Problem 1.1

Proving existence of H in two image points of 2 cameras whose projection matrices are P1 and P2. X1 and X2 are points in homogenous coordinates.

$$X_1 \equiv H*X_2$$

Let's define a point in the homogenous coordinate,

$$O\ (X_i, Y_i, Z_i, 1)$$

Now,
The image of O in P1 and P2 will be

$$X_1 = P_1*O \quad \text{and} \quad X_2 = P_2*O$$

$$\therefore (P_1)^{-1}*X_1 = (P_2)^{-1}*X_2$$

$$\text{So, } X_1 = X_2*P_1*(P_2)^{-1}$$

Now,

$$P_1*(P_2)^{-1} = H$$

$$\therefore X \equiv H*X_2$$

$\therefore$ We can say that the equation is correct to a scaling factor -> $X_1 \equiv H*X_2$

Problem 1.2.1

Total degrees of freedom are the number of elements in the matrix, from which one unit is deducted to account for scaling factor.

$\therefore$ Degrees of freedom of **h = 8**

Problem 1.2.2

To solve h, we need a total of 8 points,

$\therefore$ **4 point pairs** will be required.

Problem 1.2.3

Deriving A_i

Given $X_1^i = H*X_2^i$ -------------------> ①

$$\text{Say, } X_1 = \begin{bmatrix} a \\ b \\ 1 \end{bmatrix} \text{ and } X_2 = \begin{bmatrix} c \\ d \\ 1 \end{bmatrix}$$

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

Using Scale factor λ, we can rewrite ① as,

$$\begin{bmatrix} a*\lambda \\ b*\lambda \\ \lambda \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} * \begin{bmatrix} c \\ d \\ 1 \end{bmatrix}$$

Let's, put λ = 1 and divide 1st and 2nd row with the 3rd.

We get,

$$-h_{11}*C-h_{12}*d-h_{13}+(h_{31}*c+h_{32}*d+h_{33})*a = 0$$

and

$$-h_{21}*C-h_{22}*d-h_{23}+(h_{31}*c+h_{32}*d+h_{33})*b = 0$$

The above 2 equations can be now expressed in the matrix form as

$$A_i*h=0$$

Where,

$$A = \begin{bmatrix} -c & -d & -1 & 0 & 0 & 0 & a*c & a*d & a \\ 0 & 0 & 0 & -c & -d & -1 & b*c & b*d & b \end{bmatrix}$$

And, h = $[h_{11} \quad h_{12} \quad h_{13} \quad h_{21} \quad h_{22} \quad h_{23} \quad h_{31} \quad h_{32} \quad h_{33}]^T$

Problem 1.2.4

Trivial Solution for h will be

$$H= [0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0 \quad 0]^T$$

Here, size of h is 9*1.
A is not a full rack matrix. Since only 4 points are required to calculate h, A is an 8*9 Matrix.

Since the 8 columns are linearly independent, 8 out of the 9 vectors will be linearly dependent as well, so one of the eigen values will be zero.

The eigen vector corresponding to this 0 eigen value will map to A*h = 0

Problem 1.4.1

We have

For 1st camera: X_1 = $k_1 * [I \quad 0] * X$

For 2nd camera: X_1 = $k_2 * [R \quad 0] * X$

Here,

I: Identity Matrix

R: Rotation Matrix

X is a point in the 3D Space

$$X = \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

So,

$$X_1 = k_1 * [I \quad 0] * X$$

$$X_1 = k_1 * [I \quad 0] * \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = k_1 * \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix}$$

$$\therefore X = \begin{bmatrix} k_1^{-1} * X_1 \\ 1 \end{bmatrix}$$

Similarly, on substituting tis value of X in the equation for 2nd Camera,

$$X_1 = k_2 * [R \quad 0] * X$$

$$X_1 = k_2 * [R \quad 0] * \begin{bmatrix} u \\ v \\ w \\ 1 \end{bmatrix} = k_2 * [R \quad 0] * \begin{bmatrix} k_1^{-1} * X_1 \\ 1 \end{bmatrix}$$

$$\therefore X_2 = k_1 * R * K_1^{-1} * X_1$$

$\therefore$ on comparing with $X_1 = H * X_2$

We get,

$$H = K_2 * R * k_1^{-1}$$
$$\text{On } H = K_1 * R^{-1} * K_2^{-1}$$

Therefore, there exists a homography H such that it satisfies

$$X_1 \equiv H * X_2$$

Problem 1.4.2

We know,

$$H=K*R*k^{-1}$$

And R = $\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

So, R^{-1} = $\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Now, $H^2 = K*R(\theta)*k^{-1} * K*R(\theta)*k^{-1}$

$$H^2 = K*\begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * k^{-1}$$

$$H^2 = K*\begin{bmatrix} \cos\theta^2 - \sin\theta^2 & -2 * \sin\theta * \cos\theta & 0 \\ 2 * \sin\theta * \cos\theta & \cos\theta^2 - \sin\theta^2 & 0 \\ 0 & 0 & 1 \end{bmatrix} * k^{-1}$$

