“[ENPM 673] PROJECT 5”

ENPM 673 – PERCEPTION FOR AUTONOMOUS ROBOTS

PROJECT 5 REPORT

ADITYA VAISHAMPAYAN -116077354

ISHAN PATEL – 116169103

NAKUL PATEL – 116334877

Instructor: Dr. Cornelia Fermuller

CONTENTS

1) Pre- processing:

A. **Mosaic to RGB: The image we are given is a mosaic image thus we need to demosaic it first. To do so we used the following commands:**
   - cv2.imread('bayer4.png', cv2.IMREAD_GRAYSCALE | cv2.IMREAD_ANYDEPTH)
   - cv2.cvtColor(imageRaw, cv2.COLOR_BAYER_GR2BGR)

B. **Undistort Image:** Next Step is to undistort the image using the un-distortion 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. **Convert Image to Grayscale**: Feature detection requires a gray image, so we first converted the undistorted image to gray image.

D. **Equalization of Image:** We wanted to improve the contrast in order to detect the features in a better way, so we used histogram equalization and increased the contrast.

Figure 1: Mosaic and de-mosaiced images

2) Feature Detection:

This is one of the most important step in the algorithm. We had to use ORB for feature detection as SIFT and SURF are patented. ORB is basically a fusion of FAST key point detector and BRIEF descriptor with many modifications to enhance the performance. 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.

Fig: Detected Feature points in the image

Fig: Key-points matched in the consecutive images

3) 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.

Points before normalization

570	387.6
710.4	416.4

Points before normalization

0.562	-0.351
1.010	-0.2595

4) Fundamental Matrix:

This is the first step in estimating, the camera center. The fundamental matrix is derived by the relation Ax = 0 where

$$Ax = \begin{pmatrix} x_1'x_1 & x_1'y_1 & x_1' & y_1'x_1 & y_1'y_1 & y_1' & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_n'x_n & x_n'y_n & x_n' & y_n'x_n & y_n'y_n & y_n' & x_n & y_n & 1 \end{pmatrix} x = 0$$

$$[x_0x_1 \quad x_0y_1 \quad x_0 \quad y_0x_1 \quad y_0y_1 \quad y_0 \quad x_1 \quad y_1 \quad 1] \begin{bmatrix} F_{11} \\ F_{12} \\ F_{13} \\ . \\ . \\ F_{33} \end{bmatrix} = 0$$

Here x1’ and y1’, x2’ and y2’ 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 includes:
  - making the rank of the F matrix generated to 2
  - Retransforming 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.

Our Function Implementation:

- getFundamentalMatrix (points1, points2) → Fundamental matrix
- **Input: points1, points2, Output: Fundamental Matrix F**

- This function takes the 8-point correspondences and thus gives us back a Fundamental matrix F.

Summarizing we can say that:

1. Obtain the unnormalized points
2. Scale and translate the points in the process of normalization
3. Compute F
4. Force F to have rank 2
5. Undo normalization i.e. undo scaling and translation

5) Zhang's 8-point Algorithm:

We make an 8x8 grid out of the image and assign all the key points from the image to the grid according to their corresponding position. Then we take 8 cells at random out of those 64 grid cells, and then we take one point from each of those 8 cells. Thus, we are able to select 8 points at random out of all the key points detected in the image.

In order to get the best inliers, we need to obtain the distance of each point from the epipolar line. If the distance of the key point from the epipolar line is lesser than one pixel we consider it as a good match. Hence it is considered as an inlier. If the distance is greater than the threshold, we discard that set of key points. The Sampson's formula for calculating the error is given below:

Epipolar line

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

- x1 and x2: corresponding points in image1 and image 2.
- F1 is the fundamental matrix obtained and F2 is F' or inverse of F.
- Because for line l2, we would need F of Frame 2 with respect to Frame 1.

The entire process of selecting the 8 random points and finding inliers is repeated multiple times. We have repeated the process for 50 iterations. Once the inliers get selected, we obtain the new fundamental matrix just by using these inlier points.

