ENPM673 Project 4

Brendan Neal

April 26, 2023

Contents

1 Introduction

This project has students performing stereo vision to generate depth information on three different pairs of pictures.

2 Pipeline

2.1 Pre-processing and Feature Matching

Before implementing stereo vision, we have to perform several pre-processing steps. First, I loaded the information for each of the data sets from the calib.txt file, specifically the intrinsic matrices and the baseline. Next, I loaded the two images into my program. Next, I converted the image to gray. Then, I created an ORB object. The reason I used ORB rather than SIFT is because the SIFT license expired for my version of Python/OpenCV. Next, I found and matched the features using a BF matcher. After running into some issues regarding falsely matched features, I then siphoned the "closest" 100 features in terms of x distance from each other. This, combined with the RANSAC method described later, mitigated the potential for outliers and made the Stereo Vision calibration better. Finally, to visualize the best matches, I plotted them using the cv.drawMatches() function. These matches can be found in the results section.

2.2 Estimation of Fundamental Matrix

To calculate the fundamental matrix, I performed the following mathematical operations. First, we know the relationship of the fundamental matrix is as follows:

$$x_i^{'T} * F * x_i = 0$$

And so we can transform this equation as following:

$$Af = 0$$

where A is:

$$\begin{bmatrix} x_i^{'} * x_i & x_i^{'} * y_i & x_i^{'} & y_i^{'} * x_i & y_i^{'} * y_i & y_i^{'} & x_i & y_i & 1 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. \end{bmatrix}$$

for all corresponding points (or in this case matched features). Next, we perform SVD to solve for the fundamental matrix. Before performing SVD, I took specific steps to "normalize" the input matrices as needed.

Due to the presence of outliers/falsely matched features, we have to implement the fundamental matrix calculation within a RANSAC algorithm. To do so, I adapted my RANSAC algorithm from projects 1 and 2 to fit this specific application. I created a function that would take the feature list as an input and iterative through calculate a best-fit model for the specific pair of images. The minimum number of points needed for the calculation of the fundamental matrix is eight pairs, and thus I would randomly sample eight pairs from my feature list. Using those eight pairs, I would calculate an iteration F. If the number of inliers was greater than the current inlier max, set that model as the best model. To calculate the error, I had to create a separate function. This function takes the entire matched feature list and applies them to this equation:

$$x_i^{'T} * F * x_i$$

The error would then be the Euclidean distance found between the result and zero. I had to set a very low error threshold, as it yielded me better results. Thus, an inlier was counted if the error was less than 0.03. The function would continue to search and update the max inliers, max iterations until a best model fit was found. The estimated fundamental matrix can be found in the Results section. As a check, I also calculated the matrix rank and determinant. For every trial I ran, the matrix rank was 2 (not full) and the determinant was 0.

2.3 Estimation of Essential Matrix

To estimate the Essential Matrix, I had to set up the following equation, where K1 and K2 are the intrinsic matrices of the cameras:

$$K_2^T * F * K_1 = 0$$

Solving this using SVD yields my Essential matrix.

2.4 Decomposition of Essential Matrix

To decompose my essential matrix, I had to take the SVD once again to get my u,s, and v vectors. However, decomposing my essential matrix into my rotation and translation matrices yields four possible solutions. Using a matrix W of form:

$$\begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Thus the four possible solutions are:

$$T_1 = U(:,3), R_1 = UWV^T$$

$$T_2 = -U(:,3), R_2 = UWV^T$$

$$T_3 = U(:,3), R_3 = UW^TV^T$$

$$T_4 = -U(:,3), R_4 = UW^TV^T$$

Additionally, after this step is that if the determinant of R is -1, I had to correct the camera pose by negating both T and R.

To correctly identify the rotation and translation matrices, I had to create another set of functions that would back out the correct solution. Finding the correct solution involves me finding the Rotation and Translation matrix pair that yields positive depth. To accomplish this, I had to generate a set of 3-D points to project and count the number of positive projections. To create my 3-D data set, I used cv2.triangulatePoints() with two pairs of points projected using both camera intrinsic matrices.