We Know,

$$\cos\theta^2 - \sin\theta^2 = \cos 2\theta$$

And $2 * \sin\theta * \cos\theta = \sin 2\theta$

$$H^2 = k * \begin{bmatrix} \cos 2\theta & -\sin 2\theta & 0 \\ \sin 2\theta & \cos 2\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} * k^{-1}$$

$$\therefore H^2 = K*R(2\theta)*k^{-1}$$

H^2 is the homography corresponding to a rotation of $'2\theta'$.

Problem 1.4.3

Planar homography is not sufficient since its repeated pattern handling is inefficient. It works well only between
arbitrary image and the viewpoint. If the scene/image is planar, which is not how it is in actuality.

Also, between subregion of 2 images, there exists different homographies, corresponding to the viewpoint on subregion of same planar.

Problem 1.4.4

Consider a 3D line with Coordinates as $\begin{bmatrix}0\\0\\1\end{bmatrix}$, $\begin{bmatrix}1\\1\\1\end{bmatrix}$ and $\begin{bmatrix}4\\4\\1\end{bmatrix}$

Now,

$$\text{Perception matrix P} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix}$$

On multiplying P with line,

$$\text{X} = \begin{bmatrix}1 & 0 & 0 & 0\\0 & 1 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 1\end{bmatrix} * \begin{bmatrix}0 & 1 & 4\\0 & 1 & 4\\1 & 1 & 1\\1 & 1 & 1\end{bmatrix}$$

$$\text{X} = \begin{bmatrix}0 & 1 & 4\\0 & 1 & 4\\0 & 0 & 0\\1 & 1 & 1\end{bmatrix}$$

From the x value, we can see that line is projected to xy plane and is still preserved.

Problem 2.1.1

FAST detector and Harris corner detector are used to detect corners in a scene based on the change in intensity value. Fast detector samples a pixel and considers a 16 pixel circle around it, a threshold is defined on which change in intensity based on which pixel intensities out of four pixels on the axis are checked. The decision of the corner is made based on if the intensity of chosen pixel is above or below the threshold.

On the other hand, the **Harris Corner Detector** is a corner detection operator that uses a window around each pixel and uses sum of squared difference of the pixel values when the window is shifted by a small amount in any direction. It takes differential of the corner score into account with reference to direction directly and hence is more accurate in its detection.

Problem 2.1.2

The filter banks seen in the lectures requires a lot of computations to find binary strings whereas by utilizing less memory, faster matching and higher recognition rate, BRIEF Descriptor is an easy way to get binary descriptors. BRIEF is very fast both to build and to match. It does that by comparing intensities of the selected location pairs from the part of image with smooth patch.

Problem 2.1.3

Binary strings in BRIEF Descriptor that are used to match features can use Hamming distance as a metric for computing the match.

In Nearest Neighbor, make two sets. From the first image, pick N interest points and put them in first set. Then from ground truth data, deduce the corresponding points in the other, and put it in 2nd set. After computing the 2N associated descriptors, for each point in first set, use Nearest Neighbor to find the second one and call it a match.

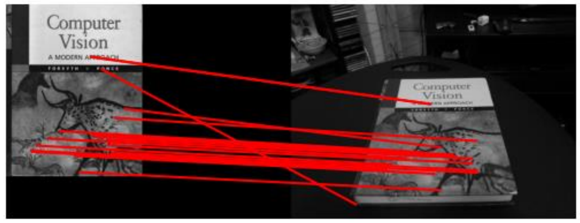
Hamming distance, as compared to Euclidean distance provides better speed-up in measuring distance because finding hamming distance is just applying XOR and bit count, which are very fast in modern CPUs with SSE instructions.

