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.**

In the beginning...

working on the bayer filter task was straight forward. Nevertheless the resulting image confuses me at the first time when i saw it. There are some areas that looks very blue. I Thought it would be a implementation error, but after some research, i figured out, that this might happen due to the reflection part of the image. Also the bayer filter just approximates the missing values, so it is very likely to see some corrupted colors in the image.

Figure 1: Bilinear bayer filter.

I implemented the bayer filter by looping through all the pixel of the provided image and differentiate between the existing pixel. While looping through i build up all three color layers. The approximation is outsourced and very straight forward. In the following is the core function:

```
for i in range(bayer_img.shape[0]):
        for j in range(bayer_img.shape[1]):
            if(i%2 == 0 and j%2 == 0):      # red
                rgb_img[i,j,0] = bayer_img[i,j]
                rgb_img[i,j,1] = calc_green(bayer_img,i,j)
                rgb_img[i,j,2] = calc_red_blue(bayer_img,i,j)

            elif(i%2 == 1 and j%2 == 1):    # blue
                rgb_img[i,j,0] = calc_red_blue(bayer_img, i, j)
                rgb_img[i,j,1] = calc_green(bayer_img,i,j)
                rgb_img[i,j,2] = bayer_img[i,j]

            else:                           # green
                rgb_img[i,j,0] = calc_red_at_green(bayer_img, i, j)
                rgb_img[i,j,1] = bayer_img[i,j]
                rgb_img[i,j,2] = calc_blue_at_green(bayer_img,i,j)

    return rgb_img
```

As one example function the following approximation of the green values. Checking the edge cases and summing up all values. Then dividing by the number of used neighbours is probably not the best approach, but it works :).

```
def calc_green(bayer_img, i, j):
    sum = 0
    count = 0
    if(i > 0):
        sum += bayer_img[i-1,j]
        count += 1
    if(i < bayer_img.shape[0]-1):
        sum += bayer_img[i+1,j]
        count += 1
    if(j > 0):
        sum += bayer_img[i,j-1]
        count += 1
    if(j < bayer_img.shape[1]-1):
        sum += bayer_img[i,j+1]
        count += 1
    return sum/count
```

Interesting Implementation Detail

Working on the fundamental matrix was a little bit more challenging. When i first implement this matrix, my resulting image had some epipolar lines offsets as shown in Figure 2. After debugging my code i finaly realised, that i miscalculated the matrix. So changing from `np.outer(p1, p2)` to `np.outer(p2, p1)` results in the right image as seen in Figure 3.

Rectifying the image was challenging and my result might not be correct. My fundamen-

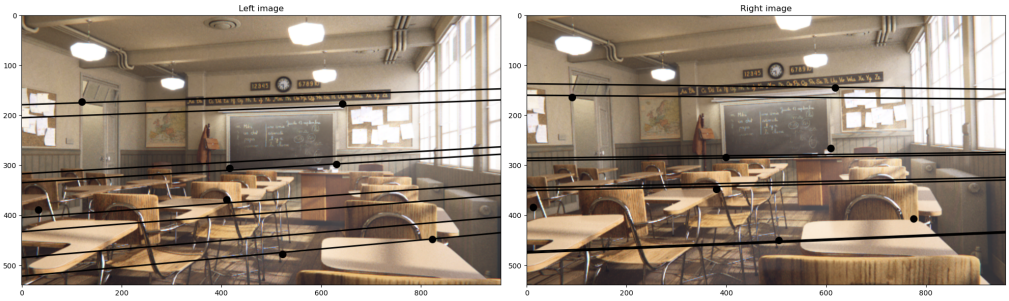

Figure 2: Slightly wrong epipolar lines

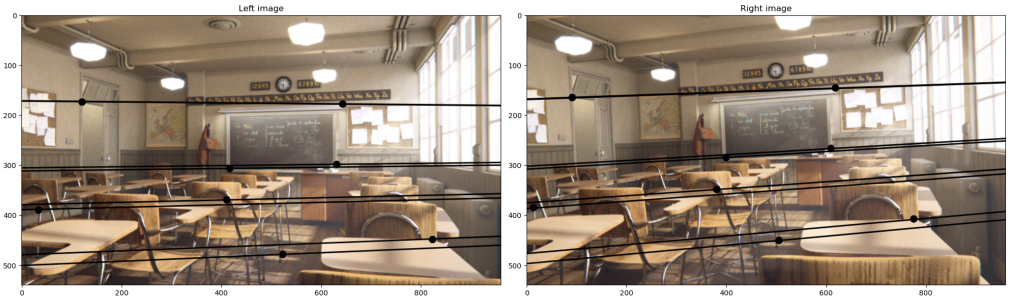

Figure 3: Final epipolar lines

tal matrix between the initial images is

$$\begin{bmatrix} -5.76277528e-10 & 7.41637512e-09 & 1.75709675e-07 \\ -6.67838520e-09 & 3.95133950e-10 & 2.30756555e-05 \\ 1.57683141e-06 & -2.33362693e-05 & 2.67644408e-04 \end{bmatrix}$$