Once I formed my 3-D data set, I created another function that would count the number of positive Z coordinates after projection. I would project these points using the each of the 4 generated rotation and translation matrices. and count the number of positive Z values. The Rotation/Translation pair with the most positive projected Z values is the correct solution. You can find my rotation and translation matrices in the Results section.

2.5 Epipolar Lines

2.5.1 Unrectified

To draw the epipolar lines, I had to create two pre-processing functions. One function is to generate an x coordinate given a y coordinate and a line equation. The other function is to resize the two images to match the sizes to the larger of the two images.

To generate the epipolar lines I created a function that takes the two sets of matching features as inputs, the fundamental matrix, the two images, and a logical True or False statement related to whether or not the inputs are rectified or not. For now, assume the inputs are not rectified. Within this function, the matching sets of features are organized into two homogeneous coordinate sets, X1 and X2. I generate the epipolar line corresponding to the first feature by performing:

$$Line2 = F * X_1$$

I generate the epipolar line corresponding to the corresponding second feature by performing:

$$Line1 = F^T * X_2$$

Using this line (which is of shape 3x1) I can then generate the X and Y coordinates corresponding to any unrectified epipolar line using these four equations:

$$Y_{min} = 0$$

$$Y_{max} = ImageWidth$$

$$X_{min} = GenerateXPoint(Y_{min})$$
$$X_{max} = GenerateXPoint(Y_{max})$$

Once I have these four points I can plot the lines and their corresponding features on their images. See the Results section to see the visual interpretation of these results.

2.5.2 Rectified

In order to rectify the image, I created another function that would take both images, both corresponding pairs of features, and the fundamental matrix as inputs and outputs the inputs as their rectified forms. First, the function uses cv2.stereoRectifyUncalibrated() in order to output two homography matrices with which I can perform perspective transformations with. To output the rectified images, I used cv2.warpPerspective() on each of the images with their corresponding homography matrix. To recitfy the feature set, I used cv2.perspectiveTransform(). Finally, to generate the rectified fundamental matrix, I performed the following calculation:

$$F_{rect} = H_2^{-1T} * (F * H_1^{-1}$$

See the Results section for these homography matrices.

Next, to redisplay these images, I passed them through my GenerateEpipolarLines() function using the rectified features, images, fundamental matrix, and a logical True to regenerate and draw the epipolar lines. The logical "True" now uses the following four equations to generate the points needed to draw the lines:

$$X_{min} = 0$$
$$X_{max} = ImageHeight - 1$$
$$Y_{min} = -Line(2)/Line(1)$$
$$Y_{max} = -Line(2)/Line(1)$$

This means that the lines are now horizontal. See the Results section to see how the new rectified images and epipolar lines look.

2.6 Disparity Map

In order to calculate disparity, I had to apply sum of squared differences (SSD). The basic concept is depicted by this equation:

$$\sum_{[i,j]\in R} (f(i,j) - g(i,j))^2$$

where f(i,j) and g(j,j) are pixel intensity information. I selected a window size of 15. First, I converted the rectified images to gray. Next, I created a series

of for loops that would search the image window by window and compute the SSD for each window subset. Iteratively, I would build my disparity image by taking the minimum value of the SSD for a particular iteration. Once I had my raw disparity image, I had to scale it to 0-255 in order to display it (since the raw SSD would be relatively small). To do so, I used the following equation:

$$(TrueDisparity) = \frac{(RawDisparity)}{(Max(RawDisparity))} * 255$$

Once I had the true disparity information, I could then display the results on an image.

To convert the disparity image into a heat map, I used the pyplot's function pyplot.getcmap('inferno') and followed the instructions to convert to a color heat map.

2.7 Depth Image

In order to convert the disparity information into depth information, I created another function. This function would take the disparity image as inputs and scale the information based on the focal length of and baseline of the cameras. The equation is as follows:

$$RawDepth[Disparity > 0] = (f * baseline)/(Disparity(Disparity > 0)$$

Then I would scale the information back to 0-255 using this equation:

$$(TrueDepth) = \frac{(RawDepth)}{(Max(RawDepth))} * 255$$

From here I can plot and display the information.