Problem 2.1.4

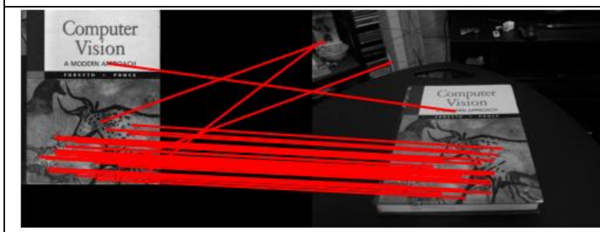
Code Snippet: matchPics

```python
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'

        # TODO: Convert Images to GrayScale
        I1 = cv2.cvtColor(I1, cv2.COLOR_BGR2GRAY)
        I2 = cv2.cvtColor(I2, cv2.COLOR_BGR2GRAY)

        # TODO: Detect Features in Both Images
        locs1 = corner_detection(I1, sigma)
        locs2 = corner_detection(I2, sigma)

        # TODO: Obtain descriptors for the computed feature locations
        desc1, locs1 = computeBrief(I1, locs1)
        desc2, locs2 = computeBrief(I2, locs2)

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

        return matches, locs1, locs2
```

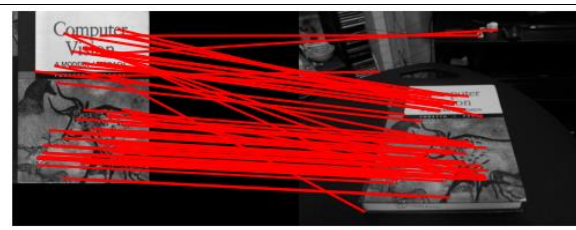
Problem 2.1.5



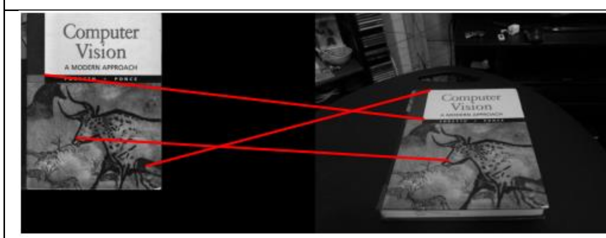
Initial Vanilla Parameters



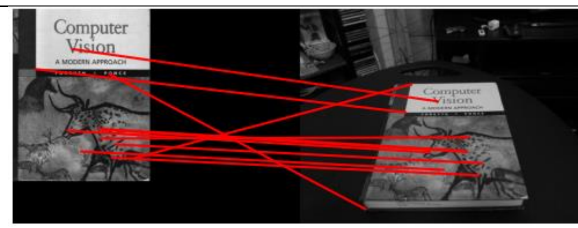
Sigma = 0.1 Ratio = 0.65



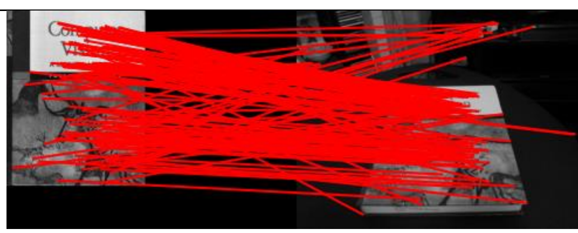
Sigma = 0.15 Ratio = 0.8



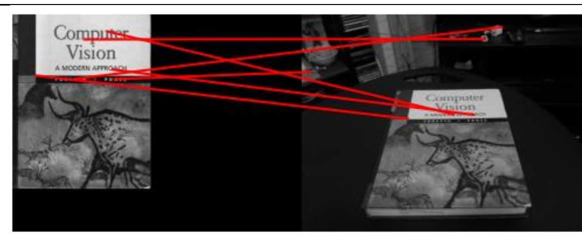
Sigma = 0.2 Ratio = 0.6



Sigma = 0.2 Ratio = 0.7



Sigma = 0.2 Ratio = 1



Sigma = 0.4 Ratio = 0.75

From the above figures it can be seen that at lower value of sigma there are more matches outside the book. Also, as the value ratio is lowered the number of matches reduces. And when the Ratio increases, even if sigma varies, that is increases or decreases, the no of matches between the images increases greatly. Another conclusion could be that ratio acts as a threshold of difference between the points, such that when ratio is high, points which are a mismatch are also considered as a match, but when the ratio is low, the only similar points are matched, which should be the case.

Problem 2.1.6

Code Snippet – briefRotTest

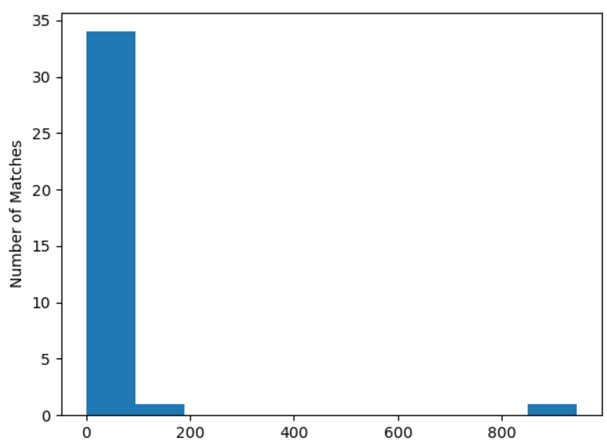
```
def rotTest(opts):

    #Read the image and convert to grayscale, if necessary
    opts = get_opts()
    image = cv2.imread('../data/cv_cover.jpg')
    hist_match = list()

    for i in range(36):

        #Rotate Image
        img_rotate = rotate(image, 10*(i+1))
        matches, locs1, locs2 = matchPics(image, img_rotate, opts)



        #Update histogram
        plotMatches(image, img_rotate, matches, locs1, locs2)
        hist_match.append(len(matches))


    print(hist_match)
    # pass


    #Display histogram
    plt.hist(hist_match)
    plt.ylabel("Number of Matches")
    plt.show()

if __name__ == "__main__":

    opts = get_opts()
    rotTest(opts)

