558 Assignment3
Evelyn Cao
ID : 260888060

Q1 Fundamental Matrix Estimation

First, manually identified 8 corresponding pixel pairs by *imtool* and stored their coordinates in an 8*4 matrix.

The fundamental matrix **F** is estimated in `fundamental_matrix.m` by the following steps:

1) Define the data normalization matrix M1 and M2 for the left and right images by the formulas below,

For the image points, the centroid is

$$(\bar{x}, \bar{y}) = \frac{1}{N}\sum_{i=1}^{N}(x_i, y_i)$$

and the average squared distance of the x and y coordinates from the centroid is:

$$\sigma_1^2 = \frac{1}{2N}\sum_{i=1}^{N}(x_i - \bar{x})^2 + (y_i - \bar{y})^2$$

We *normalize* by applying a translation and scaling as follows:

$$(x_i, y_i) \rightarrow (\frac{x_i - \bar{x}}{\sigma_1}, \frac{y_i - \bar{y}}{\sigma_1})$$

$$\boldsymbol{M} = \begin{bmatrix} \frac{1}{\sigma} & 0 & \frac{\bar{x}}{\sigma} \\ 0 & \frac{1}{\sigma} & \frac{\bar{y}}{\sigma} \\ 0 & 0 & 1 \end{bmatrix}$$

2) Build matrix **A** by normalized data and take SVD of **A** to get its null space, which is the last column of **V** with singular value 0. By this we get the reshaped 1*9 matrix **F**.

$$\begin{bmatrix} x_1^1x_2^1 & y_1^1x_2^1 & x_2^1 & x_1^1y_2^1 & y_1^1y_2^1 & y_2^1 & x_1^1 & y_1^1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_1^ix_2^i & y_1^ix_2^i & x_2^i & x_1^iy_2^i & y_1^iy_2^i & y_2^i & x_1^i & y_1^i & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ x_1^Nx_2^N & y_1^Nx_2^N & x_2^N & x_1^Ny_2^N & y_1^Ny_2^N & y_2^N & x_1^N & y_1^N & 1 \end{bmatrix} \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ F_{21} \\ F_{22} \\ F_{23} \\ F_{31} \\ F_{32} \\ F_{33} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

3) Enforce a rank = 2 constraint on **F** by taking SVD of **F** (normalized) and making $\Sigma(3,3) = 0$.

4) De-normalize by $\boldsymbol{F} \equiv {M_2}^T F_{normalized} M_1$
I also did $\boldsymbol{F} = \boldsymbol{F}/norm(\boldsymbol{F})$ in convenient to comparing **F** with the result got from the Matlab's build-in functions.

As shown below, **F1_8point** is got from function *fundamental_matrix* I defined, by *rank* function I checked **F1_8point** has rank 2. **F1_build_in** is from Matlab's function *estimateFundamentalMatrix*. We can see they are basically the same.

F by 8-points estimation of stereo 1:

```
F1_8point = 3x3
    -0.0000    0.0000   -0.0029
    -0.0000    0.0000    0.0035
     0.0029   -0.0133    0.9999

ans = 2
```

```
F1_build_in = 3x3
    -0.0000    0.0000   -0.0028
    -0.0000    0.0000    0.0035
     0.0029   -0.0134    0.9999
```

F by 8-points estimation of stereo 2:

```
F2_8point = 3x3
    -0.0000   -0.0001    0.0393
     0.0001   -0.0000    0.0667
    -0.0367   -0.0592   -0.9946

ans = 2
```

```
F2_build_in = 3x3
     0.0000    0.0001   -0.0393
    -0.0001    0.0000   -0.0667
     0.0367    0.0593    0.9946
```

Use *draw_epipolar_lines* to draw the epipolar lines with the 8 points and the fundamental matrix I estimated. More explanation is in Q2.

