Camera Calibration and Stereo Imaging

Andrew Festa
axf5592@rit.edu

Department of Computer Science
Department of Electrical Engineering
Rochester Institute of Technology

August 13, 2019

Abstract

This assignment consisted of exploring properties of cameras and ways in which this information can be applied in order to remove distortions from an image or to compute a stereo image from a set of images taken with various distances between the image centers. The first step was to actually compute the distance coefficient and the camera matrix, which contains information regarding the focal length and optical center of the camera. The second task performed was using this information to undistort an image taken by the same camera as well as highlights the effects of the undistortion. The final task was to tie it all together and to compute the stereo image from a set of images of a single scene taken from the same camera, although at increasing distances between the image centers.

1 Overview

2 Camera Parameters

Every camera, intrinsically, has certain features which distort any image taken from that camera. This manifests in several ways, including radially and tangentially. The first, radial distortion, causes straight lines to appear curved. The other, tangential distortion, occurs due to a misalignment between the vertical plane and the imaging lens. Thus, particularly for stereo imaging, it is of interest to correct for these distortion effects.

2.1 Approach and Experiment

In order to determine the intrinsic distortion effects of a camera, it is useful to consider the effect of each distortion independently[4]. The OpenCV tutorial for this problem [1] shows that the radial distortion effectively is determined by three coefficients. Furthermore, the tangential distortion can be captured in a further two coefficients, leaving a set of equations with five unknown variables. However, this hides the fact that there are several characteristics yet to be determined. Namely, intrinsic and extrinsic properties of the camera, collectively captured by a camera calibration matrix. Fortunately, this information is dependent only on the camera and so can be stored and reused for future runs.

Prior to running the experiment, it bears mentioning that OpenCV recommends the use of at least 10 images when computing the distortion coefficients and camera calibration matrix. Additionally, each input image used to calculate these values should have a set of easily identifiable points. While, theoretically, the algorithm could be extended to also take into consideration depth from the camera, for optimal results, it is desired that each input image is only translated in the xy-plane.

Assuming the above points are adhered to, or at least as much as possible, OpenCV provides useful methods for performing the required steps, the first of which is to locate the points of interest. This is done using the $findChessboardCorners()$ method, which attempts to locate a specified number of corners on a black-and-white grid. For this step, the only additional piece of information required is the pattern to search for. Given the input images used, this was set to be a $(6, 8)$ grid.

Document Version: Write-up
Updated on: August 13, 2019

The second step is to map the points in 3D space to 2D space. At this point, both the image points (the 2D points) and the object points (the 3D locations) are known and able to be passed into OpenCV to compute the distortion coefficients and camera matrix, which, as mentioned previously, can be applied to further images in order to undistort the image.

2.2 Results and Analysis

In implementing this task, the greatest challenge lay in understanding how the camera matrix was represented by OpenCV. Based on their guide, the numerical values reported correspond to the focal length (the values along the diagonal) and the optical center (the values above the 1 in the last column). The values obtained through OpenCV's computation are shown below in listing 1. Based purely on structure of the matrix, this is about as expected. However, it is known that the camera has a focal length of 18 mm. As such, it is not entirely clear as to how these values actually correlate to the physical properties of the camera, even after attempting to comprehend the mathematical formulations. As such, they are taken as accurate and simply applied in further steps.

Listing 1: Camera Matrix

```
[[8.25810410e+02  0.00000000e+00  4.79335239e+02]
 [0.00000000e+00  1.24270281e+03  5.27859023e+02]
 [0.00000000e+00  0.00000000e+00  1.00000000e+00]]
```

3 Image Distortion

After obtaining the intrinsic characteristics of the camera, the next step is to actually apply the information in order to correct for the image distortion. After this step, it is desired that straight lines in 3D space appear straight in 3D and closer points scale correctly with regards to the background.

