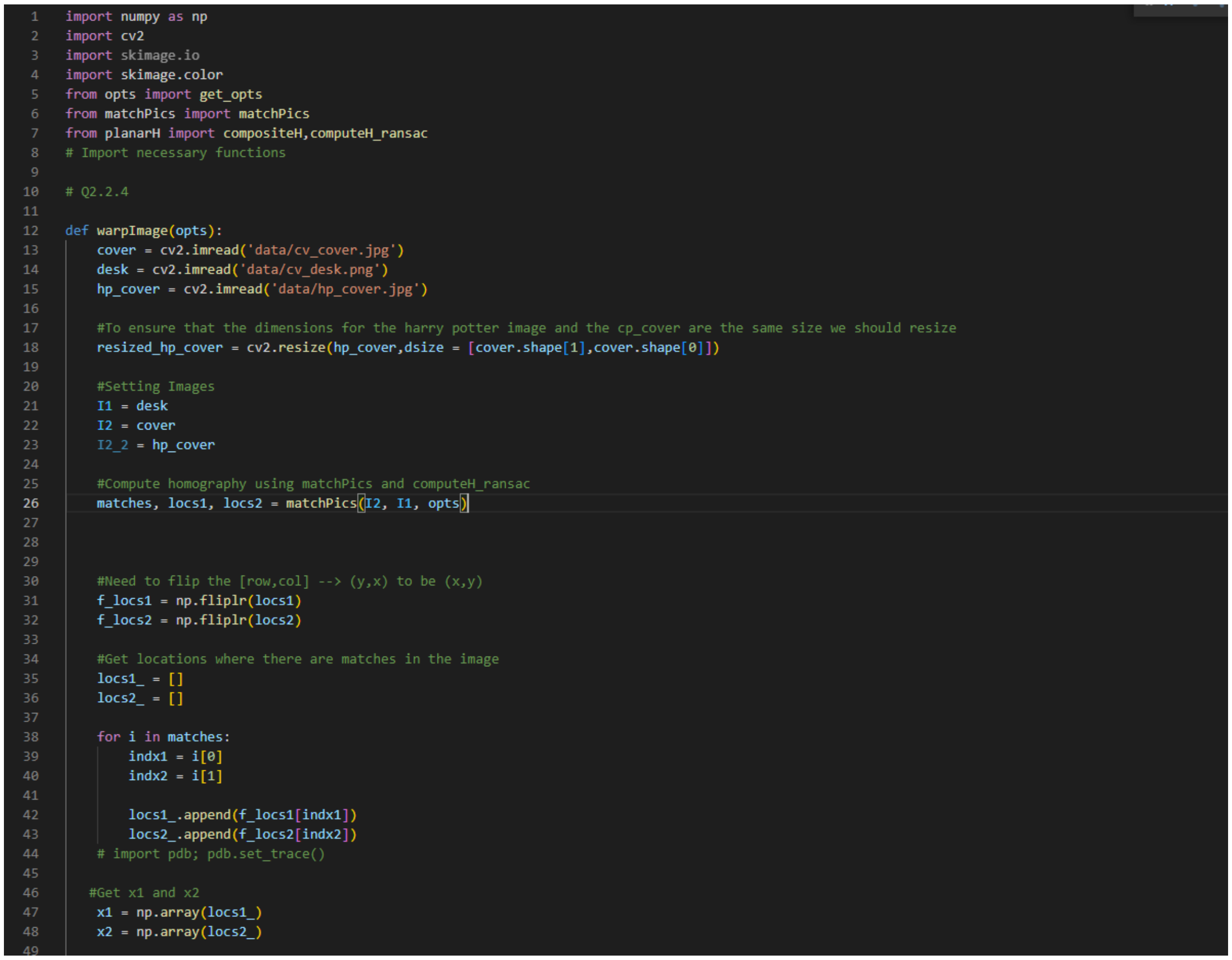


```python
def computeH_ransac(locs1, locs2, opts):
    #Q2.2.3
    #Compute the best fitting homography given a list of matching points
    max_iters = opts.max_iters  # the number of iterations to run RANSAC for
    inlier_tol = opts.inlier_tol # the tolerance value for considering a point to be an inlier

    #H will be a homography such that if x2 is a point in locs2 and x1 is a corresponding
    # point in locs1 then x1 = Hx2

    bestH2to1 = []
    # x_coord = locs1[:,0]
    # Use when computing distance bewtween points

    #Keep track of inliers on line
    current_inliers = 0
    inliers = []

    numb_inliers = 0
    ct_inliers = []
    p_prime = np.zeros((locs1.shape[0],3))


    for i in range(max_iters):
        #Select a random sample of 4 pairs of corresponding points

        indexes = np.random.choice(locs1.shape[0],4, False)
        x1 = locs1[indexes]
        x2 = locs2[indexes]

        #Compute H norm
        H = computeH_norm(x1,x2)
        numb_inliers = 0
        # print(i)

        #Calculate distance for each correspondence
        for k in range(locs1.shape[0]):
            # p_prime[0].insert(np.append(locs2[0],1))
            p_prime = np.append(locs2[k],1)
            error = np.dot(H,p_prime)

            #These are my x and y coords for x1
            x2_x = error[0]
            x2_y = error[1]
            x2_z = error[2]

            #These are my x and y coords for x2
            x1_x = locs1[k][0]
            x1_y = locs1[k][1]
```

