Steven Gambino

Assignment 3 part 2

Visual Odometry

Reading the Images

Images were first taken from the input folder in the given Bayer format. They were converted to color images using demosaic 'gbrg'. At every step, this was done for the current image and the image directly after it. Next the camera parameters were taken using the ReadCameraModel function provided. The two frames were then undistorted using the UndistortImage function provided. A Gaussian filter was then applied to each of the frames.

Point Correspondences

After getting the images and applying the filter to them, feature points were detected using Matlab's *detectSURFFeatures* and extracted with *extractFeatures.* This method uses the Speeded Up Robus Features algorithm to find features in an image. To match the features between frames, *matchFeatures* was used, taking the extracted points as input. The matched features were accurate for the most part, but did have a few wrong matches for some frames. The matched features for two frames are shown below, with frame1 points in red and frame2 points in green.

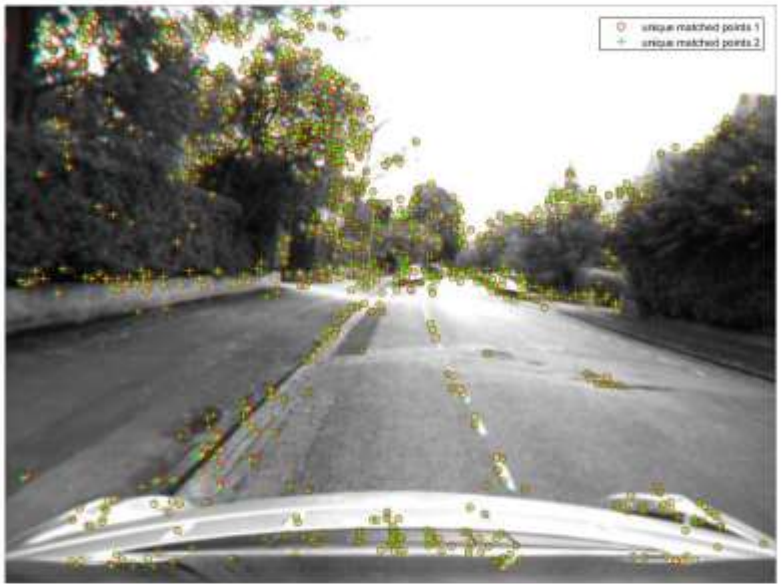


Figure 1: Matched Features between two consecutive frames

Fundamental Matrix

With the matched points, it was possible to solve for the fundamental matrix. To do this, the eight-point algorithm was used, meaning eight of the point correspondences previously found were needed. The point correspondences were chosen based on distance, meaning the lower distance between two corresponding points, the more likely it would be used for the fundamental matrix calculation. The goal was to use the best correspondences to be used in the fundamental matrix calculation. After looping and finding eight correspondences, the points were recorded as x_1 through x_8, y_1 through y_8, and x'_1 through x'_8, y'_1 through y'_8. Using these eight coordinates from each frame, a matrix A was created, as shown below.

```
A  =[x1*x1P x1*y1P x1 y1*x1P y1*y1P y1 x1P y1P 1;
     x2*x2P x2*y2P x2 y2*x2P y2*y2P y2 x2P y2P 1;
     x3*x3P x3*y3P x3 y3*x3P y3*y3P y3 x3P y3P 1;
     x4*x4P x4*y4P x4 y4*x4P y4*y4P y4 x4P y4P 1;
     x5*x5P x5*y5P x5 y5*x5P y5*y5P y5 x5P y5P 1;
     x6*x6P x6*y6P x6 y6*x6P y6*y6P y6 x6P y6P 1;
     x7*x7P x7*y7P x7 y7*x7P y7*y7P y7 x7P y7P 1;
     x8*x8P x8*y8P x8 y8*x8P y8*y8P y8 x8P y8P 1;];
```

