16-720-A Computer Vision

Sourojit Saha — sourojis
Collaboration with Arjun Sengupta and Sohan Kulkarni

Assignment 2

1.1

Let there exist a homography P_1 that projects the points form the world frame Π to first camera frame F_1. The points in the world frame are represented by x_π, and the points in the first camera plane are represented by x_1. Hence, the equation can be written as:

$$x_1 \equiv P_1 x_\pi \tag{1}$$

Similarly, the points in world frame Π can be represented in the second camera frame, F_2, whose points are represented by x_2, with a homography P_2 as:

$$x_2 \equiv P_2 x_\pi \tag{2}$$

Now, given the points in the camera frame F_2, we can find the points in the world frame Π, by pre-multiplying both sides of the equation by P_2^{-1}. This gives us:

$$P_2^{-1} x_2 \equiv P_2^{-1} P_2 x_\pi = x_\pi \tag{3}$$

Hence, combining the result of (3) with (1) and (2), we can write:

$$x_1 \equiv P_1 P_2^{-1} x_2 \tag{4}$$

Hence, we see there exists a homography, say H, that maps the points from the frame F_2 to frame F_1, given by:

$$H = P_1 P_2^{-1} \tag{5}$$

1.2 Correspondences

How many degrees of freedom does h have?

h is a 9×1 vector. However, the last element is a scale factor and can be changed to 1 (normalise), without affecting the homography. Hence, h has 8 Degrees-of-freedom.

How many point pairs are required to solve h?

One pair of points generates two equations. But, we have 8 unknowns, hence we need 4 pairs of points to solve for h.

Derive A_i.

Given that:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = w_1 \begin{bmatrix} \frac{x_1}{w_1} \\ \frac{y_1}{w_1} \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} \tag{6}$$

Writing the above equation as:

$$x_1 = \frac{h_{11}x_2 + h_{12}y_2 + h_{13}}{h_{31}x_2 + h_{32}y_2 + h_{33}} \tag{7}$$

$$y_1 = \frac{h_{21}x_2 + h_{22}y_2 + h_{23}}{h_{31}x_2 + h_{32}y_2 + h_{33}} \tag{8}$$

In the matrix form, this can we written as:

$$\begin{bmatrix} x_2 & y_2 & 1 & 0 & 0 & 0 & -x_1x_2 & -x_1y_2 & -x_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_1 & -y_1y_2 & -y_1 \end{bmatrix} \begin{bmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \\ h_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{9}$$

As $\boldsymbol{A_i h = 0}$, we can derive $\boldsymbol{A_i}$ as:

$$A_i = \begin{bmatrix} x_2 & y_2 & 1 & 0 & 0 & 0 & -x_1x_2 & -x_1y_2 & -x_1 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -x_2y_1 & -y_1y_2 & -y_1 \end{bmatrix} \tag{10}$$

Solution of $A_i h = 0$.

The above equatioon can be solved by the *Least Squares* method. The solution will be the eigen vector of $\boldsymbol{A_i}$ corresponding to the smallest eigen value of $\boldsymbol{A_i}$. $\boldsymbol{A_i}$ is full rank as we are implementing $\boldsymbol{SVD}$. $\boldsymbol{h}$ is the last column of $\boldsymbol{V}$

1.4

1.4.1 Homography under Rotation.

Given the transformation $\boldsymbol{K_1}$[I **0**] from the world frame to the camera frame of the *first* camera, we can find the inverse transformation from the *first* camera frame to the world frame, i.e. [I **0**]$^{-1}\boldsymbol{K_1^{-1}}$. After transforming to the world frame, we can use the given transformation from world frame to the camera frame of the *second* camera, i.e. $\boldsymbol{K_2}$[R **0**][I **0**]$^{-1}\boldsymbol{K_1^{-1}}$. Hence there exists a homography $\boldsymbol{H} = \boldsymbol{K_2}$[R **0**][I **0**]$^{-1}\boldsymbol{K_1^{-1}}$ from the *first* camera frame to the *second* camera frame.

