Lab 06 & 07, Stereo Vision and 3D vision

João Santos, *MRSI, 76912*

***Index Terms*—OpenCV, Computer Vision, MRSI, UA, DETI, LaTeX.**

I. Introduction

THIS report is intended to be used alongside the Python3 code developed for this Lab.

The Labs #06 and #07 are introduction classes to stereo vision, 3D vision, OpenCV and Python 3.

This report was written using LaTeX .

II. Lab #06 Exercises

Lets analyse the resolution of the proposed exercises about stereo vision.

A. Ex. 6.1: Chessboard calibration

This exercise is based on the examples of the previous class. The minor difference is that, is this one, it was required to obtain the calibration parameter for both a left and a right camera simultaneously. In order to do so, some changes had to be made to the provided code in order to store the image points of both cameras.

This information will be used in the next exercise.

B. Ex. 6.2: Stereo calibration

The goal of this exercise was to use *cv2.stereoCalibrate()* to obtain the intrinsic parameters of both camera, simultaneously.

To be possible to the calibration algorithm to converge, we had to indicate to the refereed method to use the same focal length, which comes from the fact stated earlier.

The output for the intrinsic matrices were

$$right = \begin{bmatrix} 536.826 & 0 & 334.914 \\ 0 & 535.937 & 242.392 \\ 0 & 0 & 1 \end{bmatrix}$$

and

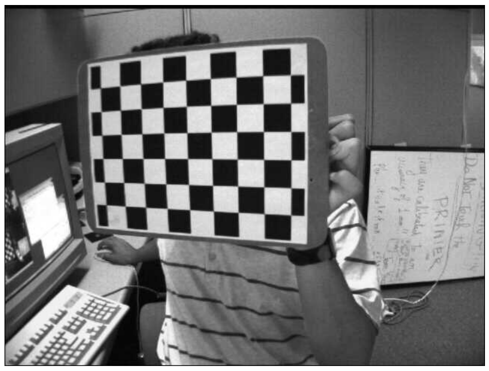
$$left = \begin{bmatrix} 536.826 & 0 & 335.265 \\ 0 & 535.937 & 241.698 \\ 0 & 0 & 1 \end{bmatrix}$$

which are, as expected, very similar given that both camera are of the same specification.

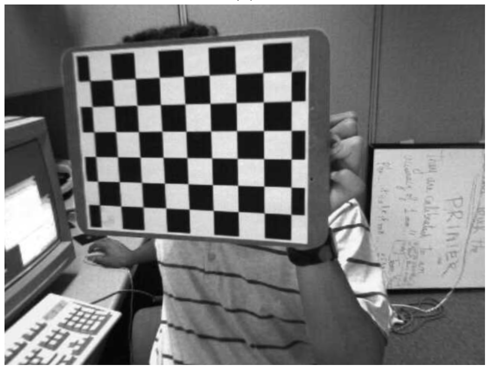
This method also outputs the distortions coefficients for both cameras alongside with rotation, translation, essential and fundamental matrices from between both cameras.

C. Ex. 6.3: Lens distortion

Using the parameters obtained and stored on the previous exercise, we are now able to rectify the lens distortion from both cameras. Figs. 1 and 2 show the output of the operation, in both cameras.

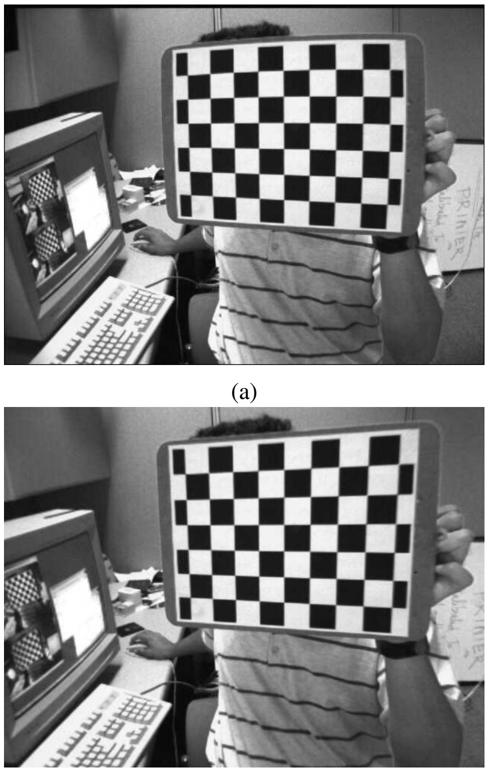


(a)



(b)

Fig. 1: Lens distortion correction for the right camera. (a) Original image and (b) rectified image.