The Singular Value Decomposition of A was then taken to find UDV^T. Then, the last column of V was used and reshaped to find the Fundamental matrix F. An SVD of F was taken to find FU,FD,FV, and the last value of FD was set to zero. The fundamental matrix *Fund* was then calculated with FU*FD*FV.

```
F = reshape(V(:,9),3,3)';
[FU, FD, FV] = svd(F);
FD(3,3)=0;
Fund = FU*FD*FV;
```

Essential Matrix

The intrinsic camera matrix *K* was found using the values obtained from *ReadCameraModel.* The Essential matrix was then found using *K,* its transpose *K'*, and the fundamental matrix *Fund*:

```
E = K' * Fund * K
```

Obtaining Rotation and Translation Matrices

To find the rotation and translation matrices, the SVD of the Essential matrix *E* was taken to get *U1,sig,V1.*

```
[U1, sig, V1] = svd(E)
```

A matrix *sig1* was then created to and used to adjust the essential matrix, calling the new one E_2.

```
sig1 = [1 0 0; 0 1 0; 0 0 0]
       E2 = U1*sig1*V1
```

The Rotation and Translation matrices were taken from E_2 by taking the SVD and utilizing matrices *W* and *Z,* as seen below.

```
W = [0 -1 0;            Z = [0 1 0;
     1  0 0;                -1 0 0;
     0  0 1;]                0 0 0;]
```

```
[U2,sig2,V2] = svd(E2);
R1 = U2*W'*V2';
R2 = U2*W*V2';
Tx = U2*Z*U2';
t1 = [Tx(3,2) Tx(1,3) Tx(2,1)];
t2 = -t1;
```

As seen, two Rotation matrices and two Translation matrices were found. To narrow down which ones to take, the algorithm took advantage of the fact that we knew it was a car that was moving forward. By knowing that the car will only move in the forward direction, the format of rotation matrices, and inspecting signs (positive/negative), the correct Translation and Rotation matrix were chosen.

Plotting Camera Trajectory

After obtaining the Rotation and Translation between two frames, it was possible to plot the trajectory of the camera. This was done by setting an initial *Rpos = [1 0 0; 0 1 0 ; 0 0 1]* and an initial *tpos = [0 0 0]* and updating them with the rotation *Rgood,* and the translation *tGood.* After each frame they were updated using:

```
Rpos = Rgood*Rpos
tpos = tpos + tGood*Rpos;
```

The x and z values of *tpos* could then be plotted to get a 2D representation of the camera trajectory.

Issues

After applying the steps above, the results came out unsatisfactory (see Results section). It seemed the fundamental matrix calculation would be off which would lead to incorrect rotations and a plot that was not representative of what the car was actually doing. The method to find the fundamental matrix was adjusted with different thresholds but without success. Although it was quoted in the assignment to ”NOT use Matlab’s Computer Vision Toolbox…”, it was used to find the fundamental matrix to obtain accurate results. The function *estimateFundamentalMatrix* was used with the matched points found earlier using the RANSAC method. After finding the fundamental matrix using Matlab, the rotation and translation matrices were found with custom functions using the method shown in the previous section.

Results

When using the custom functions for finding the fundamental matrix and rotation/translation matrices, the results hardly lead to a path, and it seemed like the car was moving in a tight circle. The path for a portion of the video is shown compared to using the Matlab functions *estimateFundamentalMatrix* and *relativeCameraPose*. The result does not look good:

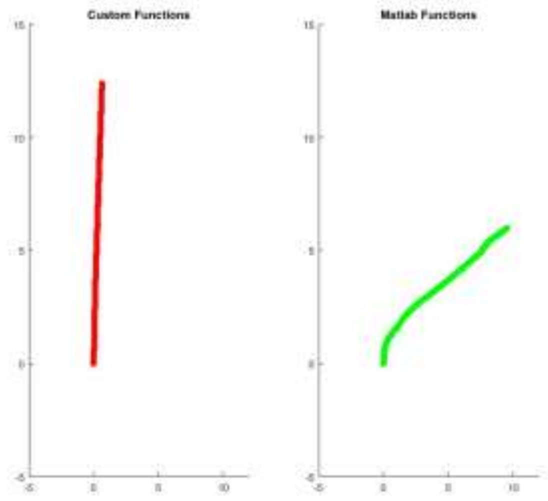


Figure 2: Custom function vs Matlab results

After struggling to get results, the *estimateFundamentalMatrix* function was used with the custom rotation/translation method. The results came out much better:

