CS5670 - Computer Vision
Homework 2 Write-up
Jing Dai (jd837) - Muhammad Khadafi (mk2333)

Task 1. SIFT

1. Difference of Gaussian (DoG) is better suited for SIFT if we want to find interest points that are more resistant to noise. Random noise, as opposed to edges have higher spatial frequency, as in there are little difference of frequency in one area regardless of the blur. Using DoG, we can eliminate the high frequency detail. Finally, working with less points of interest without noises, will simplify the search for more important points in SIFT

2. We are using the OpenCV library to implement SIFT in Python, where the function to get the key-points and descriptors are:

```
# Initiate SIFT detector
sift = cv2.SIFT()

# find the keypoints and descriptors with SIFT
kp, des = sift.detectAndCompute(img ,None)
```

3. Find_matches was implemented in the code as follows:

```
img1_coord, img2_coord = find_matches(kp1, kp2, des1, des2)
```

Taking key-points and descriptors of both images as input, and getting the coordinates of matching key-points as output. We implemented k-NN to look for the k most similar descriptor of each key-points in image1 using *my_knn(d1, d2, k)* using k=2 and the shortest euclidean distance of the descriptors. After getting the 2-nearest neighbor, we implemented ratio test, as mentioned in David Lowe's paper with the function *ratio_test(mat, arg)*, which prevents the 2-nearest neighbor to be too similar to each other.

4. The function show_matches is as follows:

```
show_matches(image1, image2, img1_coord, img2_coord, img1_name + " - " +
                    img2_name + " all keypoint")
```

Taking the two images, the matching coordinates and the name to give the title and filename. First, we pad the first image with black as wide as image 2's width, then we append image 2 to it's right. Then we loop through the marching coordinates and creates line in between them. The results are as follows:

Stop Sign 1 - Stop Sign 2 all keypoint

Stop Sign 1 - Stop Sign 3 all keypoint

Stop Sign 1 - Stop Sign 4 all keypoint

5. The function affine_matches is as follows:

```
img1_affine_coord, img2_affine_coord, affine_trans =
        affine_matches(img1_coord, img2_coord)
```

Taking the matching coordinates as input and giving matching coordinates that has been filtered by the RANSAC algorithm using the affine transformation found by least squares algorithm (*get_affine_transformation()*). We modified the RANSAC algorithm a little bit to suit our need. Instead of letting RANSAC going until we reach certain threshold and risking false fit (if the threshold is too low) or infinite loop (if threshold is too high), we run RANSAC 10000 times and taking the transformation with highest match as the accepted. We decided to do this because the 3 images are too variable, and we found that there isn't a best threshold to fit all 3 images. The downside is that we may not get the best fit, however we will always find a solution, and the more we repeat, the more chance he have to find the best solution. Lastly, once we found the filters coordinate, we run the least squares algorithm one more time fo get the average affine transformation matrix.

Here are the results:

Stop Sign 1 - Stop Sign 3 affine keypoint

Stop Sign 1 - Stop Sign 4 affine keypoint

In short, RANSAC did exactly what needed to be done and removed the outliers from the matches. The remaining matches are less that what it was and is consistent geometrically

6. The function align_images is as follows:

```
align_images(image1, image2, average_affine_trans, img1_name + " - " +
                    img2_name + " affine aligned")
```

We are taking both of the images, the average affine transformation matrix, and the name for title and filename. As described it the question, we first transform image 1

with the transformation matrix, we then take only the blue channel of image 1, and red and green channel from image 2. Lastly, we merge them together by simply adding their value. Here are the results:

Stop Sign 1 - Stop Sign 2 affine aligned

Stop Sign 1 - Stop Sign 3 affine aligned

Stop Sign 1 - Stop Sign 4 affine aligned

We can tell that the transformation is accurate when the stop sign is complete in color (the red and white are clear). The results are generally satisfactory, other that the last image. This image cannot be perfectly represented as an affine transformation from stopsign1 because the two stop signs has different collinearity for each of their lines. We can see the where the transformation made the bottom line fit, the top line didn't. This is the case when affine transformation might fail

7. One simple quantitative measure is to measure the difference (euclidean distance) between image2 and the merged image. Then the image is correctly aligned, the difference between the merged image and the original target image will be less. This will work best as comparison between different methodology, as the smaller the "failure rate" (distance) is the better algorithm. Here are the results:

```
    Stop Sign 1 - Stop Sign 2
affine success rate: 6556.36133843

    Stop Sign 1 - Stop Sign 3
affine success rate: 22705.6236206

    Stop Sign 1 - Stop Sign 4
affine success rate: 13946.5134711
```

8. Repeating the above steps of homography requires much of the same algorithm. The biggest difference is instead of using least squares fit the transformation, we use singular value decomposition (SVD).

Here are the key-points filtered by RANSAC:

Stop Sign 1 - Stop Sign 2 projective keypoint

Stop Sign 1 - Stop Sign 3 projective keypoint

Stop Sign 1 - Stop Sign 4 projective keypoint

Here are the aligned images:

Stop Sign 1 - Stop Sign 2 projective aligned

Stop Sign 1 - Stop Sign 3 projective aligned

Stop Sign 1 - Stop Sign 4 projective aligned

Here is the “failure rate” (distance):

```
Stop Sign 1 - Stop Sign 2
projective success rate: 6557.9472398

Stop Sign 1 - Stop Sign 3
projective success rate: 22706.4641017

Stop Sign 1 - Stop Sign 4
projective success rate: 13944.6895268
```

The results are quite similar for the first two images, only differs singular digit failure rate. The homography transformation did worse in both cases, and it can roughly be seen visually in stopsign2. This can be explained by the high amount of matching keypoints and the collinearity made both transformation fits nicely on the images.

The third image however, homography/projective transformation did a little better than affine. As mentioned above, the different collinearity of the two images made affine transform unreliable, the same does not apply to homography transformation. However, since projective transformation really dependant on the training points, the little amount (and non spread out) key points also made the projective transformation not perfect. One can assume, if there are matching keypoint on the bottom part of the sign, the projection will fit nicer.

*results may vary each run because of the modified RANSAC

Task 2. Epipolar Geometry and Camera Calibration

Part 1

1. Set the world coordinate frame as the camera coordinates of the left camera.
   - The right camera coordinates is given by the world coordinates rotate by 60˚ along y-axis. Therefore the rotation matrix R is:

$$\begin{pmatrix} cos(-60^o) & 0 & sin(-60^o) \\ 0 & 1 & 0 \\ -sin(-60^o) & 0 & cos(-60^o) \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \\ \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} \end{pmatrix}.$$

   - The translation vector t = $\left(\frac{\sqrt{3}}{2}d,\ 0,\ \frac{1}{2}d\right)^T$, which is the coordinates of the world origin in the right camera coordinate frame. So the matrix T_x is:

$$\begin{pmatrix} 0 & -\frac{d}{2} & 0 \\ \frac{d}{2} & 0 & -\frac{\sqrt{3}d}{2} \\ 0 & \frac{\sqrt{3}d}{2} & 0 \end{pmatrix}.$$

   .

   - K_L and K_R are both identity matrices as both have focal length of unity.
   - The fundamental matrix is:

$$F = K_L^{-T} T_x R K_R^{-1} = \begin{pmatrix} 0 & -\frac{d}{2} & 0 \\ -\frac{d}{2} & 0 & -\frac{\sqrt{3}d}{2} \\ 0 & \frac{\sqrt{3}d}{2} & 0 \end{pmatrix}.$$

2. Given the point x= $(1,\ 1,\ 1)^T$ in the left image plane, the corresponding epipolar line is:

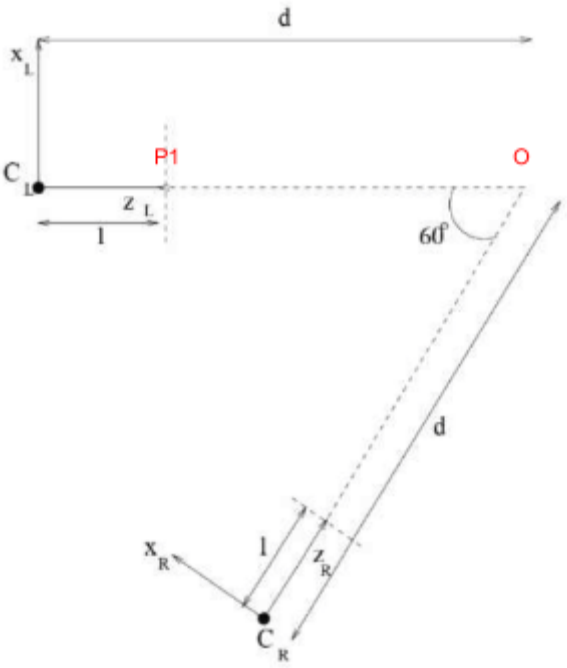
$$l = Fx = \begin{pmatrix} 0 & -\frac{d}{2} & 0 \\ \frac{d}{2} & 0 & -\frac{\sqrt{3}d}{2} \\ 0 & \frac{\sqrt{3}d}{2} & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} -\frac{d}{2} \\ -\frac{1+\sqrt{3}}{2}d \\ \frac{\sqrt{3}}{2}d \end{pmatrix},$$

   which gives us the normal vector of the epipolar line: $n = \left(-\frac{d}{2}, -\frac{1+\sqrt{3}}{2}d\right)^T$.

   Note that the epipolar of the left camera in the right image plane is: $\left(\sqrt{3},\ 0\right)^T$. We can derive that the epipolar line is: $-\frac{d}{2}(x-\sqrt{3}) - \frac{1+\sqrt{3}}{2}d\,y = 0$, which can be simplified as:

$$x + (1+\sqrt{3})y - \sqrt{3} = 0.$$

3. The potential correspondences to the left image point $(0,\ 0)^T$ is the ray from P_1 to O (labelled in the figure).



O projects to the point $(0,\ 0)^T$ on the right image plane.
In the right camera coordinate frame, the right image of $P_1 = (0,\ 0,\ 1)^T$ can be calculated as:

$$p1 = [R|t]P1 = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{\sqrt{3}}{2} & \frac{\sqrt{3}}{2}d \\ 0 & 1 & 0 & 0 \\ \frac{\sqrt{3}}{2} & 0 & \frac{1}{2} & \frac{1}{2}d \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{\sqrt{3}}{2}(d-1) \\ 0 \\ \frac{d+1}{2} \end{pmatrix}.$$

Therefore the right image of p_1 is: $(\frac{\sqrt{3}(d-1)}{d+1},\ 0)^T$. The corresponding ray is:

$$\begin{cases} x \le \frac{\sqrt{3}(d-1)}{d+1} \\ y = 0 \end{cases}.$$

4. In this part, we set the world coordinate frame by the right camera frame. Let R and T_x be the rotation and translation matrix for the left camera with respect to the right one. Suppose x_1 and x_2 are the images of a sample point X in the left and right image plane respectively, Fx_1 and $F^T x_2$ are the epipolar lines. We want these two lines to be parallel. Note that $x_2{}^T F x_1 = 0$ holds for any image points, then we have: $x_2{}^T F^T x_2 = 0,\ \forall x_2$, which shows that F is skew-symmetric (i.e. $F^T = -F$). Since $F = T_x R$, where T_x is skew-symmetric and R is orthogonal, we can derive that R and T_x should satisfy the following :

$$(T_x R)^T = -T_x R \Leftrightarrow R^T {T_x}^T = -T_x R \Leftrightarrow -R^T T_x = -T_x R \Leftrightarrow R^T T_x = T_x R .$$

Part 2.

Code:

q2.py works as follows:

- read sample points in world coordinates from the address WORLD_INFILE (by default './HW2_world.txt'), and convert them into homogeneous coordinates;
- read sample points in image coordinates from the address IMAGE_INFILE (by default './HW2_image.txt'), and convert them into homogeneous coordinates;
- follow the instructions to construct the matrix A
- use `numpy.linalg.svd` to decompose A, then p is the row of V corresponding to the smallest singular value (in s). Reshape p into 3*4 projection matrix P.
- conduct SVD on P, then solve for the projection centre

Output

1. The projection matrix is:

$$\begin{pmatrix} -1.27000127e-01 & -2.54000254e-01 & -3.81000381e-01 & -5.08000508e-01 \\ -5.08000508e-01 & -3.81000381e-01 & -2.54000254e-01 & -1.27000127e-01 \\ -1.27000127e-01 & 5.41233725e-16 & -1.27000127e-01 & -6.66133815e-16 \end{pmatrix}$$

The reprojection of the points are very close to the original image. Detailed output are as follows:

```
1.    Original image: [5.1177070095845485, 4.765384406444506]
      Reprojection: [5.1177070095845512, 4.7653844064445101]
==
2.    Original image: [5.52365450225305, 3.8703291749533313]
      Reprojection: [5.523654502253053, 3.8703291749533322]
==
3.    Original image: [7.163101712347696, 7.359420656383301]
      Reprojection: [7.1631017123477019, 7.3594206563833096]
==
4.    Original image: [5.222166277736225, 4.427958500493556]
      Reprojection: [5.2221662777362265, 4.4279585004935589]
==
5.    Original image: [5.604796139395111, 4.674836482251409]
      Reprojection: [5.6047961393951127, 4.674836482251413]
==
6.    Original image: [13.594948846275605, 10.052154948111069]
      Reprojection: [13.594948846275624, 10.052154948111079]
==
7.    Original image: [8.734521888062085, 5.564205313535146]
      Reprojection: [8.7345218880620941, 5.5642053135351492]
==
8.    Original image: [6.224339521255969, 3.9082188503387805]
      Reprojection: [6.2243395212559731, 3.9082188503387805]
==
9.    Original image: [9.747638859645045, 6.904237229708508]
```

```
      Reprojection: [9.7476388596450541, 6.9042372297085111]
==
10.   Original image: [5.090310794175585, 4.550851304977495]
      Reprojection: [5.0903107941755863, 4.5508513049774999]
```

2. The world coordinates of the projection center is:
   $(-0.91471139,\ 3.36952562,\ -3.2842370)^T$.