1.4.2 Understanding Homographies under Rotation.

Given that the homography $\boldsymbol{H}$ maps the view of the camera from its *first* orientation to its view in *second* orientation that is rotated by θ. Hence the view in the *second* orientation is given by:

$$x_{second\ orientation} = Hx_{original\ orientation} \tag{11}$$

Now, suppose, the camera is rotated again by θ, in the same sense as before, i.e. it is rotated by 2θ from the original orientation. The view in this new orientation will be given by:

$$x_{new\ orientation} = Hx_{second\ orientation} = H(Hx_{original\ orientation}) = H^2x_{original\ orientation} \tag{12}$$

Hence, we see that a homography of $\boldsymbol{H^2}$ corresponds to rotation of 2θ.

1.4.3 Limitations of Planar Homography:

The Planar Homography is not applicable when:

- The motion of the camera is not pure rotation.
- The points do not lie in a plane

Any one condition should be satisfied.

1.4.4 Behaviour of Lines Under Perspective Projections.

Coordinates in 3D are represented as $[X, Y, Z]^T$ and as $[x, y]^T$ in 2D. In homogeneous system, the points is represented as $[x, y, 1]^T$. The equation of line in 3D is given by:

$$a \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = 0 \tag{13}$$

In 2D system, it can be written as:

$$a * Z \begin{bmatrix} \frac{X}{Z} \\ \frac{Y}{Z} \\ 1 \end{bmatrix} = 0 \tag{14}$$

We see that the above equation can be written in the following form:

$$a' \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = 0 \tag{15}$$

Which is the equation of line in 2D. Therefore, lines in 3D remain lines in 2D.

2.1Feature Detection and Matching

2.1.1 FAST Detector:

Instead if calculating the whole window, as in Harris detector, FAST detector works by calculating the specific pixels. This speeds up the process of feature detection.

2.1.2 BRIEF Descriptor:

Filter banks computes the whole sliding window.This uses more computation. BRIEF instead compares the pixel intensities of a specific set of pixels within a patch. A single filter cannot be used as a good descriptor.

2.1.3 Matching Methods:

BRIEF Descriptor gives us a sequence of 1s and 0s. This can be used to compute the distance between two points of interest.

- **Hamming distance** implements XOR to compute the distance. XOR is faster to process than conventional distance metrics such as Euclidean distance.
- **Nearest Neighbor** can be used to match points. The metric used can be Hamming Distance. In such a case, lower the Hamming distance, more similar are the points.

2.1.4 Feature Matching

```
import numpy as np
import cv2
import skimage.color
from helper import briefMatch
from helper import computeBrief
from helper import corner_detection
from helper import plotMatches
from opts import get_opts
import os
```

```python

def matchPics(I1, I2, opts):

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

    matches, locs1, locs2 = None, None, None

    img1 = I1
    img2 = I2
    # TODO: Convert Images to GrayScale
    img1_grey = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
    img2_grey = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
    # TODO: Detect Features in Both Images
    locs1 = corner_detection(img1_grey, sigma)
    locs2 = corner_detection(img2_grey, sigma)
    # TODO: Obtain descriptors for the computed feature locations
    desc1, loc1 = computeBrief(img1_grey, locs1)
    desc2, loc2 = computeBrief(img2_grey, locs2)
    # TODO: Match features using the descriptors
    matches = briefMatch(desc1, desc2, ratio)

    return matches, loc1, loc2
```

2.1.5 Feature Matching and Tuning

On keeping the ratio constant (0.7 here), and increasing the sigma, the number of matched points decrease. This is because the sigma acts as threshold value. As the value increases, less points are considered as interest points.
On keeping the sigma constant (0.15 here), and increasing the ratio, the number of matched points increase, as the points are considered more similar and hence matched.
(Fig 1-6)

2.1.6 BRIEF and Rotation

BRIEF is not rotation invariant. This is because it considers a specific sequence/set of pixels in a patch. When the image is rotated, the position of the pixels change, which cause the

