CSE472 Assignment 3

Yunhoe, Ku

Due date: December 1st, 2021, before the class

1 Disparity map

1.1 Task 1. Camera center in the world coordinate system

In this task, we manipulate extrinsic matrix because we want to know the perspective of each camera, left one and right one. In general, the extrinsic matrix shows that the matrix of rotation and translation. The form of extrinsic matrix in above case is like this:

$$\begin{bmatrix}\mathbf{I}_{3\times3} & \mathbf{0}_{3\times1}\end{bmatrix}_{3\times4}\begin{bmatrix}R_{3\times3} & T_{3\times1}\\ 0_{1\times3} & 1\end{bmatrix}_{4\times4}$$

In the above case, the criteria of rotation and translation is view of "world coordinate". In this case, we can get the camera center C easily using this formula:

$$C = -R^{-1}T = -R^{T}T$$

But in our problem, the given extrinsic matrix is like this:

$$\begin{bmatrix}\mathbf{I} & \mathbf{0}\end{bmatrix}\begin{bmatrix}R^{T} & -R^{T}C\\ \mathbf{0}^{T} & 1\end{bmatrix} = \begin{bmatrix}R^{T} & -R^{T}C\end{bmatrix}$$

The first matrix of left-side equation is just projection matrix P, which is just matrix to map 3d coordinates to normalized image plane so we don't need to think about it. All we need is just second matrix of left-hand side equation. But the formation of matrix is slightly different with above thing. The reason why formula is slightly different is difference of views. Given matrix, we see the objects in camera coordinate view, not in the world coordinate view. So we just get slightly different equation but the basis doesn't change at all. So we can get the camera center as same way:

$$-R^{T}T = -R(-R^{T}C) = (RR^{T})C = IC = C$$

So, we just multiply $-R$ to $-R^{T}C$ to get the camera center and I get the camera center using this way:

```
data{k} = importdata(append('./data/cam-poses/', f_list(k).name));
cam_center{k} = -data{k}(1:3, 1:3).' * data{k}(:, 4);
```

Figure 1: Matlab code for calculating center of camera

1.2 Task 2. Image rectification

First of all, please keep in mind the first image is just concatenated image with no rectification. You can see there is no **black pixel area** which is induced by image warping like rotation. In this case, we can't make disparity map because it is not well aligned so that we can't pick two points which are in same situation.

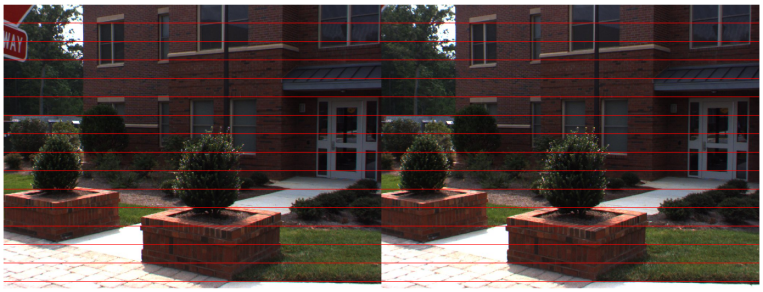

Figure 2: Concatenated image using 132.jpg(left) and 130.jpg(right) with no rectification

And this is the concatenated image of rectification result. You can see the subtle difference, especially black pixel as I mentioned before. Those pixels are affected by image warping also I mentioned before.

Figure 3: Concatenated image using 132.jpg(left) and 130.jpg(right) with rectification

But this kind of result can't make good disparity map because those black region actually doesn't match with any other pixels. So it is probably outlier of disparity map, so we just cut-off them using cropping method. I just crop those images in 700×1000 size which is concluded by simple heuristic.

Figure 4: Concatenated image using 132.jpg(left) and 130.jpg(right) with rectification and cropping

1.3 Task 3. Disparity map

I got three disparity map using different size of window and same metric, SSD. The reason why I run same codes with triple times because I want to see the effect of window size.