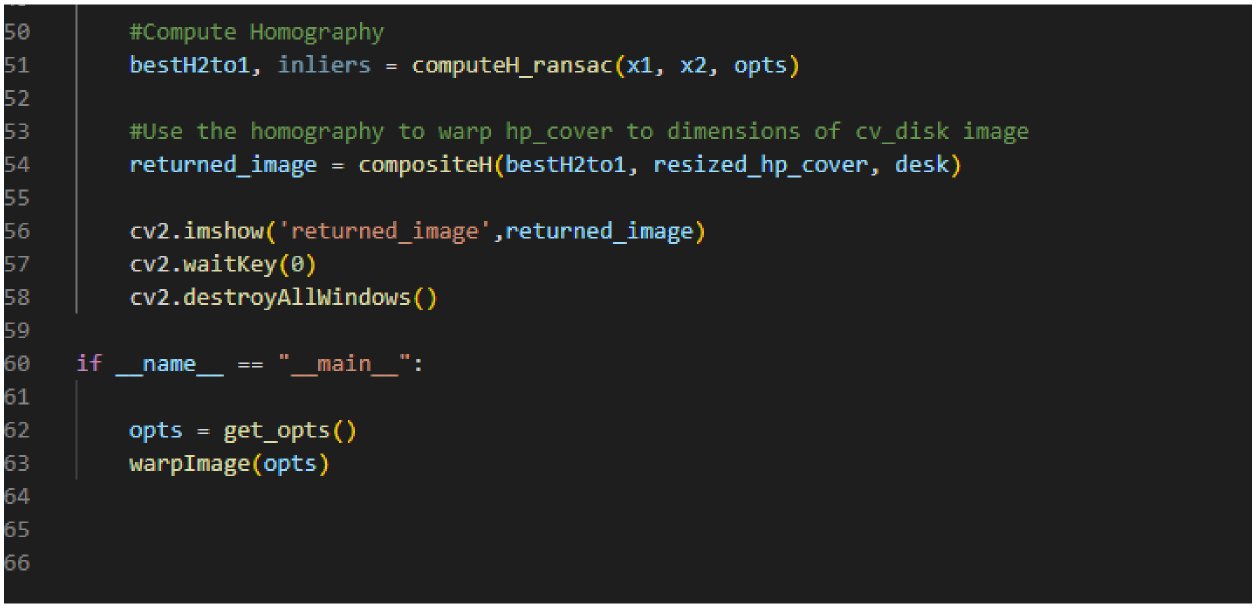


```python
            #These are my x and y coords for x2
            x1_x = locs1[k][0]
            x1_y = locs1[k][1]

            # x1_z = locs1[2]

            #Now compute the distance between the points: #Should be np.sum(np.sqrt((x2 - x1)^2/Z + (y2-y1)^2)/Z)
            distance = np.sqrt((x2_x/x2_z - x1_x)**2+(x2_y/x2_z - x1_y)**2) #Points have to be normalized divide by Z


            #Check if distance satifies the tolerance for each point pair
            if distance < inlier_tol:
                numb_inliers +=1
                ct_inliers.append(numb_inliers)
            else:
                numb_inliers = 0
                ct_inliers.append(numb_inliers)

            #Choose the H with the largest number of inliers
            if numb_inliers > current_inliers:
                current_inliers = numb_inliers
                inliers = ct_inliers
                # save H
                bestH2to1 = H

            #Recompute H with the inliers

    return bestH2to1, inliers
```