```

Orientation: 90°

Orientation: 180°

Orientation: 220°

Orientation: 360°



From the histogram, it can be seen as the image is rotated the number of features mapped dips significantly. From this we can conclude that even if brief descriptor is fast in computation, it is unable to detect similar features therefore not good in feature matching.

We can also see that there occurs maximum amount of matching when the orientation is the same as the original image (at 360° and 0°) and very less in most other cases. The amount of matches we see in these angles overshadow the ones appearing in the other angles.

Problem 2.2.1

Code Snipped – computeH (planarH.py)

```python

def computeH(x1, x2):
    #Q2.2.1
    #Compute the homography between two sets of points

    hmm = []
    for i in range(x1.shape[0]):
        hmm.append([-x2[i, 0], -x2[i, 1], -1, 0, 0, 0, x1[i, 0]*x2[i, 0], x1[i, 0]*x2[i, 1], x1[i, 0]])
        hmm.append([0, 0, 0, -x2[i, 0], -x2[i, 1], -1, x1[i, 1]*x2[i, 0], x1[i, 1]*x2[i, 1], x1[i, 1]])

    u, s, v = np.linalg.svd(np.asarray(hmm))
    H2to1 = v[-1, :].reshape(3,3)
    return H2to1

```

Problem 2.2.2

Code Snippet – compute_norm (planarH.py)

```python

def computeH_norm(x1, x2):
    #Q2.2.2
    #Compute the centroid of the points
    x1_centre = [np.mean(x1[:,0]), np.mean(x1[:,1])]
    x2_centre = [np.mean(x2[:,0]), np.mean(x2[:,1])]

    #Shifting the origin of the points to the centroid
    p1, p2 = []
    for i in range(x1.shape[0]):
        p1[i] = np.sqrt((x1[i,0] - x1_centre[0])**2 + (x1[i,1] - x1_centre[1])**2)
        p2[i] = np.sqrt((x2[i,0] - x2_centre[0])**2 + (x2[i,1] - x2_centre[1])**2)

    x1_norm = np.sqrt(2)/(np.amax(p1))
    x2_norm = np.sqrt(2)/(np.amax(p2))


    mat1, mat2, mat3, mat4 = np.eye(3)
    for i in range(0, 2):
        mat1[i, i] = x1_norm
        mat2[i, i] = x2_norm

        mat3[i, 2] = -x1_centre[i]
        mat4[i, 2] = -x2_centre[i]

    #Similarity transform 1
    T1 = mat1@mat3

    #Similarity transform 2
    T2 = mat2@mat4

    #Compute homography
    x1_homography = np.vstack((x1.T, np.ones((x1.shape[0]))))
    x2_homography = np.vstack((x2.T, np.ones((x2.shape[0]))))

    xx = T1@x1_homography
    xx = xx/xx[2, :]
    xx = xx.T[:, 0:2]

    xy = T2@x2_homography
    xy = xy/xy[2, :]
    xy = xy.T[:, 0:2]

    H = computeH(xx, xy)

    #Denormalization
    H1 = np.dot(np.linalg.inv(T1), H)
    H2to1 = np.dot(H1, T2)

    return H2to1
```

Problem 2.2.3

Code Snippet – computeH_ransac (planarH.py)

```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
    inlier_tol = opts.inlier_tol # the tolerance value for considering a point to be an inlier
    max_inliers = -1
    m1 = np.hstack((locs1, np.ones((locs1.shape[0], 1))))
    m2 = np.hstack((locs2, np.ones((locs2.shape[0], 1))))


    for i in range(max_iters):
        index_r = np.random.randint(locs1.shape[0], size=4)
        p1 = locs1[index_r]
        p2 = locs2[index_r]

        H = computeH(p1, p2)

        m = np.matmul(H, m2.T)
        m = m.T

        d1 = np.expand_dims(m[:, -1], axis=1)
        d2 = ((m/d1) - m1)
        d2 = np.linalg.norm(d2, axis = 1)
        inlier = np.where(d2<inlier_tol, 1, 0)
        if(np.sum(inlier) > max_inliers):
            max_inliers = np.sum(inlier)
            inliers = inlier
        index_x = np.where(inliers == 1)
        bestH2to1 = computeH(locs1[index_x[0], :], locs2[index_x[0], :])

    return bestH2to1, inliers
```