First of all, big window size. I use window size as 25 pixels. In general, the bigger size of window is robust to noise but it fail to make details of map and good precision:

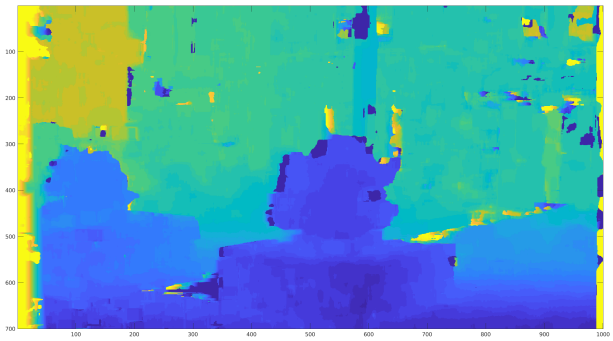

Figure 5: Disparity map using window size 25 and SSD

Next, the result with window size with 13. It makes more details than the before, but at the same time you can see more noise than before:

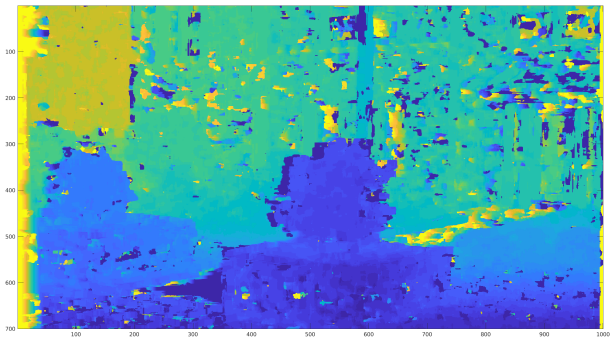

Figure 6: Disparity map using window size 13 and SSD

Last, the result with window size with 5. You can see so many noises and details at the same time so that we can't make good disparity as I mentioned before.

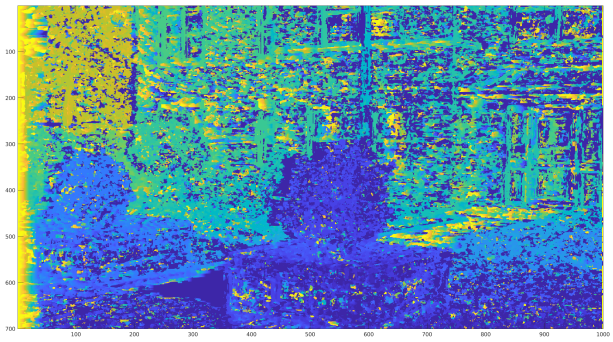

Figure 7: Disparity map using window size 5 and SSD

In this situation, we can think the result with window size 13 can make good result because it is like compromise plan as many people generally think. But the reality isn't match with our expectation. The result shows not so many, but somewhat noises along the disparity map.

There are many reasons will be there, but one of them is the case: In the below size, we can see some red boxes that can't be distinguished easily. The reason is there is not much information to notify us this area is different from other areas. Just using the result of SSD can't make good disparity matching because generally all of window has similar colors in same figures. That can make hard inference of disparity matching and lead us to get noisy disparity map. The solution of this problem is just simply we give some information for those regions like this: *"All regions like window should have same depth compared to nearby regions!"*

Figure 8: Figure to explain the problem of simple disparity matching algorithm

2 3D point reconstruction

2.1 Task 4. Triangulation for 3D coordinate reconstruction

In this task, I followed the instruction below figure:

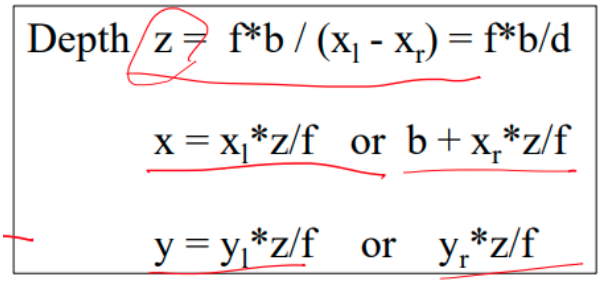

Figure 9: Figure to explain the process to get 3d points from 2D disparity map

And you can see the result below three figures which are visualization of one dimension of 3d points and submitted ply.