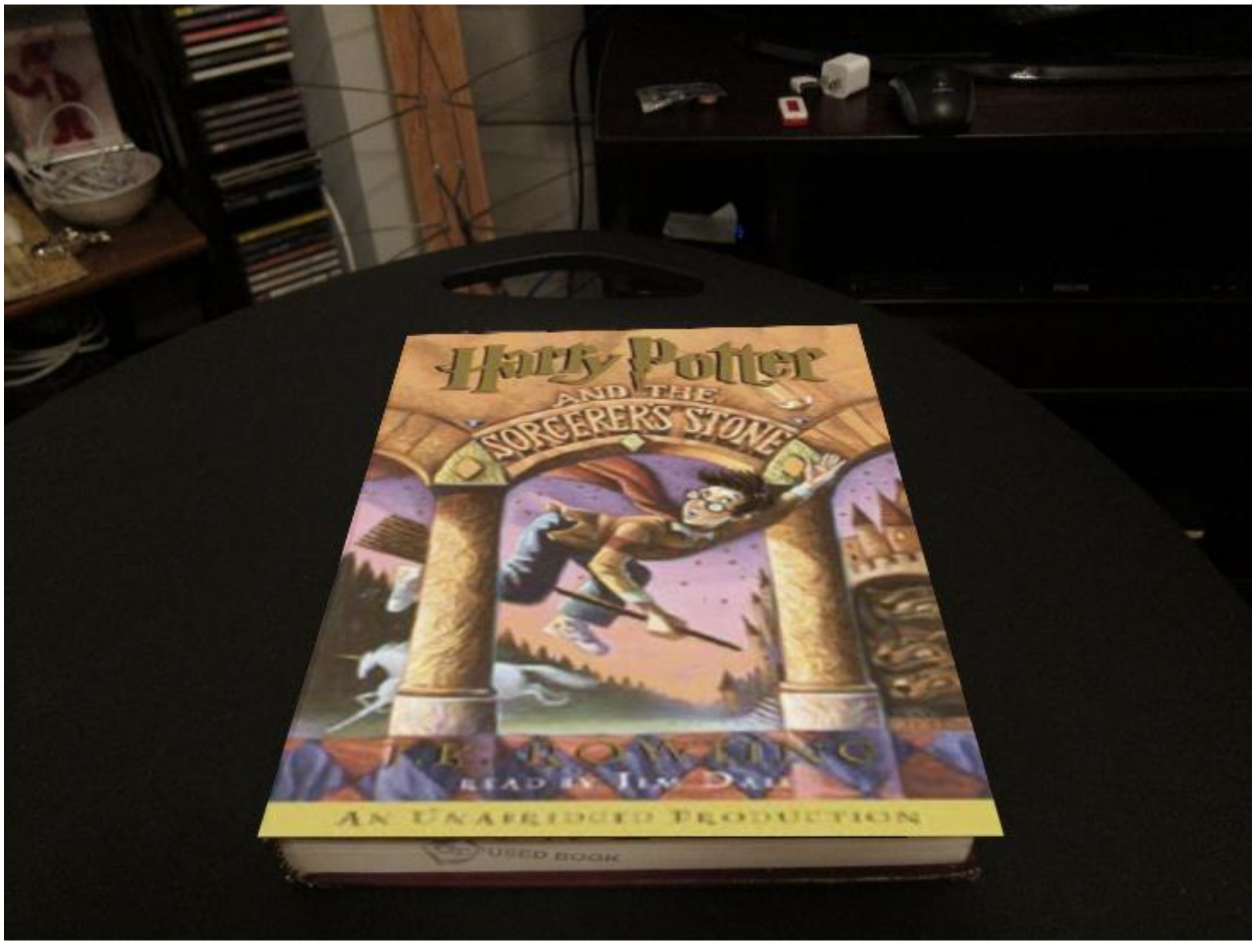
Q2.2.4



```python
import numpy as np
import cv2
import skimage.io
import skimage.color
from opts import get_opts
from matchPics import matchPics
from planarH import compositeH,computeH_ransac
# Import necessary functions

# Q2.2.4

def warpImage(opts):
    cover = cv2.imread('data/cv_cover.jpg')
    desk = cv2.imread('data/cv_desk.png')
    hp_cover = cv2.imread('data/hp_cover.jpg')

    #To ensure that the dimensions for the harry potter image and the cp_cover are the same size we should resize
    resized_hp_cover = cv2.resize(hp_cover,dsize = [cover.shape[1],cover.shape[0]])

    #Setting Images
    I1 = desk
    I2 = cover
    I2_2 = hp_cover

    #Compute homography using matchPics and computeH_ransac
    matches, locs1, locs2 = matchPics(I2, I1, opts)



    #Need to flip the [row,col] --> (y,x) to be (x,y)
    f_locs1 = np.fliplr(locs1)
    f_locs2 = np.fliplr(locs2)

    #Get locations where there are matches in the image
    locs1_ = []
    locs2_ = []

    for i in matches:
        indx1 = i[0]
        indx2 = i[1]

        locs1_.append(f_locs1[indx1])
        locs2_.append(f_locs2[indx2])
    # import pdb; pdb.set_trace()

   #Get x1 and x2
    x1 = np.array(locs1_)
    x2 = np.array(locs2_)
```



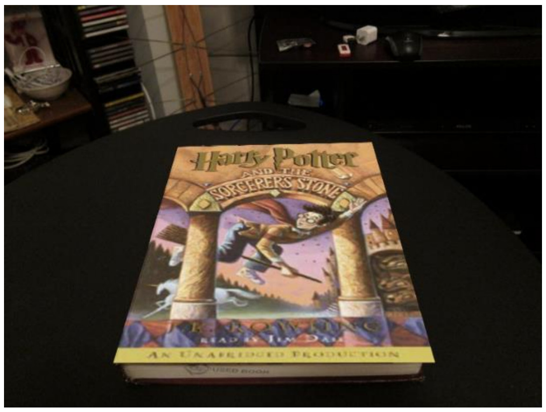
```python
    #Compute Homography
    bestH2to1, inliers = computeH_ransac(x1, x2, opts)

    #Use the homography to warp hp_cover to dimensions of cv_disk image
    returned_image = compositeH(bestH2to1, resized_hp_cover, desk)

    cv2.imshow('returned_image',returned_image)
    cv2.waitKey(0)
    cv2.destroyAllWindows()

if __name__ == "__main__":

    opts = get_opts()
    warpImage(opts)
```



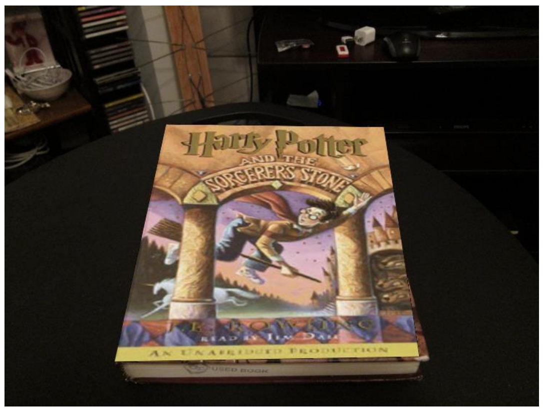
The harry potter was able to fit the cover of the book on the desk, without any problems. Though if this was an issue of the cover not filling the same space as the book, it would be largely due to the dimensions of the cv cover and harry potter cover not matching. To fix this issue, the harry potter cover would need to be resized to ensure that the dimensions matched with the cv cover before warping the harry potter cover onto the desk cover.