Figure 1: epipolar lines in Q1

Q2 Fundamental Matrix Refinement - RANSAC

RANSAC algorithm is implemented in `ransac.m`, which works as follows

1) Randomly pick 8 points from matched_list and estimate **F** with these points using the function *fundamental_matrix* same as Q1.
2) This *ransac* has two parameters, *t* and *cmin*. To determine whether a pair of pixels (x1,x2) is an inlier, calculate the distance from the x1 (in the left image) to the epipolar line got from the corresponding x2 (in the right image) and fitted **F** from step1). If the distance is less than *t* then take this pair as an inlier.
   The parameters I set are
   for stereo 1: $t = 0.5, cmin = 150$
   for stereo 2: $t = 0.5, cmin = 55$
3) If the consensus set is above *cmin*, terminate and keep **F**. Otherwise start again.

F1_ransac is got from *ransac* I defined, **F** is got from Matlab's *estimateFundamentalMatrix*. We can see they do not vary much, which means my algorithm works well. I also used *rank* function to confirm my **F1_ransac** has rank 2.

Note that since step1) picks 8 points randomly to estimate **F**, **F** is not the same every time we run *ransac*. But by setting strict parameters the results don't vary too much.

F by RANSAC of stereo 1:

```
F1_ransac = 3x3 single matrix
    0.0000    0.0000   -0.0006
    0.0000   -0.0000   -0.0378
   -0.0026    0.0335    0.9987
```

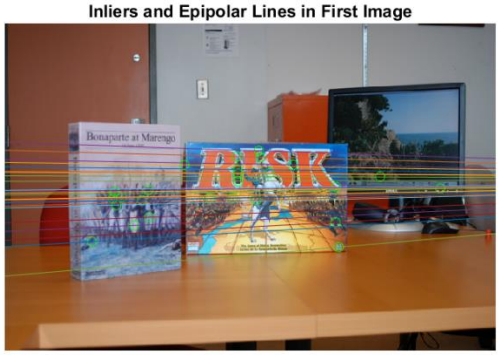
```
F = 3x3
    0.0000    0.0000   -0.0010
    0.0000   -0.0000   -0.0386
   -0.0024    0.0344    0.9987
```

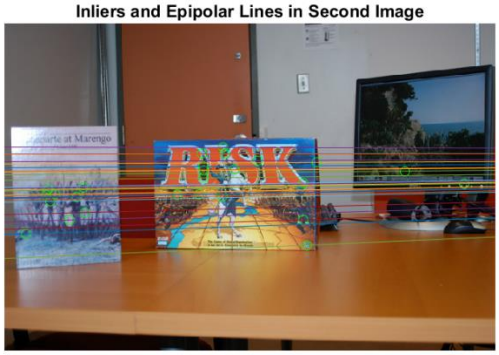
F by RANSAC of stereo 2:

```
F2_ransac = 3x3 single matrix
   -0.0000    0.0000    0.0013
    0.0000   -0.0000   -0.0754
   -0.0034    0.0723    0.9945
```

```
F = 3x3
   -0.0000    0.0000    0.0023
    0.0000    0.0000   -0.0585
   -0.0042    0.0544    0.9968
```

