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.
- Use as many pages as you need, but err on the short side If you feel you only need to write a short amount to meet the brief, th
- **Please make this document anonymous.**

bayer_to_rgb.py

I used bilinear interpolation to compute rgb. I divided the cases for pixel location. If the index of pixel is (i, j) where i and j both even, it meant the image data has value for red so that interpreted the values for green and blue. So the rgb value became (img[i, j], (img[i + 1, j + 1] + img[i + 1, j-1] + img[i - 1, j + 1] + img[i - 1, j - 1])/4, (img[i + 1, j] + img[i, j + 1]. img[i-1, j], img[i, j-1]) / 4). Similar way was used for case when i and j both odd, when the image data represents blue. The rgb value became ((img[i + 1, j] + img[i, j + 1]. img[i-1, j], img[i, j-1]) / 4, img[i + 1, j + 1] + img[i + 1, j-1] + img[i - 1, j + 1] + img[i - 1, j - 1] / 4, img[i, j]) The remaining cases are when the pixel has green value. When i is even and j is odd, the pixel on upper and down side has red value and on right and left has blue value. The red and blue values got computed by adding the values and divide by 2. When i is odd and j is even, the pixel on upper and down side has blue value and right and left has red. Similar way was used for red and blue value computation. When the pixel is on the edge, the number of pixels to add for bilinear interpolation differed. The interpolation took consideration about it, so that no index out of bound error arises. The important thing in converting the pixel pattern was type. The pixel values were unsigned 8 bit so that overflows if trying to add adjacent pixel values. I had to cast it to int to prevent it.

calculate_fundamental_matrix.py

first, I normalized the given points. By using the normalized points, I multiplied two points to get 3 by 3 matrx. It got reshaped into 1 by 9 matrix, and the matrices for 8 points concatenated to form A. I found eigenvector of $A^T A$, f, and reshaped into 3 by 3 to get F. I enforced rank 2 constraint on F, and transformed it into original units using the normalizing transformations I got when normalizing points.

rectify_stereo_images.py

I first used perspectiveTransform to get the coordinate for the edges of each images. Using the coordinates, I could get the bases for deciding how much to translate the rectified images; for vertical movement $min(y) - 20$ and $min(x)$ for horizontal movement. The homography matrix got transformed for the translation by multiplication. Using the new homography matrix, I used warpPerspective to change the perspective of image.

calculate_disparity_map.py

I tried to reduce repeating calculations by computing in advance and saving. The difference of window and its mean was saved, and also the normalized value. Using the saved sum of square roots, correlation of images is computed in triple loop, which takes lots of time, and the other computations is computed in double loop or in no loop. By doing so I could decrease the computing time. To compute disparity map, I calculated NCC. As a result, the computation takes about 160 seconds. Increasing the window size made the disparity map more blurry. Decreasing the size caused more noise. So I used window size 7, same as given size. Also increasing max_disparity makes it seem like the angle between two cameras increases, and decreasing vice versa. Also increasing the value made the result too dark, meaning most of the calculated disparity is relatively small comparing to the max_disparity. Therefore I used max_disparity 40.