Q2.2.5

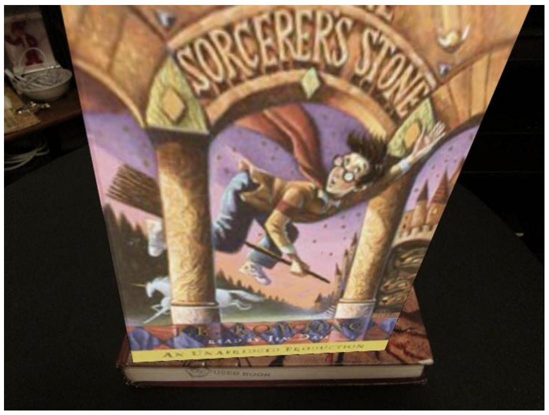
Max_Iters	Inlier_tol
700	2
500	8
1000	25
4000	2



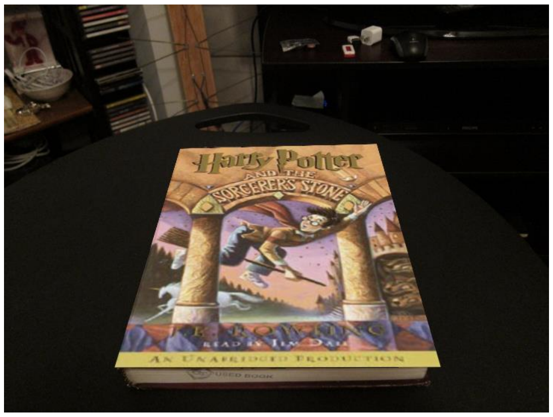
Max_Iters – 700, Inlier_tol - 2



Max_Iters – 500, Inlier_tol - 8



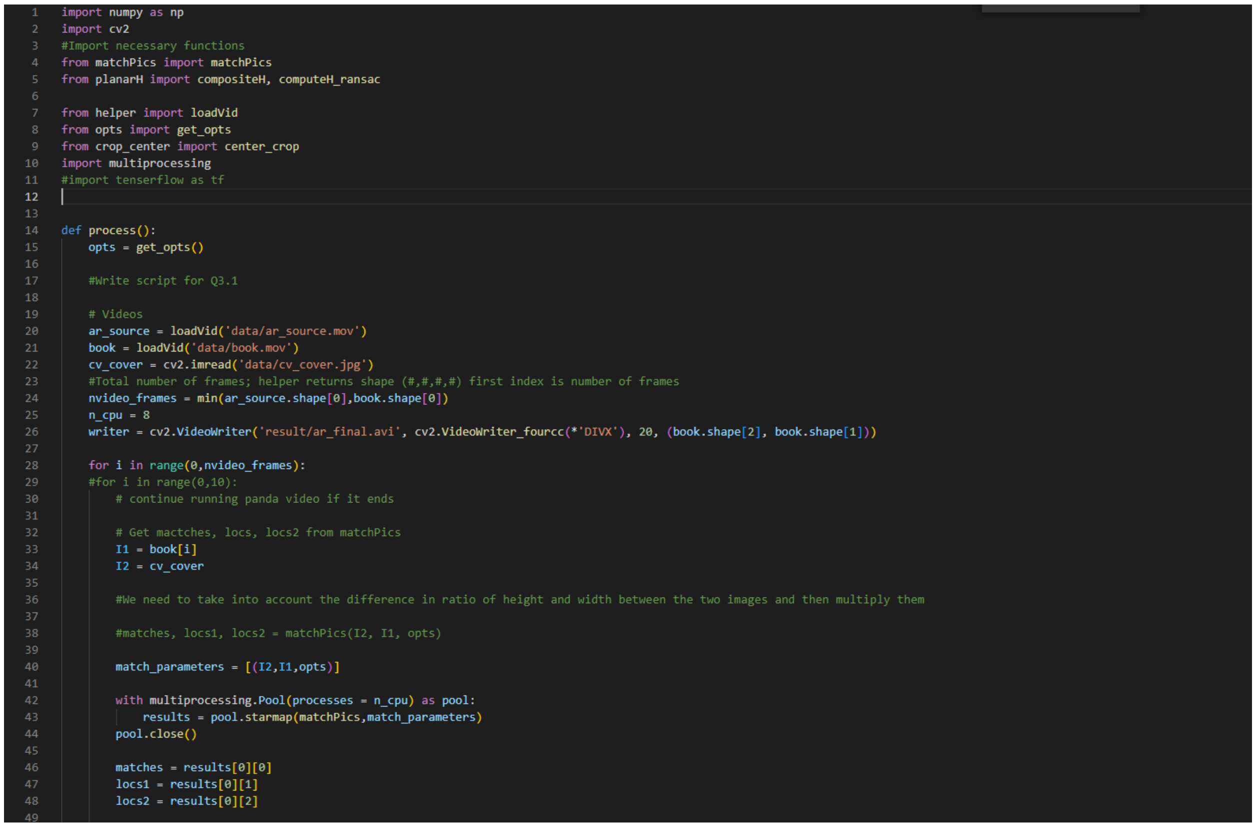
Max_Iters – 1000, Inlier_tol - 25



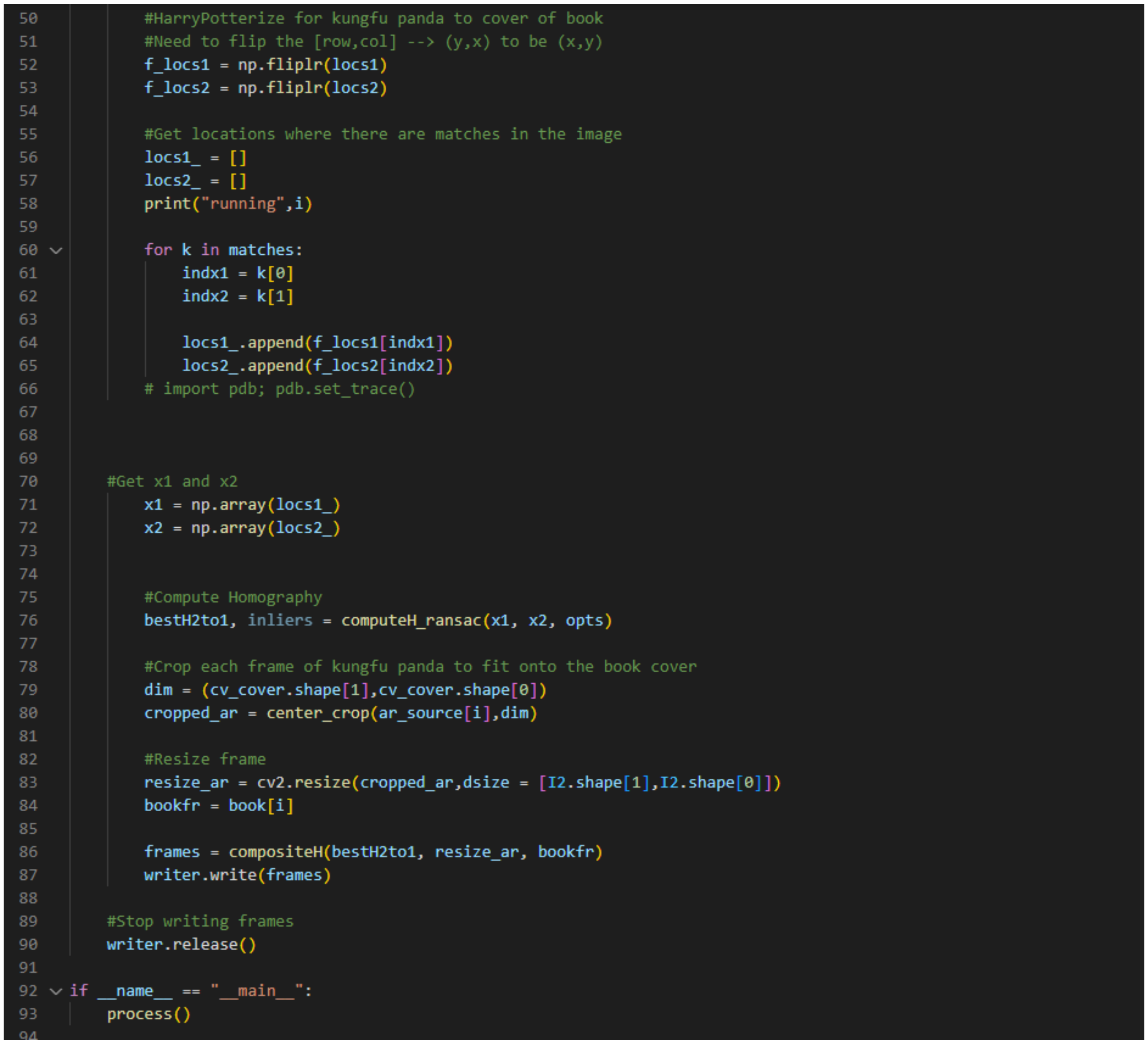
Max_Iters – 4000, Inlier_tol - 2

As seen in the images above when the maximum number of iterations is varied there does not seem to be much change in the image, placing the harry potter cover onto the desk and book cover, but when the inlier tolerance value is increased the warped harry potter cover (onto the book cover) begins to shift out of place even completely leaving the cover when the value was set to 25. This is because when the inlier tolerance is increased this also increases the chances of including outliers in the inliers list since the tolerance value is raised. Similarly, it can be inferred that larger iterations can also cause an increase in the number of outliers found (though this change is not as noticeable individually).