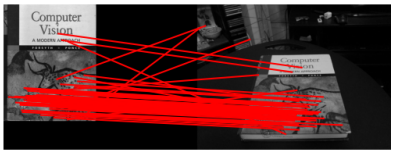

Figure 1: sigma = 0.1, ratio = 0.7, matches = 60

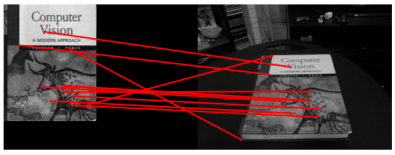

Figure 2: sigma = 0.2, ratio = 0.7, matches = 11

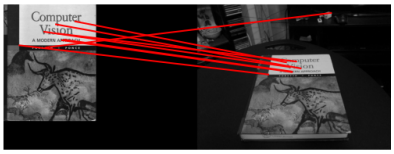

Figure 3: sigma = 0.3, ratio = 0.7, matches = 8

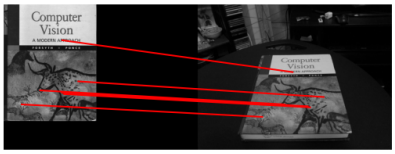

Figure 4: sigma = 0.15, ratio = 0.6, matches = 6

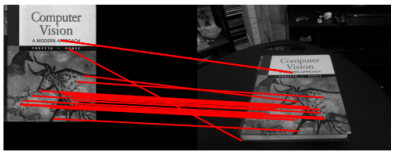

Figure 5: sigma = 0.15, ratio = 0.7, matches = 18

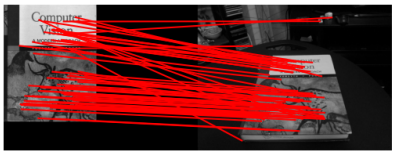

Figure 6: sigma = 0.15, ratio = 0.8, matches = 61

Figure 7: Matches when images are rotated by 0°.

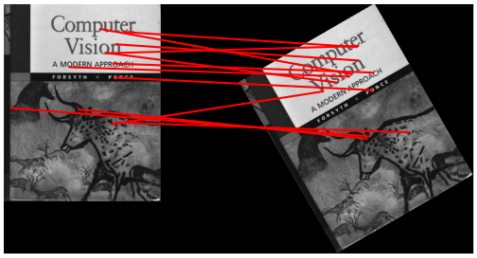

Figure 8: Matches when images are rotated by 30°.

algorithm to consider a different set of pixels when trying to match points. This causes the number of matches to decrease as the image is rotated.
(Fig 7-10)

```
import matplotlib
import numpy as np
import cv2
from matchPics import matchPics
from opts import get_opts
import os
from scipy import ndimage, misc
import helper
import matplotlib.pyplot as plt

def rotTest(opts):
```

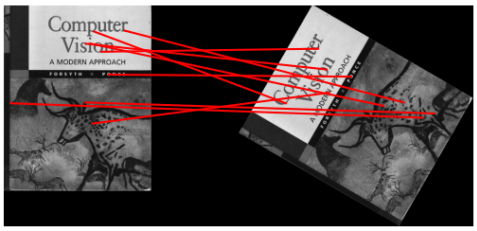

Figure 9: Matches when images are rotated by 60°.

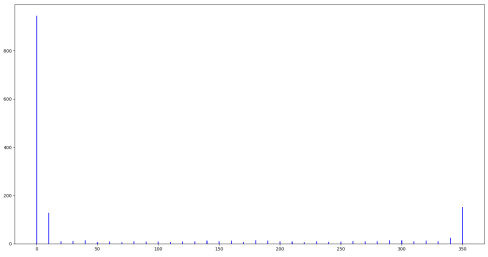

Figure 10: Histogram showing the variation of number of matches(y axis) with rotation angle (x axis). Note: Rotation angle of 360° *coincides with* 0°.

```
    path = os.path.join("../data/","cv_cover.jpg")
    img = cv2.imread(path)

    opts = get_opts()

    range_i = 36
    hist = np.zeros(range_i)
    for i in range(range_i):#36

        print(i)
        angle = i*10
        img_rot = ndimage.rotate(img, angle, reshape=True)

        matches, locs1, locs2 = matchPics(img, img_rot, opts)
        hist[i] = len(matches)

    plt.bar(np.arange(range_i)*10, hist, color='blue', alpha=0.75)
```

```
    plt.show()

if __name__ == "__main__":
    opts = get_opts()
    rotTest(opts)
```

