CSE 483 / ECE 483: Mobile Robotics

Assignment 3: Two View Sparse Reconstruction

General Instructions:

1. Read this document very carefully! We have discussed many complicacies that arise in the process of two view sparse reconstruction.
2. The sole purpose of this assignment is to equip the students with implementation skills. It is not enough to know the theory, one needs to implement it to gain a better understanding of the topic.
3. Start well in advance. This assignment will take fairly long.
4. **Plagiarism** is strictly prohibited. Any instance of plagiarism will resort to a score of zero over this assignment (irrespective of the scale and nature of plagiarism).

Deliverables:

A main folder with two sub-folders: *code* and *images* (the constituents of these folders are discussed below). Name the main folder with your *roll number*. Zip this folder and upload it.

1. Place all your codes in the *code* folder.
2. Place all the images in the *images* folder. Your code should be reading images from this folder so make sure you use appropriate path.
3. The main folder should also contain a short **report** (in pdf form) that briefly describes the implementation. The results and plots must be presented in the report.

To Do:

The objective of this assignment is to reconstruct the scene being captured from a calibrated monocular camera. For this assignment you will be reconstructing a sparse set of points in the scene using only two views (images of the same scene taken from different locations). The images are located in the `images` folder and the calibration matrix is given below:

$$K = \begin{pmatrix} 558.7087 & 0.0000 & 310.3210 \\ 0.0000 & 558.2827 & 240.2395 \\ 0.0000 & 0.0000 & 1.0000 \end{pmatrix}$$

Extract the images and save them in the *images* folder of your main deliverable folder. Write a main script **sparseReconstruction.m** for sparse two view reconstruction that has the required steps and functions described below:

Note: It is the main script (**sparseReconstruction.m**) which will be run to evaluate your assignment, so make sure it executes all the required steps and functions which are explained below (use the same function names and parameters while writing your code as we will be using them in future assignments).

1. **Extract corresponding points:** Read the two images and extract globally unique interest points (SURF) and establish correspondences between the points of each image using the codes provided in the link: Detect and match SURF feature points. To detect SURF features, as shown in the example for which the link is provided above, use the following command.

   ```
   detectSURFFeatures(img1,'MetricThreshold',10)
   ```

   (The value 10 for MetricThreshold variable allows for a large number of not so strong features to be detected and helps us to get rid of degeneracy caused when all features lie on the same plane (now you see why we have two planes in the scene). False matchings will be taken care by RANSAC routine.)

2. **Preconditioning the system:** Normalize the 2D points of each image to avoid ill conditioning of system used for Fundamental matrix estimation. To normalize subtract the mean from the points and scale them such that the average distance to the points is equal to$\sqrt{2}$. See **Normalizing 2D points** section for more details. Write a function to do this.

   ```
   function [normPts2D, T] = normalize2DPoints(pts2D)
   ```
   (Here, `T` is the transform which does the normalization; It will later be used to denormalize the F matrix.)

3. **Compute Fundamental matrix:** Write a function to estimate the Fundamental matrix (F) using the normalized 8-point algorithm in a RANSAC framework.

   ```
   function [norm_F] = estimateFundamentalMatrixRANSAC(corresponding_pts1, corresponding_pts2)
   ```

   Note: The rank of the F matrix is 2 and it could be so that due to noise or other factors the estimated F matrix is full rank i.e. rank3. In order to make the F matrix as rank 2 matrix do the following steps:

   ```
   [u d v] = svd(F);
   new_d = diag([d(1,1) d(2,2) , 0]);
   F = u * new_d * v' ;
   ```

   (The above process can bee seen as projecting the estimated F matrix on a Rank 2 manifold which will cause a bit of information loss but that is fine.)

4. **Compute rigid body transform between cameras:** First denormalize the estimated F matrix and compute compute the Essential matrix (E) using the calibration matrix and the F matrix. Essential matrix, just like F matrix, is a rank 2 matrix with both its singular values same. So again due to noise or other factors, the E matrix might not have above properties. So to convert E matrix into a valid Essential matrix do the following:

   ```
   [u d v] = svd(E);
   new_d = diag([(d(1,1)+d(2,2))/2, (d(1,1)+d(2,2))/2 , 0]);
   E = u * new_d * v' ;
   ```

   Decompose the E matrix into R and t using the function which has been provided to you with this assignment (located in the *available_codes* folder). The function prototype is shown below:

   ```
   function [R t] = decomposeEssentialMatrix(E, pts2D_1, pts2D_2, K)
   ```
   (where, `E = K'*F*K` and `F = T2*F_norm*T1`)

   Read the comments in the function to know the formats of the input and output.

