Exercise 4

Deadline: February 02, 2018

Please send your solutions to threedcv@dfki.uni-kl.de

Dense 3D Stereo Reconstruction

Theory

Normally, in dense 3D reconstruction, one of the challenges is to find the correct correspondences among all pixels between the images. As explained in the lecture, if the epipolar geometry is known, the correspondence search reduces to a 1-D search problem. In a stereo setup (two images only), image rectification may be used to further simplify the search for correspondences.

[T1] What is the advantage of a rectified image pair regarding correspondence search?

[T2] What is the advantage of a rectified image pair regarding triangulation?

[T3] Is image rectification also a good approach in case of multi-view dense reconstruction? Why? (*Hint: Consider a scenario where several images were taken around a statue.*)

Implementation

Goal

The goal of this exercise is to perform dense matching on *rectified* images and reconstruct the original 3D scene.

Requirements

For the remaining tasks, the following packages are required:

- numpy
- scipy
- python-opencv
- MeshLab
  - Ubuntu: `sudo apt-get install meshlab`
  - Windows: download binaries from www.meshlab.net

Datasets

There are two pairs of rectified images. The first pair, from the Middlebury dataset [1], consists of color images. The second pair, from the KITTI dataset [2], consists of two grayscale images. The depth is obtained by

$$Z = \frac{b \times f}{d + doffs},$$

where b is the baseline, f the focal length, d the disparity and $doffs$ the x-difference of principal points, that is $doffs = cx_1 - cx_0$. In case of the KITTI dataset, $doffs = 0$. For more details, check the explanation in the provided source code and the readme files of the datasets in the data folder.

Template

Use the provided template code to implement a dense matching and reconstruction algorithm according to the tasks below (fill in the missing gaps in the code). Note that it is necessary to account for the number of channels depending on the target dataset.

Flowers Dataset

Here, `im0.png` and `im1.png` stand for left and right images, respectively.

[I1] Pixel-based dense matching

1. For each pixel in the left image, find the corresponding pixel on the right image using a *pixel-based* cost function, i.e. the difference between individual left and right pixels.
2. Compute the disparity map and save it as a gray scale image named `pixelDisparity.jpg`.
3. Is the disparity map a good representation of the original scene? Why? (The answer should be included in your report.)

[I2] NCC-based dense matching

1. For each pixel in the left image, find the corresponding pixel on the right image using a *window-based* NCC (Normalized Cross Correlation) cost function. Use window size of 7.
2. Compute the disparity map and save it as a gray scale image named `nccDisparity7x7.jpg`.
3. Experiment with different window sizes: 3x3, 5x5, 9x9 pixels, etc. and save them to `nccDisparity3x3.jpg`, `nccDisparity5x5.jpg` and so on. See if they can give better results compared to window size of 7.
4. Comparing the window-based results to that obtained with the pixel-based cost function of **[I1]**, has the quality of the disparity map improved? Why? (The answer should be included in your report.)

KITTI Dataset

[I3] Pixel-based dense matching

1. For each pixel in the left image, find the corresponding pixel on the right image using a *pixel-based* cost function, i.e. the difference between individual left and right pixels.
2. Triangulate all valid correspondences to reconstruct the original 3D scene.
3. Save at least one screenshot of MeshLab with the reconstructed point cloud and add it to your report.
4. Is the reconstructed point cloud a good representation of the original scene? Why? (The answer should be included in your report.)

[I4] SSD-based dense matching

1. For each pixel in the left image, find the corresponding pixel on the right image using a *window-based* SSD (Sum of Squared Differences) cost function. Use window size of 5.
2. For a given pixel in the left image, if 2 or more pixels in the right image have SSD scores less than $1.5 \times min\_ssd\_score$, consider it as an outlier and reject the pixel. This will serve as a simple outlier filtering scheme.
3. Compute the disparity map and save it as a gray scale image named `ssdDisparity5x5.jpg`.
4. Triangulate all valid correspondences to reconstruct the original 3D scene.
5. Save at least one screenshot of MeshLab with the reconstructed point cloud and add it to your report.
6. Experiment with different window sizes: 3x3, 7x7, 9x9 pixels, etc. and save the disparity maps to `ssdDisparity3x3.jpg`, `ssdDisparity7x7.jpg` and so on.
7. In this case, what is the best window size? (The answer should be included in your report. You may compare the results visually.)

Now, based on your results from both Flowers and KITTI datasets:

- **[Q1]** What is the influence of the size of window on the reconstruction quality?
- **[Q2]** What is the influence of the size of window on the processing time?

Observations:

1. If your algorithm is not fast enough, for the Flowers dataset you may reconstruct areas as small as 200x200 pixels. However, for the KITTI dataset try to reconstruct the full image. As explained in the template code, you should work with downsized versions of the datasets.
2. Make sure your code executes the tasks above by simply calling `python flowers.py` and `python kitti.py` (include the data directory alongside your scripts in your `.zip` file).

Good Luck!

References

[1] `http://vision.middlebury.edu/stereo/data/scenes2014`.

[2] `http://www.cvlibs.net/datasets/kitti/eval_stereo_flow.php?benchmark=stereo`.