Figure 2 shows the epiploar lines calculated by **F1_ransac** and **F2_ransac**,





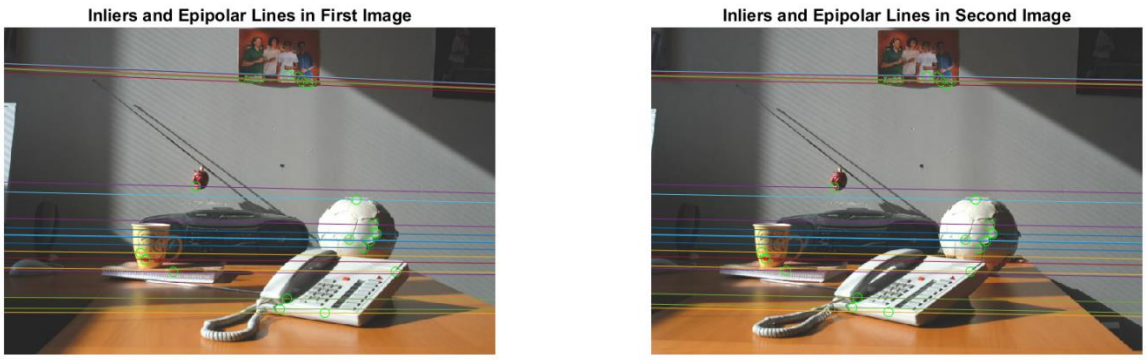


Figure 2: epipolar line in Q2

Compared to Q1, the **F** matrix and epipolar-line images look inconsistent. But this does not mean what we have done in Q1 was wrong, after all Q1 and Q2 estimated **F** using the same function. This is because Q1 just used 8 points, F could be totally different based on which 8 points we chose, so the result was unstable. In conclusion, the RANSAC algorithm is much more robust.

Q3 Rectification

Use the fundamental matrix got from Q2, the rectify algorithm works as follows

1) Get epipoles e_1 and e_2 by calculate right and left null space of **F**
2) By epipole $e = (e_u, e_v, 1)$, define rectifying homography **H** as

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 \\ -\frac{e_v}{e_u} & 1 & 0 \\ -\frac{1}{e_u} & 0 & 1 \end{bmatrix}$$

   Here first I set **H1**=**H2** as above, but the results were not good. The right rectified image always seemed compressed vertically compared the left one. So I modified **H2** a bit to get better results

$$\boldsymbol{H2} = \begin{bmatrix} 1.1 & 0 & -5 \\ -\frac{e_v}{e_u} & 1 & 0 \\ -\frac{1}{e_u} & 0 & 1 \end{bmatrix}$$

3) In the function *rectify*, apply **H** to pixel points to get new pixel position and copy the RGB intensity values from the original image.

Figure 3 shows the rectified images with 8 matching features for each stereo pair. After

rectification the matching features lie on horizontal lines basically.
But the results are still not very ideal, we can see some features are not lined on horizontal lines, like the left-bottom corner of the book in stereo1, and the right-bottom corner of the telephone in stereo2.
I think there are two reasons. First, we did not define **H's** very well. There are a lot of 3*3 matrix which can map epipoles to infinity in the x-direction. The same **H** may not work for every stereo image, and H1, H2 can be different. Second, quality of **F** depends on what matching points we choose. By observing the SIFT features got from *SurfFeaturepoints*, for example I found the features of stereo1 concentrated mostly on the book and box planes. It is better to use features from different planar surfaces and depths.

