Homework 3 Writeup

Instructions

- Describe any interesting decisions you made to write your algorithm.
- Show and discuss the results of your algorithm.
- Feel free to include code snippets, images, and equations.
- **Please make this document anonymous.**

Interesting Implementation Detail

Bayer Image Interpolation

When conducting interpolation, I used convolution(correlation) to fill empty pixels effectively. These are two different filters I used. (filter for red and blue pixels are identical.)

```
rb_conv = np.array([
    [0.25, 0.5,  0.25],
    [0.5,  1.,   0.5 ],
    [0.25, 0.5,  0.25],
])
g_conv = np.array([
    [0.,   0.25, 0.  ],
    [0.25, 1.,   0.25],
    [0.,   0.25, 0.  ],
])
```

Also, I padded images with 'reflect' mode before conducting convolution, so that the size of image was not lost.

Stereo Rectification

When I apply 'warpPersepctive' function with original h matrix, upper part of the transformed image may be truncated. This is because some points may be transformed into negative coordinate. In order to fix it, it need to be linearly transformed, and it was done by transformation matrix.

```
move_m = np.float32([
    [1, 0, -left_edge + 10],
    [0, 1, -right_edge + 10],
    [0, 0, 1],
])
img1_rectified = cv2.warpPerspective(img1, h1 @ move_m, (rectified_h + 20, rectified_w + 20))
img2_rectified = cv2.warpPerspective(img2, h2 @ move_m, (rectified_h + 20, rectified_w + 20))
```

Calculate Disparity Map

When calculating disparity map, huge calculations are required due to repetitive operations. In order to conduct these effectively, I saved some frequently used values so that we don't have to calculate it many times. Image differences (each pixel is subtracted by mean of surrounding pixels) and its norm was calculated by convolution and saved.

```
1 img1_diff = cv2.filter2D(img1_gray, -1, avg_filter)
2 img2_diff = cv2.filter2D(img2_gray, -1, avg_filter)
3
4 norm1 = np.sqrt(cv2.filter2D(img1_diff ** 2, -1, np.ones((w, w)) / (w
      * w))).astype(np.float)
5 norm2 = np.sqrt(cv2.filter2D(img2_diff ** 2, -1, np.ones((w, w)) / (w
      * w))).astype(np.float)
```

A Result

Bilinear interpolation

The results of bilinear interpolation was quite good. However, when I zoomed in the result image, I could see some defects around white pixels. I think this is the limitation of bilinear interpolation.

Figure 1: Bilinear interpolation of bayer$_i mg1, bayer_i mg2$

Figure 2: Limitation of bilinear interpolation

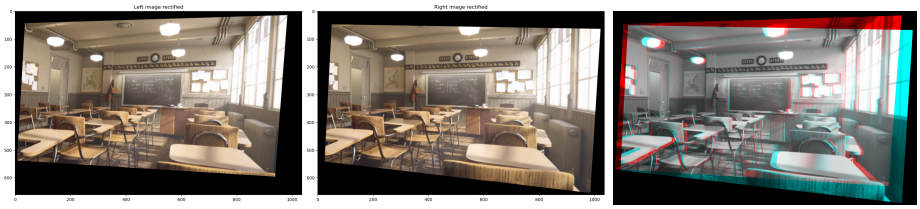

Figure 3: result of stereo rectification

Stereo Rectification

Disparity Map

I didn't changed disparity range, 40, because I think this is appropriate range according to 'true disparity map', whose range is about -40 0. Instead, I changed window size for ncc and agg and found best parameters that minimize EPE value.

First, getting best window size for NCC. According to the result, we can say that window size 41 is the best parameter.

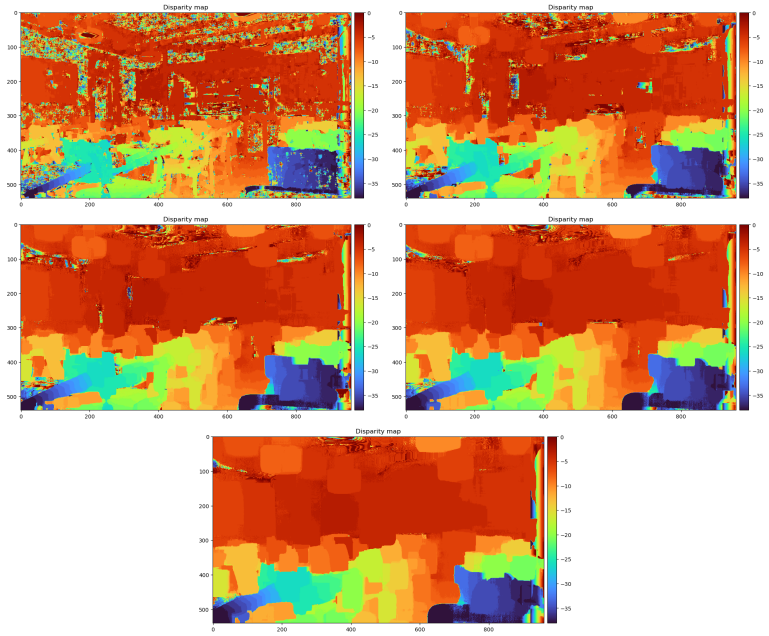

Figure 4: disparity map by ncc window size (11, 21, 31, 41, 51)

And then, I changed agg window size while ncc window size is fixed to 41. The best parameter for agg window size might be 31.

*** Disparity computation time for best parameter: 7.17 seconds**

NCC window size bpr (%)	EPE
11 30.74	4.84
21 22.23	3.48
31 20.00	3.04
41 19.66	2.92
51 20.08	2.93

Table 1: EPE and bad pixel rate by ncc window size

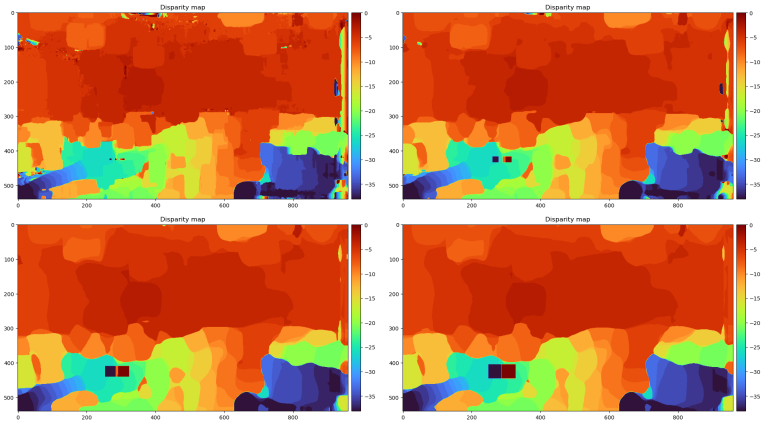

Figure 5: disparity map by agg window size (5,17,31,41)

AGG window size bpr (%)	EPE
13 16.56	2.61
21 15.88	2.49
31 15.61	2.44
41 15.74	2.43

Table 2: EPE and bad pixel rate by agg window size