Project 3
ENPM673

Sakshi Kakde
M. Eng. Robotics
University of Maryland
College Park, MD, 20742
Email: sakshi@umd.edu

I. Calibration

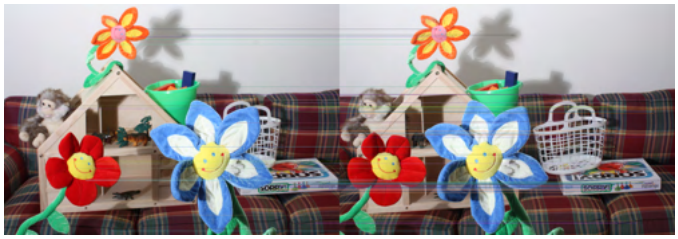
A. Feature Detection and Matching

The following steps were followed to extract the feature pairs from the images:

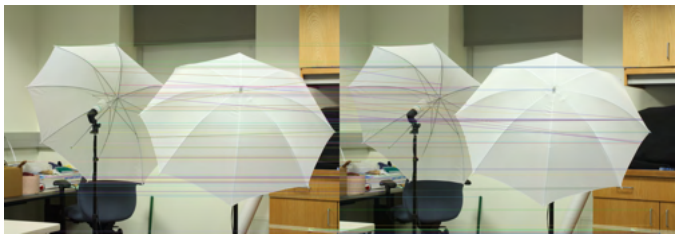
1) Convert the image to grayscale.
2) Use SIFT feature detector to get the corner points.
3) Use Brute-Force matcher to get the matching corners. The OpeenCV function *cv2.BFMatcher* can be used.
4) The result of *bf.match(des1,des2)* line is a list of DMatch objects. This DMatch object has the following attributes:[3]
   a) DMatch.distance - Distance between descriptors. The lower, the better it is.
   b) DMatch.trainIdx - Index of the descriptor in train descriptors
   c) DMatch.queryIdx - Index of the descriptor in query descriptors
   d) DMatch.imgIdx - Index of the train image.

   Sort the obtained matches based on the *distance* parameter to get the first *n* best matches. Here, *n* is set as 100.
5) The result obtained is shown in fig. 1. It can be seen that there are a few wrong matches obtained. These can terribly affect the results. To filter these wrong feature pairs, we will use RANSAC. It will be discussed in the following sections.

(a)



(b)



(c)

Fig. 1. SIFT features matched using Brute-Force matcher

B. Estimation of Fundamental Matrix and RANSAC

The fundamental matrix is calculated using the 8-point algorithm.[4][5]. If the F matrix estimation is good, then terms $x_2^T.F.x_1$ should be close to 0, where x_1 and x_2 are features from image1 and image2. Using this criteria, RANSAC can be used to filter the outliers. The results after RANSAC are shown in fig. 2. It can be seen that the outliers are significantly reduced.

C. Estimation of Essential Matrix

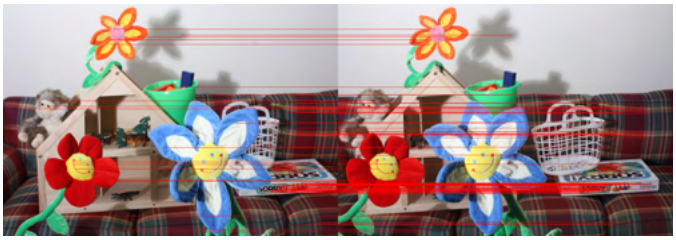
We will obtain a Fundamental matrix from the above section. We already know the intrinsic parameters of the cameras(K_1 and K_2). The Essential matrix can be estimated as:

$$E = K_2^T.F.K_1$$

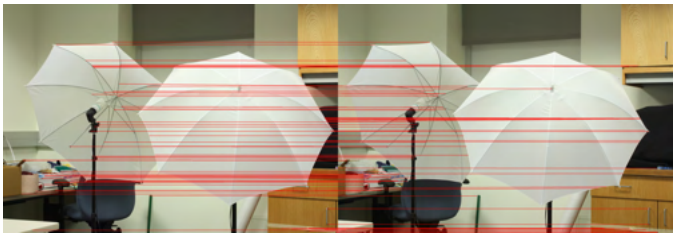
D. Estimation of Camera Pose

As described in [6] page 258, camera pose(R and C) can be estimated using the essential matrix E. We will be estimating the pose for camera 2 with respect to camera 1 which is assumed to be at world origin. We will get four solutions, where two will be twisted pairs and one will be flipped by 180^o. Refer fig. **??**. For a correct pair of camera pose, the Z value of the 3D points shall be positive. For each set of R and C, I calculated the 3D points and chose the R and C set with maximum values for positive Z for both the cameras. Refer fig. 4 a sample plot of 3D points.