To convert the depth image into a heat map, I used the pyplot's function pyplot.getcmap('inferno') and followed the instructions to convert to a color heat map.

3 Results

These results were taken from some of my better RANSAC runs. Running the code may yield better or worse results.

3.1 Feature Matching

3.1.1 Art Room

Figure 1 shows the matched features for the Art Room.

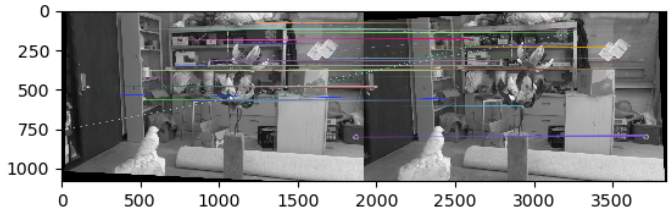

Figure 1: Matched Features

3.1.2 Chess

Figure 2 shows the matched features for the chess images.

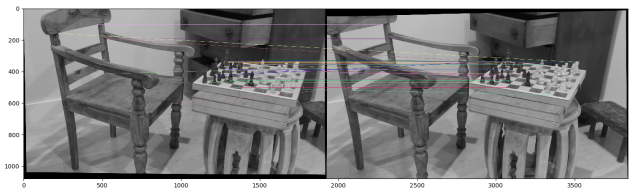

Figure 2: Matched Features

3.1.3 Ladder

Figure 3 shows the matched features for the chess images.

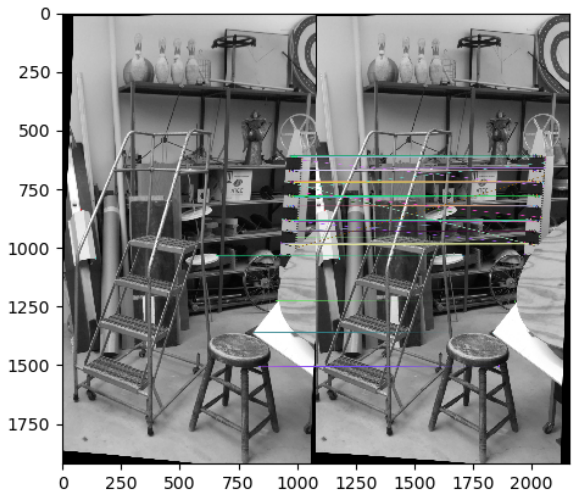

Figure 3: Matched Features

3.2 Estimation of Fundamental Matrix

3.2.1 Art Room

$$\begin{bmatrix} -2.27e-9 & -2.16e-8 & 1.49e-4 \\ 6.32e-8 & -3.54e-8 & -2.52e-3 \\ -1.56e-4 & 2.50e-3 & 1.49e-2 \end{bmatrix}$$

Matrix Rank: 2
Determinant: 0

3.2.2 Chess

$$\begin{bmatrix} -9.76e-8 & -1.22e-6 & 6.81e-4 \\ 2.54e-6 & -2.16e-6 & -4.26e-3 \\ -8.84e-4 & 4.29e-3 & 9.49e-2 \end{bmatrix}$$

Matrix Rank: 2
Determinant: 0

3.2.3 Ladder

$$\begin{bmatrix} 5.60e-8 & -4.97e-7 & 2.57e-4 \\ 1.32e-6 & 1.10e-6 & -3.90e-3 \\ -1.16e-3 & 2.88e-3 & 9.24e-1 \end{bmatrix}$$

Matrix Rank: 2
Determinant: 0

3.3 Estimation of Essential Matrix

3.3.1 Art Room

$$\begin{bmatrix} -1.55e-3 & -1.6e-2 & 5.47e-2 \\ 4.33e-2 & -1.11e-2 & -9.97e-1 \\ -4.94e-2 & 9.99e-1 & -1.24e-2 \end{bmatrix}$$