Problem 2.2.4

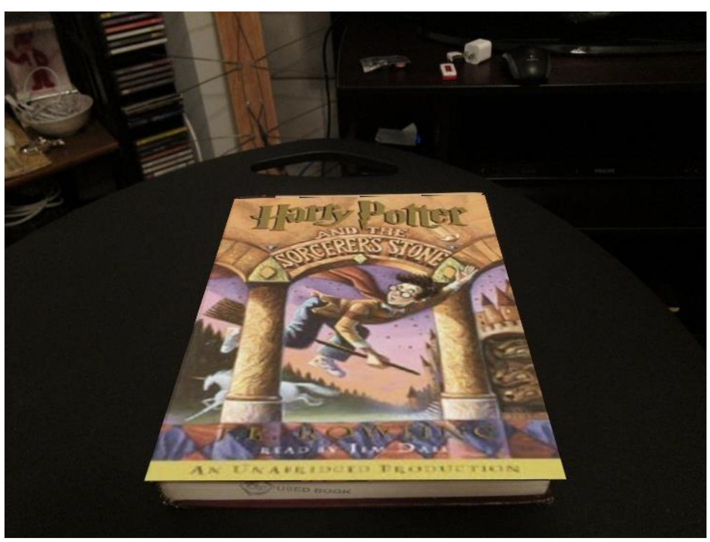
Code Snippet – composite (planarH.py)

```python
def compositeH(H2to1, template, img):

    mask=np.ones(template.shape)
    template=cv2.transpose(template)
    mask=cv2.transpose(mask)

    #Mask
    mask1 = cv2.warpPerspective(mask, np.linalg.inv(H2to1), (img.shape[0],  img.shape[1]))

    #Warp mask by appropriate homography
    warp_mask = cv2.transpose(mask1)

    #Warp template by appropriate homography
    warp_template = cv2.warpPerspective(template, np.linalg.inv(H2to1), (img.shape[1], img.shape[1]))  #check and

    #Use mask to combine
    template = cv2.transpose(warp_template)
    img[np.nonzero(warp_mask)] = template[np.nonzero(warp_mask)]
    composite_img = img

    return composite_img
```