(a)

(b)

Fig. 2: Lens distortion correction for the left camera. (a) Original image and (b) rectified image.

D. Ex. 6.4: Epipolar lines

Building upon the previous exercise, we should now complete it in order to now compute and draw the epipolar lines.

Notice that the method *cv2.computeCorrespondEpilines()* requires, as an argument, the index of the image that contains the points.

Figs. 3 and 4 show to examples of epipolar lines.

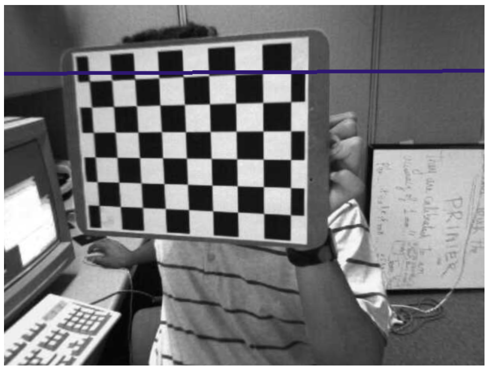


Fig. 3: Epipolar line selected on the left image and drawn on the right one.

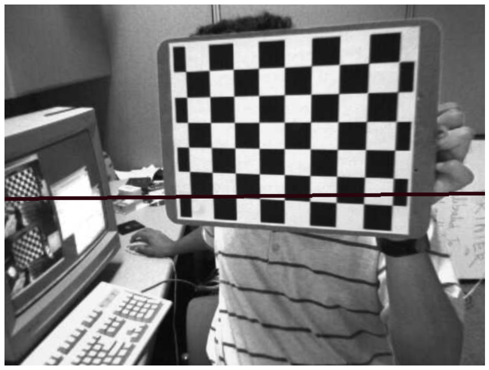


Fig. 4: Epipolar line selected on the right image and drawn on the left one.

E. Ex. 6.5: Image rectification

In the final exercise of this class, the goal was to rectify the images using rectification maps so that, when drawing epipolar lines, they would be not only on the same row within the image but also on both images too.

Figs. 5 and 6 show the outcomes. This particular example shows that the calibration procedure had some mismatch because the selected point in one image was (marginally) outside the epipolar line on the other one.

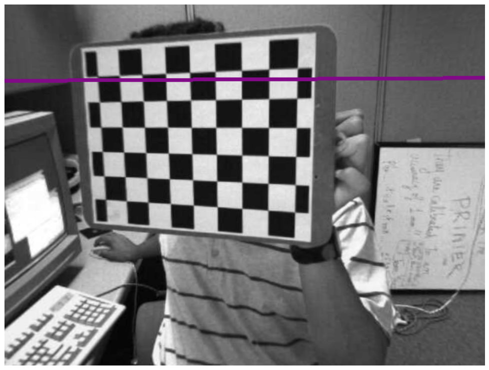


Fig. 5: Epipolar line selected on the left image and drawn on the right one.

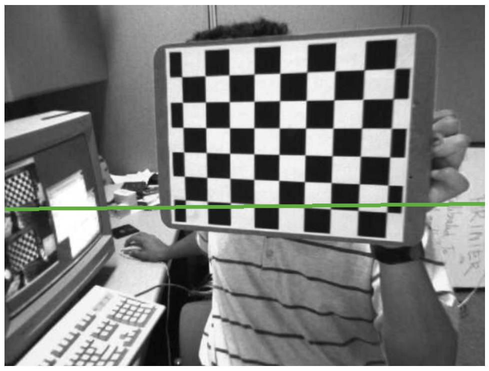


Fig. 6: Epipolar line selected on the right image and drawn on the left one.

III. Lab #07 Exercises

Lets analyse the resolution of the proposed exercises about 3D vision.

A. Ex. 7.1: Disparity map

Using the code listing provided for Stereo Clock Matching, we were asked to compute the disparity map of two gray-level images, taken in the same instant to the same scene, from two different view points.

Given this, the left image on Fig. 7 and the corresponding right image (not displayed) combine to output the disparity map on Fig. 8.

