Stereo

EECE5639 Computer Vision — Project 3
Fall Semester 2021

Cheung, Spenser & Liming, Tim
Dept. of Electrical and Computer Engineering
Northeastern University
Boston, MA, USA

***Abstract* — In this project we explore stereo vision and finding disparity in two images. We used SIFT to find correspondence pairs in both images to estimate the fundamental matrix. By getting the fundamental matrix, we reduce the search area for finding corresponding points on the epipolar lines, which we use to calculate and produce a disparity map.**

I. Introduction

In order to produce a disparity map of two images, we use epipolar lines and correspondence pairs to help us. Using corresponding pairs found with the SIFT detector, we compute the fundamental matrix and then run RANSAC on our values to eliminate any outliers.

With the inliers from RANSAC, we then compute the epipolar lines for the left and right images. Using the epipolar lines, we calculate the line equations which we then use as a guide to reduce our search space for matching points. Then with the matching points we found, we can then calculate the disparity using the matching points we found.

Code for our project can be found in our Github repository[1][2], linked in the Appendix.

II. Description of Algorithms

The first task of the project involved reading in a pair of stereo images that are taken from the same location and making them grayscale. We made use of the OpenCV Python package `cv2` and used its `imread` function to read in an image and then used the `cvtColor` function to transform the images to grayscale. We then read all the images in a folder and store them into an array ahead of time, which allowed us to speed up the processing part of the program.

Once we had our images in an easy to use format, we used the `resize` function to reduce their size. The scaling factor used in this application was `0.5` for both dimensions. We ended up not needing the scaled down images, as our algorithm ran quickly on the images at their original resolution.

Continuing on with our SIFT detector that we used from Project 2, we found features and descriptors in both of our stereo image pairs using the built-in functions provided by the `cv2` python package. We started by initializing our detector with `xfeatures2d.SIFT_create` and then found the keypoints and descriptors with `detectAndCompute`. We used `FlannBasedMatcher` to initialize our matcher and then used `knnMatch` to find our

matches. Once we had matches across the stereo pair, we needed to filter down our matches to only the best ones which was done with a simple ratio test. We can display our matches and draw the best lines with the `drawMatchesKnn` function [3].

Once we've identified some good matches, we can use those image points to calculate the Fundamental Matrix for our stereo pair. The Fundamental Matrix can be found using the `findFundamentalMat` function which gives us the Fundamental Matrix (F) and a mask that gives us our inliers. We determined our inliers using the RANSAC method to eliminate outliers. We were able to do this using the `FM_RANSAC` flag in the `findFundamentalMat` function [4].

With our inliers identified, we can use our points to plot the epipolar lines. The `computeCorrespondEpilines` function in the `cv2` package does this nicely for us and with the help of a simple `drawlines` helper function, we can draw the epilines that were calculated [4].

Finally, we can find the disparity map using the `StereoSGBM_create` function to initialize our stereo instance and then use the `compute` function on our stereo instance to generate our disparity map. A simple `imshow` with an `hsv` or `gray` flag completes the process [5][6].

III. Observations

One of the first observations we noticed was that the matches were much more consistent with the stereo pairs than with the images from Project 1. This is the benefit of comparing images taken from a stereo rig compared to the images that are produced when a camera rotates about an axis.

The first set of images that we worked with was a stereo pair of a wall with a watchtower, perhaps at an old prison or castle. The filename was labelled "Cast" so we will assume it is a castle from here on.

The output of our first function, the SIFT detector can be seen below in Figure 1. This function found points of interest in each of the images and attempted to find the matches between the two. We used a ratio test to pare down the number of matches and only display the matches that had the highest chance of being a "good" match.

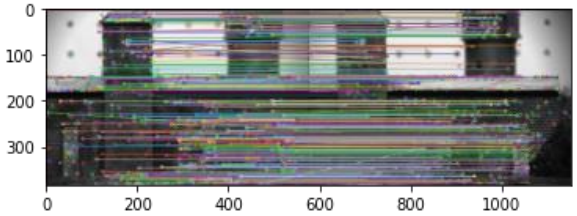

Figure 1: Corresponding Features (Cast)

The fact that it was a stereo pair meant that since the images were nearly identical, almost every single point of interest could be found in the other image. This meant that the matches were almost entirely horizontal lines (since we plotted the images side-by-side) with no vertical variation since the images are the same size and shape.