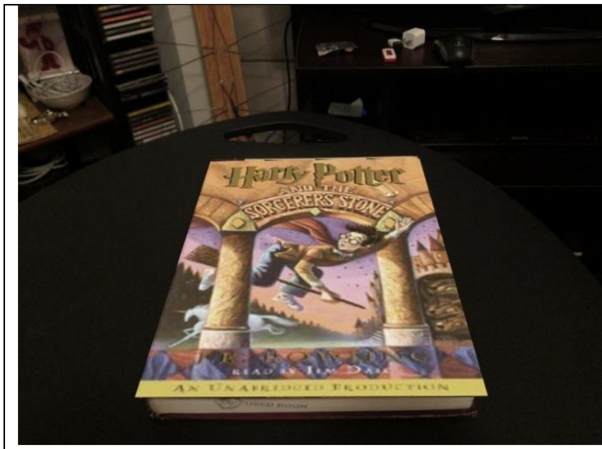
Code Snippet – HarryPotterize.py

```python
# Q2.2.4

cv_cover = cv2.imread('../data/cv_cover.jpg')
cv_desk = cv2.imread('../data/cv_desk.png')
hp_cover = cv2.imread('../data/hp_cover.jpg')

def warpImage(opts):
    matches,locs1,locs2 = matchPics(cv_cover, cv_desk, opts)

    a = (cv_cover.shape[1], cv_cover.shape[0])
    hp_cover_new = cv2.resize(hp_cover, a)

    # plotMatches(cv_desk, cv_cover,  matches, locs1, locs2)

    x1 = locs1[matches[:, 0], 0:2]
    x2 = locs2[matches[:, 1], 0:2]

    bestH2to1, inliers = computeH_ransac(x1, x2, opts)

    # hp_cover = cv2.resize(hp_cover, cv_cover.shape[1],cv_cover.shape[0])
    composite_image = compositeH(bestH2to1, hp_cover_new, cv_desk)
    plt.imshow(composite_image)
    plt.show()
    cv2.imwrite('../data/warped_image.jpeg', composite_image)
    # pass



if __name__ == "__main__":

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

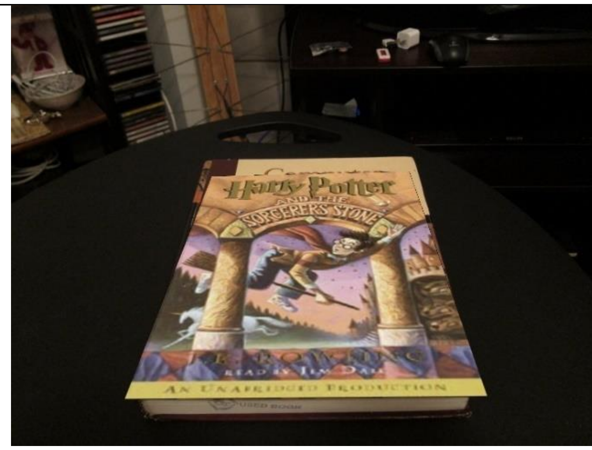
Problem 2.2.5



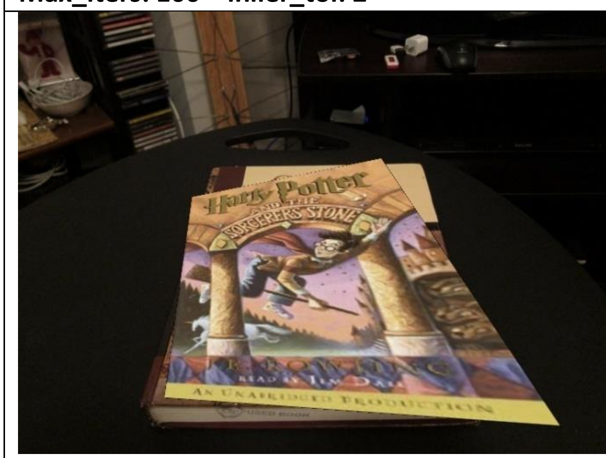
Initial image without parameter changes



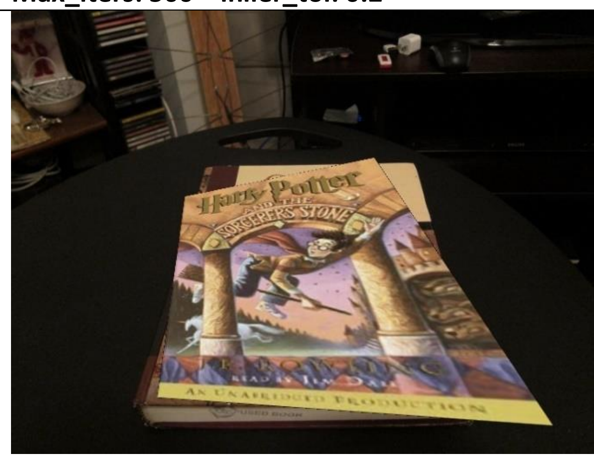
Max_iters: 100 inlier_tol: 2



Max_iters: 500 inlier_tol: 0.2



Max_iters: 2500 inlier_tol:40



Max_iters: 5000 inlier_tol: 20

Some conclusions that we can boil down are:

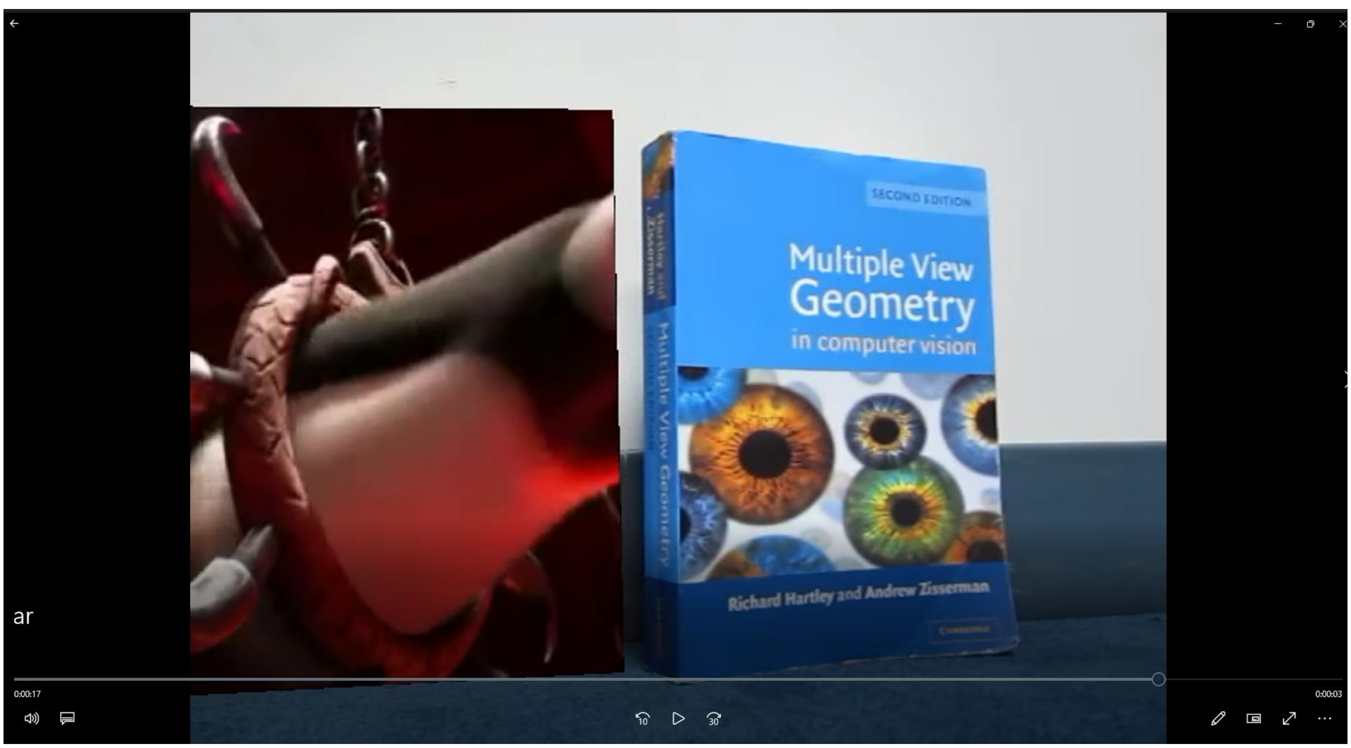
- As the number of iterations are increased the number inliers is not affected up to a certain value.
- If the tolerance is high, large number of points are used to calculate the homography and hence fewer no of iterations will be required for doing the same
- However, if the tolerance is very small, even a large no of iterations can't help in calculating the homography, there is no change in the no of inliers detected.
- On increasing the inlier tolerance tremendously, the number of inliers increases but the image gets distorted

Problem 3.1

Code Snippet

