Project 3: Visual Odometry

ENPM673 Perception for Autonomous Robots - Spring 2017

University of Maryland

Project Report by

Anirudh Topiwala

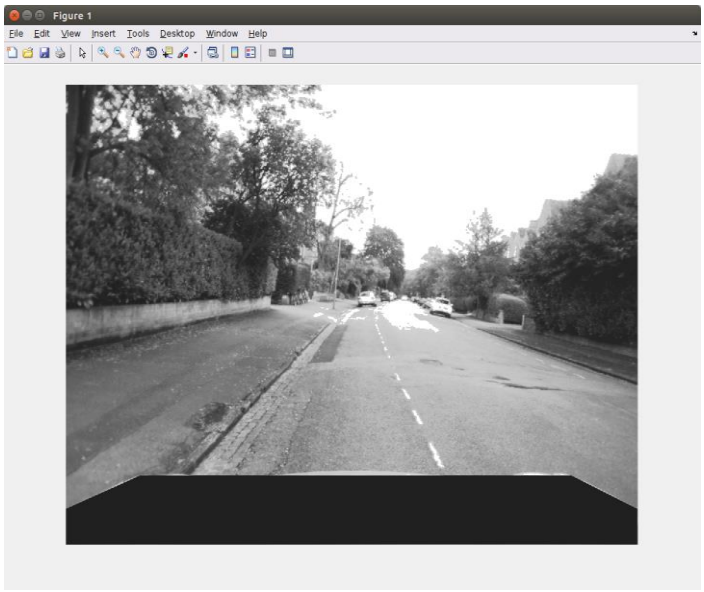
UID: 115192386

Contents

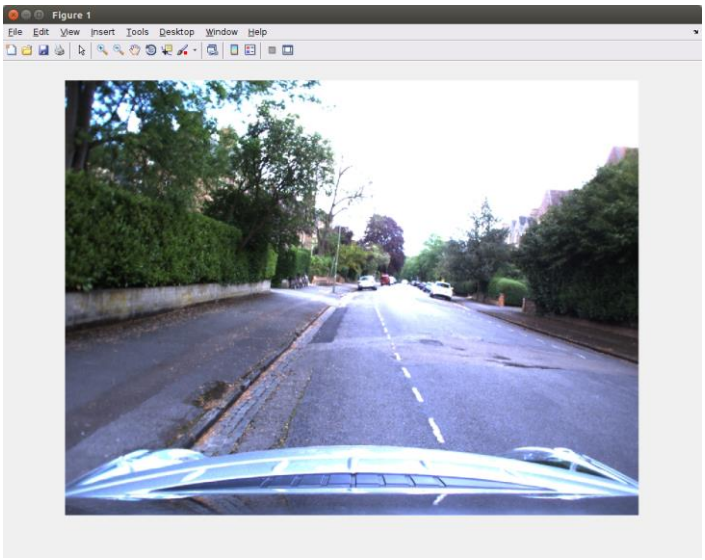
Chapter 1
Brief Pipeline
1) Pre- processing:

(a) **Mosaic to RGB:** The given image is a mosaic image, so the first step is to demosaic image in order to further process it. This is done by using demosaic function with GRBG configuration.
(b) **Undistort Image:** Next Step is to undistort the image using the undistortion lookup table (LUT) produced by given function ReadcameraModel. Undistorting the image gives us a better image which is then used for feature detection and to compute visual odometry.
(c) **Inserting Shape on the Front Hood of the Car:** this is done because, the front hood remains always stationary with respect to camera, that is, there should be no change in correspondence points between images when car moves forward. To ensure that no features are detected here, a black trapezium of the size of the from hood is added on the image.
(d) **Convert Image to Grayscale:** As feature detection requires a 2D image. The RGB image is converted into Grayscale using rgb2gray function.
(e) **Equalization of Image:** As I wanted to improve the contrast of the image for better feature detection, I have used histeq() to equalize the image or increase contrast of the image.

Final preprocessed Image compared to RGB image.



