3D DEPTH VISION WITH 2D STEREO IMAGES

Kanav Singla | 1004997827

This report is about 5 pages with an appendix as the 6th page.

All the code and the video can be found on my Git hub via this link under StereoVision.

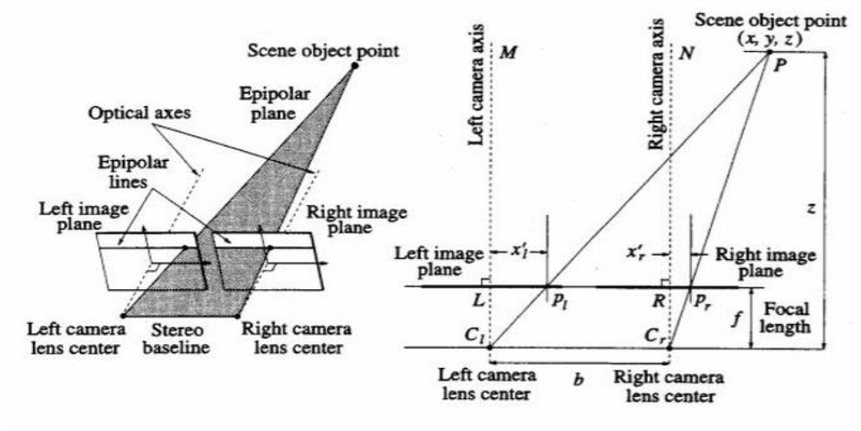

Figure 1: The epipolar lines and triangular geometry. [1]

1 Introduction

Computer vision in its essence aims to achieve the functionality of human vision in perceiving and analyzing the information collected from the 3-dimensional environment around it. Having the ability to process information along these dimensions, especially depth gives robots an exceptional amount of maneuverability and functionality to accomplish daunting tasks.

Stereo vision tries to achieve this by mimicking the human eye's method of 3D reconstruction, by using 2 cameras (spaced at some distance) to provide depth to a pair of 2D images (left and eye right views) of the same subject.

In the context of this assignment, we have been given the dataset of these images. The focal length of the cameras used to take those images is same and they are perfectly aligned horizontally. Keeping these assumptions in mind, the algorithm should be able to demonstrate construction of stereo vision point clouds to depict a 3D space for the corresponding 2D images.

The algorithm basically constitutes of solving two key problems; matching and reconstruction. Matching basically deals with identifying and correlating pixels in one image to the corresponding pixels in the second one. This is achieved by a method called SAD (sum of absolute differences), which gives us a set of matched pixels from which we can calculate the corresponding disparities. Reconstruction uses the results from matching to map a 3D world by giving the image a depth perception.

In the following report, we shall be discussing this algorithm in more detail alongside a discussion of a background theory that is used by the algorithm. Furthermore, we shall analyze the approaches taken, acknowledging the limitations and hence helping us draw insightful conclusions.

2 Background

The above-mentioned sets of assumptions imply that both images are perfectly perpendicular to the same epipolar plane hence giving us a horizontal epipolar line (at the intersection of the Epipolar plane and image planes). Hence, the matched pixels shall lie on this line and the matching is perfectly horizontal. The features of the object point lying on this plane can then be geometrically calculates by using the geometry of similar triangles/triangulation.

The diagram on top of this page has been taken from a chapter in the book Machine Vision, [1]. Applying similarity of triangles on two pairs of triangles PMC_l, P_lLC_l *and* PNC_r, P_rRC_r gives us the following:

Using the first pair of triangles: $\frac{x}{z} = \frac{x'_l}{f}$

Using the second pair gives us: $\frac{x-(N-M)}{z} = \frac{x'_r}{f}$

Combining these, gives depth: $z = \frac{f(N-M)}{x'_r - x'_l}$

N-M can also be written as b, the baseline distance; which is basically the distance (in mm) between the two cameras. ***f*** is the focal length of the camera being used and $x'_r - x'_l$ is the relative disparity difference for the specific pixel in two images.

The matching part of the algorithm helps us find the two matching pixels/sections from which we can calculate the corresponding disparity. This disparity ultimately using the above equation provides us with depth.

3 Process Description

The algorithm is implemented by the following steps:

3.1. SAD Block Matching

The aim of this step is to be able to match features in the two pictures which are horizontally shifted. Which breaks down to essentially choosing a pixel in the right picture and then trying to match it to the corresponding pixel in the left one.

But the above process can lead to an extreme time complexity making the search practically impossible. So, we deal with this in following ways: Firstly, by downscaling the original pictures resulting in lest pixels to be compared reducing the time complexity of the complete algorithm. We end up using a downscale of half for our final algorithm. To support this, time analysis on different image sizes is provided bellow [source].

Secondly, by comparing regions instead of individual pixels. So as seen in the figure bellow, we will be taking a small region of pixels in the right picture and search for the closest matching region of pixels in the left picture. The white box in the bottom picture (right view) is compared to boxes in the above picture (left view) to find the closest matching box.

Figure 2: The image at top and bottom corresponds to left and right views respectively depicting block matching.

Thirdly, since we assumed that the pictures are only horizontally shifted, we can ignore any vertical disparities hence narrowing the search space which leads to a practical time complexity. This leads to a row wise search, so starting with the same coordinates on the left picture we go rightwards up to the maximum allowed disparity and choose the closest matching block to the white block in the right picture. Now that we have set up the structural framework for the search, we are going to explore the actual comparison of these block windows to find the closest match.

Block Comparison

We create a template region window in right image of an appropriate size (5 x 5 is used in my case), which is compared to a same sized window in the left image. For each comparison, we use an operation called SAD (Sum of absolute differences). We take the sum of an absolute difference of each pixel in the template window (in right image) to the corresponding pixel in the block (left image). The images are converted to grayscale beforehand so that the intensity of each pixel ranges from 0 to 255. A lower sum basically depicts less difference and more similarity of the blocks giving us the closest matching block in the left image. So, all the SAD values are computed for the potential matching blocks, but we end up choosing the lowest SAD value block as the one closest matching to the template.

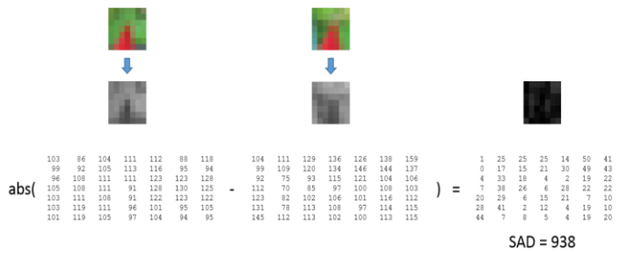

Figure 3: Depiction of SAD on blocks [2]

3.2. Disparity, Depth & Coordinates

Once we are done with matching and comparison of one region, we can calculate the disparity as the horizontal pixel distance between the centers of white and green block in the figure 2.

Now there are two approaches that were considered in getting the disparities of the whole picture. Firstly, assigning this disparity value to a specific coordinate (a single point) and then iterating to the next column for the same row and repeating the region matching process again to get another disparity value for the next point. This approach is very time costly and requires sub pixel estimation for a reasonable result.

Figure 4: Downscaled input image in grayscale used for disparities and depth map.

The second approach rather populates the complete white region with the same disparity calculated from one round of region matching, hence we can iterate in blocks instead of single points reducing the time complexity significantly. By setting the appropriate parameters such as the block size (relatively small so that not a lot of information is lost) and maximum disparity, this approach gives us similar results to the first approach. We end up using this approach for getting the disparity map.

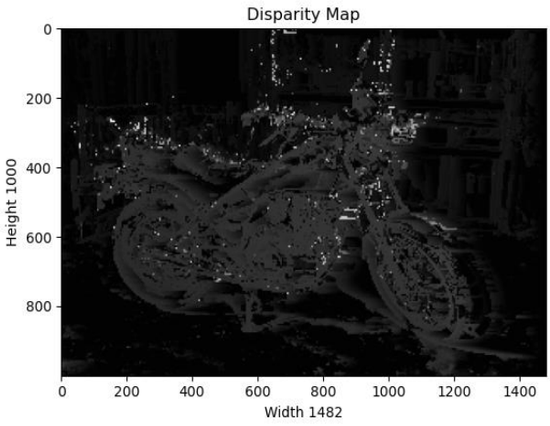

Figure 5: Disparity Map.

As seen above the computed disparity map is in grayscale and we can see that disparities of the objects closer to the camera is more (hence they are whiter) compared to once in the back. This agrees with the depth formula, $z = \frac{f(N-M)}{x'_r - x'_l}$

Couple of things worth mentioning here is since we downscaled the image, we had to change certain parameters (such as maximum disparity allowed, and block size) proportionately from the ones provided in the dataset. Secondly, since there isn't much texture in the pixels on the ground right in front of motorcycle (there are no distinct features), it becomes hard to match similar Regions. Hence, the results from the disparity values are quite inconsistent to the actual depths. Masking these values (by setting these disparities to be high) gives us a much cleaner disparity and depth map.

The depth map can now easily be constructed from the disparity matrix, by applying the depth formula for each disparity. Once the depth matrix is formed, we can map it in 'jet_r' color to get an idea of how different objects lye at varying depth in the space.

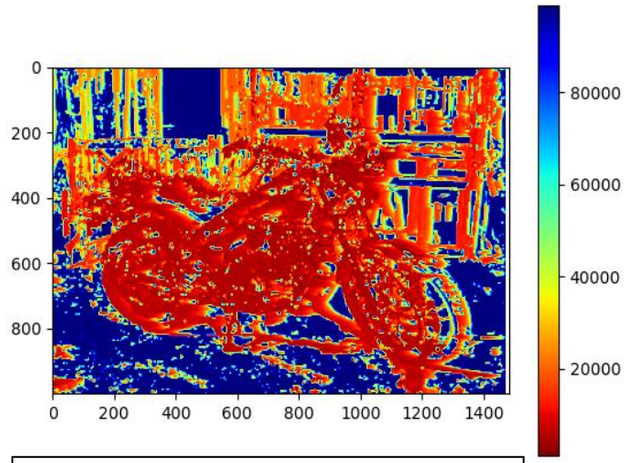

Figure 6: The Depth Map in jet-r scale.

As seen in the above image, the features are clearly differentiated in depth. The box shelves and the bench are with a lighter shade of red compared to the actual motorcycle which is closer to the camera.

Next we compute the exact coordinates of each pixel in the scene seen keeping the x and y centerline offsets in mind. These coordinates are later used to construct point clouds. We will be using the formulas we derived above to get the following coordinates:

$$z = \frac{f(N-M)}{x'_r - x'_l}$$

$$x = z\frac{x'_l}{f} \qquad y = z\frac{y'}{f}$$

parameters (such as maximum disparity allowed, and block size) proportionately from the ones provided in the dataset.

3.3. Point Cloud Construction

Given the coordinates as calculated above, we the location of each pixel in 3D space for the point cloud. Now all we need in addition to this is the exact R G B values for each pixel to be potted. We can get these by assessing the original colored picture downscaled. The only thing to consider is that OpenCV represents images as BGR so we need to invert the indexes when assessing the specific RGB values. We make a ply (polygon format) file with each row containing the respective X, Y, Z (coordinates) and R, G, B values concatenated.

The output ply file can then be graphed using any ply file plotters. The point cloud shown bellow was constructed by opening the ply file in MeshLab.

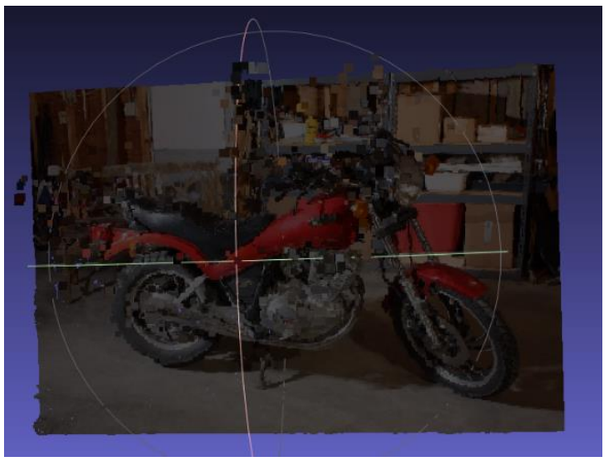

Figure 7: Straight view of the point cloud

Other views can be seen bellow:

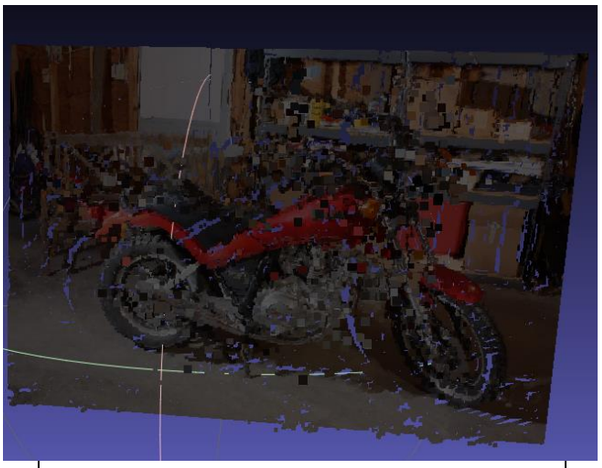

Figure 8: Slightly right rotated and zoomed in view

3.4. Limitations & Improvements

The algorithm described above provides us with reasonable results in most of the sections but still has certain drawbacks. While the maps came out decently clean, they can be improved by having more data points to fill in the visible gaps and make the maps smooth. Although as noted above, using the original picture without downscaling will result in an impractical compute time. For reference, it took the above algorithm about 93 mins [Appendix 1] just to get the disparity map with no downscaling (on intel core I7 8th Gen processor). Using better hardware can result in more realistic compute times and hence we can get more pixels to fill in the gaps producing smoother maps.

The point cloud seems to have some discrepancies especially around the edges of various features. Since the depth values are discrete numbers rather than being continuous, it results in multi-layering of the point cloud in depth axis (figure 9). Since the expectation is a continuous 3D surface, having more points to fill in the layered gaps could help in overall smoothening.

Other limitation of the point cloud is that it is susceptible to some noise. There seems to be some redundant pixels (especially in front of the motorcycle as seen in figure 8 & 9) that can be filtered/masked out from the depth map using different metrics resulting in less noise. An example of one such metric is masking out all the points with zero or illogical disparities. Alongside, we can also mask off the points with disparity values way above the average.

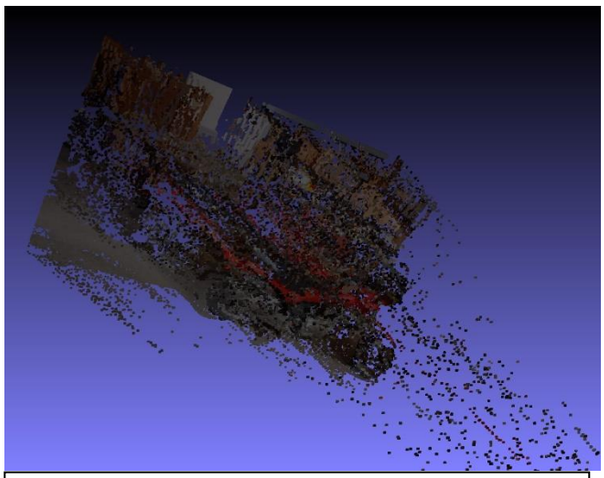

Figure 9: Left and top rotated, zoomed out view

4 Image Rectification

In this report, one of the biggest assumptions we had was that that the image planes are parallel to the camera plane resulting in a horizontal epipolar line. As a result of which, matching points were found to be on the same horizontal line. Although this assumption made our task in hand easier, it's not a practical assumption. In real world, the robots using these algorithms could have cameras have a relative orientation in respect to each other as compared to just a horizontal orientation seen in our case.

To deal with such a problem, if we know the relative orientation of the image planes; we can quantize this orientation in the form of different matrices. These matrices can then be used to map the images onto the same plane. In this way, the respective epipolar lines can be wrapped such that the matching points on the same line [3]. The success of this approach shall heavily depend on precise calculations of this relative orientation between cameras. Once the images are rectified, the above algorithm can be implemented for reconstruction of 3D spaces around these mobile robots.

References

[1] Chapter 11 depth, individual assignment recommended resource. https://www.cse.usf. edu/∼r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter11.pdf.[Ac-cessed on 10/4/2020].

[2] Stereo Vision Tutorial - Part I. (2014, January 10). Retrieved from http://mccormickml.com/2014/01/10/stereo-vision-tutorial-part-

[3] A. Fusiello, E. Trucco, and A. Verri. A compact algorithm for rectification of stereo pairs. Machine Vision and Applications

Appendix:

1.Runtime of the disparity algorithm for a fully scaled image is about 93 minutes.

```
PS C:\Users\Kanav\Desktop\1\begin\Praxis> & C:/Python/Python38/python.exe c:/Users/Kanav/Desktop/1/begin/Praxis/main.py
Row number 0 Percent complete 0.0 %
Row number 50 Percent complete 2.5 %
Row number 100 Percent complete 5.0 %
Row number 150 Percent complete 7.5 %
Row number 200 Percent complete 10.0 %
Row number 250 Percent complete 12.5 %
Row number 300 Percent complete 15.0 %
Row number 350 Percent complete 17.5 %
Row number 400 Percent complete 20.0 %
Row number 450 Percent complete 22.5 %
Row number 500 Percent complete 25.0 %
Row number 550 Percent complete 27.5 %
Row number 600 Percent complete 30.0 %
Row number 650 Percent complete 32.5 %
Row number 700 Percent complete 35.0 %
Row number 750 Percent complete 37.5 %
Row number 800 Percent complete 40.0 %
Row number 850 Percent complete 42.5 %
Row number 900 Percent complete 45.0 %
Row number 950 Percent complete 47.5 %
Row number 1000 Percent complete 50.0 %
Row number 1050 Percent complete 52.5 %
Row number 1100 Percent complete 55.0 %
Row number 1150 Percent complete 57.5 %
Row number 1200 Percent complete 60.0 %
Row number 1250 Percent complete 62.5 %
Row number 1300 Percent complete 65.0 %
Row number 1350 Percent complete 67.5 %
Row number 1400 Percent complete 70.0 %
Row number 1450 Percent complete 72.5 %
Row number 1500 Percent complete 75.0 %
Row number 1550 Percent complete 77.5 %
Row number 1600 Percent complete 80.0 %
Row number 1650 Percent complete 82.5 %
Row number 1700 Percent complete 85.0 %
Row number 1750 Percent complete 87.5 %
Row number 1800 Percent complete 90.0 %
Row number 1850 Percent complete 92.5 %
Row number 1900 Percent complete 95.0 %
Row number 1950 Percent complete 97.5 %
elapsed time... 92.93517645200093 mins
Disparity map....

Calculating depth....
```