```
#Write script for Q3.1
opts = get_opts()

book = loadVid('../data/book.mov')
source = loadVid('../data/ar_source.mov')
cv_cover = cv2.imread('../data/cv_cover.jpg')


def video(cv_cover, frame, arr, opts):
    m, locs1, locs2 = matchPics(cv_cover, frame, opts)
    x1 = locs1[m[:, 0], 0:2]
    x2 = locs2[m[:, 1], 0:2]

    H2to1, inliers = computeH_ransac(x1, x2, opts)
    arr = arr[45:310,:,:]
    cover_width = cv_cover.shape[1]
    width = int(arr.shape[1]/arr.shape[0]) * cv_cover.shape[0]

    r_ar = cv2.resize(arr, (width,cv_cover.shape[0]), interpolation = cv2.INTER_AREA)
    h, w, d = r_ar.shape
    cropped_ar = r_ar[:, int(w/2) - int(cover_width/2) : int(w/2) +  int(cover_width/2), :]

    r = compositeH(H2to1, cropped_ar, frame)

    return r

a, b, c = book[1].shape
out = cv2.VideoWriter('arrr.avi', cv2.VideoWriter_fourcc('X', 'V', 'I', 'D'), 25, (b, a))

for i in range(source.shape[0]):
    frame = book[i]
    ar = source[i]
    print(i)
    final_vid = video(cv_cover, frame, ar, opts)
    out.write(final_vid)