Final Pre- Processed Image



RGB Image

2) Feature Detection

This is one of the most important step in the algorithm. SURF features are detected and tracked across all the frames. Once the tracker detects the corresponding points in the current frame, this information can be used to compute fundamental matrix and then translation and rotation between two frames. The detected corresponding points is displayed in the final output video as well as the final trajectory.

3) Visual Odometry:

(a) Fundamental Matrix

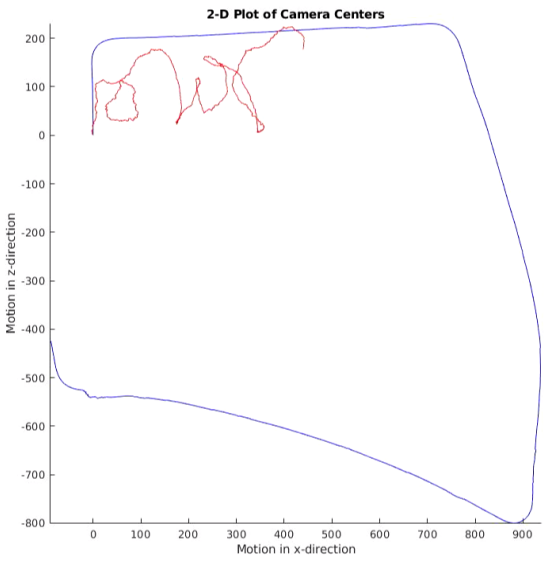
1. Methodology

1.1 Getting Fundamental Matrix: (getfundamentalmatrix.m)

- This is is the first step to estimate the camera center. The fundamental matrix is derived by the relation

$$\mathbf{A}\,\mathbf{f} = \begin{bmatrix} x'_1 x_1 & x'_1 y_1 & x'_1 & y'_1 x_1 & y'_1 y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_n x_n & x'_n y_n & x'_n & y'_n x_n & y'_n y_n & y'_n & x_n & y_n & 1 \end{bmatrix} \mathbf{f} = 0$$

- Where (x1,y1) and (x1',y1') are normalized corresponding Points. The solution to this problem is the last column of the V matrix generated by [U S V] = svd(A).
- Further operation include:
    - making the rank of the F matrix generated to 2
    - Re Transforming back the fundamental matrix by using the transformation matrices generated by normalizing the points.
    - Diving F by its norm
    - Negating F if the last element of F or F(3,3) is negative). This is an additional constraint and discussed later in the report.
- **Normalization:** the points are normalized by shifting all the points around the mean of the points and enclosing them at a distance of sqrt(2) from the new origin or the new center.
- The result from the using a normal8 point algorithm is shown below. It is compared with the matlab's normal 8point algorithm output. Note: the frames here are started from 650 to quickly visualize the turn. Also all the points are used to calculate the fundamental matrix rather than just 8 point as per the discussion mentioned here.



https://www.cc.gatech.edu/~hays/compvision/results/proj3 /html/vkhurana9/index.html

Because of the incorrect output, I adopted RANSAC.

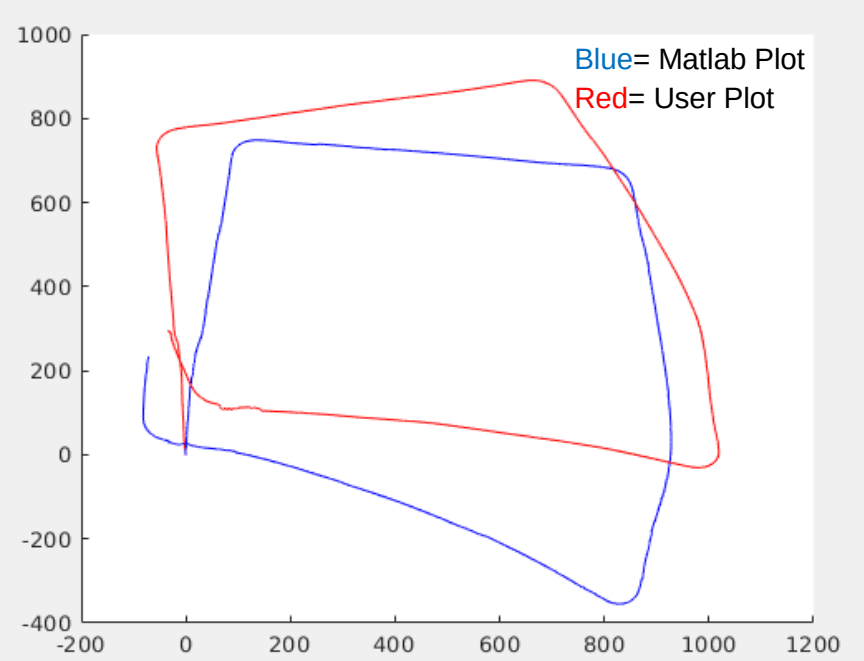
1.2 Ransac: (Ransac.m)