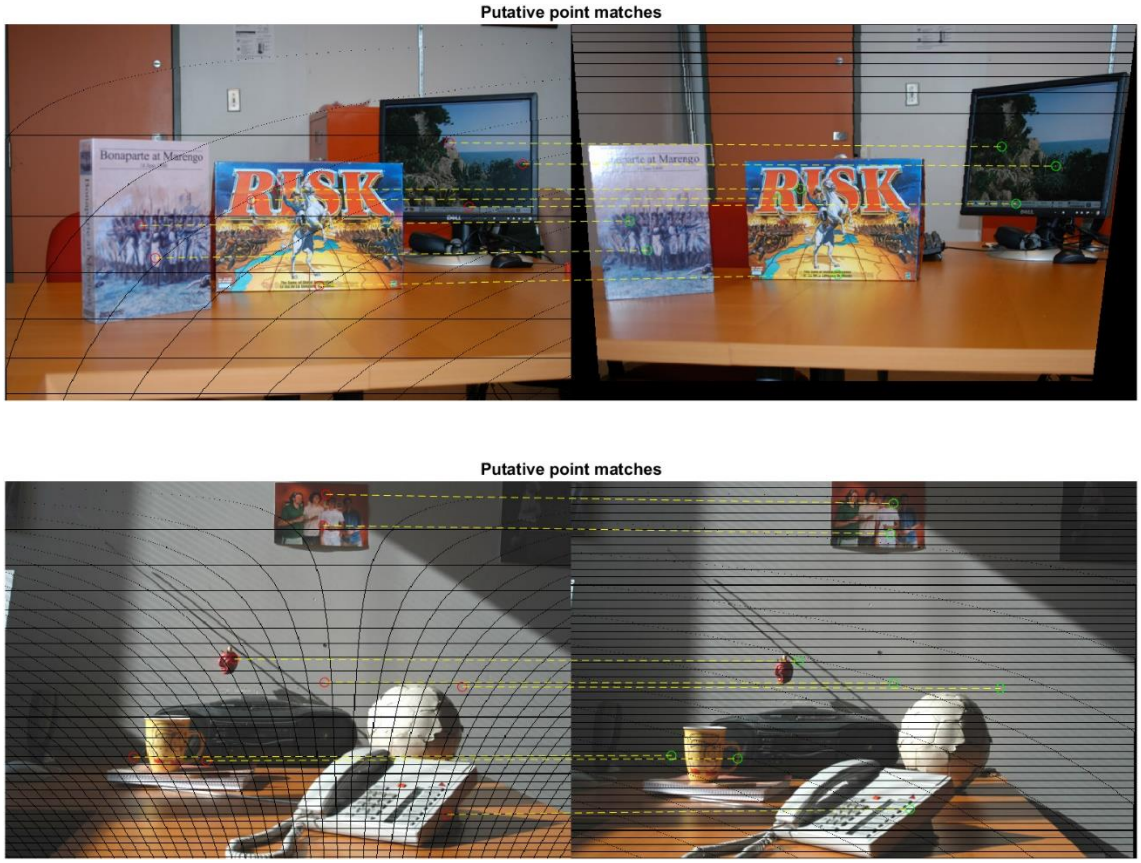


Figure 3: rectified images with matching features

Q4 Reconstruction

In the question I implemented SVD, Louget-Higgen's algorithm to decompose essential matrix, and TRIANG algorithm to reconstruct 3D scene point.

Louget-Higgen's algorithm works as follows (referred from internet)

1) The essential matrix can be written as $\boldsymbol{E} = [\boldsymbol{T}]_{\times}\boldsymbol{R}$ where $[\boldsymbol{T}]_{\times}$ is a skew-symmetric matrix which is actually a cross product $\boldsymbol{T} \times \boldsymbol{X}$, and **R** is a rotation.
   A skew-symmetric matrix has the following property,

SVD of a skew-symmetric matrix:

$$\hat{\mathbf{a}} = \begin{bmatrix} -\mathbf{e}_2 & \mathbf{e}_1 & \mathbf{e}_3 \end{bmatrix} \begin{bmatrix} ||a|| & 0 & 0 \\ 0 & ||a|| & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{bmatrix} \quad \text{where } \mathbf{e}_3 = \mathbf{a} / \|\mathbf{a}\|$$

One singular value is 0 and the other two = ||**a**||

$$\hat{\mathbf{a}} = \begin{bmatrix} \mathbf{e}_1 & \mathbf{e}_2 & \mathbf{e}_3 \end{bmatrix} \begin{bmatrix} ||a|| & 0 & 0 \\ 0 & ||a|| & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{e}_1^T \\ \mathbf{e}_2^T \\ \mathbf{e}_3^T \end{bmatrix}$$

2) Let $\hat{\boldsymbol{t}}$ be the unit vector of **T,** we have

$$\hat{\boldsymbol{t}}^{\boldsymbol{T}}\boldsymbol{E} = \hat{\boldsymbol{t}}^{\boldsymbol{T}}[\hat{\boldsymbol{t}}]_{\times}\boldsymbol{R} = \boldsymbol{0}$$

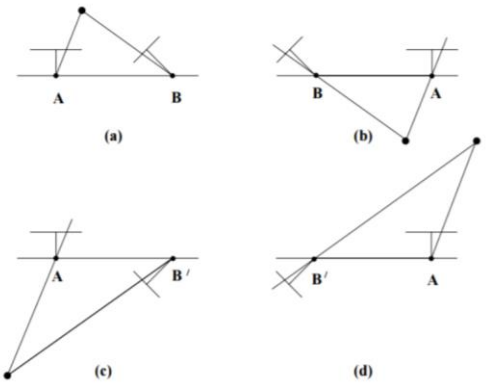
   Then we can decompose $[\hat{\boldsymbol{t}}]_{\times}$ and **E** by the properties above.

