Stereo Vision: 3D Vision with 2D Cameras

Young Seok Seo

April 9, 2020

1.0 Introduction

Many complex tasks in computer vision involve obtaining three-dimensional coordinates of an object. This project uses mathematical relationships along with various software tools to create a 3D image from two 2D images.

2.0 Theory

The motivation for computer stereo vision comes from stereopsis, which is how human eyes perceive depth. Through the left and right eyes, humans see the same object in slightly different ways; this is due to the effect of parallax, where the apparent position of an object changes based on the angle of the line of sight. With the distance between two eyes and the displacement of the object position in each eye, trigonometry and ratios can be used to gain depth.

To obtain a 3D image using computer stereo vision, one must have access to two, flat 2D images as well as calibration values for the cameras with which the images were obtained. Two cameras, positioned a known distance apart, mimic the human eyes; they capture the same scene, which are offset due to the difference in line of sight. The binocular disparity (further referred to as "disparity"), is defined as the pixel distance of the same image point in the two images:

$$d = x_l - x_r$$

where x_l and x_r are the pixel x-coordinate of the image point in the left and right images, respectively. Assuming that the object is located at *P(x, y, z)* as defined in Figure 1, the following proportionality relationships can be applied to calculate the z-coordinate, or the depth:

$$\text{Left image:} \quad x / z = x_l / f$$

$$\text{Right image:} \quad (x - b) / z = x_r / f$$

where b is the distance between the two cameras, and f refers to the focal length of the cameras. Combining these two equations, the z can be calculated:

$$z = (b * f) / (x_l - x_r)$$

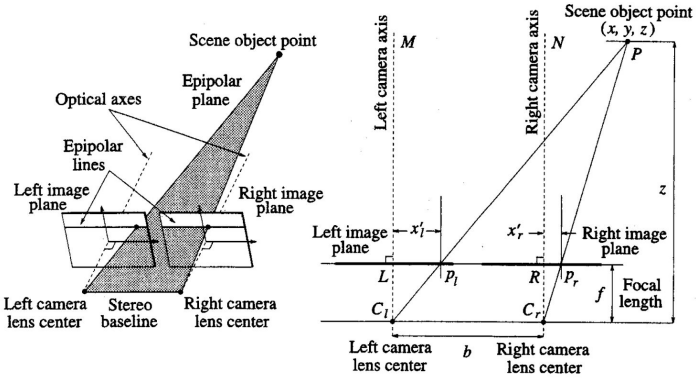

Figure 1. Stereo vision with two cameras with focal length f*, placed* b *units apart [1].*

The most important task comes in calculating the disparity between each point in the two images. A very useful technique that can be used to obtain disparity is *region matching*. The process involves creating a window around each pixel in the left image, and finding a window in the right image which matches it the most closely. Here, it is assumed that the two cameras were positioned at identical heights, which eliminates the need for pixels outside of the row to be examined during this step. The resulting matrix of disparity values allows for a complete 3D reconstruction of the image using x-, y-, and z-coordinates [1].

3.0 Process

The left and right images were obtained from the 2014 Middlebury Stereo Dataset [2].

Figure 2. Left image of the test set (recycling bin).

The images were pre-processed, analyzed to calculate disparity for each pixel, then post-processed to produce the final result.

3.1 Pre-processing

The original images were 2864 x 1924, with RGB values. The image was downloaded and converted to grayscale with the Python Pillow image processing library. As the resolution was quite high, the image was resized to ⅛ of the original dimensions to drastically reduce runtime. For further analysis, the intensity values were used to create a NumPy array. A *Process* class was created to handle these procedures.

```
im = Process(image_path)
im.resize_image(width, height)
array = im.get_array()
```

Figure 3. Pre-processing function, with the Process class.

3.2 Calculating disparity

As described under **2.0**, region matching was used to compare the two images. This process was executed multiple times with different hyperparameter values to obtain the optimal set.

Two different methods were examined for evaluating the similarity between the windows in the left and right images:

1. *Correlation coefficient*: Treating the two windows as matrices, the correlation coefficient between them were calculated, using the formula listed below:

$$r(d_x, d_y) = \frac{\sum_{(x,y)\in S}\left[f_1(x,y) - \bar{f}_1\right]\left[f_2(x+d_x, y+d_y) - \bar{f}_2\right]}{\left\{\sum_{(x,y)\in S}\left[f_1(x,y) - \bar{f}_1\right]^2 \sum_{(x,y)\in S}\left[f_2(x+d_x, y+d_y) - \bar{f}_2\right]^2\right\}^{1/2}}.$$

   where S refers to the window, f_1 and f_2 refer to the left and right images, respectively, and d_x refers to the disparity between the two pixels; d_y was removed from the calculations under the assumption that the two points lie on the same y-value. The distance between the pixel in the left image and the pixel with largest $r(d_x, d_y)$ in the right image was assumed to be the disparity value.