- After not getting good results just by using the 8 point algorithm or normalized 8 point algorithm Ransac is adopted.
- Here, the trick is to take random 8 points from the list of corresponding points and then calculating the fundamental matrix using normal 8 point algorithm.
- Then error is calculated by using sampson's formula. Here x1 and x2 relate to all corresponding points in image1 and image 2. F1 is the fundamental matrix obtained and F2 is F' or inverse of F. This is because for line l2, we would need F of Frame 2 with respect to Frame 1.

Epipolar line

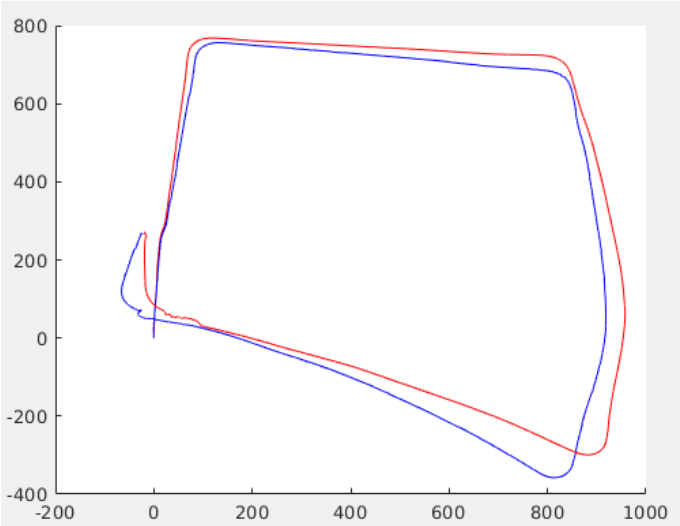
$$e = \frac{|ax+by+c|}{\sqrt{a^2+b^2}} = \frac{|\mathbf{x}_2^T\mathbf{l}|}{\sqrt{\mathbf{l}_1^2+\mathbf{l}_2^2}} = \frac{|\mathbf{x}_2^T\mathbf{F}\mathbf{x}_1|}{\sqrt{(\mathbf{F}_1\mathbf{x}_1)^2+(\mathbf{F}_2\mathbf{x}_1)^2}}$$

- Once we have the error, we check if the error is under a limit (discussed later), if yes then the points taken to calculate the fundamental matrix are termed as inliers.
- This entire process of selecting the 8 random points and finding inliers is repeated many times to get robust amount of inliers. In my case, I have used 1000 iterations.
- Once we have the inliers selected we find the new fundamental matrix just by using these inlier points.
- This method, gives a better output than the normal 8 point algorithm but still is not following the matlab path completely.