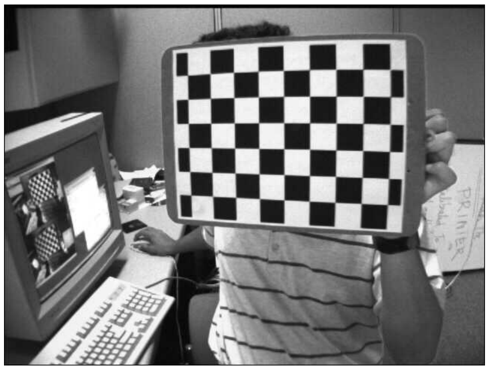


Fig. 7: Left image used for the disparity map.

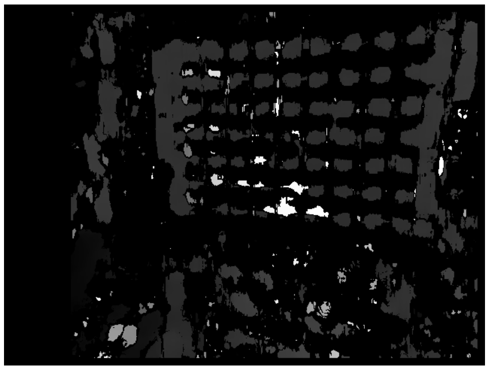


Fig. 8: Disparity map.

B. Ex. 7.2: 3D reconstruction

The *cv2.stereoRectify()* used to compute the rectification transforms for each head of the calibrated stereo camera outputs the Q parameter which is an 4×4 matrix representing the disparity-to-depth mapping. In the previous exercise, this output was

$$Q = \begin{bmatrix} 1.0 & 0.0 & 0.0 & -3.221\text{e}{+}02 \\ 0.0 & 1.0 & 0.0 & -2.431\text{e}{+}02 \\ 0.0 & 0.0 & 0.0 & 5.359\text{e}{+}02 \\ 0.0 & 0.0 & -3.002\text{e}{-}01 & 0.0 \end{bmatrix}$$

which, in conjunction with the disparity map and the *cv2.reprojectImageTo3D()* method, outputs the 3D point cloud coordinates of the scene.

C. Ex. 7.3: Visualization of point cloud

The Open3D library provides visualization tools for the user to take advantage. On Fig. 9, using the mentioned tools, we can see the point cloud built on the previous exercise. Note that the point cloud that we visualized has been filtered to remove both *NaN* and *+/-Inf* values. The author also chose to only display point which have the absolute value of Z coordinate between 25 and 75.

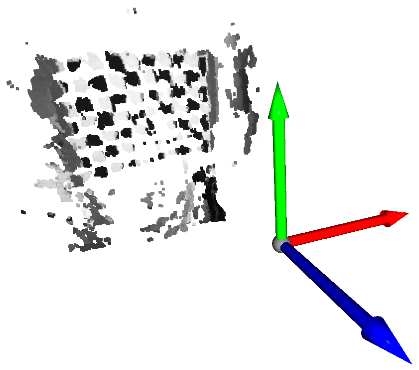
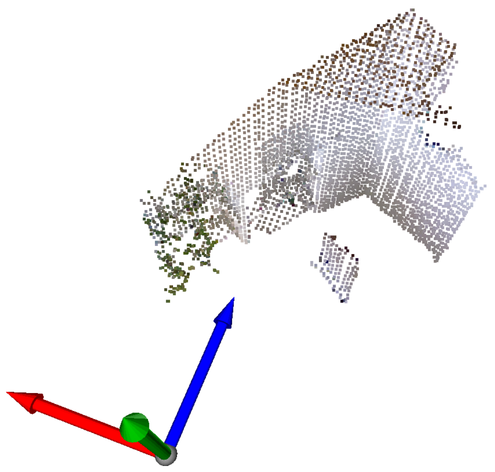


Fig. 9: Filtered point cloud visualization.

D. Ex. 7.4: PCD (point cloud data) 3D format

Open3D also allows us to read and save point clouds in the native and standard format (.pcd) instead of the NumPy way (.npz). Fig. 10 shows two point cloud of a office, taken and viewed from the same perspective (files *filt_office1.pcd* and *filt_office2.pcd*).


