Assignment 1

ECE 736: 3D Image Processing and Computer Vision

Pranav Rawal

Department of Electrical & Computer Engineering, McMaster University

Question 1: Camera Calibration

1.1: Linear camera calibration

There were two Matlab scripts associated with solving this question:

- **LinearCalib.m -** function which takes two inputs. First being the set of homogeneous coordinates for the 3D features of the calibrate object, and the second input being the matching 2D image points corresponding to the calibration object. The function outputs the project matrix containing the intrinsic and extrinsic parameters of the camera. The process by which the function computes the projection matrix is highlighted below:
    1. Assembling the A matrix from the 2D and 3D points supplied as inputs to the function.
    2. Taking the SVD of A and taking the last column of the **V** as the projection matrix represented as a column vector of dimension (1x12).
    3. The projection matrix is then reshaped into the (3x4) matrix form and outputted.

- **LinearCameraCalibration.m –** a script which computes the projection matrix using the LinearCalib function then proceeds to separate the intrinsic and extrinsic parameters from the projection matrix following the process highlighted in the notes. The resulting calibration (K), rotation (R), and translation (t) matrices are provided below:

```
K =

  1.0e+03 *

   1.2383    0.0835    0.0003
        0    1.2354    0.0003
        0         0    0.0010
```

```
R =

    0.8484    0.0891   -0.5173
   -0.2528    0.9363   -0.2528
    0.6125    0.6278    0.6125
```

```
t =

    5.5302
    6.4160
    3.2347
```

1.2: 3D to 2D projection

There were three Matlab scripts associated with solving this question:

- **LinearCalib.m –** function used to produce the projection matrix from the last question.
- **CameraProject.m** – function which takes the 3D feature coordinates of the calibration object and the projection matrix from LinearCalib function and outputs a (2xn) dimension matrix containing the projected 2D image points.
- **threeD_to_twoD_prj.m** – script which verifies the correctness of the projected 2D points from CameraProject function with the original 2D points provided using a simple percent error calculation of each data point. The max error is 0.0036% in the projected u-values and 0.0028% in the projected v-values. This script also projects some other 3D points saved in '3Dpoints.txt' to see how the new points are mapped to the image plane. The contents of '3Dpoints.txt' are as follows:

```
0.0000    0.0000    0.0000    1.0000
1.0000    1.0000    1.0000    1.0000
5.0000    5.0000    5.0000    1.0000
0.0000    1.0000    5.0000    1.0000
5.0000    0.0000    5.0000    1.0000
3.0000    2.0000    1.0000    1.0000
```

And here are the projected 2D points from the script:

u	v
300.500000104155	287.436111100618
300.499999964107	287.436277603625
300.500002204733	287.433613723252
1436.70135250631	681.054963997815
300.499998039523	4083.23153010537
-155.858508850059	287.436610619179

These results show that the 3D points (0, 0, 0), (1, 1, 1), (5, 5, 5) are all mapped to approximately the same 2D image point (300.5, 287.436). This is due to the fact the all of these

points are scalar multiples of one another and thus lie on the same line of sight. Therefore, from the camera's perspective all three of these points lie on the same point. The slight inconsistencies between the projected 2D points are likely due to the inaccuracy in the collection of the 2D point coordinates from which the projection matrix is based on. These inaccuracies in the 2D points means that an exact solution of the projection matrix is not possible, however, the inaccuracies are relatively small and that is reflective of the fact that 37 samples were used to solve for the projection matrix as opposed to 6 samples, which is the minimum requirement.