Matrix Rank: 2
Determinant: 0

3.3.2 Chess

$$\begin{bmatrix} -2.54e-1 & -6.23e-1 & 1.17e-1 \\ 7.24e-1 & -4.38e-1 & -5.32e-1 \\ 3.11e-1 & 6.43e-1 & -1.55e-1 \end{bmatrix}$$

Matrix Rank: 2
Determinant: 0

3.3.3 Ladder

$$\begin{bmatrix} 2.57e-2 & -2.94e-1 & 5.74e-2 \\ 5.61e-1 & 2.87e-2 & -8.26e-1 \\ 2.32e-2 & 9.55e-1 & 2.90e-2 \end{bmatrix}$$

Matrix Rank: 2
Determinant: 0

3.4 Decomposition of Essential Matrix

3.4.1 Art Room

Rotation Matrix:

$$\begin{bmatrix} 0.999 & -0.005 & 0.027 \\ 0.005 & 0.999 & -0.011 \\ -0.027 & 0.0111 & 0.999 \end{bmatrix}$$

Translation Matrix:

$$\begin{bmatrix} 0.998 \\ -0.055 \\ 0.017 \end{bmatrix}$$

3.4.2 Chess

Rotation Matrix:

$$\begin{bmatrix} 0.916 & -0.328 & 0.232 \\ 0.379 & 0.898 & -0.224 \\ -0.136 & 0.293 & 0.946 \end{bmatrix}$$

Translation Matrix:

$$\begin{bmatrix} 0.730 \\ -0.038 \\ 0.946 \end{bmatrix}$$

3.4.3 Ladder

Rotation Matrix:

$$\begin{bmatrix} 0.955 & 0.026 & 0.293 \\ -0.032 & 0.999 & 0.013 \\ -0.293 & -0.022 & 0.956 \end{bmatrix}$$

Translation Matrix:

$$\begin{bmatrix} 0.953 \\ -0.056 \\ 0.295 \end{bmatrix}$$

3.5 Epipolar Lines

3.5.1 Art Room

Figure 4 shows the unrectified epipolar lines for the art room images.

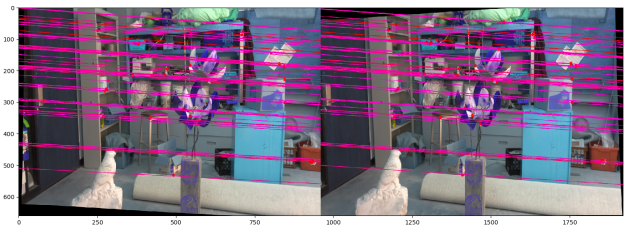

Figure 4: Unrectified Epipolar Lines

Printed below are the corresponding homography matrices used in rectification.

$$\text{H1} = \begin{bmatrix} -2.42e-3 & -3.20e-5 & 2.34e-1 \\ 1.57e-4 & -2.50e-3 & -1.00e-1 \\ 6.36e-8 & -3.45e-8 & -2.53e-3 \end{bmatrix}$$

$$\text{H2} = \begin{bmatrix} 9.94e-3 & 5.02e-5 & -4.47e+1 \\ -5.95e-2 & 9.98e-1 & 3.38e+1 \\ -9.50e-6 & -4.80e-7 & 1.01e00 \end{bmatrix}$$

Figure 5 shows the rectified epipolar lines for the art room images.

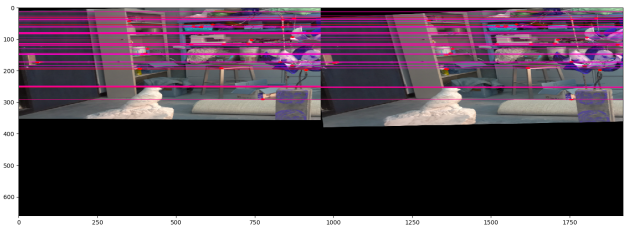

Figure 5: Rectified Epipolar Lines

3.5.2 Chess

Figure 6 shows the unrectified epipolar lines for the chess images.

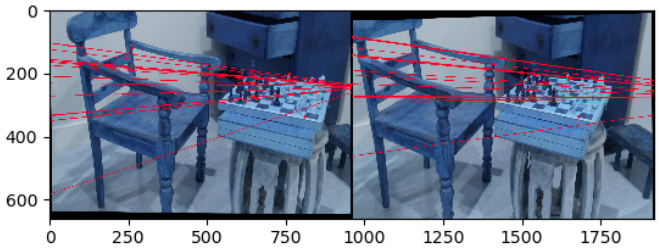

Figure 6: Unrectified Epipolar Lines

Printed below are the corresponding homography matrices used in rectification.

$$\text{H1} = \begin{bmatrix} 4.99e-3 & -2.61e-5 & -3.16e00 \\ 1.05e-3 & -4.74e-3 & -2.57e-1 \\ 3.00e-6 & -2.94e-6 & -4.93e-3 \end{bmatrix}$$

$$\text{H2} = \begin{bmatrix} 7.38e-1 & -1.72e-1 & 3.06e+2 \\ -1.93e-1 & 1.07e00 & 3.54e+1 \\ -4.37e-4 & 1.02e-4 & 1.14e00 \end{bmatrix}$$

Figure 7 shows the rectified epipolar lines for the chess images.

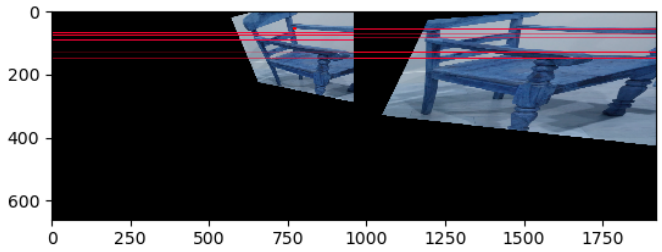

Figure 7: Rectified Epipolar Lines

3.5.3 Ladder

Figure 8 shows the unrectified epipolar lines for the ladder images.

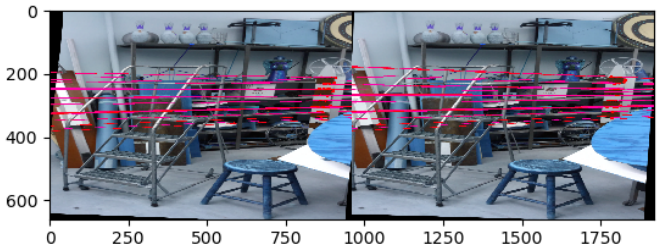

Figure 8: Unrectified Epipolar Lines

Printed below are the corresponding homography matrices used in rectification.

$$\text{H1} = \begin{bmatrix} -1.70e-3 & 4.09e-5 & -1.21e00 \\ 1.32e-3 & -2.88e-3 & -1.41e00 \\ 1.58e-6 & 1.42e-7 & -4.68e-3 \end{bmatrix}$$

$$\text{H2} = \begin{bmatrix} 8.07e-1 & 1.44e-1 & 1.77e+2 \\ -1.26e-1 & 9.97e-1 & 1.22e+2 \\ -2.01e-4 & -3.58e-6 & 1.19e00 \end{bmatrix}$$

Figure 9 shows the rectified epipolar lines for the ladder images.

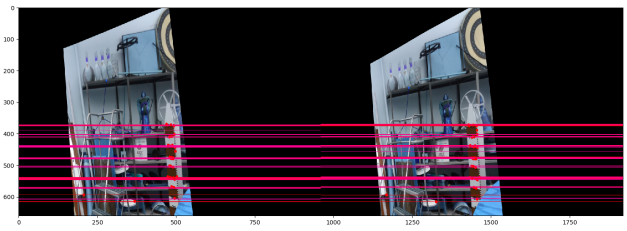

Figure 9: Rectified Epipolar Lines

3.6 Disparity Map

3.6.1 Art Room

Figure 10 shows the black and white disparity image for the art room images.