(a)



(b)



(c)

Fig. 2. Feature matches after RANSAC

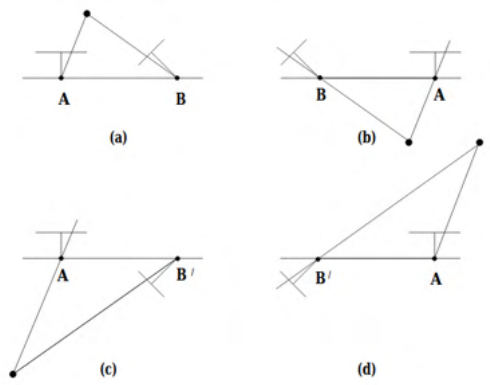


Fig. 3. The four possible solutions for calibrated reconstruction from E.[6]

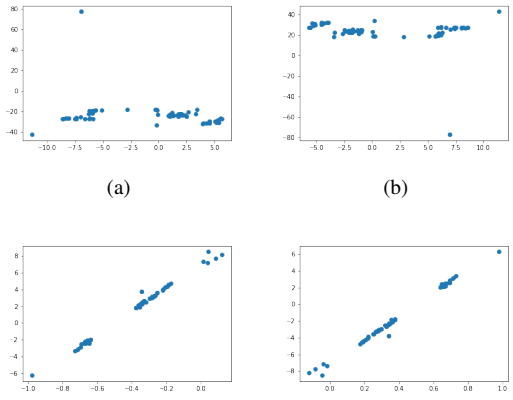
R and C values for Dataset1:

$$R = \begin{pmatrix} 9.99999106e-01 & -3.56189670e-04 & -1.28880478e-03 \\ 3.57173209e-04 & 9.99999645e-01 & 7.62990309e-04 \\ 1.28853255e-03 & -7.63449953e-04 & 9.99998878e-01 \end{pmatrix}$$

$$C = \begin{pmatrix} 0.99984035 & -0.01729798 & 0.00447798 \end{pmatrix}$$

R and C values for Dataset2:

$$R = \begin{pmatrix} 9.99992784e-01 & 7.62529585e-04 & 3.72175066e-03 \\ -7.67192905e-04 & 9.99998922e-01 & 1.25172418e-03 \\ -3.72079217e-03 & -1.25457045e-03 & 9.99992291e-01 \end{pmatrix}$$



(a) (b) (c) (d)

Fig. 4. 3D points for all the R and C sets for Dataset3. a)Flipped by 180 b)Correct set c)Twisted d)Twisted

$$C = \begin{pmatrix} 0.99994608 & 0.00691254 & 0.00774903 \end{pmatrix}$$

R and C values for Dataset3:

$$R = \begin{pmatrix} 0.99989106 & -0.00278445 & -0.01449525 \\ 0.00235965 & 0.9995696 & -0.02924118 \\ 0.01457043 & 0.02920379 & 0.99946728 \end{pmatrix}$$

$$C = \begin{pmatrix} 9.99428908e-01 & -9.36341324e-04 & -3.37784042e-02 \end{pmatrix}$$

II. Rectification

Using the fundamental matrix and the feature points, we can obtain the epipolar lines for both the images. Refer fig. 5. The epipolar lines need to be parallel for further computations to obtain depth. This can be done by reprojecting image planes onto a common plane parallel to the line between camera centers[1]. Using the matched image pairs and the estimated F matrix, the images can be rectified using the OpenCV function *cv2.stereoRectifyUncalibrated*. We will obtain two homography matrices, one for each image.