2.2 Homography Computation

2.2.1 Computing the Homography

```
def computeH(x1, x2):

    rows = x1.shape[0]
    a = np.zeros((2*rows, 9))
    for i in range(rows):
        row1 = np.asarray([-x2[i,0], -x2[i,1], -1, 0, 0, 0, x2[i,0]*x1[i
    ,0], x2[i,1]*x1[i,0], x1[i,0]])
        row2 = np.asarray([0, 0, 0, -x2[i,0], -x2[i,1], -1, x2[i,0]*x1[i
    ,1], x2[i,1]*x1[i,1], x1[i,1]])

        a[2*i] = row1
        a[(2*i)+1] = row2

    u,s,vt = np.linalg.svd(a)

    eig_val = s[-1]
    eig_vect = vt[-1,:]
    H2to1 = eig_vect.reshape(3,3)
    H2to1 = H2to1/H2to1[-1,-1]

    return H2to1
```

2.2.2 Homography Normalization

```
def computeH_norm(x1, x2):
    mean1 = np.mean(x1, axis=0)
    mean2 = np.mean(x2, axis=0)

```

```
    x1_trans = x1-mean1
    x2_trans = x2-mean2

    x1_trans_dist = ((x1_trans[:,0]**2) + (x1_trans[:,1])**2)**0.5
    x2_trans_dist = ((x2_trans[:,0]**2) + (x2_trans[:,1])**2)**0.5

    scale1 = (2**0.5)/np.max(x1_trans_dist)
    scale2 = (2**0.5)/np.max(x2_trans_dist)

    t1 = np.array([[scale1, 0, 0],
                   [0, scale1, 0],
                   [0, 0, 1]]) @ np.array([[1, 0, -mean1[0]],
                                           [0, 1, -mean1[1]],
                                           [0, 0, 1]])


    x1_homo = np.hstack((x1, np.ones((x1.shape[0], 1))))
    x1t = t1@x1_homo.T

    t2 = np.array([[scale2, 0, 0],
                [0, scale2, 0],
                [0, 0, 1]]) @ np.array([[1, 0, -mean2[0]],
                                        [0, 1, -mean2[1]],
                                        [0, 0, 1]])

    x2_homo = np.hstack((x2, np.ones((x2.shape[0], 1))))
    x2t = t2@x2_homo.T

    H2to1 = computeH(x1t.T, x2t.T)

    H2to1 = np.linalg.inv(t1)@H2to1@t2

    return H2to1
```

2.2.3 Implement RANSAC

```
def computeH_ransac(locs1, locs2, opts):

    max_iters = opts.max_iters  # the number of iterations to run RANSAC
   for
    rows = locs1.shape[0]
    if rows<4:
```

```
        print("Less than 4 points.")
        return -1, -1
    else:
        score_prev = 0
        bestH2to1 = np.zeros((3,3))
        for i in range(max_iters):
            idx = np.random.choice(range(rows), 4, replace=False)
            x1 = locs1[idx,:]
            x2 = locs2[idx,:]

            H = computeH_norm(x1,x2)

            idx_subset = np.setdiff1d(range(rows), idx)

            locs1_subset = locs1[idx_subset,:]
            locs2_subset = locs2[idx_subset,:]

            locs2_subset_homo = np.hstack((locs2_subset, np.ones((
    locs2_subset.shape[0], 1))))

            locs1_pred = H@locs2_subset_homo.T
            locs1_pred = locs1_pred.T
            locs1_pred[:,0] = locs1_pred[:,0]/locs1_pred[:,2]
            locs1_pred[:,1] = locs1_pred[:,1]/locs1_pred[:,2]
            locs1_pred = locs1_pred[:,0:2]

            diff = locs1_pred-locs1_subset
            dist = (diff[:,0]**2 + diff[:,1]**2)**0.5

            inlier_tol = opts.inlier_tol
            inliers= np.zeros(diff.shape[0])
            for m in range(len(inliers)):
                if dist[m]<inlier_tol:
                    inliers[m] = 1

            score = np.sum(inliers)
            if score > score_prev:
                score_prev = score
                bestH2to1 = H

    return bestH2to1, inliers
```