(a)

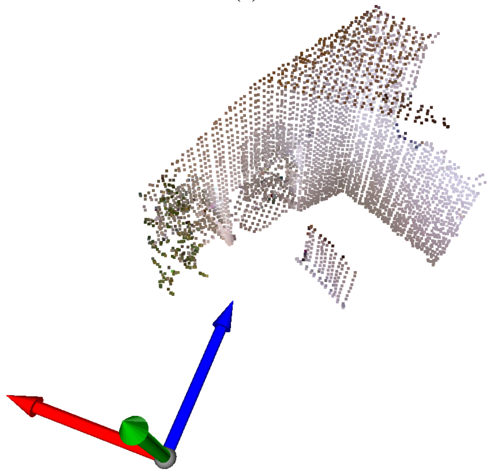

(b)

Fig. 10: Two point clouds from the same scene.

E. Ex. 7.5: ICP alignment

The *registration_icp()* method provided by Open3D allows us to compute the ICP algorithm. This algorithm allows us to align a point cloud.

On Fig. 11 we can see the combined result of the ICP algorithm. The red point cloud is the reference point cloud (the source), in green is the point cloud that we want to align to (the target) and, finally, in blue is the source point cloud aligned to the target.

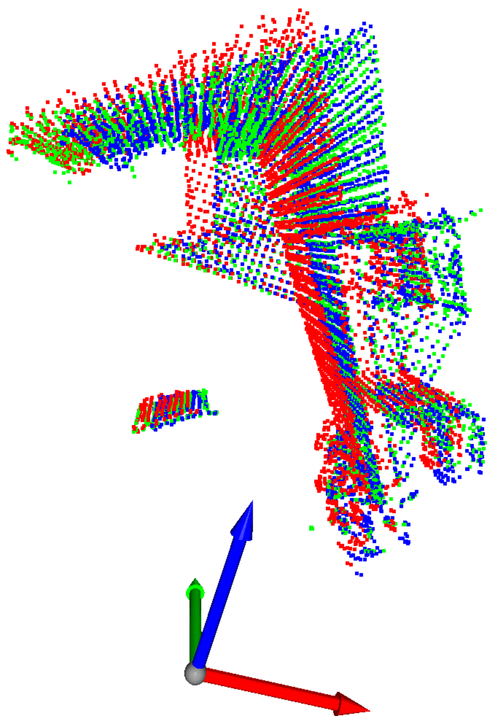


Fig. 11: ICP algorithm output.

The refereed method, using a threshold of 0.1 outputs:

$$fitness = 9.849246\text{e}{-01}$$
$$inlier\_rmse = 2.288494\text{e}{-02}$$
$$correspondence\_setsize = 3920$$
$$transformation =$$
$$\begin{bmatrix} 9.979\text{e}{-01} & 9.784\text{e}{-04} & 6.473\text{e}{-02} & 3.726\text{e}{-03} \\ -1.186\text{e}{-03} & 9.999\text{e}{-01} & 3.183\text{e}{-03} & -8.172\text{e}{-03} \\ -6.472\text{e}{-02} & -3.253\text{e}{-03} & 9.978\text{e}{-01} & -3.038\text{e}{-03} \\ 0.0 & 0.0 & 0.0 & 1.0 \end{bmatrix}$$

Notice that is we used a threshold of 0.01, the result would be very different, and the alignment would not work as expected. An example can be found on Fig. 12.

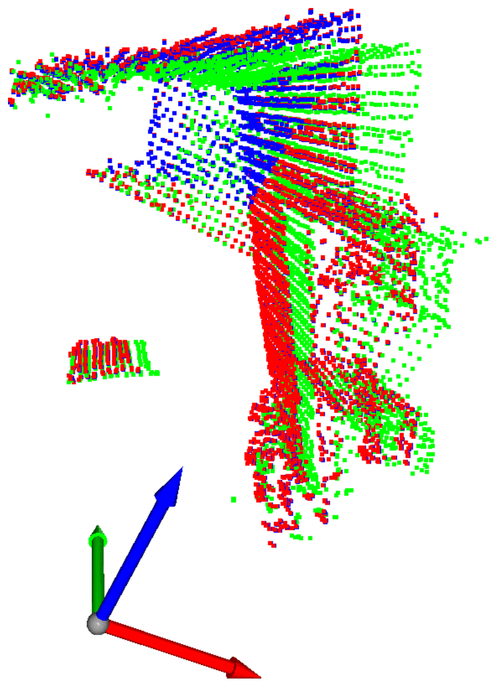


Fig. 12: ICP algorithm output with threshold 0.01.