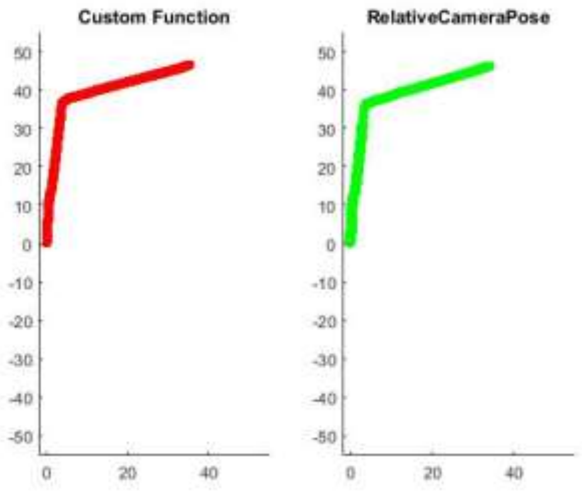


Figure 3: Using estimateFundamentalMatrix and custom R|t functions for frames 30-1440

This method was done for different portions of the video, and then for the whole video, always being compared to the *relativeCameraPose* method. For these portion of the video comparisons, the actual coordinates are not accurate since it was started in the middle of the video. The whole path result contains accurate coordinates.

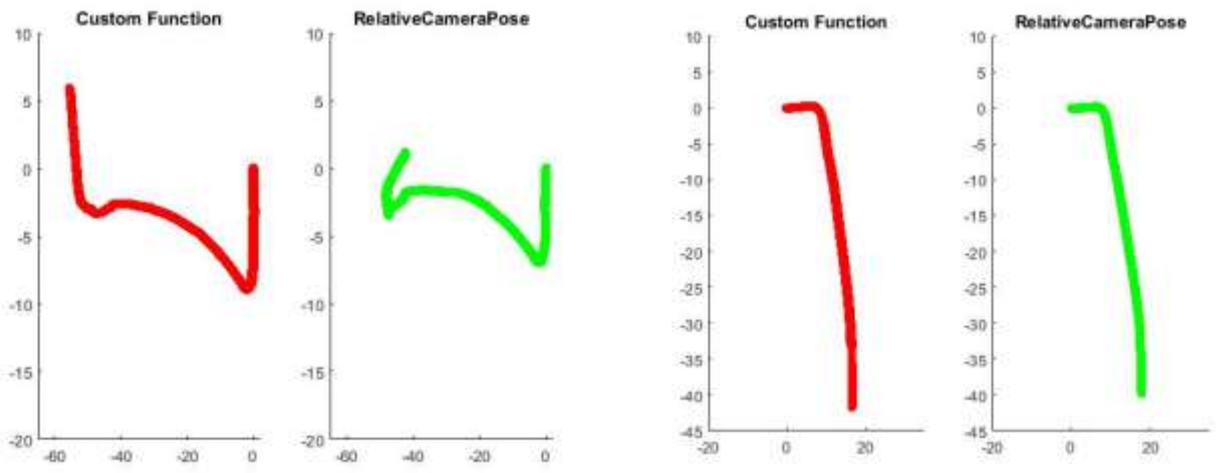
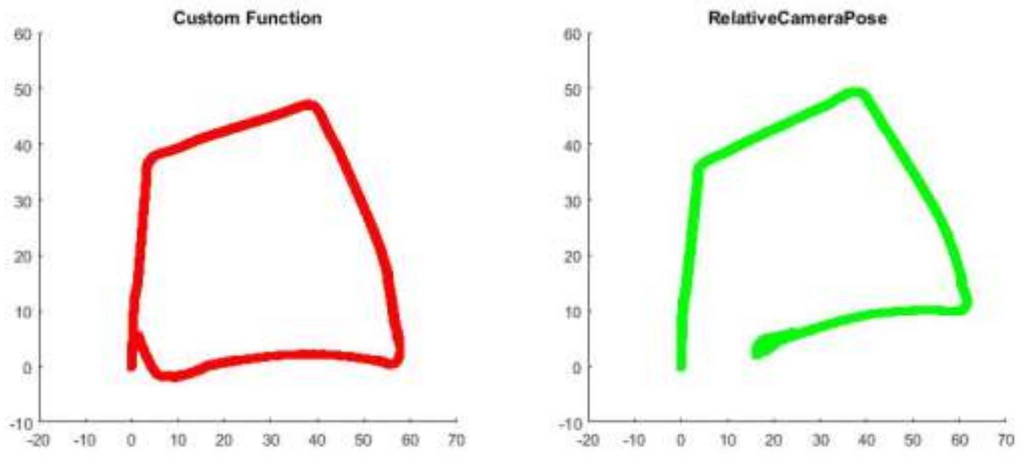
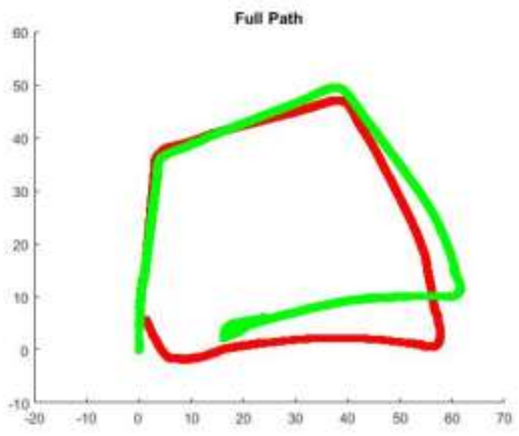


