University of Maryland

College Park

PROJECT 2

ENPM673 - PERCEPTION FOR AUTONOMOUS ROBOTS

Camera Pose Estimation using Homography

Panoramic Stitching of Four Images

Author:

Aaqib Barodawala (119348710)

Instructor and TAs:

Dr. Samer Charifa

Arunava Basu, Madhu NC,

Ritvik Oruganti

PROBLEM 1:
Perform camera pose estimation using homography by computing rotation and translation between the camera and the coordinate frame whose origin is located on any one of the corners of the sheet of paper.

SOLUTION:

The solution begins with importing all the necessary libraries for OpenCV, math libraries (NumPy and SymPy), itertools along with matplotlib library for basic plotting. The video is then read in using OpenCV's "read()" function to get each frame of the video. Once the frames are separated, each frame is then converted to Gray colour space and a Gaussian Blur is applied to effectively apply canny edge detector algorithm. This edge detector separates the edge coordinates of the paper from the entire frame and gives a binary image after setting appropriate threshold values. After finding coordinates of each edge point, the coordinates are then thrown in the "hough_transform" function. Here, the detected points are converted to polar coordinates and a voting mechanism is applied to search for peaks in this parameter space. The peaks can then be extracted and used as proper detected points for further image processing.

After appropriate sorting and extracting higher peaks, the points are then passed to the "check_perpendicular" function where it checks if the angle between each pair forms a perpendicular angle or not. If it does, it means that a corner is detected and the pair is further stored for processing. After getting a list of pair of polar points that form a corner, their line equations are further solved to extract the corner at which they intersect. Once intersection/corner points are detected, all the adjacent points are clustered into a single point using Manhattan distance. If the distance between the points are less than 14.6, they are grouped together and the final four corners can then be calculated by finding appropriate means of all X and Y coordinates for each clustered points.

Once the four corners are perfectly detected for each frame, the homography matrix can then be calculated by following the steps mentioned below:

1. Form the A matrix below, where x_n, y_n represents the world coordinates and x'_n, y'_n represents the corner coordinates:

$$A = \begin{bmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 & -x'_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 & -y'_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 & -x'_2 \\ 0 & 0 & 0 & x_2 & y_2 & 0 & -y'_2x_2 & -y'_2y_2 & -y'_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 & -x'_3 \\ 0 & 0 & 0 & x_3 & y_3 & 0 & -y'_3x_3 & -y'_3y_3 & -y'_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 & -x'_4 \\ 0 & 0 & 0 & x_4 & y_4 & 0 & -y'_4x_4 & -y'_4y_4 & -y'_4 \end{bmatrix}$$

2. Compute eigen vectors of $A^T A$ with the smallest eigen value.
3. Form the Homography Matrix by reshaping the resulting 9x1 matrix.

To calculate the rotation and translation matrix, we can decompose homography matrix with the steps mentioned below:

$$H = \lambda K[r_1 \quad r_2 \quad t]$$

$$K^{-1}H = \lambda[r_1 \quad r_2 \quad t]$$

$$A = K^{-1}H$$

$$[A_1 \quad A_2 \quad A_3] = \lambda[r_1 \quad r_2 \quad t]$$

$$Norm(A_1) = \lambda_1$$

$$Norm(A_2) = \lambda_2$$

$$\lambda = \frac{(\lambda_1 + \lambda_2)}{2}$$

$$[A_1 \quad A_2 \quad A_3]/\lambda = [r_1 \quad r_2 \quad t]$$

$$r_3 = r_1 \times r_2$$

Form the rotation matrix and further decompose the rotation matrix to get roll, pitch, yaw and translation components in x, y and z direction. Plot the varying values over all the frames.

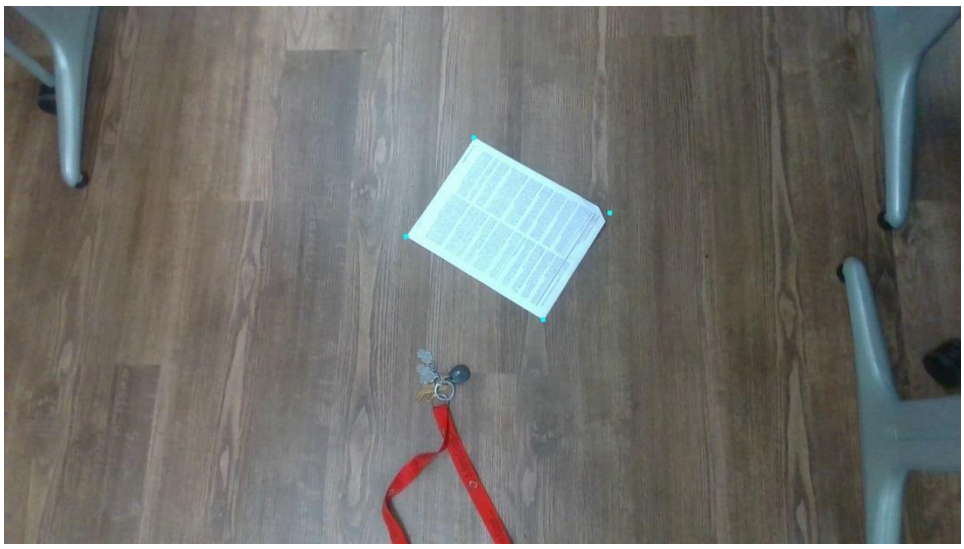
PROBLEMS FACED:

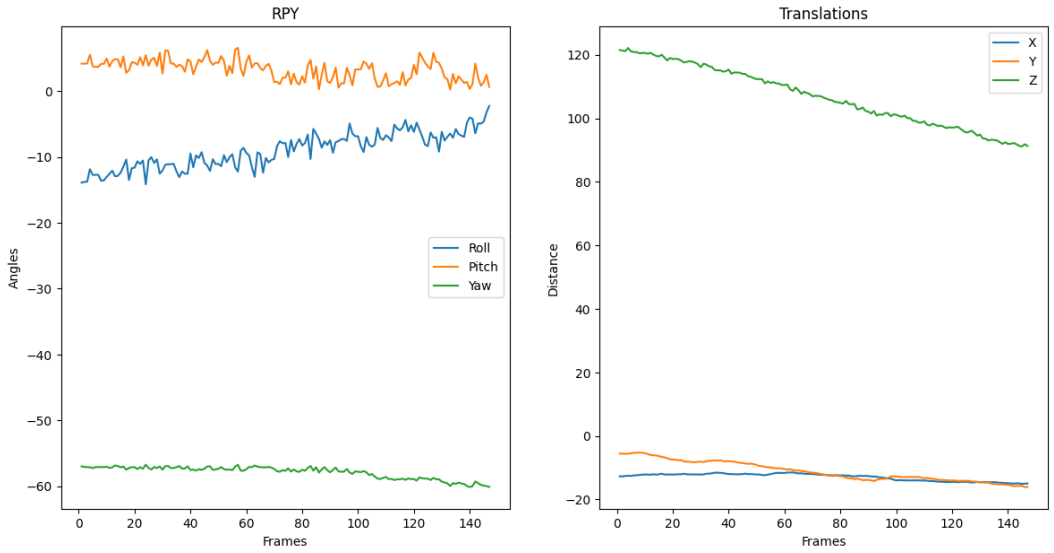
1. Detecting the corners for each frame posed to be an issue as some of the edge points were missing in those particular frames. Tweaking the threshold values for Canny Edge Detector solved the issue.

2. Clustering the points together posed to be challenge at first but after careful consideration and proper implementation of python logic, the issue was resolved.

3. The script ran slowly on the backend, causing long processing time for each frame. Using numpy arrays for particular parts helped to mitigate the problem.

RESULTS:




RPY
Angles
Frames
Roll
Pitch
Yaw
Translations
Distance
Frames
X
Y
Z

PROBLEM 2:

Stitch four images taken at the same camera position (only rotation and no translation) and develop a panoramic image.

SOLUTION:

The solution begins with reading in the four images using OpenCV's "read()" function. For each image, features (key points and descriptors) are extracted using Scale-Invariant Feature Transform (SIFT) algorithm which is available with OpenCV. Furthermore, the Brute Force Matcher algorithm was further implemented to match the descriptors of each image with one another. After finding the matches, sorting and extracting the best matches from the key points helps to get the coordinates in each image for which a match was found. The pair of coordinate points for which a match was found are stored in an array for further processing.

RANSAC is used to calculate and further optimize the homography matrix. Four random pairs are selected from the array that was previously generated. The homography is then calculated inside the RANSAC function to further calculate and compare with the best homography matrix found. The comparison is done by using an error function wherein the matrix calculated before is used to predict the source coordinate points and the error is then calculated by subtracting their norms. If the error value is less than threshold set, the pair is added to the list of inliers and the process repeats again to find the best homography matrix. If the number of inliers exceeds the threshold, the best homography matrix computed is then returned to stitch the images. This way, the best homography matrix is found between two images and are stitched using OpenCV's "warpPerspective()" function.

PROBLEMS FACED:

Major problem faced was stitching all the four images together. This problem was not rectified.

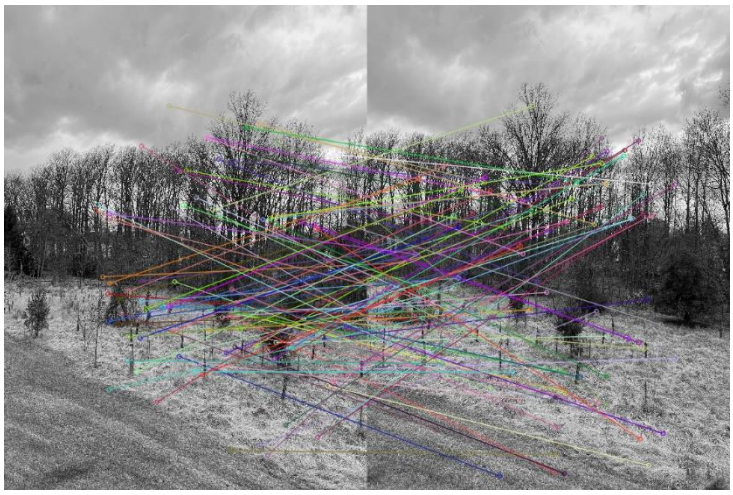
RESULTS:

Stitching Image 1 and Image 2:

Stitching Image 2 and Image 3:

Stitching Image 3 and Image 4:

Matched Features 1-2:



Matched Features 2-3:

Matched Features 3-4: