CSCE 867: Computer Vision
Project #2
Due time: 11:59:59 pm, Wednesday, April 17
Team members: Denise Davis, Jared Gentry, Wesley Cherry

Introduction

Passive stereo is a method of computer stereo vision that uses multiple (at least two) cameras to recover a 3D scene/object from 2D images. The general strategy of the passive stereo method is camera calibration, correspondence, and 3D reconstruction from matched 2D points. The last step, 3D reconstruction, requires image rectification. Image rectification is the process of projecting multiple images onto a common image plane. Once the images have been rectified and are on the same image plane, 3D information can be retrieved from the images, such as a disparity map. In this project a 3D scene was recovered from a stereo camera system through the strategy outlined above.

First, the two cameras (left and right) were calibrated using the methods developed from project 1 of this class. This calibration yielded a matrix of intrinsic camera parameters, along with rotation and translation matrices for each camera. The rotation and translation matrices were used to estimate the relative rotation and translation between the two cameras. The results of this process are shown below:

*Note: Python files cv_proj2_left.py and cv_proj2_right.py were used for camera calibration of left and right cameras. The relative rotation and translation matrices were calculated using the Python file rectification.py.

<u>Left and Right Camera Matrices of Intrinsic Camera Parameters:</u>

```
Wleft = [ [401.7720860407226, 0, 184.15045638684873],
          [0, 396.12224129404564, 118.14563010963332],
          [0, 0, 1] ]

Wright = [ [412.2085911472977, 0, 187.92643715729065],
           [0, 407.7880359726471, 131.29161875061976],
           [0, 0 ,1] ]
```

*Note: Focal length of right camera is slightly larger. Both u_o and v_o are close to the center of image (160,120).

<u>Rotation and Translation Matrices of Left Camera:</u>

```
Rleft = [[0.7552643,-0.65459403,0.03290133],
         [0.13783671,0.09968584,-0.98542568],
         [-0.64178723,-0.74880731,-0.16551967]]

Tleft = [[5.963266213979592],[21.688085517596715],[-74.4625329655395]]
```

Rotation and Translation Matrices of Left Camera:

```
Rright = [[0.57126946,-0.8203492,0.02604603],
          [0.16564024,0.08176389,-0.98279091],
          [-0.80410364,-0.56575378,-0.1825924 ]]

Tright = [[12.806194509975613],[23.191573457787968],[-77.59926135564851]]
```

Relative Rotation and Translation Matices Between Two Cameras:

```
R = [[ 0.96931207  0.03924488 -0.24297926]
     [-0.02870172  0.99944941  0.0126986 ]
     [ 0.2433389  -0.00485996  0.96992665]]

T = [[-26.21509412]
     [ -0.13775712]
     [ -2.20047669]]
```

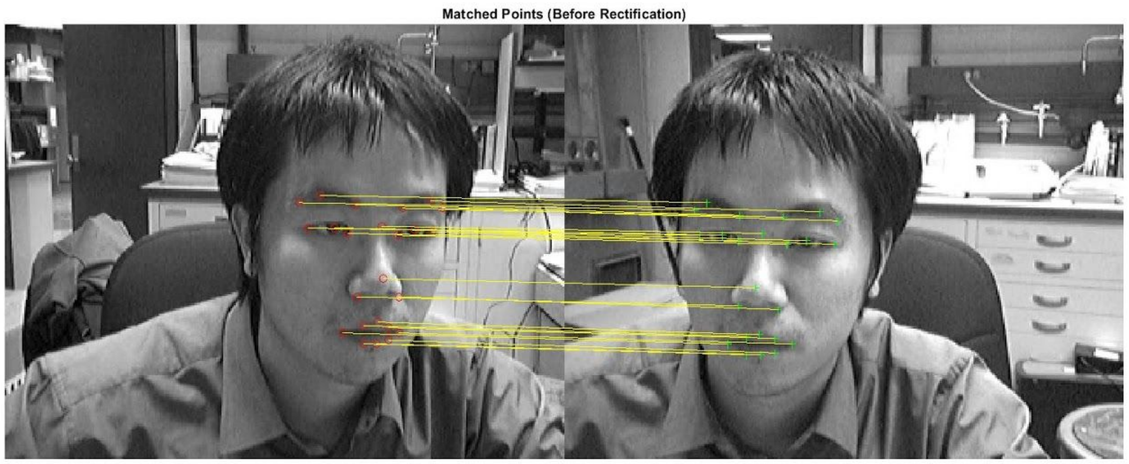

Figure 1. Connecting similar landmarks between Left and Right Image. This is not conclusive. Epipolar Lines have to be calculated to show real relationship between two images.

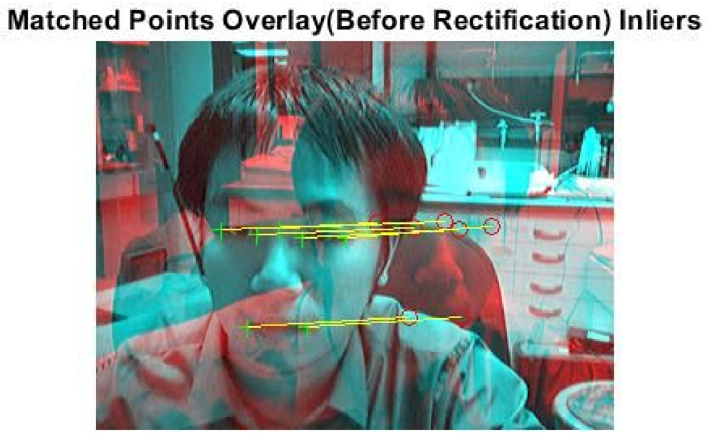

Figure 2. Overlay original images with the six landmarks. Corner of eyes (4) and corner of mouth (2).

Discussion and Results of Computing E and F

The next step in this project was to calculate the essential matrix E, and the fundamental matrix F. Our first attempt to calculate F was with the landmark files. The non-coplanar requirement was questionable with these. F is not dependent upon the images, but rather the individual cameras and their intrinsic parameters. So, the calibration files and checkerboard were more appropriate for this task. We derived F from the checkerboard files and the previous camera calibration step described above. Our calculation of F is based upon the 8-point algorithm presented in class [Lecture 20]. This is a two step process whereby the singularity constraint is enforced, meaning the epipolar lines will intersect at the same epipole.

*Note: The F matrix was calculated and the epipolar lines were generated using the Matlab file f_matrix.m.

$SVD(A)=U_A D_A V_A^T$

Then we construct F_{start} from v:

$$F_{start} = \begin{matrix} -1.44E-05 & -1.45E-05 & -0.0001 \\ -2.08E-05 & -1.58E-06 & -0.04979 \\ 0.0073 & 0.05542 & -0.9972 \end{matrix}$$

After enforcing singularity constraint, setting the last singularity of the D_f, diagonal matrix, to zero.

$SVD(F_{start})=U_f D_f V_f^T$

Then we reconstruct F with new Df:

$$F = \begin{matrix} -2.293e-06 & -1.6175e-05 & -0.0001 \\ -2.0849e-05 & -1.5624e-06 & -0.04978 \\ 0.00730 & 0.0554 & -0.9972 \end{matrix}$$

For this situation, there was not a large change in the fundamental matrix after applying the singularity constraint. We do understand that this is a necessary step in the process to ensure epipolar lines pass through the same epipole.

And then E, the essential matrix is derived from intrinsic parameters, W_r and W_l, and F.

$$E = \begin{matrix} -0.3798 & -2.6502 & -1.0667 \\ -3.4044 & -0.2524 & -21.3549 \\ 1.8206 & 21.3112 & 0.7936 \end{matrix}$$

Using our calculated F matrix, we calculated epipolar lines for both unrectified images and as depicted in Figure 3 and 4. All figures displayed in this report were produced with Matlab. We did find a difference between our calculated F matrix and that predetermined using Matlab library. The difference did not appear with the unrectified images. It did, however, showup with the rectified. We are still investigating this difference as it was found late in the process, tonight. Those results appear in Figure 6.

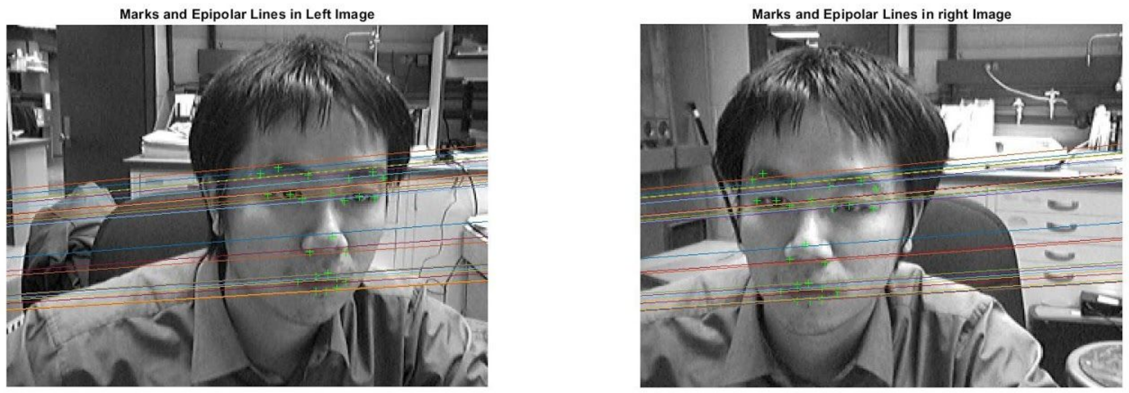
Marks and Epipolar Lines in Left Image
Marks and Epipolar Lines in right Image

Figure 3. Epipolar Lines Calculated for Unrectified Left and Right Images.

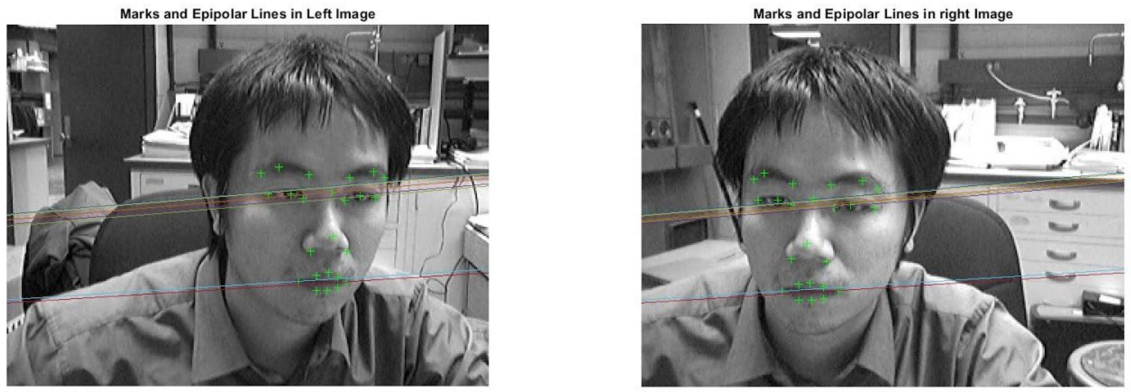
Marks and Epipolar Lines in Left Image
Marks and Epipolar Lines in right Image

Figure 4. Epipolar Lines Calculated for 6 Landmarks only from Unrectified Left and Right Images.

Discussion and Results of Rectification

The purpose of image rectification is to project the images onto the same image plane. In order to do this there are several overarching steps that need to be followed. The first of these steps is to construct the left camera's rectified rotation matrix (R_{lrect}). With this value calculated, the points for the image taken with the left camera can be reprojected. Each point in the original left image has its position in the rectified image calculated. In this step the average of the left and right cameras' intrinsic parameters matrices are used as the two will be slightly different, and that difference needs to be accounted for. In our implementation, the reprojection of image points used forward mapping as opposed to backward mapping, meaning that calculations were performed onto the original image points to find out where they belonged in the rectified image instead of vice versa. This choice was made because our rectified images didn't contain notable holes where no image pixels were projected, which is the issue that backward mapping is supposed to address. By dividing the W_{rect} matrix by 1.1, the reprojected image was essentially downsampled and holes disappeared. After the left image is reprojected, calculations can begin to be done on the right image. The first step to rectify the right image is to calculate the rotation matrix (R_{rrect}) for the right camera. This is done by multiplying the R_{lrect} matrix and the R matrix. With the R_{rrect} calculated, the image points for the right camera can be calculated using the same method used for left camera points. Then, after all of the pixels for the right image have been reprojected, both of the images have been rectified and the rectification process is complete. Both images should now be on the same image plane and 3D information can be retrieved from them. If done correctly, the epipolar lines will be parallel, and the conjugate epipolar lines will be collinear and parallel to the base line.

In our implementation of the rectification we followed the major steps of the rectification process. All linear algebra calculations and matrix multiplications were completed by Python's numpy library. All image processing, including importing and exporting images, was done by Python's OpenCV library. We did experience several issues with our results though. The first of these issues was that the resulting rectified images were flipped. Points that were on the right side of the image were placed on the left. We were unable to find the exact cause of this issue, but it could be related to the way the image was displayed. Since the images were flipped, we had to use an opencv postprocessing function to flip the image so that it was displayed correctly. Another issue that we experienced was that our rectified images were rotated by 180 degrees so that the rectified images were upside down. We were able to correct this by multiplying our R_{lrect} matrix by negative one. When the final rectified images were created, there were some issues with the rectified images. As one can see in the images of the rectified images with the epipolar lines shown below, the epipolar lines are parallel, but they do not connect between the images. This is an issue that we were unable to remedy.

R_{lrect} (after taking negative of it):

```
[[ 9.96481943e-01  5.23639081e-03  8.36439983e-02]
 [ 5.23639081e-03 -9.96481943e-01 -0.00000000e+00]
 [-8.33497339e-02 -4.37992664e-04  9.93003682e-01]]
```

W_{rect} (after dividing by 1.1):

```
[[369.9912169    0.         169.1258607 ]
 [  0.         365.41376239 113.38056766]
 [  0.           0.           0.90909091]]
```

R_{rrect}:

```
[[ 0.98610552  0.04393381 -0.16092941]
 [ 0.03367645 -0.99572779 -0.01392626]
 [ 0.16085709 -0.00853476  0.98338743]]
```

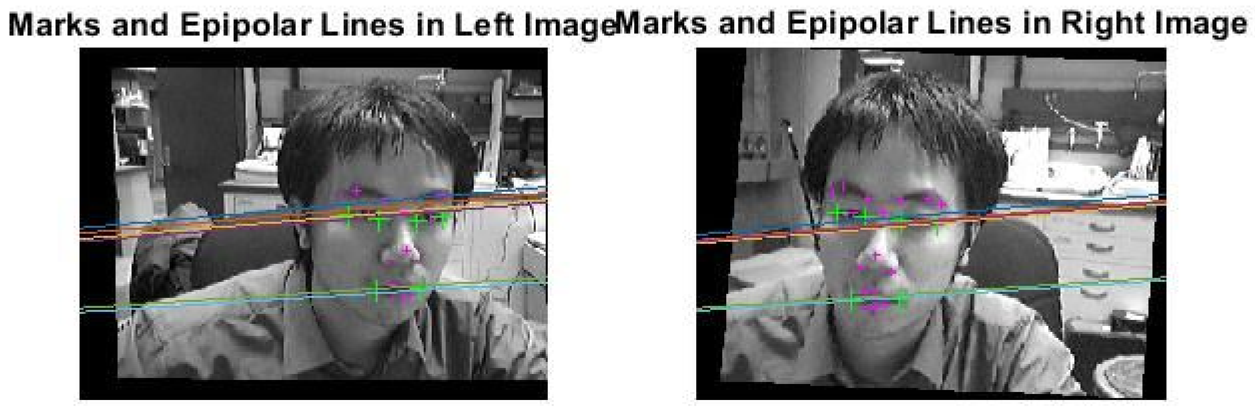
Marks and Epipolar Lines in Left Image
Marks and Epipolar Lines in Right Image

Figure 5. Preliminary Results of image rectification and post-processing. Epipolar Lines for 6 landmarks calculated for the rectified images.

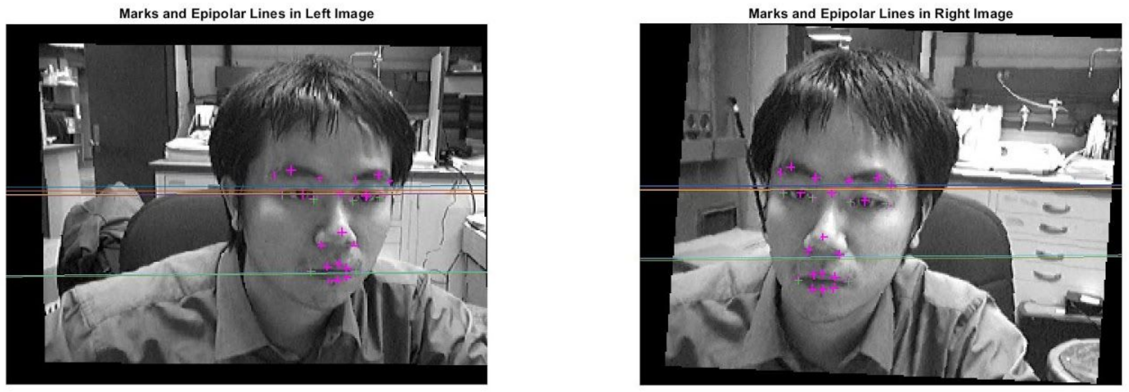
Marks and Epipolar Lines in Left Image
Marks and Epipolar Lines in Right Image

Figure 6. Epipolar Lines calculated using Matlab's F Matrix. Shows better agreement.

Summary and Conclusion

In this project we implemented passive stereo vision, so that 3D information can be extracted from two 2D images taken from slightly different perspectives. In our implementation we were able to get the first three steps in the project sheet to work correctly, but we had some issues relating to the rectification process. The issue that caused the rectified images to be flipped was one that we had to rectify using a post-processing function, and the issue that caused our rectified images to be rotated by 180 degrees had to be solved by multiplying the R_{lrec} by -1. On top of this, we were unable to get the epipolar lines between the two rectified images to connect although they were parallel. We were unable to find the causes of these three issues due to the complex nature of the project and the difficulty with debugging a program of this nature. A project of this nature involved a steep learning curve that caused us a large amount of difficulty, especially with the image processing portions of the assignment, which is something that no one in our group had experience with. Overall, our project followed the major steps involved with extracting 3D information from 2D images, and despite a few small errors, we came close to reaching that goal.

References

- Lecture 4: https://dropbox.cse.sc.edu/pluginfile.php/255282/mod_resource/content/3/lect4.pdf
- Lecture 17: https://dropbox.cse.sc.edu/pluginfile.php/256513/mod_resource/content/1/lect18.pdf
- Lecture 18: https://dropbox.cse.sc.edu/pluginfile.php/256513/mod_resource/content/1/lect18.pdf
- Lecture 19: https://dropbox.cse.sc.edu/pluginfile.php/256561/mod_resource/content/1/lect19.pdf
- Lecture 20:https://dropbox.cse.sc.edu/pluginfile.php/256561/mod_resource/content/1/lect20.pdf
- Matlab: https://www.mathworks.com/help/vision/ref/showmatchedfeatures.html