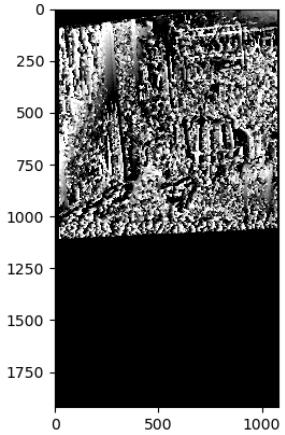

Figure 10: Black and White Disparity Image

Figure 11 shows the heatmap disparity image for the art room images

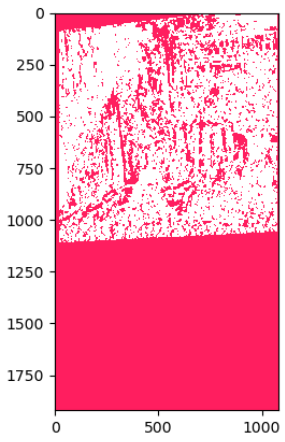

Figure 11: Heatmap Disparity Image

3.6.2 Chess

Figure 12 shows the black and white disparity image for the chess images.

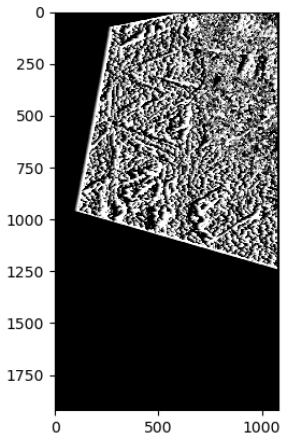

Figure 12: Black and White Disparity Image

Figure 13 shows the heatmap disparity image for the chess images

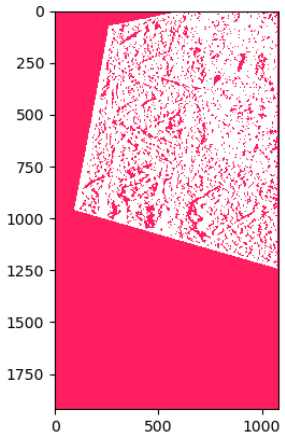

Figure 13: Heatmap Disparity Image

3.6.3 Ladder

Figure 14 shows the black and white disparity image for the ladder images.

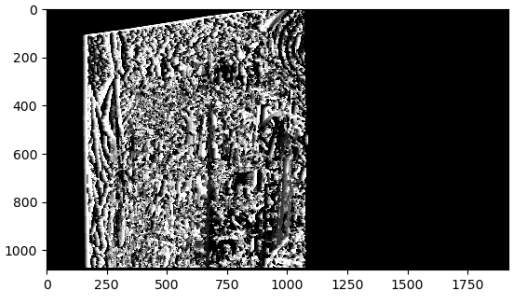

Figure 14: Black and White Disparity Image

Figure 15 shows the heatmap disparity image for the ladder images

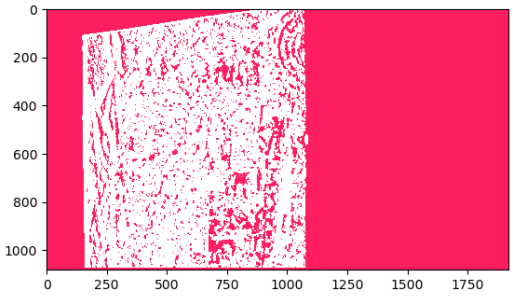

Figure 15: Heatmap Disparity Image

3.7 Depth Image

3.7.1 Art Room

Figure 16 shows the black and white depth image for the art room images.

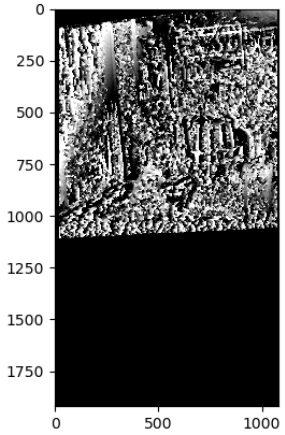

Figure 16: Black and White Depth Image

Figure 17 shows the heatmap depth image for the art room images

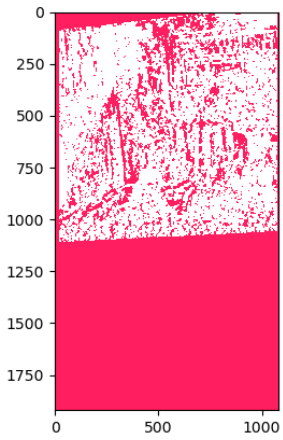

Figure 17: Heatmap Depth Image

3.7.2 Chess

Figure 18 shows the black and white depth image for the chess images.