1.3 Ransac Using Zhang's Robust Fundamental Matrix Estimation. (Ransac.m)

- The problem with just randomly selecting 8 points is that, there may be repeated selection of points or maybe the point selection is clustered to some part of the image.
- To prevent this, I have used Zhang's method to divide the image into a 8 by 8 grid.
- Then by randomly choosing the grids, and by randomly choosing a point inside this grid (if there are multiple points) I have selected the 8 points.
- Once we have the 8 ,points the original Ransac code can again be executed and final fundamental matrix is found by the inliers generated.
- The output after applying this is:

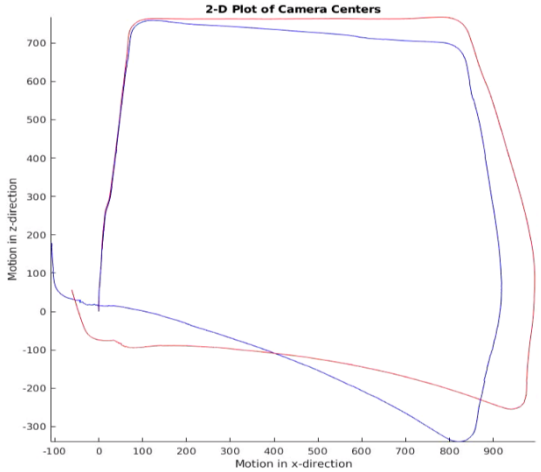


Blue= Matlab Plot
Red= User Plot

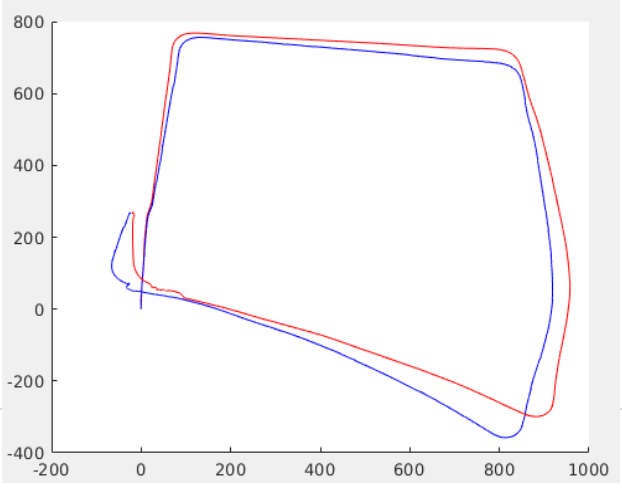
Variations in output depending upon the Error limit:

To select the best value of error threshold or to see the variation in output depending upon the inliers generated, I have outputs for two error values of 1 and 0.1.

When error limit is set as 1



When error limit is set as 0.1



Seeing that the best results are when error is set as 0.1 or when number of inliers are minimum, threshold is set to 0.1.

Constraints Added for Part a (Fundamental Matrix Calculation)

1. The first thing i ensured is that the last element of the fundamental matrix is always positive. This was implemented after comparing the Fundamental matrix generated by matlab. This may be because we always want the scaling factor to be positive and the last element of F represents the Scaling Factor.
2. Secondly the error threshold is currently kept at 0.1 to detect inliers, this can be played with to get the most optimal error threshold.
3. Thirdly, Calculating fundamental just by using the inlier points is giving me better results than any other method. Therefore, using this constraint works out the best for me.

(b) Getting Camera Pose

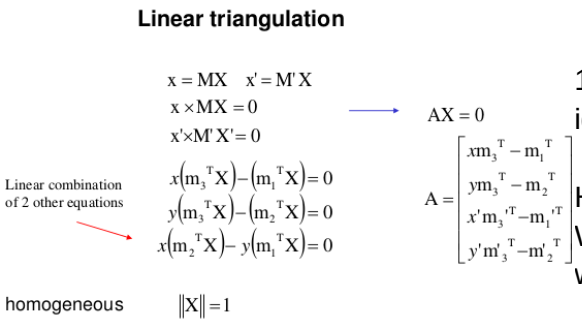
Methodology:

(a) Calculating Essential Matrix: the essential matrix is simply calculated by **E= k' *F* k** where k is the lower triangular intrinsic matrix.
(b) Getting the all the solutions of rotations and translations: (essentialmatrixbreakdown.m)

The steps for this are as follows:
1) Regenerate the Essential matrix with rank 2 constraint.
2) Decompose the E again and Use 'W' as the diagonal matrix to find the two pairs of rotation.
    a) W is generated by converting cross product to dot product.
    b) rot1 = u * W' * v'; %Prove given in"Multiple View Geometry" Page 258, -9.14
       rot2 = u * W * v';
3) Get translation as it is the last column of U .
    a) Constraint used is that T(3) or relative Z component should always be positive, as car is moving forward.
4) Therefore we only get 1 translation as solution, this reduced the total pairs of rot and trans to 2 and therefore reducing computation time.
5) Another constraint used is that, the determinant of rotation matrix generated should always be positive, if not then negate the rotation matrix. This is done as det(rot) can never be negative.

(c) Get Best pair or Triangulation: (getbestpair.m)

- Here, the general methodology is that for each corresponding point we triangulate the point in world coordinates. This is done by taking the last column of V which is generated by decomposing A.



The important points to keep in mind:

1) Here m1 and m2 are first two rows of Horigin which is an identity matrix as we consider frame 1 to be origin.

2) m3 and m4 are third and the fourth rows of **inv(H).** H is generated by taking a pair of rotation and translation. We take inv(H) because we want rot and trans of Frame 2 wrt Frame 1.

$AX = 0$ Homogenous system:

X is last column of V in the SVD of $A = U\Sigma V^T$

- Once we get X or the point in 3D world, we find Xdash or position of X wrt to other frame which is Xdash.
- Now if only X and Xdash are positive, that is the point is front of both cameras
  - And if either of only one rotation and translation pair gives the above results, then we return this pair as are selected pair. If not, then we move on to the next set of corresponding points.
  - This is because if both the sols of rot and trans gives us x and xdash as positive then there is some error.
  - Also, if no x and xdash turns out to be positive, we pass identity as the solution.
- Constraint: If translation(3) is negative we negate the entire translation as we want positive z movement.

(c) Continuous Generation of H matrices and extracting the last column to get camera center.

This is done to convert relative rotations and translations wrt world. This is done by using

h= h*[newR newT ; 0 0 0 1]

Once we have h, we just plot the last column to get the plot between x and z.

Constraints Added for Part b (Relative Camera Pose))

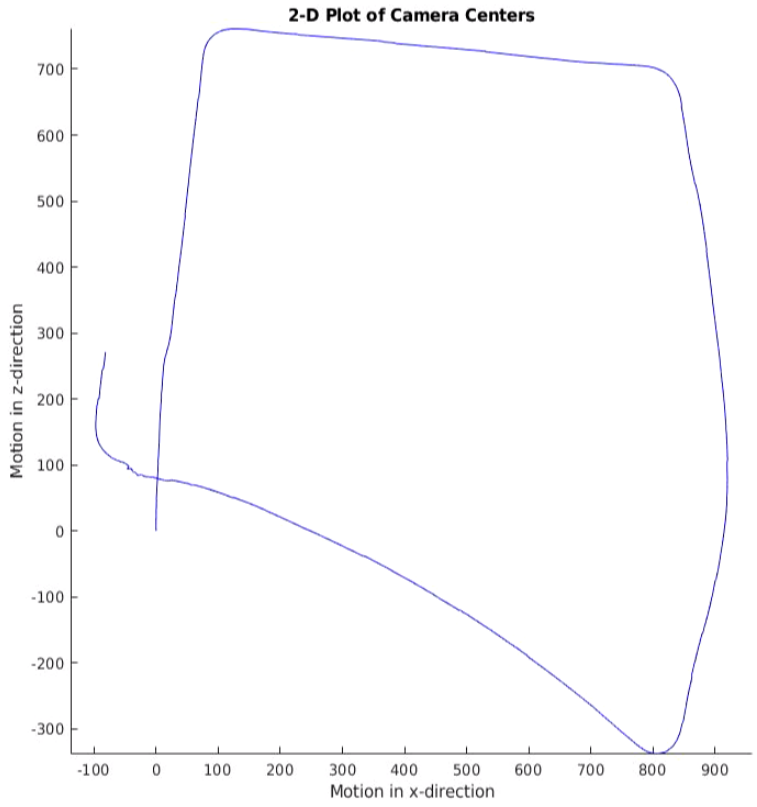
1) det(R) should always be positive.
2) Translation(3) or relative movement in Z should always be positive.