2. *Sum of Absolute Differences (SAD)*: The SAD approach calculates the difference between each pixel inside the two windows (left and right). The pixel in the right image whose window has the smallest SAD value is considered to be the matching pixel, and the distance to that pixel is taken as the disparity.

$$SAD = \sum_x \left| x_l - x_r \right|$$

After testing with both methods, the SAD method was chosen for higher accuracy as well as enhanced runtime. The *calib.txt* file included in the images' source folder provided a conservative search range for region matching (*ndisp*), which was scaled down proportionally to the resized image. As the points in the right image generally displayed a shift in the negative x-direction, the range was chosen as [- (ndisp * scale_factor), 0]. The possibility of searching in both positive and negative directions was explored, but resulted in extended runtime and more noise.

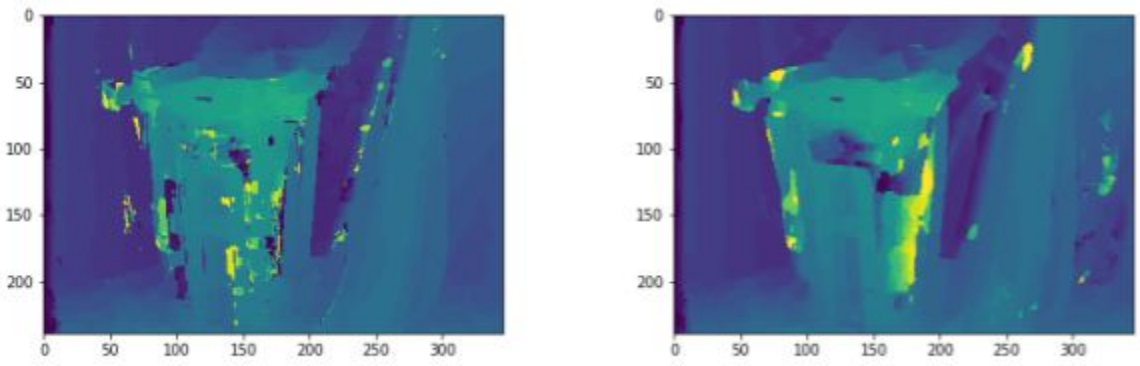

Figure 4. LEFT: Correlation Coefficient method; RIGHT: SAD method.

3.3 Post-processing

The resulting plots contained areas of noise and discontinuities, which called for a set of post-processing procedures.

The noisy pixels were assumed to originate from the "flat" areas, where the disparity values were spiking due to most SAD values being very similar. In order to capture and "denoise" these values, the following steps were introduced:

1. Check if the pixel's disparity value exceeds a threshold (>25).
2. If the pixel meets the above criteria, create a relatively large window around the pixel.
3. Calculate the mode (most common value) in the window, and assign it to the pixel.

In order to account for the discontinuous regions among pixels, the following method was utilized for every pixel:

- *Smoothing by average*: Inside a small window, the average disparity value was calculated. This was compared to the disparity of the pixel in the center, and if the difference was greater than 5 units, the average value was assigned to the pixel. This created a "smoothing" effect, allowing for a more gradual change between two regions.

Furthermore, in order to remove any extraneous noise pixels unaffected by the mode and average methods, all pixels with disparity larger than a threshold was set to a fixed value. This allowed for more distinction between the points of interest during testing as well.

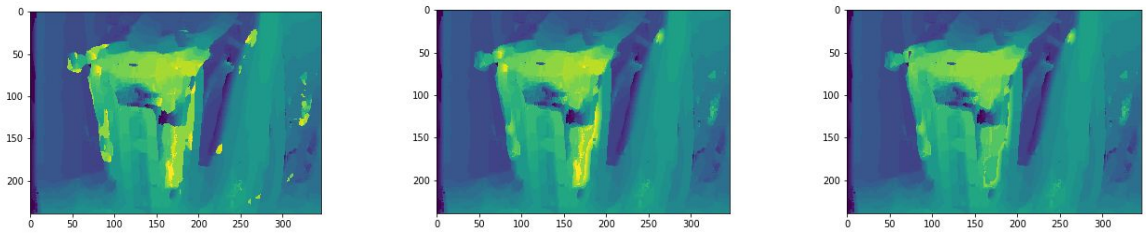

Figure 5. LEFT: after simple threshold; MIDDLE: after threshold and smoothing; RIGHT: after threshold, smoothing, and mode denoising.

3.4 Obtaining depth

The depth was calculated using the process outlined in **2.0**. The variables in the equation were replaced in order to utilize the coefficients in the *calib.txt* file. The resulting formula is listed below (*d* = *disparity*):

$$Z = baseline * f / (d + doffs)$$

4.0 Results

The resulting coordinate values, as well as the RGB values from the original resized image, were collected in a .txt file. The information was entered into Cloud Compare, an open source point cloud reconstruction software.

Figure 6. 3D reconstruction of the test image.

6.0 References

[1] https://www.cse.usf.edu/~r1k/MachineVisionBook/MachineVision.files/MachineVision_Chapter11.pdf

[2] http://vision.middlebury.edu/stereo/data/scenes2014/