$$[\hat{\boldsymbol{t}}]_{\times} = \boldsymbol{S}\boldsymbol{Z}\boldsymbol{R}_{90^\circ}\boldsymbol{S}^T = \begin{bmatrix} \boldsymbol{s}_0 & \boldsymbol{s}_1 & \hat{\boldsymbol{t}} \end{bmatrix} \begin{bmatrix} 1 & & \\ & 1 & \\ & & 0 \end{bmatrix} \begin{bmatrix} 0 & -1 & \\ 1 & 0 & \\ & & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{s}_0^T \\ \boldsymbol{s}_1^T \\ \hat{\boldsymbol{t}}^T \end{bmatrix}$$

$$\boldsymbol{E} = [\hat{\boldsymbol{t}}]_{\times}\boldsymbol{R} = \boldsymbol{S}\boldsymbol{Z}\boldsymbol{R}_{90^\circ}\boldsymbol{S}^T\boldsymbol{R} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

3) Match orthogonal and diagonal matrices $\boldsymbol{S} = \boldsymbol{U}$, $\boldsymbol{Z} = \boldsymbol{\Sigma}$**.** It gives us $\hat{\boldsymbol{t}}$ as the last column of **U,** and $\boldsymbol{R} = \pm\boldsymbol{U}\boldsymbol{R}_{\pm\frac{\pi}{2}}^{\boldsymbol{T}}\boldsymbol{V}^{\boldsymbol{T}}$

   This generates four possible **R** but only situation (a) is right.



   We can select the right **R** by check its determinant and diagonal values.

In Q4.m function *decomposeE* implements the algorithm above. With T and R recovered from E, function *triang* implements the TRIANG algorithm. Note that **T** we got is just a direction vector, so the 3D scene point (X, Y, Z) *triang* got are not the exact X, Y and Z value in the left

camera coordinate, they are scaled by a constant. But by comparing value of Z's we can get a sense of the relative ordering in depth of these scene points.

Figure 4 and 5 shows that scene point with smaller depth is labelled by a bigger number

Figure 4: relative ordering in depth of points in stereo 1

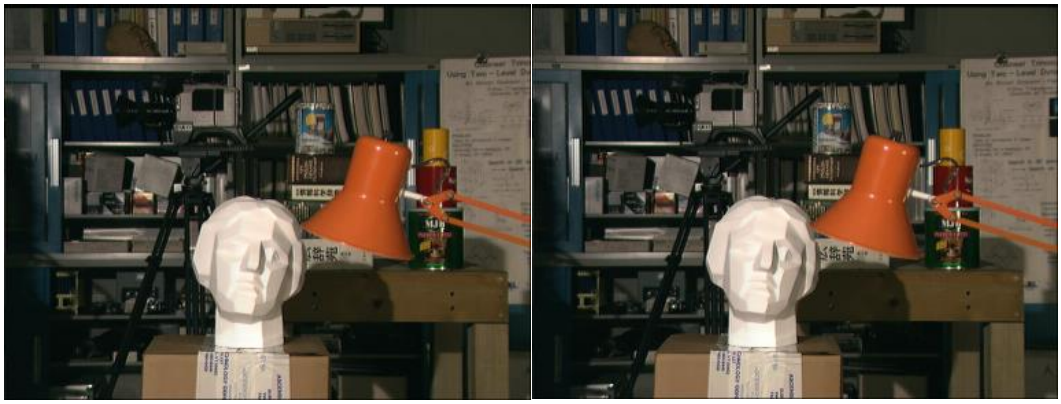
Figure 5: relative ordering in depth of points in stereo 2

Q5 Dense Reconstruction - Disparity

Function *disparity* outputs the disparity map and works as follows

1) There are two parameters, *winSize* and *dispMax*. *winSize* decides the window size and *dispMax* decides the disparity range.
2) We want to find *d* such that $I_l(x_l) \approx I_r(x_l - d)$. Slide a window first horizontally and then vertically along the images. On the right image, shift the window to the left by a value from 1 to *dispMax*. The value which minimizes the cost function $SSD = \sum(I_l(x_0, y_0) - I_r(x_0 - d, y_0))^2$ over the window is the disparity.

