Robot Vision: Assignment 1

Braulio Sespede, Jesus Soto

April 25, 2020

1 Introduction

In the field of robotics, it is fundamental to make sense of images captured by cameras. In particular, understanding 3D physical coordinates from a given pair of images enables robots to navigate in 3D space. To achieve this goal it is first necessary to extract the parameters that characterize a particular camera mathematically. This is achieved by means of camera calibration, which allows us to describe the projective transformation of real objects in 3D space into a 2D pixel-based image (and vice-versa).

Such camera model is composed by the internal parameters of the capturing lenses and sensors, also known as intrinsic parameters. These intrisic parameters are the different focal lengths f_x, f_y, central points c_x, c_y, as well as the radial k_1, k_2, k_3 and tangential p_1, p_2 distortion coefficients.

In order to extract 3D coordinates from 2D images (without the use of deep learning), it is necessary to capture two or more images of the same scene (captured at the same time). The relative orientation and position of the capturing devices is a key step to make sense of the geometry of a scene, as they enable the rectification of images. We call such relative pose the extrinsic parameters of the lenses setup. The extrinsic parameters can by described by a rotation matrix R, and a translation vector T.

The rectification process, enabled that by the use of extrinsics, allows us to warp the images to undo the effect of tangential and radial distortion, as well as computing stereo correspondences efficiently.

A disparity map can be computed using the rectified image pairs and a stereo matching algorithm that makes use of them. Disparity maps describe the horizontal distance between matching pixels, which can then be used to compute the depth of each pixel in an image, enabling the reconstruction of it's 3D position.

As previous mentioned, dense stereo algorithms try to match every single pixel of the stereo pair to each other. To this end, and considering images are rectified, stereo algorithms look for matching pixels along a 1D horizontal line. This search is usually performed using different similarity metrics between blocks surrounding the pixels. Algorithms such as the semi-global matching (SGBM), then optimize the cost-volume to reduce noise and choose better candidate disparities using dynamic programming techniques [1].

2 Implementation

Our implementation breaks down the different steps of depth estimation into distinct classes using *OpenCV's* functionality. As a side note, we used the *Boost* library in order to perform filesystem operations (e.g. retrieving all the images in a certain path or checking a certain scene has left/right stereo pairs).

2.1 Camera calibration

This can be found in the *CameraCalib* class. This class looks for all the checkerboard images in a given folder, and then uses $findChessboardCorners$ to find all the image points in a given image. Finally, we refine the detected corners to achieve sub-pixel precision using $cornerSubPix$. These image points correspond to 3D coordinates in the coordinate system given by our checkerboard. These correspondences in all given frames (or at least in those where the checkerboard corners are visible), are used to $alibrateCamera$ method to iteratively discover the correct camera parameters that minimize the re-projection error to the images. Finally, we store the results within a struct that contains the results for further use.

2.2 Stereo calibration

This can be found in the *StereoCalib* class. This class receives a folder containing a stereo pair of corresponding checkerboard images, and individually computes the intrinsic parameters of each camera using the *CameraCalib* class. Then, it iterates over the frames to find frames where both the left and right camera have seen the checkerboard. Then the image points found in the previous step are merged into a single vector, and a corresponding vector of the same size is built for the checkerboard 3D coordinates (shown in Figure 1). Finally, we call *stereoCalibrate* in order to compute the relative rotation and translation from the left to right camera, and store the results in a corresponding struct for further use.

Figure 1: Corner detection for camera calibration of a stereo pair.

2.3 Stereo matching

This functionality can be found in the *StereoMatch* class. This class receives a stereo calibration which contains both intrinsics and the corresponding extrinsics to the pair. It provides functionality to rectify the chosen scene by means of the $stereoRectify$ and $initUndistortRectifyMap$ method, which computes the necessary projection matrices using the stereo calibration. Our rectification method then undistorts the image pairs using *remap* and the previously computed matrices (shown in Figure 2. The class also provides functionality to compute disparity maps given a rectified stereo pair. This is implemented using the SGBM algorithm [1]. Finally, it reproject the disparity map into a point cloud format using $reprojectImageTo3D$ and stores it to disk using our $PLYFile$ class.

Figure 2: Rectified scene with the camera calibration parameters.

3 Results

In the following section we will provide the numerical results on the given data. Results are expressed in millimeters or pixels where it corresponds.

3.1 Camera calibration

First we computed the intrinsics for each of the cameras using the corresponding checkerboard images in *calib_data/left* and *calib_data/right*, and the provided checkerboard size. The result of the calibration procedure is shown in Table 1.

