EEP 596 Computer Vision

HW #7

The purpose of this homework is to explore 3D computer vision concepts. You are free to work with fellow students but should turn in your own work. Specifically, you should be able to explain anything in your assignment if asked.

You should turn in your assignment as **“Assignment7.py”, "kermit.ply" and “Report.pdf,”** in Gradescope.

The questions are as follows.

1. **Stereo images. (Ungraded).** Load `tsukubai_left.png` and `tsukubai_right.png`. Convince yourself: Are these stereo images rectified? Why or why not? Convert the images to grayscale for the next few problems.
2. **Scanlines.** Crop out a single row (scanline) of intensities for both images at row = 152 and columns ranging from 102 to 202 (i.e., 100 pixels in each image). Calculate maximum disparity for the pixel strips and return it.
3. **Auto-correlation.** Shift the original right image to the right by 0, 1, 2, 3, …, n pixels, where n = 30 is the maximum disparity. When you shift the image, delete the rightmost columns and fill in the columns on the left with zeros (zero-padding) to ensure that the image dimensions remain the same. You should now have n+1 “right” images, all the same size, each associated with a different disparity. Now compute the pixel-wise absolute difference between each image and the original *right* image. This will yield an absolute difference image. The value of all these images at a particular (x,y) pixel will yield the “auto-correlation” for that pixel. (It’s not really the auto-correlation in the classic sense.) Select pixel (152,152) from your scanline above and plot the 1D auto-correlation (as a function of disparity). Return the auto-correlation values as a list.
4. **Smoothing.** Now we want to smooth our auto-correlation function to reduce the effects of noise. Use box filtering (without normalization). This is equivalent to convolving the image with a square kernel filled with all 1s. Apply such a 5x5 kernel to each of the absolute difference images to smooth them. Don't worry about border effects. Plot the smoothed 1D auto-correlation plot. Return the auto-correlation values as a list.
5. **Cross-correlation.** Now repeat the previous two problems by computing the pixel-wise absolute difference between each of the shifted “right” images and the original *left* image. (That is, the only thing that changes is comparing with the left image instead of the right image. The shifted images remain the same.) The value of all the results at a particular (x,y) pixel will yield the cross-correlation for that pixel. Select pixel (152,152) and plot the 1D cross-correlation. Return the cross-correlation values as a list.
6. **Disparity map.** Now compute the estimated left-right disparity map by selecting, for each pixel, the `arg min` of the smoothed cross-correlation tensor above. That is, for each pixel, select the disparity that yields the minimum value for the smoothed cross-correlation. Return the disparity map as a numpy grayscale image.

7. **Right-left disparity map**. Now compute the estimated right-left disparity map by comparing shifted “left” images with the original right image. You do not need to display any intermediate results for this problem. Return the final right-left disparity map as a numpy grayscale image.
8. **Left-right disparity check**. A classic technique for ensuring robust results is to compare the left-right disparity map with the right-left disparity map, and to retain disparity values only when they agree. Now that you have both disparity maps, this step should be easy. If $d = d_L(x,y)$ is the disparity of the left-right disparity map at (x,y), then this implies that (x,y) in the left image corresponds to (x-d,y) in the right image. Therefore, you need to check that $d_R(x-d,y) = -d$. Use this test to create a cleaned-up disparity map with disparities retained only where they agree, and zero disparities where they disagree. Return your cleaned-up disparity map as a numpy grayscale image.
9. **3D reconstruction.** Use your cleaned-up disparity map to create a 3D point cloud of the scene. Use the colors in the original RGB images to assign colors to the points. Save your result in a PLY file as `kermit.ply` and turn it in on Gradescope. PLY files are ASCII files with a simple format: In the header you specify the number of vertices, along with the properties stored for each vertex (e.g., x y z nx ny nz r g b); then after the header there is one line per vertex. For your assignment, you should just output six columns (x y z r g b) for each matched pixel, ignoring the normal components. Manually set the focal length and baseline to nominal values in order to achieve visually plausible results. Suggestion: Use orthographic projection to get your x,y,z coordinates. Orthographic is simpler and will lead to a more aesthetically pleasing point cloud, even though it is less accurate mathematically than perspective projection.