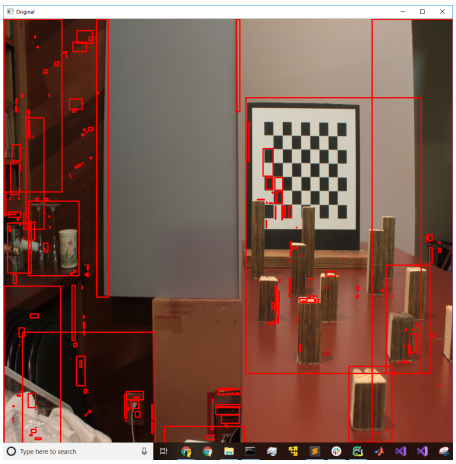
3.1 Approach and Experiment

As with the previous task of determining the camera matrix and distortion coefficients, OpenCV provides a clear and concise walk-through of undistorting an image[1], given the aforementioned values. While OpenCV has several methods of accomplishing this goal, both produce roughly the same results. Thus, the approach taken was the simpler and more straightforward of the two. This requires only applying the built-in function $cv2.undistort()$ on the image of interest. Under the hood, this essentially re-maps each point in the provided image to its corrected location in a new image, using the camera matrix value and distortion coefficients. This remapping is done for each pixel based on equations 1 and 2.

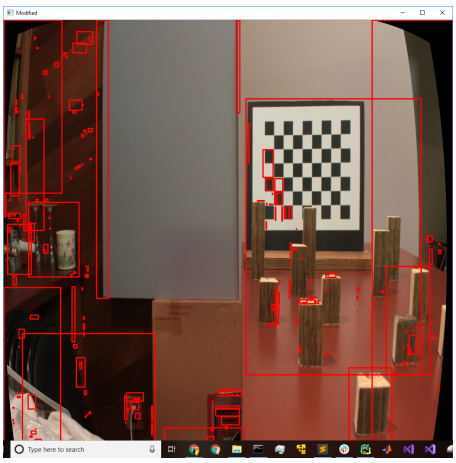
$$\begin{aligned} x_{corrected} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\ y_{corrected} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \end{aligned} \tag{1}$$

$$\begin{aligned} x_{corrected} &= x + 2p_1 xy + p_2(r^2 + 2x^2) \\ y_{corrected} &= y + p_1(r^2 + 2y^2) + 2p_2 xy \end{aligned} \tag{2}$$

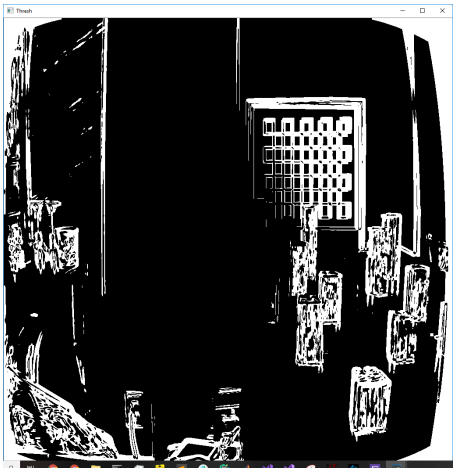
In the above equations, the k_i terms, along with the p_i terms, constitute the distortion coefficients. The camera matrix serves to apply a further rotation and translation based on the focal length and image centers. This is a fairly straightforward computation and, as such, is not computationally expensive. At the end of the process, lines which are straight in the real world should appear straight in the image.



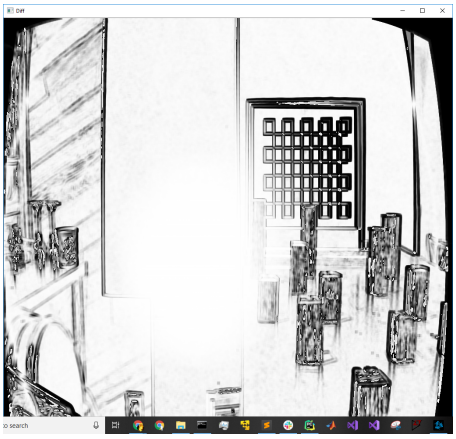
(a) Original Image



(b) Modified Image



(c) Thresholded Image



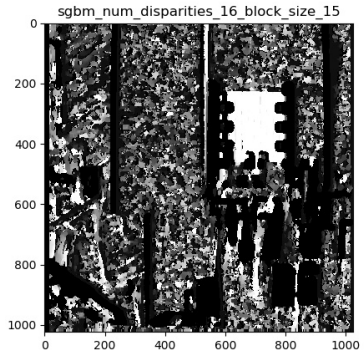
(d) Image Difference

Figure 1: Undistorting Image

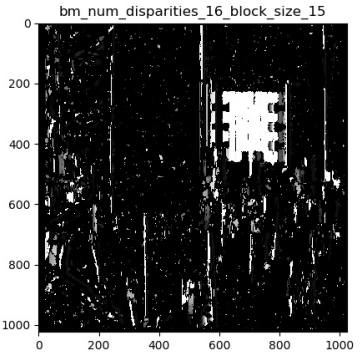
3.2 Results and Analysis

This next step, again, was fairly straightforward with the code. However, the twist was in displaying the images in a manner that highlighted the key differences. For this, it was necessary to fall back on the characteristics desired of an undistorted image as well as the assumption of how distortion is expected to alter an image. It is expected that radial distortion causes straight lines to bend, and this distortion becomes worse for points farther from the center of the image. Thus, an image that best shows a difference once undistorted would be one where there are several prominent straight lines close to the camera (and thus larger). Fortunately, there was one such image in the provided set where a chessboard was rather close and normal to the plane of the imaging lens. The effect of undistorting this image, along with various difference comparisons, are shown in figure 1.

The top two images in the figure show the original as well as the modified image, where each has highlighted those areas which were altered. As can be seen, the corners of the modified image bow in, which is understandable on the basis that the radial distortion is most present at these locations. Thus, when remapping occurs, it has a greater affect on those points which are further from the center of the image.



(a) SGBM: NumDisparities=16, BlockSize=15



(b) BM: NumDisparities=16, BlockSize=15

Figure 2: Initial Disparity Map

4 Disparity Map

Up to this point, effectively only a single image has been considered at a time. However, for a stereo image, the interesting information lies in the difference between two images.

4.1 Approach and Experiment

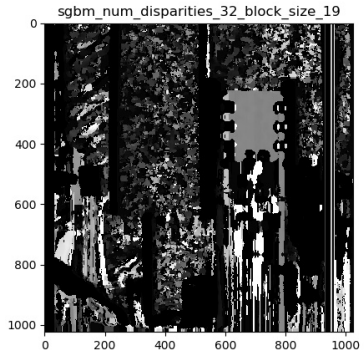
In order to perform the final task for this assignment, the OpenCV tutorials were once again the main source of information[2]. However, this was augmented by code and explanation provided by Timotheos Samartzidis[3]. In the former guide, the guide essentially relies on two function calls: $createStereoBM$ and $stereo.compute()$. The former takes in two parameters which specify how many distinct objects to search for and the block size for over which the algorithm will search. The second method also takes in two parameters in the form of two images (a left image and a right image). The resulting image is, in some regards, a segmentation of the image with respect to depth. The closer an image is, the brighter it appears in the disparity map.

4.2 Results and Analysis

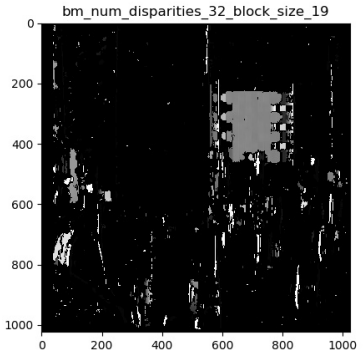
Initially, it seemed that the guides were not usable due to the fact that the methods were unable to be located, and this was not remedied by including the open-contrib libraries. However, it would appear that the guide, as well as the post by Timotheos, are out-of-date with regards to the version of OpenCV used. As such, the actual name of the factory methods for creating the disparity map generator were incorrect. The correct signature of the factory method was found to be $StereoBM_create()$ as well as another for comparison, $StereoSGBM_create()$. Fortunately, these methods took in parameters in the same fashion used by their predecessors and acted in a similar fashion, which reduced the amount of further changes.

Using the parameter values used in the guides ($numDisparities = 16, blockSize = 15$) yielded the results shown in figure 2. However, both of these calls have two degress of freedom in the form of the number of disparities as well as the block size. Note that for these results, the input images were two captures apart. Thus, as each capture was specified as being a distance of 16.375 mm apart, it was assumed that the image centers for these input images was 32.75 mm apart.