cv2.destroyAllWindows()
out.release()
```

16720B – hw2
mrohitth



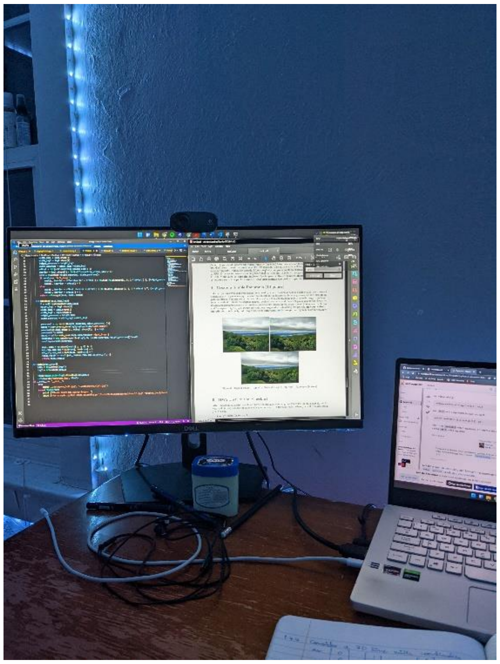
Problem 4

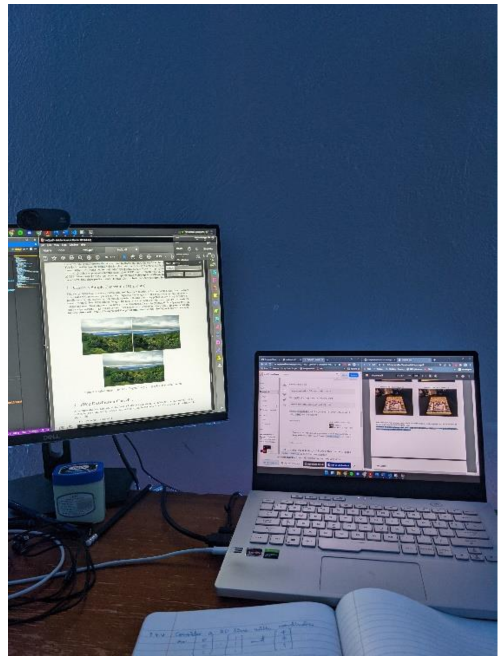
Code Snippet

```
def Panorama(im1, im2, H2to1):

    def blendmask(im):
        mask = np.ones((im.shape[0], im.shape[1]))
        mask[0,:] = 0
        mask[im.shape[0]-1,:] = 0
        mask[:, 0] = 0
        mask[:, im.shape[1]-1] = 0
        mask = distance_transform_edt( mask)
        mask = mask / np.max(mask)
        return mask

    h1, w1, _ = im1.shape
    h2, w2, _ = im2.shape
    lefttop = np.array([0,0,1]).T
    righttop = np.array([w2-1,0,1]).T
    rightbot = np.array([w2-1,h2-1,1]).T
    leftbot = np.array([0,h2-1,1]).T

    proj_lt = H2to1@lefttop
    proj_lt /=proj_lt[2]

    proj_rt = H2to1@righttop
    proj_rt /=proj_rt[2]

    proj_rb = H2to1@rightbot
    proj_rb /=proj_rb[2]

    proj_lb = H2to1@leftbot
    proj_lb /=proj_lb[2]


    tx=0
    ty=0

    tx = int(max((-proj_lt[0], -proj_lb[0], 0)))
    ty = int(max((-proj_lt[1], -proj_rt[1], 0)))

    W = max(proj_rb[0], proj_rt[0]).astype(int) + tx
    H = max(proj_lb[1], proj_rb[1]).astype(int) + ty

    M = np.array([[1, 0, tx], [0 , 1, ty], [0, 0, 1]]).astype(float)

    img_wr = cv2.warpPerspective(im2, M @ H2to1 , (W, H))
    img_wr = img_wr/255

    I = np.identity(3)
```

```python
    img_wl = cv2.warpPerspective(im1, M @ I , (W, H))
    img_wl = img_wl/255


    mask_r = blendmask(im2)
    wmask_r = cv2.warpPerspective(mask_r, M @ H2to1 , (W, H))
    mask_l = blendmask(im1)
    wmask_l = cv2.warpPerspective(mask_l, M @ I , (W, H))

    sum_mask = wmask_r + wmask_l


    with np.errstate(divide='ignore', invalid='ignore'):
        wmask_r = wmask_r / sum_mask
        wmask_r[np.isnan(wmask_r)] = 0
        wmask_l = wmask_l / sum_mask
        wmask_l[np.isnan(wmask_l)] = 0



    wmask_l = np.expand_dims(wmask_l, axis = 2)
    wmask_l = np.tile(wmask_l, (1,1,3))

    wmask_r = np.expand_dims(wmask_r, axis = 2)
    wmask_r = np.tile(wmask_r, (1,1,3))

    img_pano = img_wr * wmask_r + img_wl * wmask_l
    img_pano[np.isnan(img_pano)] = 0


    return img_pano

im1 = cv2.imread('../data/pano_left.jpg')
im2 = cv2.imread('../data/pano_right.jpg')

matches, locs1, locs2 = matchPics(im1, im2, opts)
x1 = locs1[matches[:, 0], 0:2]
x2 = locs2[matches[:, 1], 0:2]

bestH2to1, inliers = computeH_ransac(x1, x2, opts)

img_pano2 = Panorama(im1, im2, bestH2to1)

plt.figure(figsize = (20,10))

if(img_pano2 is not None):
    img_pano2 = cv2.cvtColor(np.float32(img_pano2), cv2.COLOR_BGR2RGB)
plt.imshow(img_pano2)
plt.show()
```

Initial Images





Panaroma