● The issue with **just** randomly selecting 8 points, is that there are chance of the same points getting selected randomly or the point being clustered at some part of the image.

● Hence, we used the Zhang's method of obtaining the 8x8 grid from the image.

● Then by randomly choosing the grids, and by randomly choosing a point inside this grid (if there are multiple points) we have selected the 8 points.

● Once the 8 points are obtained, the RANSAC function is rerun till the optimal fundamental matrix is obtained.

Constraints Added for Fundamental Matrix Calculation:

1. We try to keep the last element of the fundamental matrix positive. because we always want the scaling factor to be positive and the last element of F represents the Scaling Factor.
2. Secondly, the threshold for the error is kept at around 0.5, however we can play with it to obtain the optimal result.
3. Finally, we calculate the fundamental matrix just by using the inlier points. This gives a better result than any other method, hence we use this constraint.

Our Function Implementation:

- Ransac_eight-points (joint list) → best_inliers, Fundamental
- **Input: Point correspondences, Output: Best Inliers and Fundamental Matrix**
- The ransac_eight_points function takes I a list of both the corresponding keypoints matches and it returns the best inlier points and the optimal Fundamental matrix.

6) Getting Camera Pose:

Following is the method to estimate the camera pose:

Calculating Essential Matrix: the essential matrix is (3x3) matrix which is simply calculated by **E= k' *F* k** where k is the camera intrinsic matrix i.e. the calibration matrix.

For **F** matrix computation, the singular values of **E** are not necessarily [1,1,0] due to the noise in **K**. This can be corrected by reconstructing it with [1,1,0] singular values, *i.e.*

$$E = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

Getting the all the solutions of rotations and translations:
The steps for this are as follows:

- We regenerate the essential matrix using the rank 2 constraint.
- Decompose the E again and use 'W' as the diagonal matrix to find the two pairs of rotation.
- Get translation as it is the last column of U.

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

Constraint used is that the third element of the translation matrix T[3] or relative Z component in the translation matrix should always be positive, as car is moving forward. Hence, we get only 1

translation as solution. This way we reduce the total pairs of rotation and translation to 2 and therefore reducing computation time.

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.

The camera pose consists of 6 degrees-of-freedom (DOF) Rotation (Roll, Pitch, Yaw) and Translation (X, Y, Z) of the camera with respect to the world. Since the **E** matrix is identified, the four camera pose configurations are $(C1,R1)$,$(C2,R2)$,$(C3,R3)$ and $(C4,R4)$ where $C \in \mathrm{R}3$ is the camera center and $R \in SO(3)$ is the rotation matrix, can be computed.

Thus, the camera pose can be written as $P = KR[I_{3x3} \; -C]$. These four pose configurations can be computed from **E** matrix. Let $\mathbf{E}=U*D*V^T$ and W as given above. Thus, we get the four resulting configurations as:

1. $C_1 = U(:,3)$ and $R_1 = UWV^T$
2. $C_2 = -U(:,3)$ and $R_2 = UWV^T$
3. $C_3 = U(:,3)$ and $R_3 = UW^TV^T$
4. $C_4 = -U(:,3)$ and $R_4 = UW^TV^T$

Here, we also need to ensure that the det(R) has to be one and if If $det(R)=-1$, the camera pose must be corrected *i.e.* $C=-C$ and $R=-R$.

Our Function Implementation:

- get_R_T(F) → P (projection matrix)
- **input: Fundamental matrix, Output: Projection Matrix**
- essential matrix is counted within it and it returns a projection matrix containing the four poses.

7) Linear Check for Chirality Condition:

The important points to keep in mind:

1) Here m1 and m2 are first two rows of P. It is an identity matrix because frame 1 is considered as an origin.
2) m3 and m4 are third and the fourth rows of **inv(P).** P 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.
3) Upon getting the X or the point in 3D world, we find X' or position of X wrt to other frame which is X'.
4) If only X and X' 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.
5) It is because if both the solutions of rotation and translation gives us x and x' as positive then there is some error and also, if no x and x' turns out to be positive, we pass identity as the solution.