Question 2: Fundamental Matrix Estimation from Point Correspondences

There were three Matlab scripts associated with solving this question:

- **funMatrix.m** – script that estimates the fundamental matrix using the normalized 8-point algorithm, computes the epipolar lines corresponding to the image points from the two images, plots the image points and their corresponding epipolar lines onto their respective images and finally calculates the distance from the image point and its corresponding epipolar line.
- **normalizeTransformation.m** – function which outputs a 3x3 transformation matrix which can normalize (3xn) dataset containing 2D points (in homogeneous coordinate system). normalizes the dataset to have mean 0 and standard deviation of 1 (computing the z-score). It is similar to the built in normalize function in Matlab which normalizes the data to have mean 0 and standard deviation 1. The difference being that instead of simply outputting a normalized dataset, it outputs a transformation matrix which can normalize your dataset or denormalize, which is very useful in the normalized 8-point algorithm.
- **linspaceNDim.m –** this function was written by Steeve AMBROISE and was posted on the Mathworks File Exchange website, I have added the link to the page in the references. This function adds to the built in Matlab function, linspace, by generating a N-dimensional matrix. This function was used to plot the epipolar lines on the images.

The normalized 8-point algorithm was used instead of the regular 8-point algorithm because without the normalization the distance from the image points to their corresponding epipolar lines was quite high even with the rank 2 constraint on the estimated fundamental matrix. The four images associated with the 8-point algorithm are shown below.

Figure 1: Image 1 Set 1, 8-point Algorithm

Figure 2: Image 2 Set 1, 8-point Algorithm

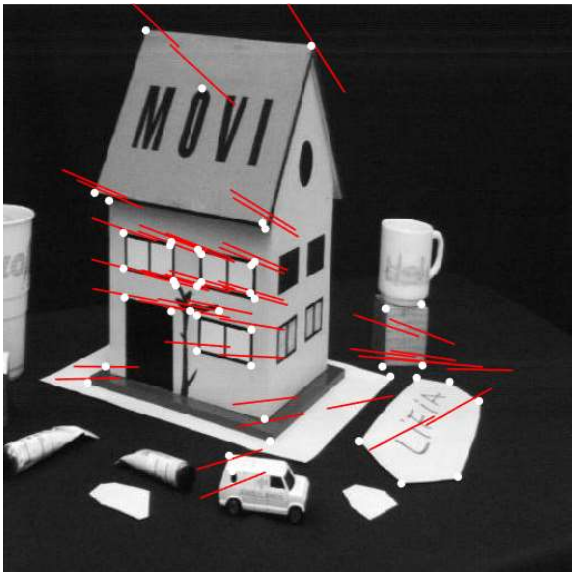

Figure 3: Image 1 Set 2, 8-point Algorithm

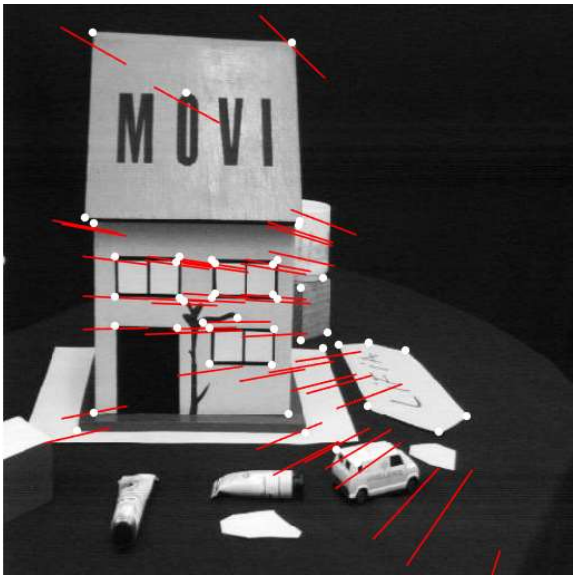

Figure 4: Image 1 Set 2, 8-point Algorithm

```
d1 =

   28.0257


d2 =

   25.1629
```

```
d3 =

    9.7014


d4 =

   14.5682
```

Where d1 and d2 represent the average distance in image 1 and 2 in set 1, respectively.

And d3 and d4 represent the average distance in images 1 and 2 in set 2, respectively.

It is evident from the four figures shown on the last page along with the distance values that the performance of the estimated fundamental matrix is not sufficient.