(4) Plot Using Matlab Function

To compare results, matlab plot is also generated. The steps after we get corresponding points are as follows:

```
%% Project 2- Anirudh Topiwala

function  [xc, yc, zc,H,h]= getmatlabplot(matchedPoints1,matchedPoints2,intrinsic,h,xc,yc,zc,c,H)

%% Generating Fundamental Matrix
    cameraParams = cameraParameters('IntrinsicMatrix',intrinsic');


    [F, inliers] = estimateFundamentalMatrix(matchedPoints1, matchedPoints2,'Method','RANSAC');
    % [E,inliers] = estimateEssentialMatrix(inliers1.Location, inliers2.Location,cameraParams);

    %% A debugging tool to see how many inliers are generated with respect to the total number of matched points.
    % h=msgbox( sprintf('inliers for c #%d is %d with tot points %d', c,sum(inliers), size(inliers1.Location,1)));
    % pause(1);
    % delete(h);

    %% Getting Relative Pose
    [newR,newT] = relativeCameraPose(F,cameraParams,matchedPoints1.Location(inliers,:), matchedPoints2.Location(inliers,:));
    Estore(:,:,c-1)=F;    %  Storing Fundamental matrices generated

    %% Continous Generation of H matrices and extracting the last column to get camera center.
    h= h*[newR' newT' ; 0 0 0 1];
    xc(c)=h(1,4); yc(c)=h(2,4); zc(c)=h(3,4);
    H(:,:,c-1)=h;



end
```

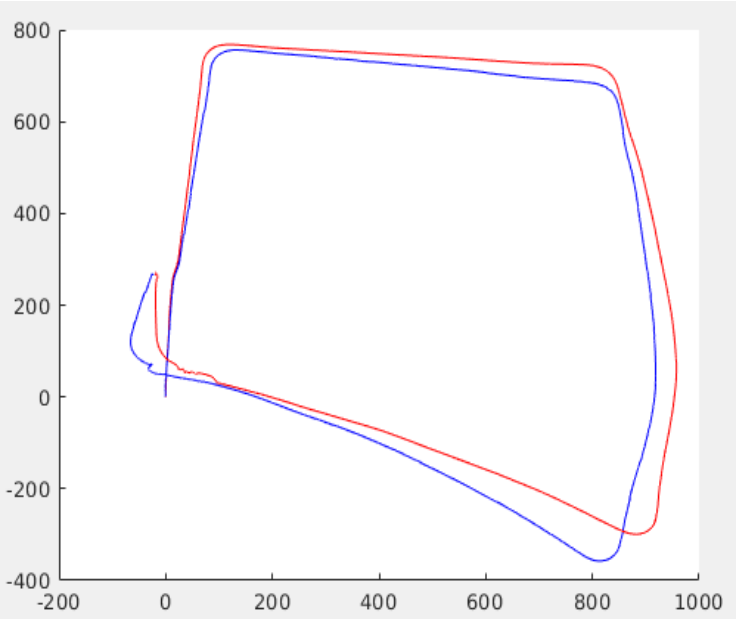
Once we start getting h, we can keep plotting the last column to get the movement of camera center.



Matlab Plot

4. Result

The final output by using Ransac with Zhang's Robust estimation method and error threshold as 0.1 when compared against matlab's output is as follows:



Blue= Matlab Plot
Red= User Plot

As we can see there is a difference between the matlab plot and the user defined functions plot. **This drift can be accounted because of the use of RANSAC as we are always choosing random points.** To further improve the drift, we can increase the number of iterations in Ransac to get better inliers.

To visualize this output please check the attached video file.

5. References:

1) Multi-View Geometry in Computer vision by Richard Hartley
2) Coursera, Robotics Perception Course Lectures, UPenn, Prof. Kostas
3) https://www.cc.gatech.edu/~hays/compvision/results/proj3 /html/vkhurana9/index.html
4) https://www.youtube.com/watch?v=K-j704F6F7Q&t=3122s (Dr. Mubarak Shah's Lecture)
5) Lecture Notes