and after the rectification

$$\begin{bmatrix} -5.38484670e-10 & 6.99822217e-09 & 2.61685986e-07 \\ -6.30455169e-09 & 3.75878066e-10 & 2.22806043e-05 \\ 1.41856603e-06 & -2.25191360e-05 & 2.72713084e-04 \end{bmatrix}$$

Rectified images should fulfill some constraints. It has to satisfy

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = He \text{ and } Fe = 0$$

Using this two equation results into

$$FH^{-1} \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} = 0$$

My resulting matrix gives the output

$$FH_1^{-1} = \begin{bmatrix} -6.92031957e+22 \\ -4.59557519e+21 \\ -1.95043071e+19 \end{bmatrix} \text{ and } FH_2^{-1} = \begin{bmatrix} 8.22532072e+24 \\ -1.27867658e+23 \\ 2.55403046e+21 \end{bmatrix}$$

Figure 4 shows the initial image with the final image after applying my rectification shown in Figure 5. As you can see there is still cropping. So far i was not able to find the bug in my code.

Figure 4: Inital images

Figure 5: After rectification

Disparity map

First of all: my computer has a very bad performance. Running the code on a friends computer results in way better times than on my. This made it very hard to know if i fulfill the time limit or not.
I implemented two version of the disparity map. One version uses the NCC as required in the task, the other version uses the SAD, since it is much more preferment (it is out-commend in my code, but can easily be switched to).
For the NCC part i used multiprocessing to parallel the calculation of the NCC values and the Aggregation. I tried to use as many vectorization calculation offered by numpy. The following code shows my multiprocessing. It splits up the calculation for each disparity value and leads to a better performance.

```
1 maps = np.array(pool.map(functools.partial(worker_new, img1=img1_gray,
      img2=img2_gray, rows=rows,cols=cols, window=window), range(
    DISPARITY_RANGE)))
```

My first version gave me just a very blurry disparity map that is shown in Figure 6, after reworking my algorithm using more slicing and vectorization and also using

multiprocessing, i could reach some good results as shown in Figure 7. Using SAD instead of NCC results in slightly better results in fact of disparity map and huge better results in performance. The resulting disparity image can be seen in Figure 8.

Finding the best parameters for the disparity map quickly shows that the best disparity-range has to be 40. Smaller values result in a very noisy image that appears to be more random distributed than calculated. Using bigger values don't use the higher disparity at all.
The window-size is set to 9 and the aggregation is set to 15. This choice seems to be a good mix between 'removing noise' and not 'removing too much details' and scored good results.
Using this parameters my algorithm scored an EPE of 3.5450 and a Bad pixel ratio of 29.19% with a time of $316.77s$. The corresponding image is shown in Figure 7.

My computer has a Intel Core i7-1165G7 @ 2.80GHz and 16GB RAM.