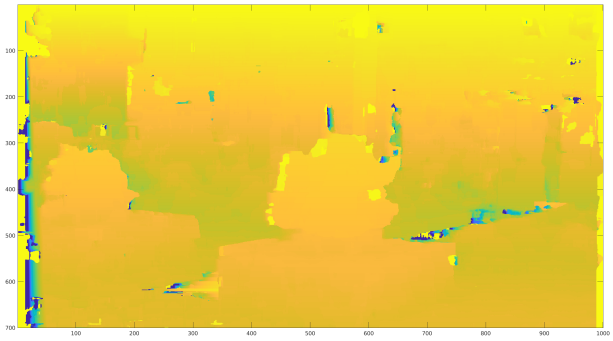

Figure 10: Figure of x coordinates of 3d reconstructed points

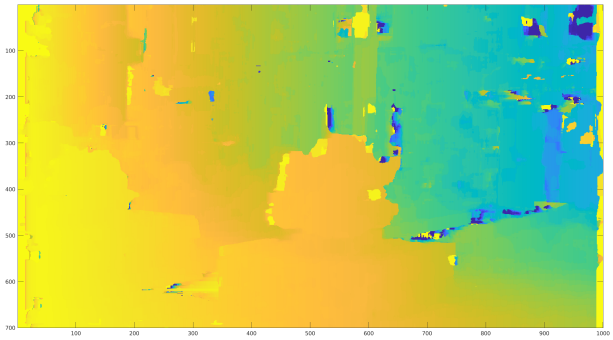

Figure 11: Figure of y coordinates of 3d reconstructed points

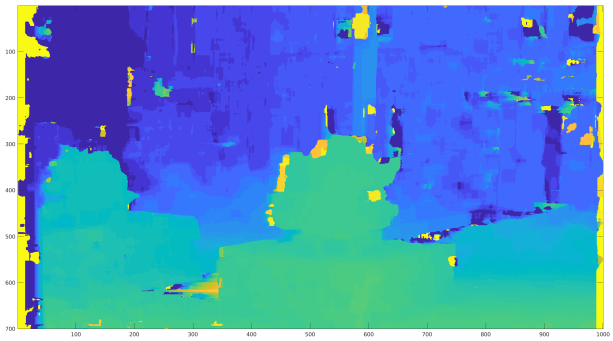

Figure 12: Figure of z coordinates of 3d reconstructed points

2.2 Task 5. Point cloud visualization

In this task, we just use the result of task 4 and visualize it in 3D world. Even though I discard some points which have infinity z values, you can see some noisy points(e.g. redundant points in the sense of depth). I just keep these points because that is a evidence of limitation of our simple methods. The result is like this:

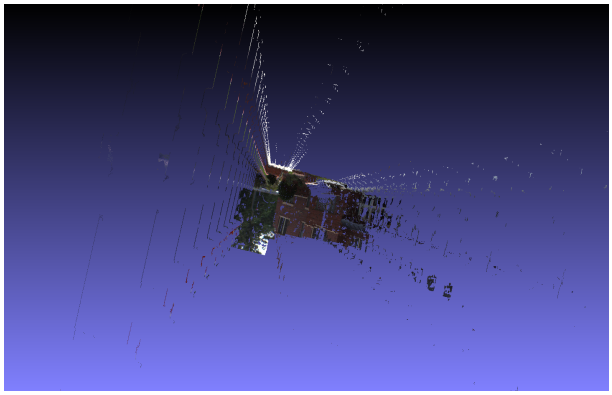

Figure 13: Partial view figure of reconstructed 3d point clouds

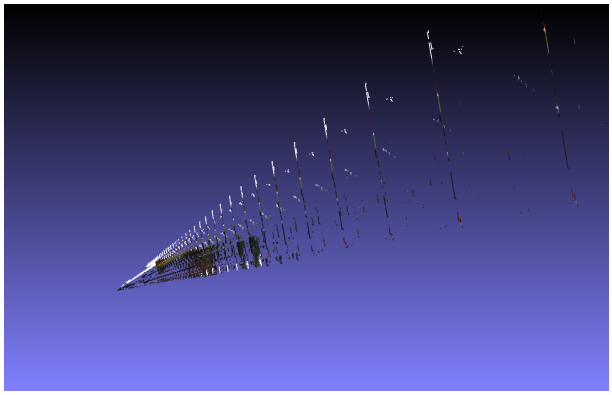

Figure 14: Partial side view figure of reconstructed 3d point clouds

As stated in the description, we can see **stairstep artifacts** and this can be solved in optional task.

3 Extra credits

3.1 Disparity map refinement

In this section, we use pyramid interpolation to solve problem which is called **stairstep artifacts**. By following the formula in the assignment description, we can get result like this:

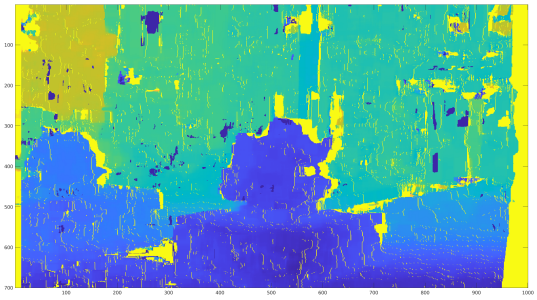

Figure 15: Disparity map with window size 25 using pyramid approximation

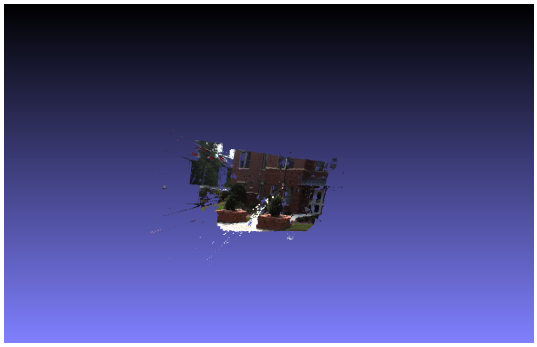

Figure 16: Partial view figure of reconstructed 3d point clouds using pyramid approximation

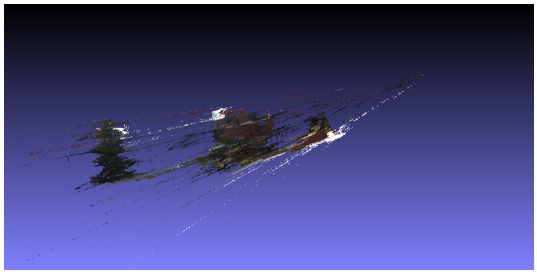

Figure 17: Partial side view figure of reconstructed 3d point clouds using pyramid approximation

We can see result like this: 1) We can get more detailed disparity map even though there is not many noise increasing. 2) More clear 3D map in certain perspective in meshlab. 3) Removing stairstep artifacts in side view of 3d point clouds.

3.2 Scene reconstruction

In this section, I can't implement this method due to lack of time but I think have some ideas:

1. Pick a reference images and compute all disparity map within those pairwise images.
2. Assume that apply above optional task method, we can get several not discrete, but continuous 3D point clouds.
3. Scatter those results in the meshlab and check the outliers like interrupt forming the floor of 3D reconstructed images and so on.
4. Drop them using some clear criterion which is induced from above step e.g.) Remove all the 3d points which have z smaller than 3.
5. Get the good result.

4 Appendix

1. Github link: https://github.com/samsara-ku/unist_cse472