The next task was to find the Fundamental Matrix. We did this using the `findFundamentalMat` function which gave us the Fundamental Matrix and a mask to separate our outliers from the inliers. The Fundamental Matrix for the Cast stereo pair can be seen below in Table 1.

```
[[ 5.90953464e-06  3.80211634e-04  6.53554551e-03]
 [-3.75307116e-04  1.82828811e-06  2.16854278e-01]
 [-1.05497262e-02 -1.94678717e-01  1.0000000e+00]]
```

Table 1: Fundamental Matrix (Cast)

With our inliers identified, we can plot the epilines. The Cast photo set produced epilines that converged on a point outside the upper right of the images. This was surprising since we were expecting the epilines to be very similar to the matches we found from the first run of the SIFT algorithm and whose results we saw in Figure 1. The inlier correspondences for the Cast image pair can be seen below in Figure 2.

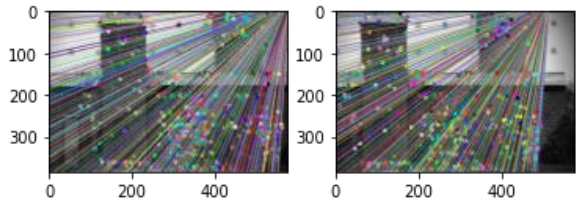

Figure 2: Inlier Correspondences (Cast)

With our matches narrowed down to the inliers, we were then able to find our disparity map. We did this using the `StereoSGBM_create` function process described in the previous section. Using the `'hsv'` flag, we generated our color disparity map. This color disparity map can be seen below in Figure 3.

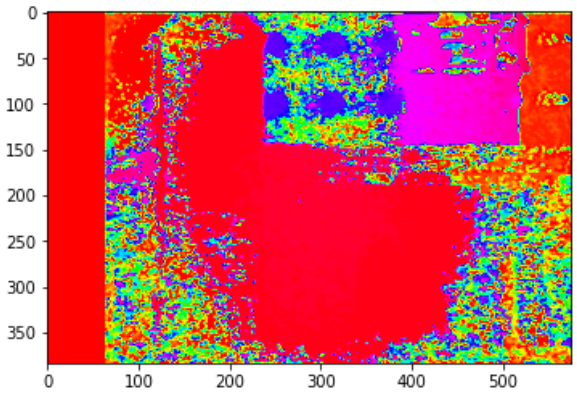

Figure 3: Color Disparity Map (Cast)

Following a similar process as the color disparity map, we were able to generate the gray disparity map using the `'gray'` flag. The gray disparity map for the Cast stereo pair can be seen below in Figure 4.

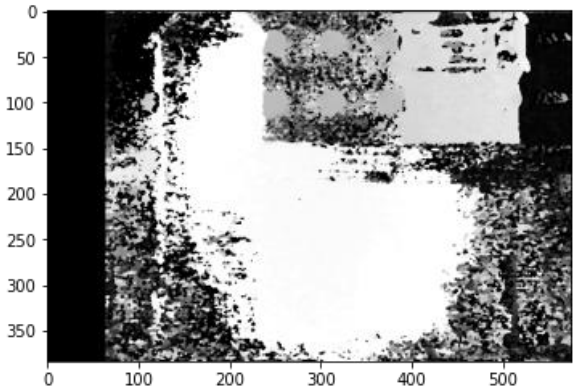

Figure 4: Gray Disparity Map (Cast)

The noisiness of our disparity maps for the Cast image set made us wonder if we set our disparity parameters incorrectly. Experimentation in modifying these parameters produced similar results so we concluded that the challenging brick structure and encroaching vegetation proved confusing for the algorithms. The dot artifacts also might have thrown off the algorithms. We wonder if results might be better with a higher resolution input image.

Following the same process as outlined for the Cast images, we searched for corresponding features in the Cones stereo pair. The resulting matches can be seen below in Figure 5. Not that some of the matches detected are not perfectly horizontal, making a faint 'X' shape on the right hand side.

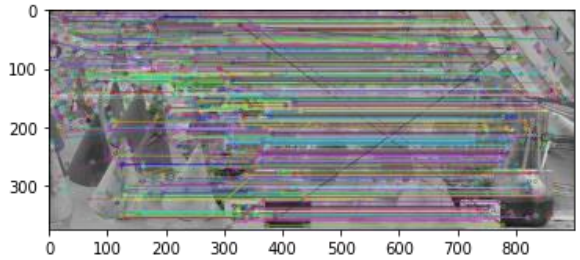

Figure 5: Corresponding Features (Cones)

Next we found the Fundamental Matrix and the masks for our outliers. The Fundamental Matrix for the Cones stereo pair can be seen below in Table 2.

```
[[ 9.31322575e-10  1.94132328e-04  5.56106567e-02]
 [-1.74582005e-04  9.88505781e-06 -2.36595085e+12]
 [-5.61523438e-02  2.36595085e+12  1.0000000e+00]]
```

Table 2: Fundamental Matrix (Cones)

With our inliers identified, we can plot the epilines. The Cones photo set produced epilines that were all perfectly horizontal which was what we expected from stereo images since they are nearly the same image with a slight shift. The inlier correspondences for the Cones photo pair can be seen below in Figure 6.