H1 and H2 for Dataset1:

$$H1 = \begin{pmatrix} -1.15475093e-03 & -4.70006490e-05 & 1.41241253e-01 \\ 4.45887978e-05 & -1.16719800e-03 & -7.25446885e-02 \\ 1.98170032e-08 & 1.05875225e-09 & -1.20159276e-03 \end{pmatrix}$$

$$H2 = \begin{pmatrix} 9.75471955e-01 & 1.99701916e-02 & 1.65948261e+01 \\ -3.68106371e-02 & 9.99455937e-01 & 5.55413315e+01 \\ -1.62774779e-05 & -3.33238030e-07 & 1.02465312e+00 \end{pmatrix}$$

H1 and H2 for Dataset2:

$$H1 = \begin{pmatrix} -1.08362452e-03 & 8.50489604e-05 & 1.41154172e-01 \\ -2.63704557e-05 & -1.01419715e-03 & 3.96780519e-02 \\ -9.66334294e-09 & 8.94180240e-10 & -1.00119579e-03 \end{pmatrix}$$

$$H2 = \begin{pmatrix} 1.01113841e+00 & -1.54810603e-02 & -7.13063314e-01 \\ 2.30469543e-02 & 9.99764338e-01 & -3.29543091e+01 \\ 7.81638700e-06 & -1.19672991e-07 & 9.88862879e-01 \end{pmatrix}$$

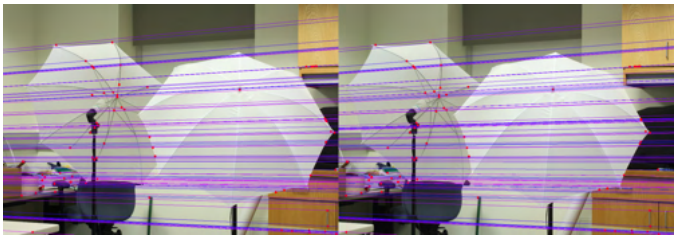
H1 and H2 for Dataset3:

$$H1 = \begin{pmatrix} 7.99948872e-04 & -8.08372864e-06 & -2.16033989e-01 \\ 8.49500790e-05 & 7.16564764e-04 & -1.33401830e-01 \\ 6.18261266e-08 & -7.33742903e-10 & 6.19156503e-04 \end{pmatrix}$$

$$H2 = \begin{pmatrix} 1.12672655e+00 & -3.53734946e-02 & -1.51898810e+02 \\ 1.18025921e-01 & 9.96787284e-01 & -1.71439946e+02 \\ 8.59587876e-05 & -2.69866962e-06 & 8.75501253e-01 \end{pmatrix}$$

(a)

(b)

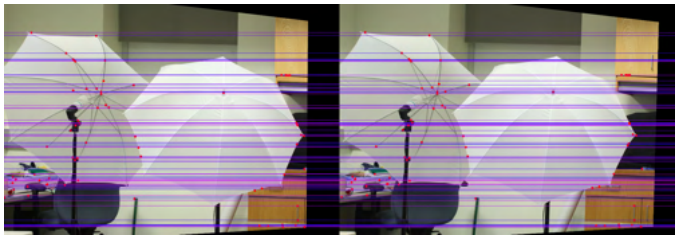


(c)

Fig. 5. Epipolar lines for unrectified images.

(a)

(b)



(c)

Fig. 6. Epipolar lines for rectified images.

After rectification, the epipolar lines will be parallel. Refer fig. 6. After the images are warped, we need to transform the feature points as well using the obtained homography matrices. Also, the F matrix will get modified as the epipolar geometry has changed. It will be $H_2^{-T}.F, H_1^{-1}$.

III. Correspondence

For every pixel in image 1, we try to find a corresponding match along the epipolar line in image 2. We will consider a window of a predefined size for this purpose, so this method is called block matching. Essentially, we will be taking a small region of pixels in the left image, and searching for the closest matching region of pixels in the right. image[2]. I tried three block comparison methods:

1) Sum of Absolute Differences (SAD)
2) Sum of Squared Differences (SSD)
3) Normalized Cross-Correlation (NCC)

Normalized Cross-Correlation (NCC) gives the best results but takes a lot of computation time. Sum of Absolute Differences (SAD) takes the least computation time, but the results are not that great. Sum of Squared Differences (SSD) takes a moderate computation time and the results are decent. I have used Sum of Absolute Differences (SAD). A simple approach to implement the block matching algorithm would be to use nested for loops. However, the approach is not efficient. In the following section, I will be discussing, in brief, my vectorized approach which significantly reduces the computation time.