Q3.1

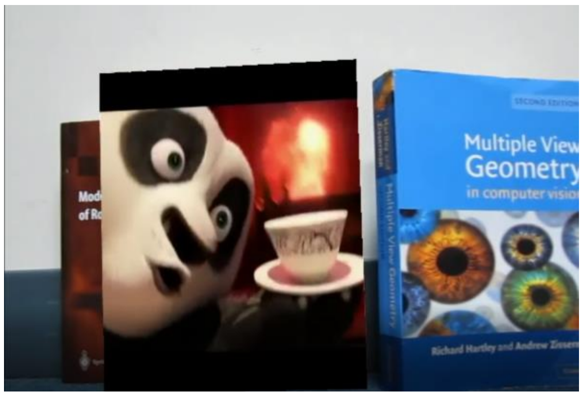


```python
import numpy as np
import cv2
#Import necessary functions
from matchPics import matchPics
from planarH import compositeH, computeH_ransac

from helper import loadVid
from opts import get_opts
from crop_center import center_crop
import multiprocessing
#import tenserflow as tf


def process():
    opts = get_opts()

    #Write script for Q3.1

    # Videos
    ar_source = loadVid('data/ar_source.mov')
    book = loadVid('data/book.mov')
    cv_cover = cv2.imread('data/cv_cover.jpg')
    #Total number of frames; helper returns shape (#,#,#,#) first index is number of frames
    nvideo_frames = min(ar_source.shape[0],book.shape[0])
    n_cpu = 8
    writer = cv2.VideoWriter('result/ar_final.avi', cv2.VideoWriter_fourcc(*'DIVX'), 20, (book.shape[2], book.shape[1]))

    for i in range(0,nvideo_frames):
    #for i in range(0,10):
        # continue running panda video if it ends

        # Get mactches, locs, locs2 from matchPics
        I1 = book[i]
        I2 = cv_cover

        #We need to take into account the difference in ratio of height and width between the two images and then multiply them

        #matches, locs1, locs2 = matchPics(I2, I1, opts)

        match_parameters = [(I2,I1,opts)]

        with multiprocessing.Pool(processes = n_cpu) as pool:
            results = pool.starmap(matchPics,match_parameters)
        pool.close()

        matches = results[0][0]
        locs1 = results[0][1]
        locs2 = results[0][2]

```

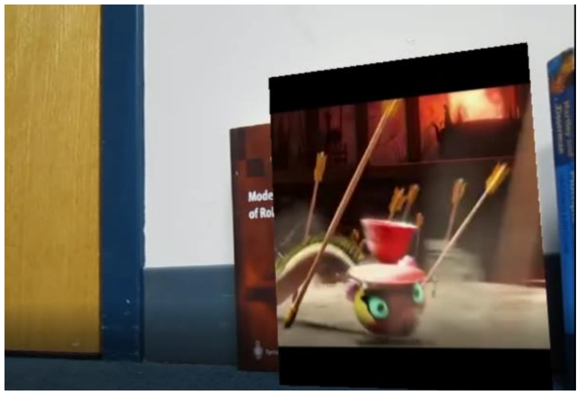


```python
        #HarryPotterize for kungfu panda to cover of book
        #Need to flip the [row,col] --> (y,x) to be (x,y)
        f_locs1 = np.fliplr(locs1)
        f_locs2 = np.fliplr(locs2)

        #Get locations where there are matches in the image
        locs1_ = []
        locs2_ = []
        print("running",i)

        for k in matches:
            indx1 = k[0]
            indx2 = k[1]

            locs1_.append(f_locs1[indx1])
            locs2_.append(f_locs2[indx2])
        # import pdb; pdb.set_trace()



    #Get x1 and x2
        x1 = np.array(locs1_)
        x2 = np.array(locs2_)


        #Compute Homography
        bestH2to1, inliers = computeH_ransac(x1, x2, opts)

        #Crop each frame of kungfu panda to fit onto the book cover
        dim = (cv_cover.shape[1],cv_cover.shape[0])
        cropped_ar = center_crop(ar_source[i],dim)

        #Resize frame
        resize_ar = cv2.resize(cropped_ar,dsize = [I2.shape[1],I2.shape[0]])
        bookfr = book[i]

        frames = compositeH(bestH2to1, resize_ar, bookfr)
        writer.write(frames)

    #Stop writing frames
    writer.release()

if __name__ == "__main__":
    process()
```

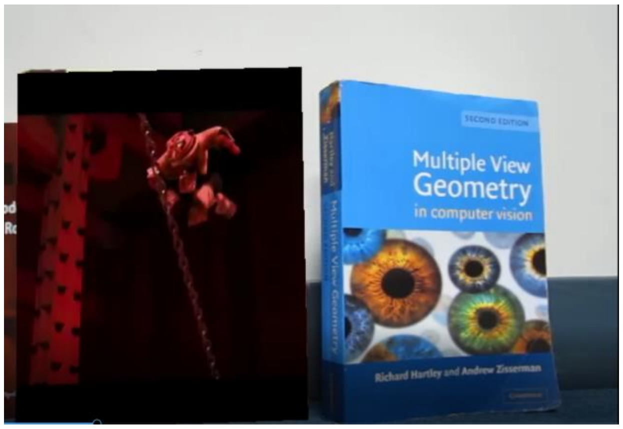
Center:


SECOND EDITION
Multiple View Geometry in computer vision
Richard Hartley and Andrew Zisserman

Left:

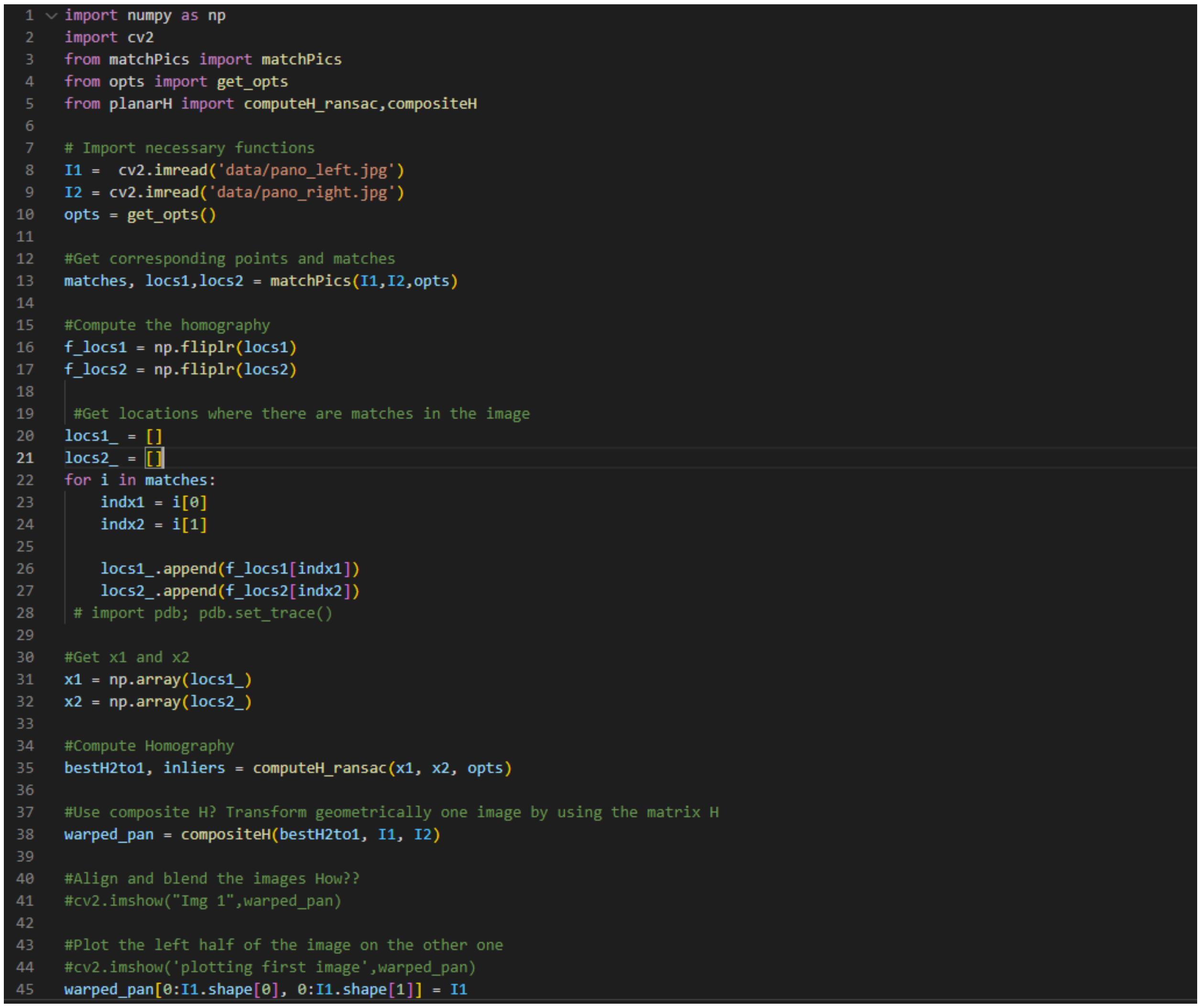


Right:

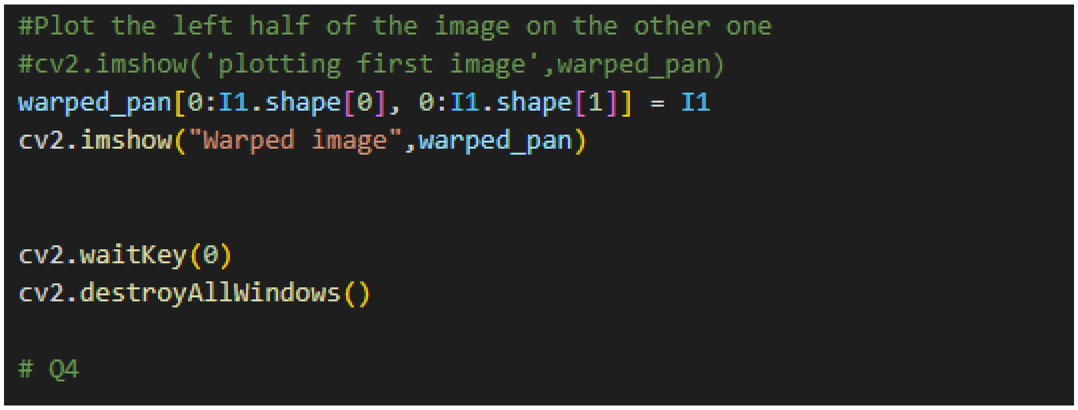

SECOND EDITION
Multiple View Geometry in computer vision
Richard Hartley and Andrew Zisserman