A Result

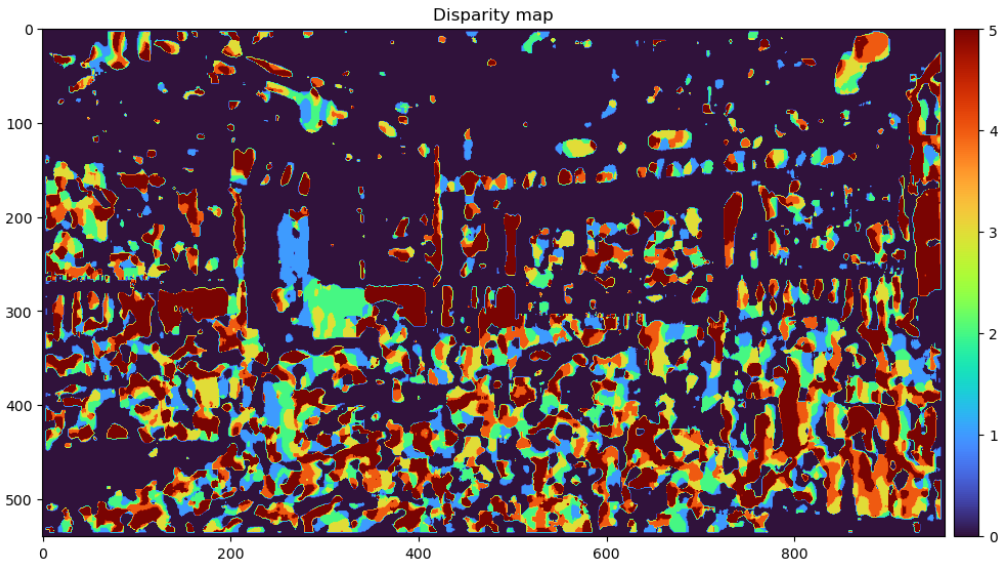

Figure 6: First image (params also not adjusted)

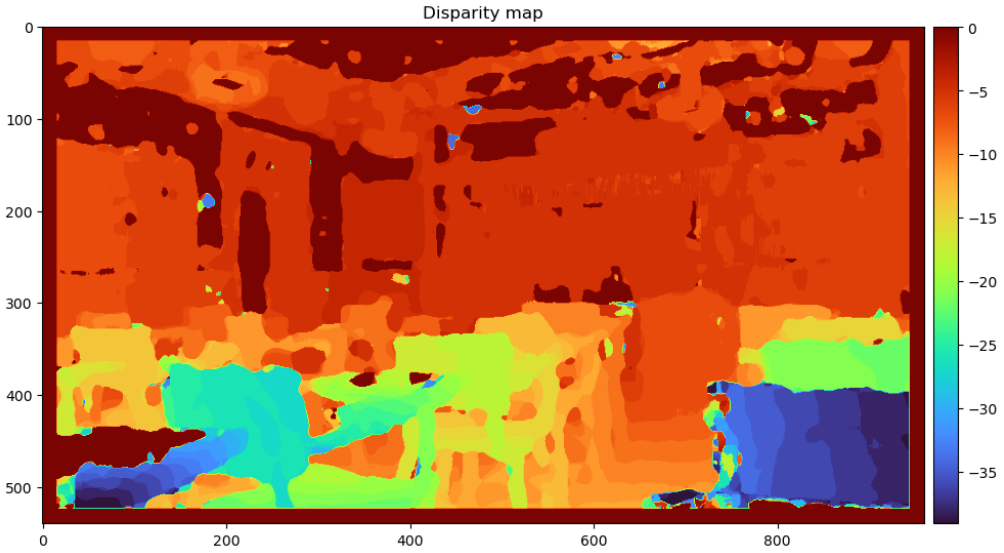

Figure 7: Result with NCC

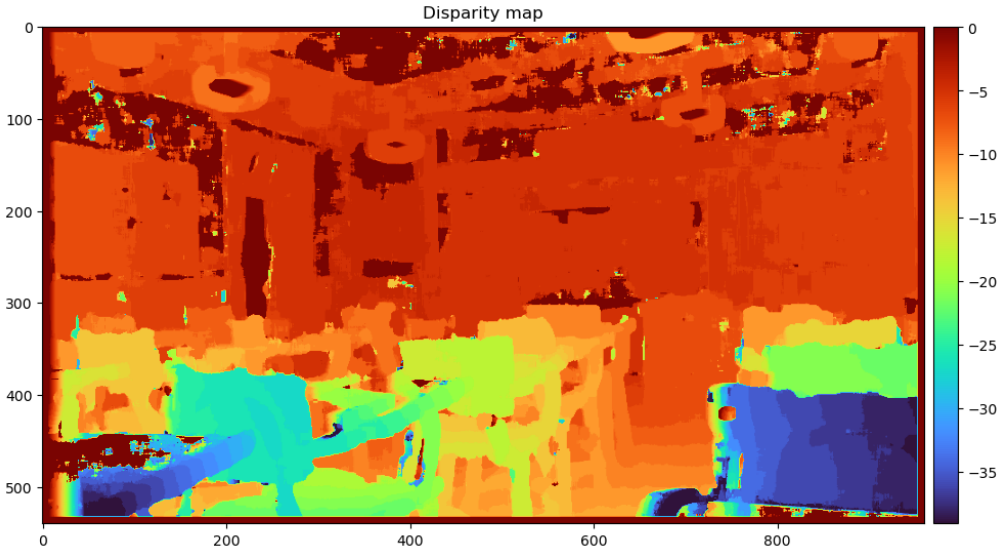

Figure 8: Result with SAD