6) If translation (3) i.e. Z is negative, we negate the entire translation as we want positive z movement.

Continuous Generation of H matrices and extracting the last column to get camera center.
We do continuous generation of H matrices and column extraction to get camera center and thus convert relative rotations and translations w.r.t world. We use the following formula to do so:
H= H * [newR newt; 0 0 0 1]

Once, we obtain H, we plot the last column to get the plot between x and z.

Our Function Implementation:

- Tri_pts(inliers) pt correspondances(p1,p2 2 projection matrices) → 3D projected points
- **Input: Corresponding points that are inliers, Output: 3D projected points**
- This function takes in the inliers and both the projection matrices and it returns a 3D set of points.

8) Output Comparison (Extra Credits):

Time required by using OpenCV Inbuilt Function: 15 min

Time required by using our functions: 45 - 50 min

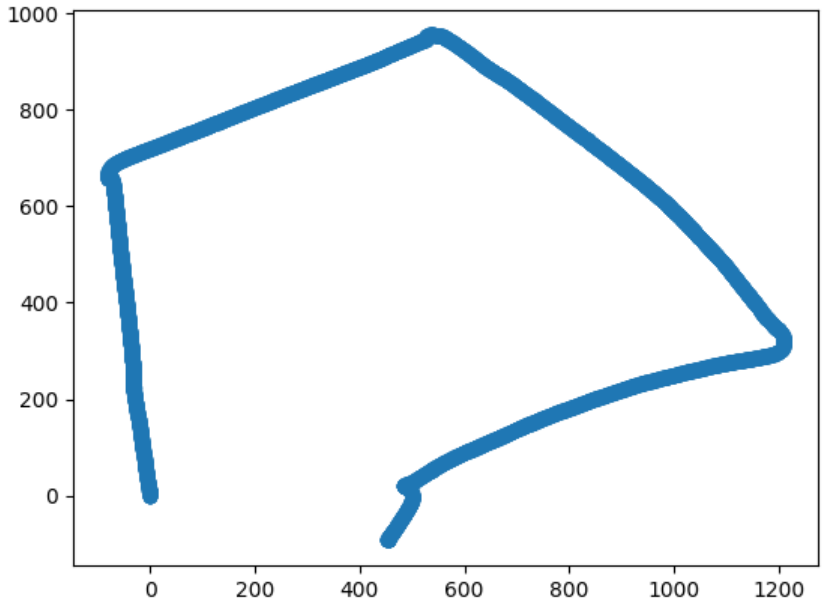

Fig: Output Trajectory obtained using OpenCV functions

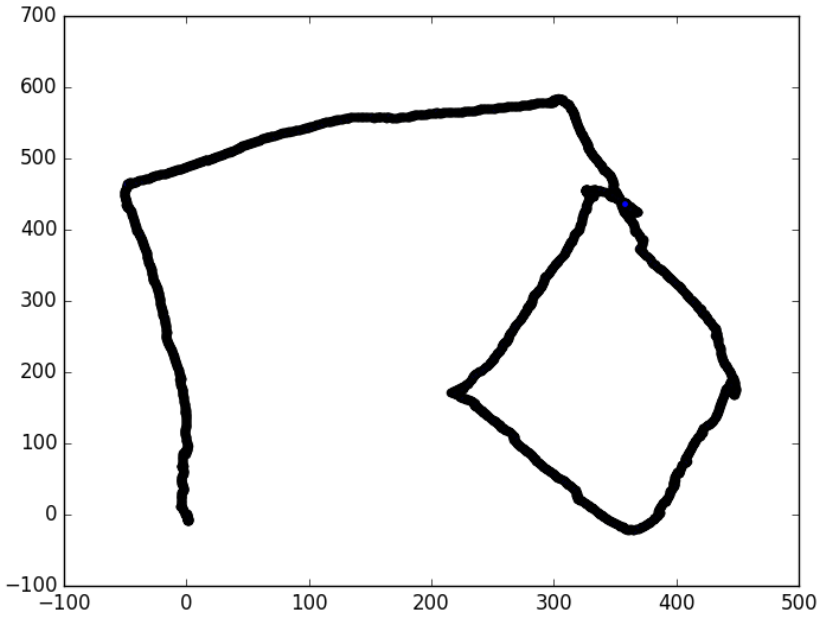

Fig: Trajectory Obtained using our Functions

We can see that trajectory that has been obtained using our functions is slightly skewed. Also, the fourth turn that has been obtained in our defined function isn't proper. The trajectory obtained by the OpenCV function is smooth whereas that obtained by our function is very jagged. Whereas the one that is obtained by OpenCV is just rotated a little. The remaining trajectory created is proper but has a wrong rotation. If had been rotated properly the trajectory plotted would be correct.

9) Non-linear Triangulation (Extra Credits):

Since we are given two camera poses and linearly triangulated points, we can refine the location of 3D world points that will help reduce the reprojection error. This error can be calculated easily by measuring the error between the measured and projected 3D points. Since the formula projects back to the image coordinates and we are dividing by the Z coordinates, the minimization problem is usually non-linear.

$$\min_{x} \sum_{j=1,2} \left(u^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X}\right)^2 + \left(v^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X}\right)^2$$

where, j = index of each camera,

$\tilde{X}$ – the homogeneous representation of X (world point)

P_1^{jT} – each row of the camera projection matrix.

10) PnP (Perspective n points) Technique (Extra Credits):

Given ($n \geq 3$) 3D reference points in the object frame-work and their corresponding 2D projections onto the images, the technique to determine the orientation and position of a fully calibrated perspective camera is known as the perspective-n-point (PnP) problem.