By sweeping over possible values for these two parameters, it was explored how they affected the resulting disparity map. The best results were found when $numDisparities = 32$ and $blockSize = 19$, which yielded the results shown in figure 3. However, the actual values for these parameters was of less interest than the

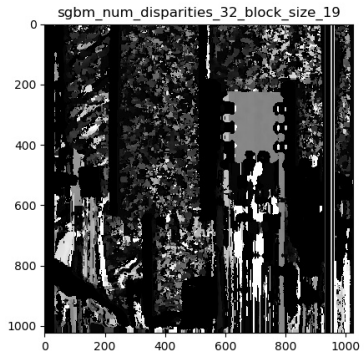


(a) SGBM: NumDisparities=32, BlockSize=19

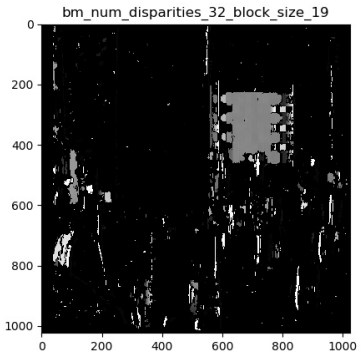


(b) BM: NumDisparities=32, BlockSize=19

Figure 3: Parameter Sweep Disparity Map



(a) SGBM: NumDisparities=32, BlockSize=19



(b) BM: NumDisparities=32, BlockSize=19

Figure 4: Increased Baseline Distance Disparity Map

role they played in the final disparity map created. It was found that the value for *numDisparities* did not have as large an effect as anticipated. This is believed to be due to the fact that each input image had a larger number of object at different depths than the images used by the guides. However, it was found that value for *blockSize* played a rather large role, especially when sweeping over the lower range of values. Beyond about 55, the resulting disparity map tended to stabilize and not change as much. In fact, for the BM computation, it tended to just highlight the chessboard in the background at this point for all values of *numDisparities*. Below this arbitrary threshold of $blockSize = 55$, the disparity maps appeared as if they contained a high level of noise, a level which was worsened at the low range and decreased with increasing values for *blockSize*.

In addition to these degrees of freedom, there were multiple provided images which increased in distance between their image centers. As such, this introduces a third degree of freedom: the distance between the left and right images. As such, the next experiment was to use images which were further apart (65.5 mm, as this is roughly the distance between pupils for the average human). Applying the intuition gleaned from the previous trials with respect to the best values for $numDisparities = 16$ and $blockSize = 15$, this yielded the results shown in figure 4.

With regards to improving the results, it would likely prove beneficial to first align the images prior to computing the disparity maps due to slight differences in height which may be difficult to detect by the

human-eye.

5 Conclusion

While each individual task was able to be completed and built upon in order to complete subsequent tasks, the end result of the disparity maps leaves room for improvement. Part of the issue may be in the difficulty in the supplied test images, as they break several of the assumptions made by the algorithm. One such assumption is that a single object tends to be located at a single depth. However, the board upon which the blocks rest extends from the foreground of the image to the background of the image. Furthermore, there are not many feature points along this board and it is fairly uniform in texture. Thus, the algorithm could reasonably be expected to not perform well on this object of interest.

Still, part of the shortcomings of the results are due to non-optimized pre-conditions for several stages. The most important of this is in not explicitly aligning the stereo images. This is partially implemented by a function which attempts to draw the epilines on the right and left images, respectively. However, this makes use of the SIFT detector in OpenCV, which is not included in the general open-source variant of the distribution, althought another keypoint detector could have been used in its place.

References

[1] OpenCV. Camera calibration. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_calibration/py_calibration.html, 2013.

[2] OpenCV. Depth map from stereo images. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html, 2013.

[3] Samartzidis, T. Open cv stereo – depth image generation and filtering with python 3+, ximgproc and opencv 3+. http://timosam.com/python_opencv_depthimage, 2017.

[4] Wikipedia. Distortion (optics). https://en.wikipedia.org/wiki/Distortion_%28optics%29, 2019.