Figure 4: Comparisons of the custom R|T function with relativeCameraPose function. Left picture is for frames 1370-2400. The right is 2400-3870

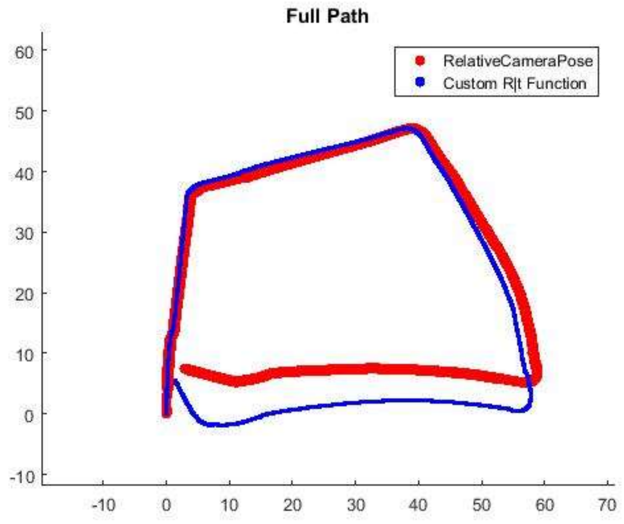
The final path output is shown below:



In the figure shown above, it is seen that the *relativeCameraPose* has a very hard right turn recorded toward the end. Another run using *relativeCameraPose* was done and gave different results. This is probably from using the RANSAC method, where the same output will not come out twice due to randomness. Custom shown in red, Matlab in green.



Another run for the Matlab functions was done:



In the image above, the custom is in blue (plotted thinner to better see a comparison) and the Matlab functions are in red. They are very similar until the midway when they start to differ. The last straightaway is a little off. It is noted this run gave better output for *relativeCameraPose* as opposed to the one above shown in green.

Conclusions

The method described did not work well, and future adjustments need to be made to correctly estimate the fundamental matrix. The extraction of the Rotation and Translation matrices did work well however, and when paired with Matlabs function to get the fundamental matrix, gave good results. The comparison between the two are close, but that is sort of expected given that the fundamental matrix was found in the same way. Finding the fundamental matrix using a different method or approach will probably lead to better results.