5. **Reconstruct the scene:** A function to algebraically triangulate (reconstruct) the interest points:

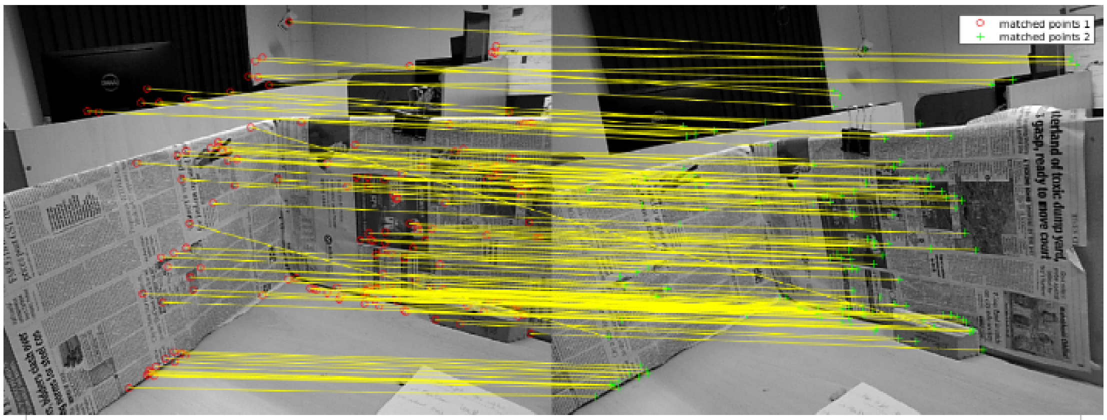
   ```
   function [pts3D] = algebraicTriangulation(pts2D_1, pts2D_2, ProjMat_1, ProjMat_2)
   ```
   (Where `ProjMat_1 = K*[eye(3,3) [0 0 0]']` and `ProjMat_2 = K*[R t]`)

6. **Results and plots to be displayed**:

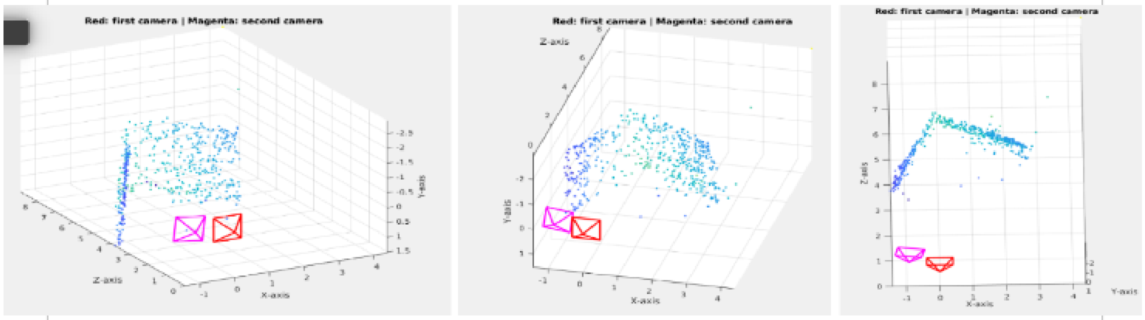
   (a) The main script should in the end print the F matrix, R, and t. Make sure the results are readable.

   (b) A montage plot having both the images with the correspondences shown (see below). (Hint: the link in step 3 has the functions to display the required format)

   (c) A 3D plot of the reconstructed (triangulated) points with the two camera frustums (see figure below). The code to for plotting the camera frustum is provided with the assignment (located in the *available_codes* folder).



Montage showing two images and matched features



Final result shown from different views

First and second camera frustums are shown in red and magenta color, respectively. In the top image we see a very sparse set of feature points but if you use the right MetricThreshold as explained in step-3 you will have a large number of interest points.

Normalizing 2D points:

When we directly use image coordinates in the linear system to solve for fundamental matrix estimates, SVD has numerical stability issues. To address this, we typically transform the points (coordinates of features) in each image such that they are centered at the origin and their mean distance from the origin is sqrt(2).

A step-by-step approach will be something similar to the following.

1. Compute the mean of all points. Call this *mu* = [*mu*(1); *mu*(2)].
2. Compute the average distance of all points from the origin. Call this *d*.
3. Compute the scaling factor. Call this *scale.* Mathematically, *scale = sqrt(2) / d*
4. Construct a homogeneous transform matrix *T* that will be applied to each point. In Matlab notation, *T* = [*scale*, 0, -*scale* * *mu*(1); 0, *scale*, -*scale* * *mu*(2); 0, 0, 1]
5. (Pre-)Multiply each image feature (in homogeneous coordinates) by *T* to obtain the transformed image coordinates.
6. Perform fundamental matrix estimation using these transformed image coordinates.
7. **Important**: You will have to *undo* this transformation after computation of F. Specifically, assume that your F computation is from the equation x2^T * F * x1 = 0 (T here is transpose, not the *T* matrix from step 4). If *T1* and *T2* are the normalizing transform matrices for images 1 and 2 respectively, then the fundamental matrix F obtained from solving the normalized system must be modified as *T2*^(transpose) * F * *T1*, to obtain the fundamental matrix for the original system. This is a more numerically stable way to compute F.