2.2.4 Automated Homography Estimation and Warping

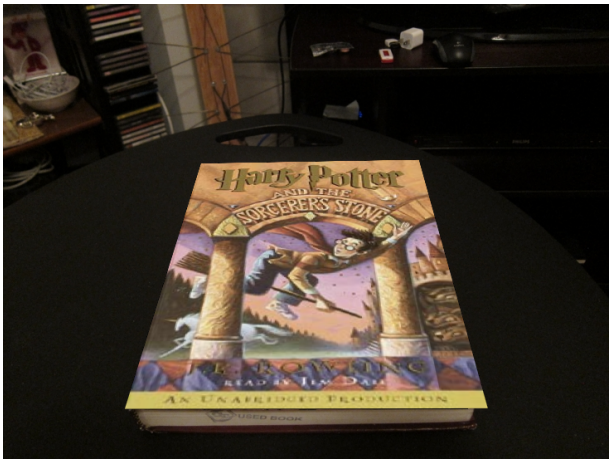

Figure 11: Warped image of Harry Potter book cover on CV book cover.

Increasing the ratio from 0.7 to 0.75 helped cover the book completely in this case.(Fig 11)

```
from configparser import Interpolation
import numpy as np
import cv2
from planarH import compute_x1x2
from planarH import generate_x1_x2
import skimage.io
import skimage.color
from matchPics import matchPics
from planarH import computeH_ransac
from opts import get_opts
from planarH import compositeH

def warpImage(opts):
    img1 = cv2.imread('../data/cv_desk.png')
    img2 = cv2.imread('../data/cv_cover.jpg')
    img3 = cv2.imread('../data/hp_cover.jpg')
    img3 = cv2.resize(img3, (img2.shape[1], img2.shape[0]), interpolation=
   cv2.INTER_AREA)
    matches, loc1, loc2 = matchPics(img1, img2, opts)
    x1, x2 = generate_x1_x2(loc1, loc2, matches)
    H, inliers = computeH_ransac(x1, x2, opts)
    print('H: ',H)
    print('Inliers: ', inliers)
    img_out = cv2.warpPerspective(img3, H, (img1.shape[1], img1.shape[0]))
    composite_img = compositeH(H, img_out, img1)
```

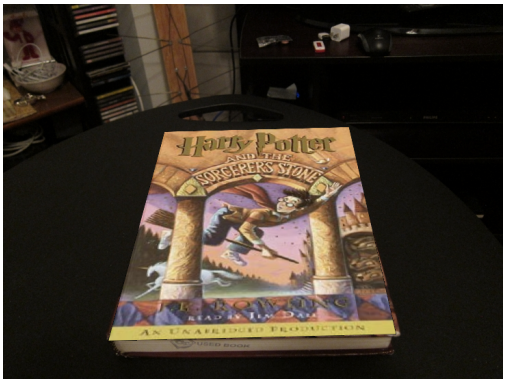

Figure 12: Iteration = 500, Tolerance = 10

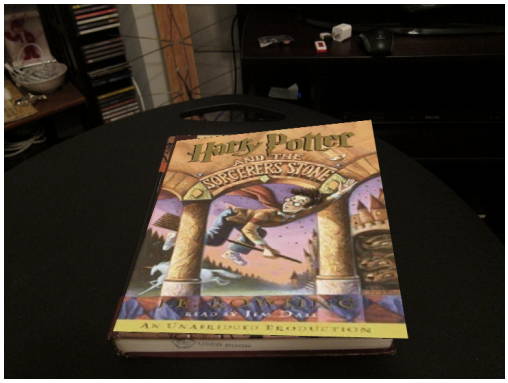

Figure 13: Iteration = 500, Tolerance = 15

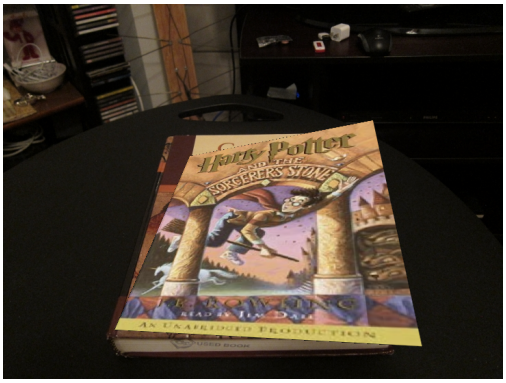

Figure 14: Iteration = 500, Tolerance = 20

```
25      cv2.imshow("img_out",composite_img)
26      cv2.waitKey(0)
27      return
28  if __name__ == "__main__":
29      opts = get_opts()
30      warpImage(opts)
```

2.2.5 RANSAC Parameter Tuning

When the iteration is kept constant (500 here), and the tolerance in increased, we see that the warp becomes more imperfect. This is because, more points that are far away, are considered as good fit and included.