A. Vectorised Approach

1) For every pixel, we have a window of size 11×11. I flatten this window to get a 1D array of size 121. Let's call this a feature descriptor. So, for each pixel, we will have a feature descriptor of size 121.
2) For the feature descriptor of each pixel in the left image, we need to subtract the feature descriptors of all the pixels in the left image. To avoid a for loop, I repeat the array elements along axis 1 for the left image and along axis 0 for the right image. After this step, I will obtain two 3D arrays representing a row in left and right images. Refer fig. 7.

IV. Compute Depth Image

A. Disparity Map

After we get the matching pixel location, the disparity can be found bu take the absolute of the difference between the source and matched pixel location. the results after normalizing are shown in fig. 9 and **??**.

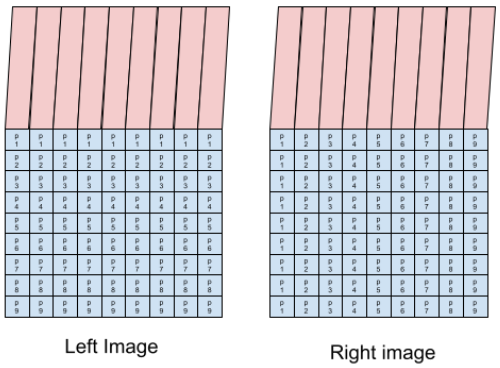


Fig. 7. 3D vector obtained for a row from left and Right image. For simplicity in understanding, each row is assumed to have 9 pixels. The pink array along the depth is the feature descriptor.[6]

3) If we subtract the arrays obtain in step 2, and take a sum along the depth, we will obtain the SAD for each pixel. Finding the minimum along axis 1 will give me the matching pixel location.

B. Depth Map

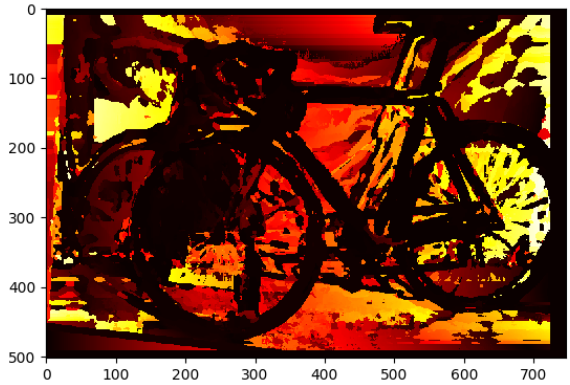
If we know the focal length(f) and basline(b), the depth can be calculated as:

$$d = \frac{f * b}{disparity}$$

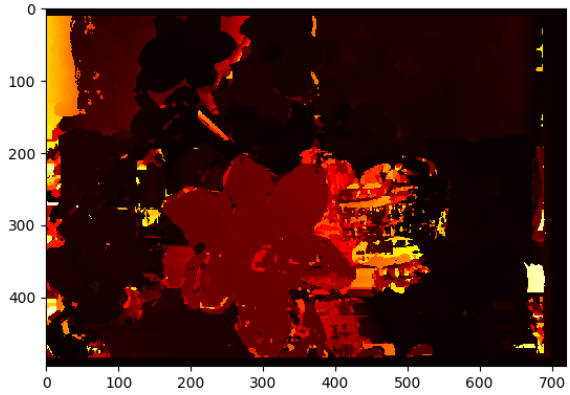
The depth results are shown in fig. 10 and 11.

REFERENCES

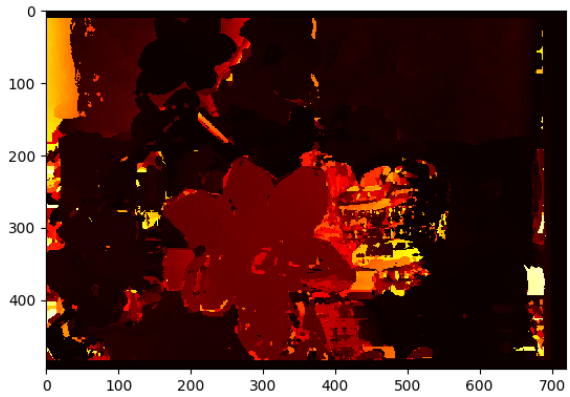
[1] http://www.cs.cmu.edu/Ĩ6385/lectures/lecture13.pdf
[2] http://mccormickml.com/2014/01/10/stereovisiontutorialparti/
[3] https://opencvpythontutroals.readthedocs.io/en/latest/py_tutorials/py_feature2
[4] https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj3/html/sda
[5] https://www.youtube.com/watch?v=Opy8xMGCDrE
[6] https://www.cambridge.org/core/books/multipleviewgeometryin-computervision/0B6F289C78B2B23F596CAA76D3D43F7A



(a)

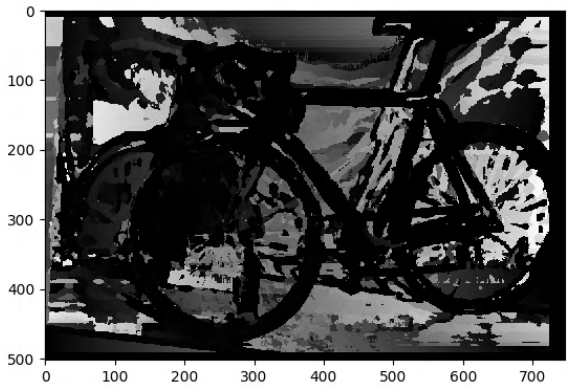


(b)

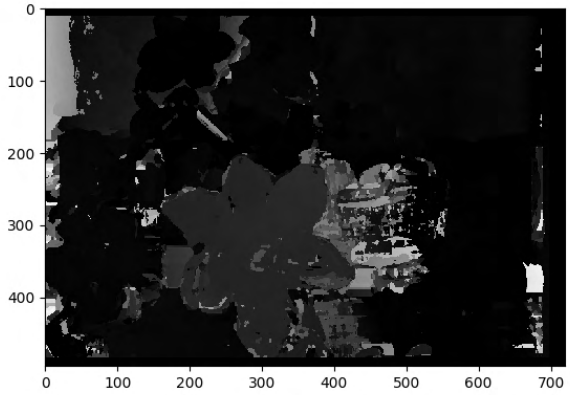


(c)

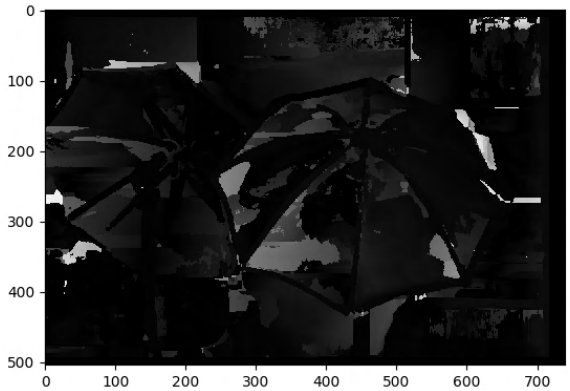
Fig. 8. Diparity images(heat map)



(a)

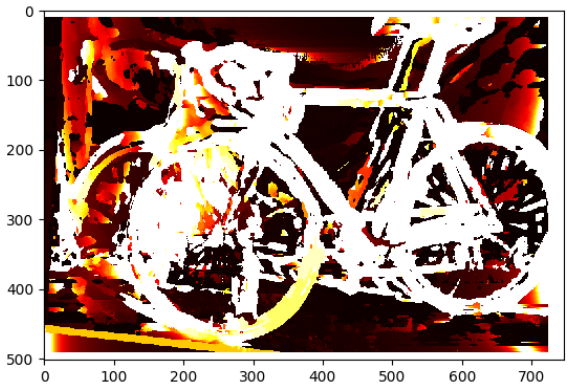


(b)

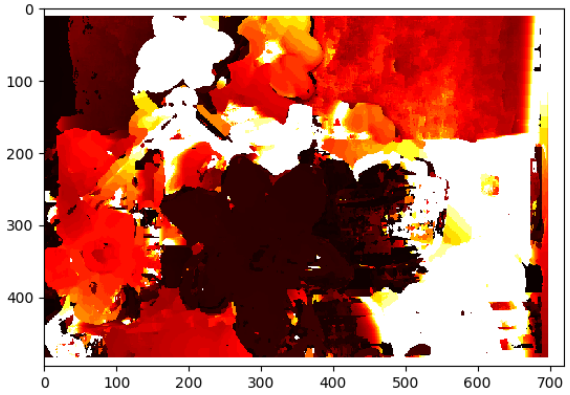


(c)

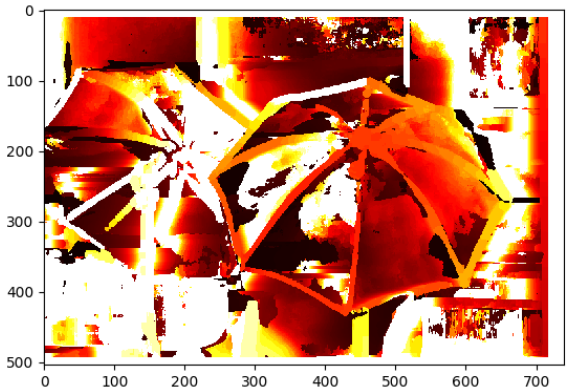
Fig. 9. Disparity images(Gray scale)



(a)

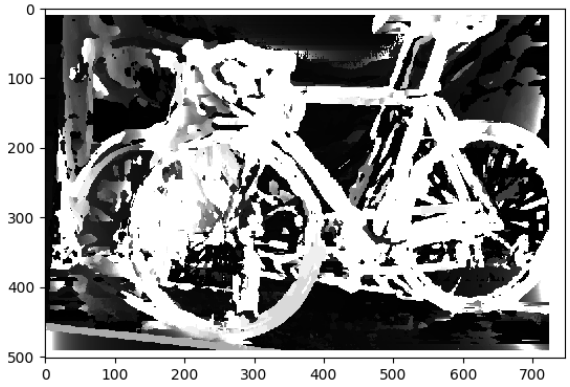


(b)

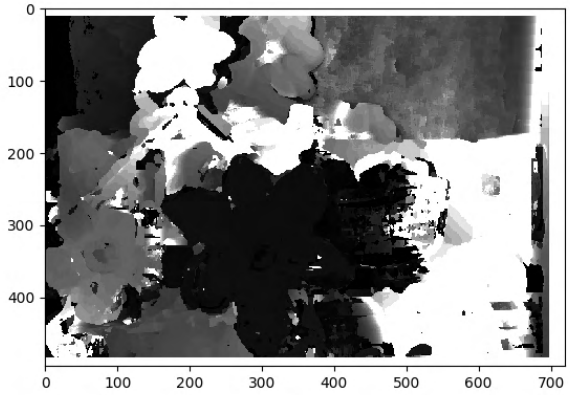


(c)

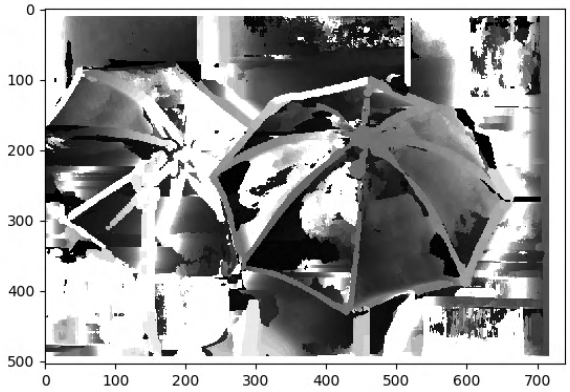
Fig. 10. Depth map(Heat map)



(a)



(b)



(c)

Fig. 11. Depth map(Gray scale)