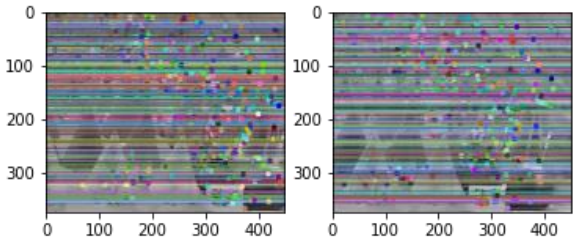

Figure 6: Inlier Correspondences (Cones)

Note the absence of the 'X' matches that appeared in the total correspondences set. This validates that we have successfully filtered out our outliers.

With our inliers identified for the Cones images, we could plot our disparity maps using the same process as detailed above for the Cast images. The color disparity map for the Cones stereo pair can be seen below in Figure 7.

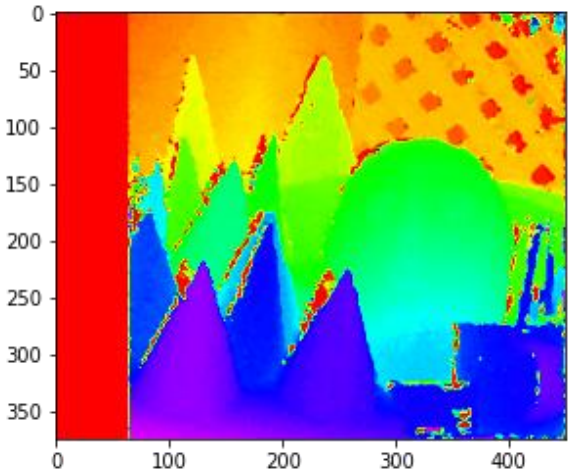

Figure 7: Color Disparity Map (Cones)

Following a similar process as the color disparity map, we were able to generate the gray disparity map using the `'gray'` flag. The gray disparity map for the Cones stereo pair can be seen below in Figure 8.

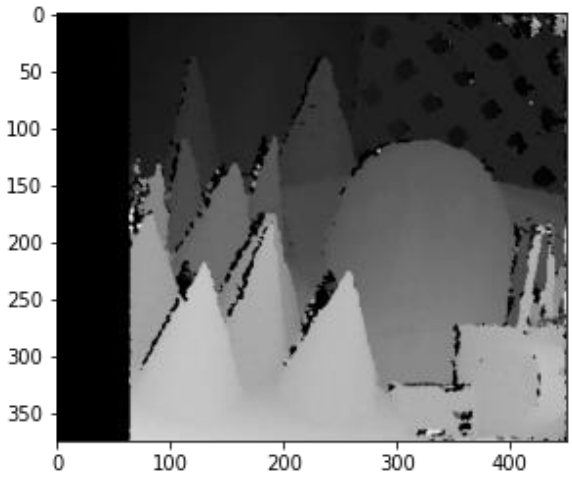

Figure 8: Gray Disparity Map (Cones)

IV. Conclusions

In this project, we demonstrated how to find correspondences between two images and use those correspondences to determine the fundamental matrix and disparity map for stereo vision. We compared and contrasted our approach with other approaches on the web to showcase how different approaches yield different results.

Once we had a good set of corresponding points, we calculated the fundamental matrix; which we used to find epipolar lines. With the epipolar lines, we found the line equations by multiplying the fundamental matrix by corresponding points in the left image. We then use a window to search along the lines until we find good matches. From those matches, we calculate disparity by subtracting the two points.

The disparity values can be used in the future to calculate depth, by using the focal length and baseline distance. Disparity is an important component of stereo vision.

V. Appendix

[1] Project 3 Repository
https://github.com/scheung97/EECE5639_ComputerVision/tree/main/Project3

[2] Project 3 Code
https://github.com/scheung97/EECE5639_ComputerVision/blob/main/p3.ipynb

[3] OpenCV » Feature Matching
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html

[4] OpenCV » Epipolar Geometry
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_epipolar_geometry/py_epipolar_geometry.html

[5] OpenCV: Depth Map from Stereo Images
https://opencv24-python-tutorials.readthedocs.io/en/latest/py_tutorials/py_calib3d/py_depthmap/py_depthmap.html

[6] OpenCV: samples/python/stereo_match.py
https://fossies.org/linux/opencv/samples/python/stereo_match.py