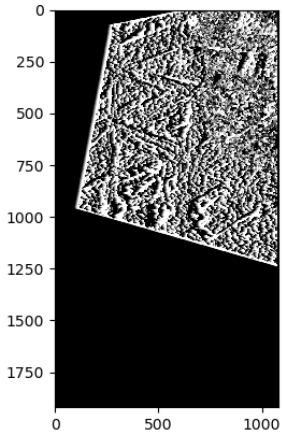

Figure 18: Black and White Depth Image

Figure 19 shows the heatmap depth image for the chess images

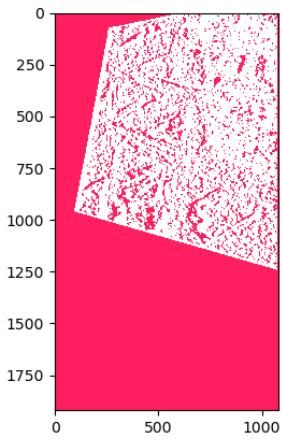

Figure 19: Heatmap Depth Image

3.7.3 Ladder

Figure 20 shows the black and white depth image for the ladder images.

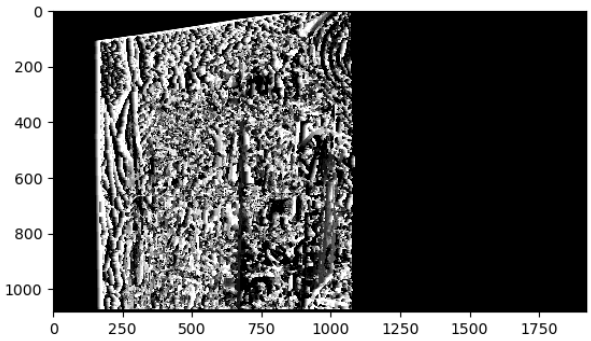

Figure 20: Black and White Depth Image

Figure 21 shows the heatmap depth image for the ladder images

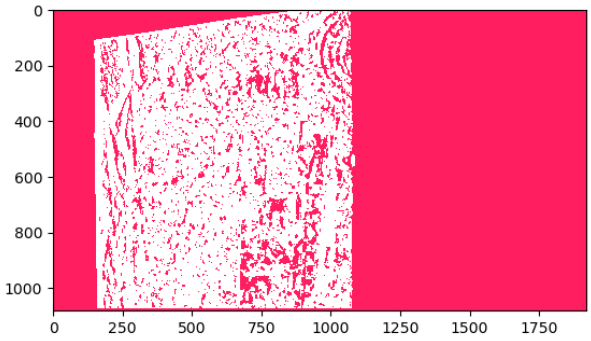

Figure 21: Heatmap Depth Image

4 Problems Encountered

4.1 Pre-processing and Feature Matching

I had to use the ORB feature detector because the SIFT license has expired for my version of OpenCV. Additionally, there were some falsely matched features that I tried to filter out via RANSAC, however the algorithm will still sometimes still catch those and make my program fail.

4.2 Estimation of Fundamental Matrix

I had to add code not mentioned in the notes to force the resulting matrix to have rank 2 and determinant zero, which I didn't think would work. However, the results are pretty good so I believe that it was the right move.

4.3 Estimation of Essential Matrix

No major issues here, just a lot of debugging.

4.4 Decomposition of Essential Matrix

No major issues here, just a lot of debugging.

4.5 Epipolar Lines

Drawing the epipolar lines was very difficult, however I think the issues come from the feature detection and matching rather than the algorithm itself. My algorithm consistently works for the art room images, which means the code works properly. However, due to the "weaker" features in the other two data sets, the code will sometimes fail or yield not ideal results.

4.6 Disparity Map

Same as above.

4.7 Depth Image

Same as above.