Since the results of Q3 were not ideal, I used another two pairs of stereo images took by rectified camera to test my algorithm. These two disparity maps seemed reasonable. Disparity and depth are inversely related, large disparity represents small depth. In the disparity map, red part represents object in the front, yellow part represents something deeper, and blue part represents the background which has largest depth.



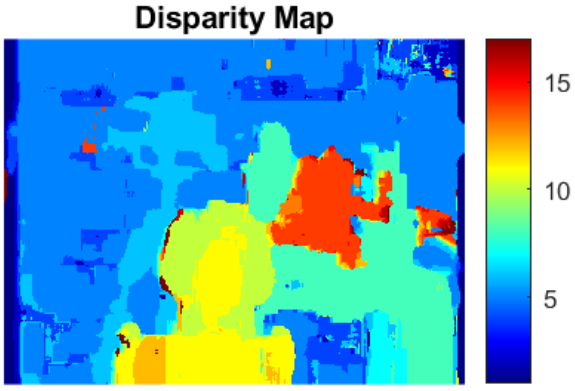


Figure 6: stereo pair 3 and its disparity map

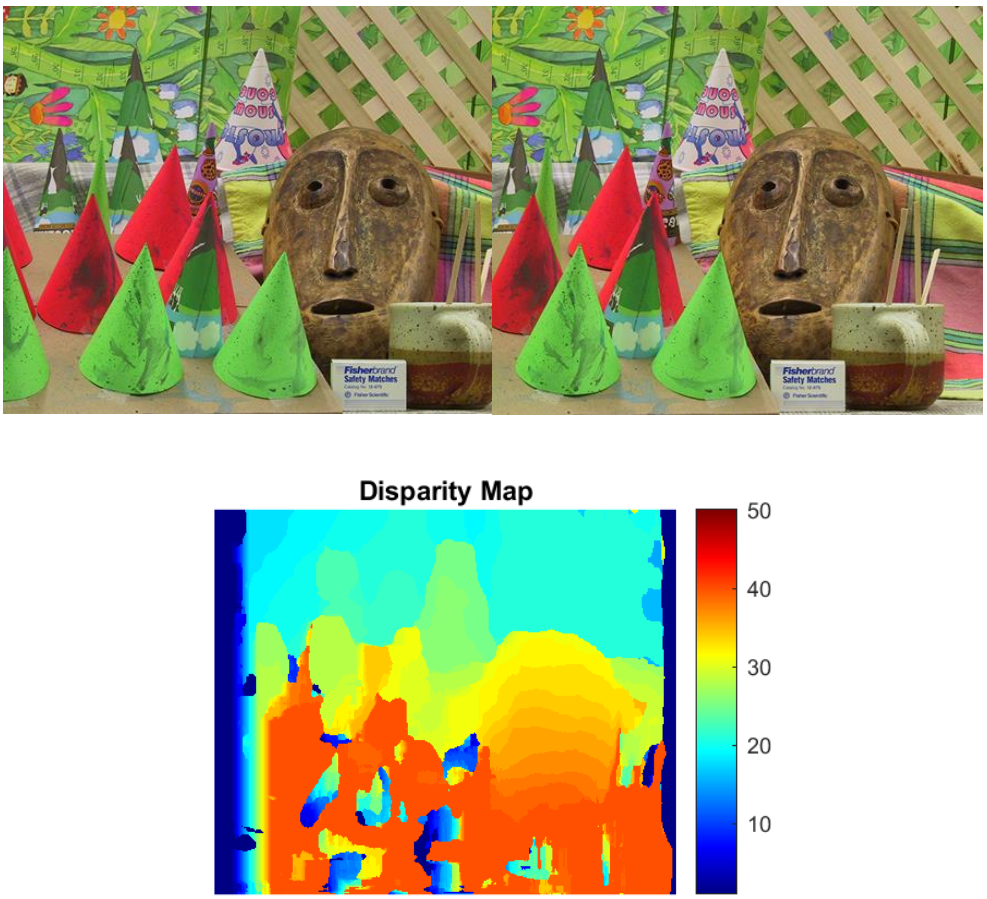


Figure 7: stereo pair 4 and its disparity map

Figure 8 and 9 show the disparity map and depth map of Q3 rectified images. The results are noisy as the shapes of the objects are not clear at all, but we can still get a sense of the relative depth relation among the objects, i.e. foreground vs background. Take stereo1 as