This is what motivated the implementation of the normalized 8-point algorithm. Using the normalizeTransformation function, a transformation matrix can be implemented which is used to both normalize and denormalize the dataset to get improve the estimation of the fundamental matrix from the SVD operation. The results are provided below, and they clearly illustrate how the normalization improves the 8-point algorithm:

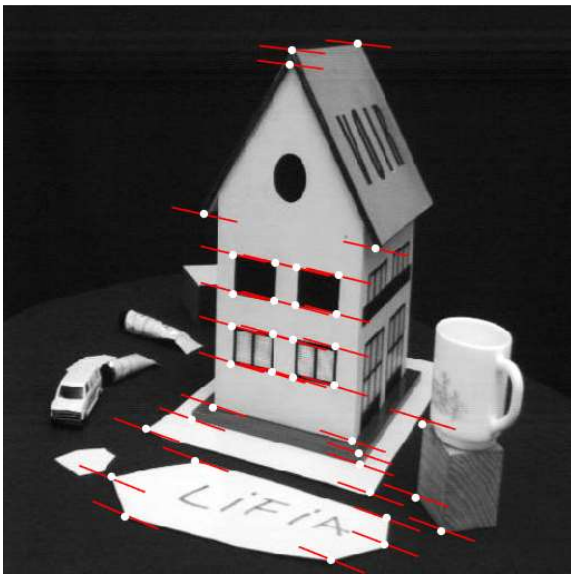

Figure 5: Image 1 Set 1, Normalized 8-point Algorithm

Figure 6: Image 2 Set 1, Normalized 8-point Algorithm

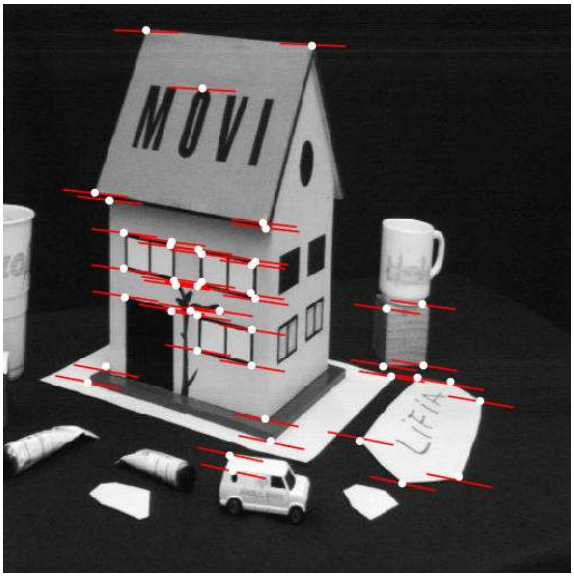

Figure 7: Image 1 Set 2, Normalized 8-point Algorithm

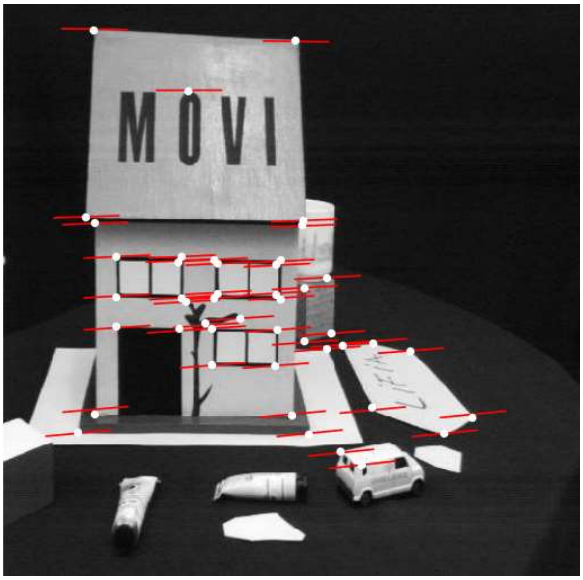

Figure 8: Image 2 Set 2, Normalized 8-point Algorithm

```
d1 =

    0.8830

d2 =

    0.8230
```

```
d3 =

    0.8912

d4 =

    0.8934
```

References

Ambroise, S. (2014). Linearly spaced multidimensional matrix without loop. *MathWorks*, https://www.mathworks.com/matlabcentral/fileexchange/22824-linearly-spaced-multidimensional-matrix-without-loop