NonLinearPnP: Given 3D-2D point correspondences, and linearly estimated camera pose (C, R), we need to refine camera pose (position and orientation), such that the reprojection error is minimized.

$$\min_{C,R} \sum_{j=1,2} (u^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X})^2 + (v^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X})^2$$

Where, all the notations are same as non-linear triangulation, and $P = KR\,[I_{3x3}\ -C]$ is the camera projection matrix. However, since the rotation matrix had not been enforced with orthogonality, we use a compact representation using a quaternion (**R = R(q)**). Hence, the equation becomes:

$$\min_{C,q} \sum_{j=1,2} (u^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X})^2 + (v^j - \frac{P_1^{jT}\tilde{X}}{P_3^{jT}X})^2$$

11) Challenges Faced:

1. Broadcasting errors
2. Dimensionality matching error
3. Errors in optimizing the functions as it took very long in running
4. Error in using the OpenCV functions and also playing with the error scale i.e. the distance of the key point from the epipolar line
5. Trial and error about finding the Fundamental matrix with using the normalized points after passing into our functions
6. Long run times because it reads all the images first making a list of them and then accesses each image from it.
7. We also had to experiment the key point feature tracking algorithm. Orb create gave less feature key points as compared to goodFeaturesToTrack hence we ultimately decided to use goofFeaturestotrack.
8. Demosaicing the image. The given function for demosaicing the image didn't work for us so we ha-d to add an additional argument of reading the depth.

   imageRaw1 = cv2.imread(images[i], cv2.IMREAD_GRAYSCALE | cv2.IMREAD_ANYDEPTH)

9. We faced issue in sorting the key points for passing into the function of the fundamental function. We used BF matcher for matching the corresponding descriptors and sorted them.
   matches = sorted (matches, key = lambda x:x.distance)

12) Key Observations:

- Using Normalized points makes the outputs better.
- Orb create functions gives a faster result than sift and surf points.
- Making a prior video containing demosaiced undistorted images results in faster processing of the frames because, our program takes frames first, demosaics them and undistorts them resulting in increased computation time. If all this preprocessing was done earlier, then it will save time. But the video formed was around 1 GB hence it couldn't be uploaded.
- Sorting the key points before sending them to our functions was a better idea because that made the processing easier.

13) 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