example, we can see that the top part of the image has smallest disparity, which should be the door and wall. In contrast, the book, box and monitor part have large disparity shown in orange since they are in front. The right corner should be the table which is in front of the wall, even the monitor. But this party is relatively big so our algorithm cannot find the disparity inside the window, taking it as the background or actually NaN as a result.
We also observe that the algorithm fails when dealing with boundaries of objects. This is because half of the boundary part belongs to another depth or is half-occluded, so the disparity is not consistent or even we cannot find correspondence.

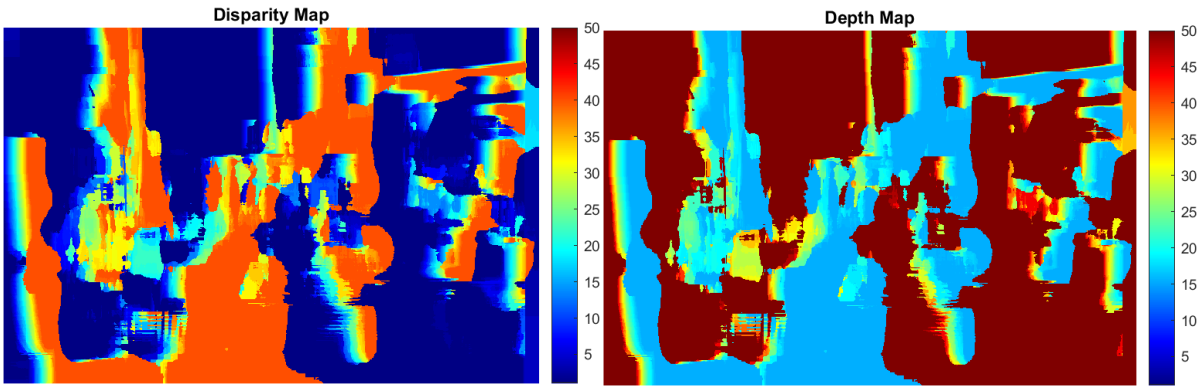


Figure 8: disparity and depth map of stereo pair 1

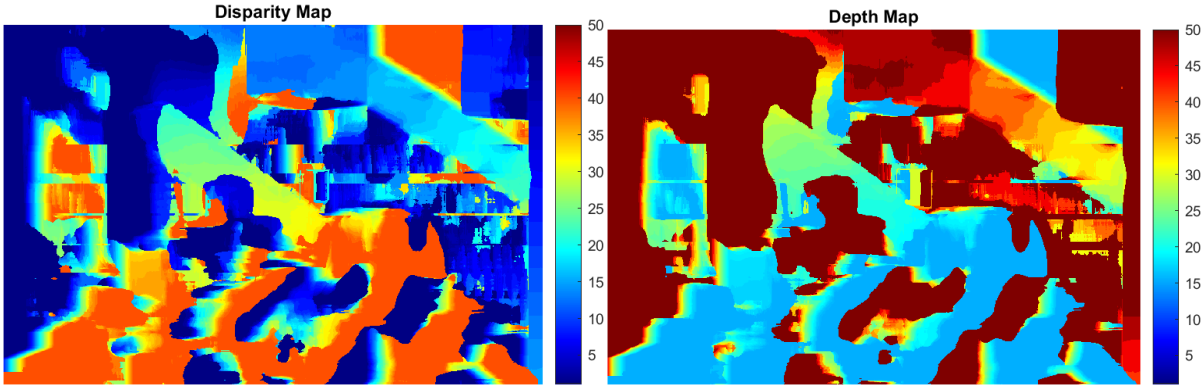


Figure 9: disparity and depth map of stereo pair 2