Video Link: Here

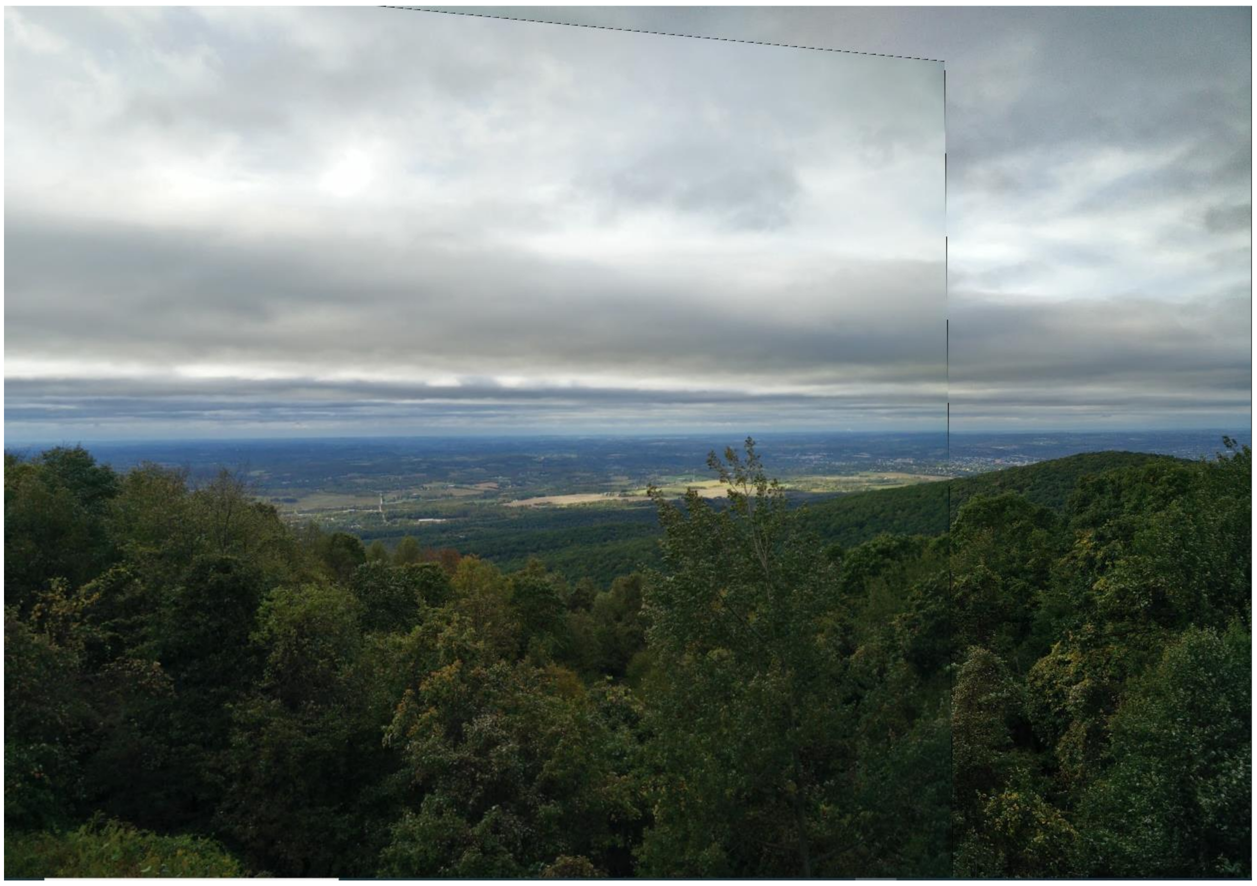
Q4.1



```
import numpy as np
import cv2
from matchPics import matchPics
from opts import get_opts
from planarH import computeH_ransac,compositeH

# Import necessary functions
I1 =  cv2.imread('data/pano_left.jpg')
I2 = cv2.imread('data/pano_right.jpg')
opts = get_opts()

#Get corresponding points and matches
matches, locs1,locs2 = matchPics(I1,I2,opts)

#Compute the homography
f_locs1 = np.fliplr(locs1)
f_locs2 = np.fliplr(locs2)

 #Get locations where there are matches in the image
locs1_ = []
locs2_ = []
for i in matches:
    indx1 = i[0]
    indx2 = i[1]

    locs1_.append(f_locs1[indx1])
    locs2_.append(f_locs2[indx2])
 # import pdb; pdb.set_trace()

#Get x1 and x2
x1 = np.array(locs1_)
x2 = np.array(locs2_)

#Compute Homography
bestH2to1, inliers = computeH_ransac(x1, x2, opts)

#Use composite H? Transform geometrically one image by using the matrix H
warped_pan = compositeH(bestH2to1, I1, I2)

#Align and blend the images How??
#cv2.imshow("Img 1",warped_pan)

#Plot the left half of the image on the other one
#cv2.imshow('plotting first image',warped_pan)
warped_pan[0:I1.shape[0], 0:I1.shape[1]] = I1
```



```
#Plot the left half of the image on the other one
#cv2.imshow('plotting first image',warped_pan)
warped_pan[0:I1.shape[0], 0:I1.shape[1]] = I1
cv2.imshow("Warped image",warped_pan)


cv2.waitKey(0)
cv2.destroyAllWindows()

# Q4
```



I am not sure what the issue could have been, besides maybe the images I was using being too close together and causing a weird effect with generating the panorama but I was unable to generate a panorama with my own images. Though I was able to get accurate results for the sample images that were used to help me start and run the .py file initially.