When the tolerance is kept constant (20 here), and the iterations are increased, the warp becomes more perfect, this is because with increasing iterations, the RANSAC algorithms

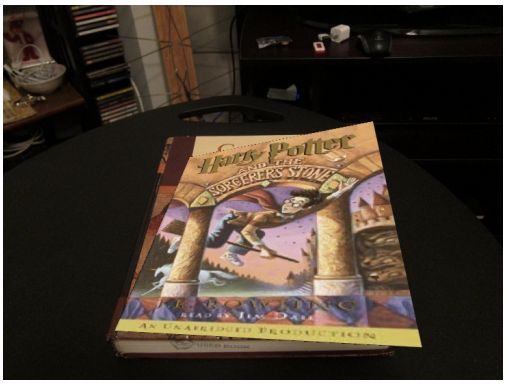

Figure 15: Iteration = 10, Tolerance = 20

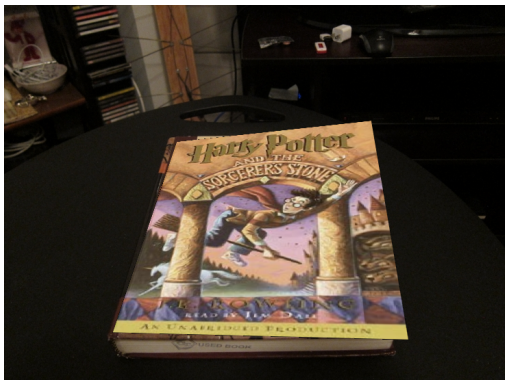

Figure 16: Iteration = 500, Tolerance = 20

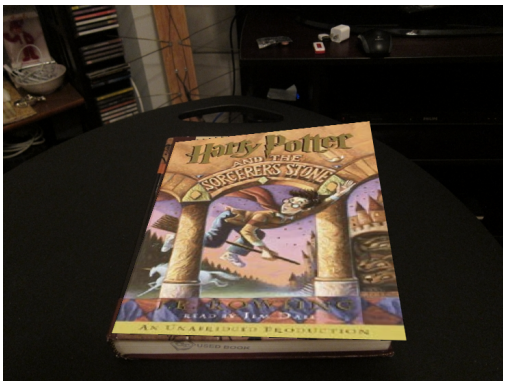

Figure 17: Iteration = 1000, Tolerance = 20

can search are bigger sample space.(Fig 12-17)

3 Augmented Reality Application

3.1 Incorporating AR Video

(Fig 18-20)

Figure 18: Frame 5 of AR video.

Figure 19: Frame 103 of AR video.

Figure 20: Frame 316 of AR video.

4 Extra Credits

4.1 Realtime AR

Finding matching points between two images is time consuming. The time increase as the size of the image increases. This limits the application to real-time AR applications. However, the speed can be increased if the size of the image is reduced. This can be done by application of spatial-pyramid. The original source and target images can be sub-sampled, saving the Laplacian of Gaussian of the image and the sub-sampled image. Then computing the matching points between the sub-sampled images, followed by computing the homography matrix. Then the images can be up-sampled and overlayed on each other as required.

4.2 Panorama

(Fig 21-23)

```
import numpy as np
import cv2
img_left = np.asarray(cv2.imread('../data/left_img.jpg'))
img_right = np.asarray(cv2.imread('../data/right_img.jpg'))
print(img_left.shape)
scale = 0.5
height = 300
width = 600
dim = (width, height)
```

Figure 21: Left Image

Figure 22: Right Image

```
10 img_left = cv2.resize(img_left, dim)
11 img_right = cv2.resize(img_right, dim)
12 stitcher = cv2.Stitcher_create(cv2.Stitcher_PANORAMA)
13 error, result = stitcher.stitch((img_right, img_left))
14 cv2.imshow("left", img_left)
15 cv2.imshow("right", img_right)
16 cv2.imshow("result", result)
17 cv2.waitKey(0)
```

Figure 23: Panorama image formed by combining the left and right images.