MEE / MEEE

Visão Computacional / Computer Vision

Stereo Vision

Stereo vision is the technique where one uses two cameras for depth perception. As such one not only obtains colour information about the scene, but also 3D information. This information can be extremely valuable for scene understanding and for applications such as dimensional quality control. As discussed in the theoretical classes, obtained the 3D information is achieved by determining the disparity between the corresponding pixels in the two images, with the corresponding pixels being found by analysing scanlines. These scanlines assume that the cameras are parallel with each other so, the first step is to transform the images into virtual parallel planes, in case the cameras are not aligned, using the *Fundamental Matrix*. If the extrinsic parameters of the two cameras are unknown, i.e., if the translation and/or rotation form one camera to the other is unknown, then it needs to be computed.

1. Epipolar Geometry

OpenCV provides for the findFundamentalMat function which allows computing the fundamental matrix by providing a set of points. You can choose between the 7-point, 8-point, RANSAC and the Least-Median methods. To test the use of this function, run through the tutorial_py_epipolar_geometry. Beware that, depending on the OpenCV version you are using, you might need to change the line

```
sift = cv.SIFT()
```

with

```
sift = cv.SIFT_create()
```

This tutorial also shows how to use the computeCorrespondEpilines function on one image given corresponding points on the other image. You will find two sets of images provided for this class that you can test the tutorial with.

2. Depth from Stereo

OpenCV provides with several functions which allow to compute depth from stereo, such as the StereoBM (stereo block matching) [1] and the StereoSGBM (stereo semi global block matching) [2].

In the tutorial_py_depthmap webpage you find an example on using StereoBM to compute the depth map. **Run it with the provided images** and point the mouse cursor to different parts of the image to see their distance from the camera on the z axis.

Experiment changing the algorithm to StereoGSBM and compare the results. You can also find a similar example to the previous on in the stereo_match.py example, but one which generates a 3D point cloud and exports it to a PLY point cloud file which you can open with 3D viewing software.

For this class we have also provided for the images *CV_D435_IR_<left|right>_0<0|1>.png* images. These images were taken using an Intel Realsense device placed above the Robotics Laboratory entrance. The images are from the left and right infrared cameras, with the difference between them being that in the "0" ones the infrared projector is turned off, while in the "1" ones the projector is on. **Computed the depth from stereo for both pairs of images** and confirm that, by adding texture to the image using the infrared projector, one is able to have much improved results.

Try now to run the algorithm with the left.jpg and right.jpg images that you used in Section 1. As you can confirm, the results are not good. This happens because the stereo algorithms you have been using expect the cameras to be parallel to each other. When the cameras are not parallel to each other, we need to first rectify the images using the fundamental matrix.

Implement the pipeline shown in Figure 1 to generate the depth map, even for uncalibrated cameras. You have already implemented steps 1, 2 and 3 in the first example of this class (Section 1). Step 6 was also already implemented in the second example of this class (beginning of Section 2). For step 5 you can use the stereoRectifyUncalibrated OpenCV function, which will generate two homography transformations, one for each image, which you can then use as an input to the warpPerspective OpenCV function (as in the previous class) in order to generate two rectified images, one from each image. Having the two rectified images, you can run step 7, i.e., feed those images to the stereo module and compute depth as previously done. Nevertheless, and as you will confirm from your results, having a good hardware placement is important and can lead to much improved results for stereo vision.

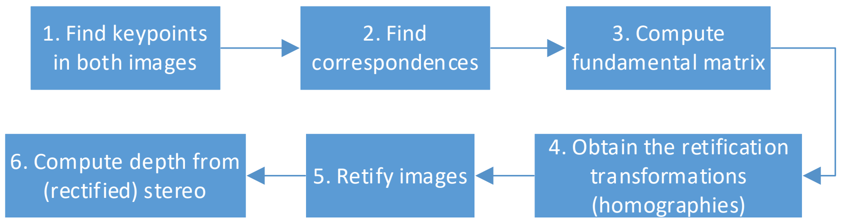
1. Find keypoints in both images
2. Find correspondences
3. Compute fundamental matrix
4. Obtain the retification transformations (homographies)
5. Retify images
6. Compute depth from (rectified) stereo

Figure 1 - Depth from uncalibrated stereo pipeline.

Although not discussed for stereo vision applications, both post-processing and pre-processing filters can have an important impact on the quality of the results, as can be seen, for instance, in the example here.

References

[1] K. Konolige, 'Small Vision Systems: Hardware and Implementation', in *Robotics Research*, London, 1998, pp. 203–212, doi: 10.1007/978-1-4471-1580-9_19.
[2] H. Hirschmuller, 'Stereo Processing by Semiglobal Matching and Mutual Information', *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 328–341, Feb. 2008, doi: 10.1109/TPAMI.2007.1166.