Intrinsics (left)		**Intrinsics (right)**	
Parameter	**Value**	**Parameter**	**Value**
fx	1392.593367	fx	1380.541194
fy	1398.827811	fy	1387.160351
cx	1038.206933	cx	1140.343492
cy	649.594959	cy	570.677301
k1	-0.162799	k1	-0.160555
k2	0.002626	k2	-0.004342
p1	0.00035	p1	0.000253
p2	0.000533	p2	0.000019
p3	0.016031	p3	0.01989
RMSE	0.432577	RMSE	0.356803

Table 1: Intrinsic parameters.

3.2 Stereo calibration

Next, we computed the extrinsics between the given pair of checkerboard images. Since the assignment didn't indicate whether we should optimize the previously computed intrinsics during the extrinsics computation, we decided not to do it. Results of this step are shown in Table 2.

Extrinsics	
Parameter	**Value**
r00	0.999998
r01	-0.002028
r02	-0.00009
r10	0.002029
r11	0.999936
r12	0.011128
r20	0.000068
r21	-0.011128
r22	0.999938
tx	-120.174236
ty	0.151629
tz	-3.794028
RMSE	0.495310

Table 2: Extrinsic parameters.

3.3 Stereo matching

Before computing the disparity map, we rectified the images as shown in Figure 2. As can be seen in said figure, the distortion in the images was rather subtle, but present nonetheless. Finally, we experimented with several SGBM parameters and achieved best results (shown in Figure 3) with the configuration shown in Table 3.

SGBM algorithm	
Parameter	**Value**
Block size	5
Min disparity	0
Number disparities	512
SGBM directions	8
SGBM P1	600
SGBM P2	2400

Table 3: SGBM algorithm parameters.

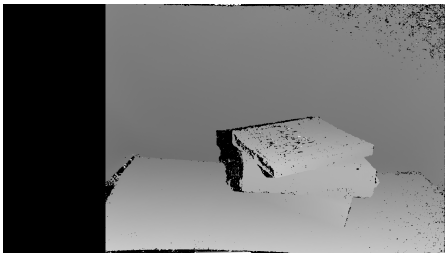

Figure 3: Disparity map

After testing different parameter values, we discovered several things. (i) The *minDisparity* help the algorithm find objects that are further away, but setting the value to a sensible value can reduce interference from out of scene areas and reduces unneeded computation. (ii) The *numDisparities* parameters describe the search range of the stereo matching algorithm and setting it to a sensible value can also reduce unneeded computation. (iii) Setting the *blockSize* can help the stereo matching algorithm identify matching pixels easier, but if set too high, can produce edge fattening effects near boundaries. On the other hand, if its too low, it can produce noisy disparity maps. (iii) The mode of the SGBM algorithm indicates in how many directions the algorithm should optimize during its dynamic programming approach. Using more directions can result in better quality results, as disparity values from neighboring pixels are taken into account. (iv) $P1$ and $P2$ are parameters that penalize big changes in disparity, enforcing smoothness.

3.4 Point clouds

To the end the assignment we projected the disparity map into 3D and measured the size of a book. The resulting measures are shown in Table 4. The measurement was performed using the measurement tool provided by *MeshLab* (Figure 4 and Figure 5 shows the points used for measurement).

Book size (mm)	
Width	135.8
Length	212.8
Height	17.4

Table 4: Measured book sides.

Figure 4: Point cloud of the book image and its measurements.

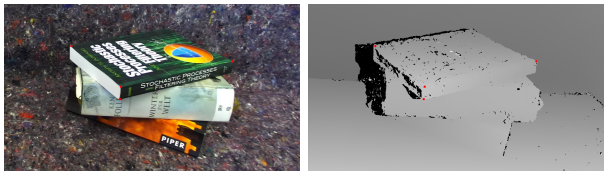

Figure 5: Point cloud of the book image and its measurements.

3.5 Other scenes

To end the assignment, we computed the point clouds for other captures using the same parameters. Among them, we noticed one of that looked particularly wrong (Figure 6). After inspecting the RGB images we concluded that it might be due to the textureless part of the laptop screen, making it hard for the stereo matching algorithm to find correct correspondences.

Figure 6: Laptop scene, showing artifacts in textureless region.

4 Conclusion

To conclude this report, we will mentioned a few takeaways from this experience:

- Camera calibration is a crucial step in the field of robot vision as it can mathematically express the image formation process. Furthermore, it enables the correction of any distortions caused by lenses.
- Even though there are certain limitations to stereo matching algorithms, it is possible to reproject captured images and measure the objects represented in images.
- In comparison to global stereo matching algorithms, SGBM results in excellent performance while maintaining quality. In contrast to local stereo algorithms, SGBM "sees" a bigger picture by including information across a larger amount of pixels, thus resulting in less noisy disparity maps.
- The use of the OpenCV library streamlines the 3D reconstruction of scenes, as it provides an intuitive and simple to use API.

References

[1] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.