CMSC 426 – Computer Vision

Final Project: Visual Odometry

In this project I am given frames of a driving sequence taken by a camera in a car. Using this sequence of frames, I will perform a series of steps to reconstruct and visualize the 3D trajectory of the camera.

Alessandro Russo

3.1 – Compute Intrinsic Matrix:

Using theReadCameraModel.py as follows: fx, fy, cx, cy, _ , LUT = ReadCameraModel('./Oxford dataset reduced/model') I could extract the camera parameters. Using the lecture notes I could compute the intrinsic matrix by plugging in the values respectively. Here fx = ax, fy = ay, cx = px, cy= py and s is set to 0. S refers to the skewness.

$$\mathbf{K} = \begin{bmatrix} \alpha_x & s & p_x \\ 0 & \alpha_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

3.2 – Load and Demosaic Images:

The input images are in Bayer format from which I recovered the color images using the demosaic function using: img = cv2.imread(filename,flags=-1) and color_ image = cv2.cvtColor(img, cv2.COLOR _BayerGR2BGR). I undistorted the two consecutive using UndistortImage.py as follows: undistorted_image = UndistortImage(color_image,LUT)

3.3 – Estimate Keypoints:

To start I used sift_create() and took all the features and descriptors of all the images .I then used bf.knnMatch() over all the pairs of images. 376 pairs in total. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. The closest k are returned and k was set to 2. I then used a simple ratio_test that compares the two points and makes sure that one is closer to the previous frame point by a ratio greater than 0.8. Otherwise the match pair is ignored. I had to tune the ratio value to further improve my trajectory. Lastly I decided to keep track of all the feature lines between images by using cv2.drawMatchesKnn().Lastly I iterate through the matches list for each pair of frames and extract the points.

3.4 – Estimate Fundamental Matrix:

To calculate a fundamental matrix from the corresponding points in two images, I used the cv2 function cv2.findFundamentalMat(p1, p2, FM_RANSAC, ransacReprojThreshold = 1, confidence = 0.97) that returned the fundamental matrix and a corresponding mask vector. The parameter FM_RANSAC is a non-deterministic algorithm that estimates parameters of a mathematical model from a set of observed data that contains outliers. It will keep iterating through until it reaches a confidence of 0.97. Lastly, I can uses the mask vector to find the index at which good points and their matching points lie on the epipolar. These points should satisfy the equation x’Fx = 0 and his can be simply done by new_points = old_points[mask.ravel() == 1].

3.5 – Recover Essential Matrix:

The formula for the essential matrix can be found below. Both the essential and fundamental matrices can be used for establishing constraints between matching image points. However the essential matrix can only be used in relation to calibrated cameras since the inner camera parameters are recognized in order to achieve normalization. This explain the formula below. I implemented this like so:

K.T * F * K = E

essential_matrix =np.dot(np.dot(intrinsic_matrix.T, Fundamental_matrix), intrinsic_matrix)

3.6 – Reconstruct Translation and Rotation Matrix from Essential Matrix:

Cv2.recoverPose returns the relative camera rotation and translation from an estimated essential matrix and the corresponding points in two images.

points, R_est, t_est, mask_pose = cv2.recoverPose(e_mat, p1, p2, intrinsic_matrix)

e_mat refers to the essential matrix between two consecutive frames.

p1 and **p2** are the points corresponding to the matches between two consecutive frames.

The **intrinsic_matrix** is the camera matrix defined at the start.

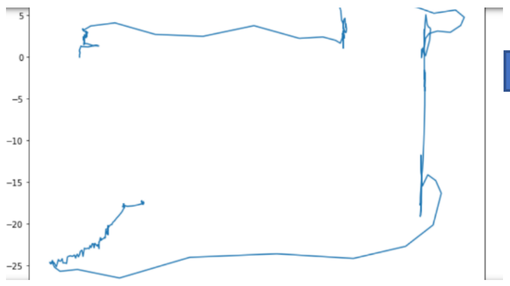
4 – Reconstruct the Trajectory:

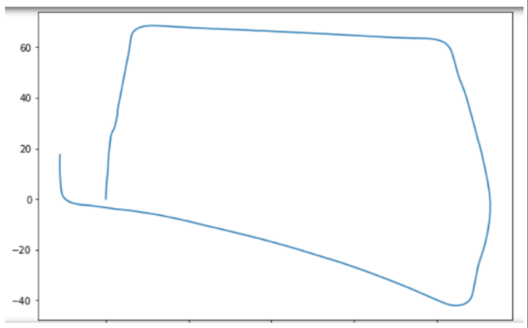
Using the notes provided by the professor I first had to create a 4x4 matrix that takes the Rotation matrix and translation vector and stacks them on top [0, 0, 0, 1]. I then applied a chain rule to calculate the new RT matrix. I then solved to find an x_0 vector that satisfies the equation below.

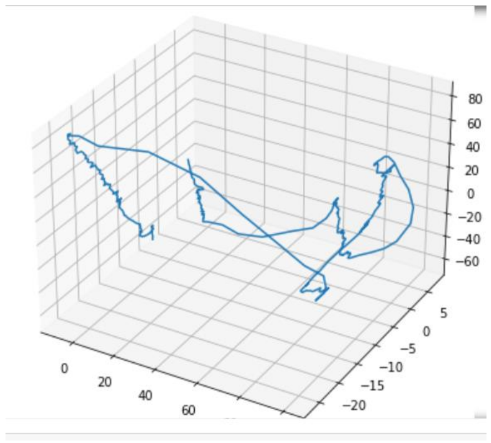
$$[0,0,0,1]^t = \begin{bmatrix}\mathbf{R}_j & \mathbf{t}_j \\ \mathbf{0} & 1\end{bmatrix}\begin{bmatrix}\mathbf{R}_{j-1} & \mathbf{t}_{j-1} \\ \mathbf{0} & 1\end{bmatrix}\dots\begin{bmatrix}\mathbf{R}_1 & \mathbf{t}_1 \\ \mathbf{0} & 1\end{bmatrix}X_0^j.$$

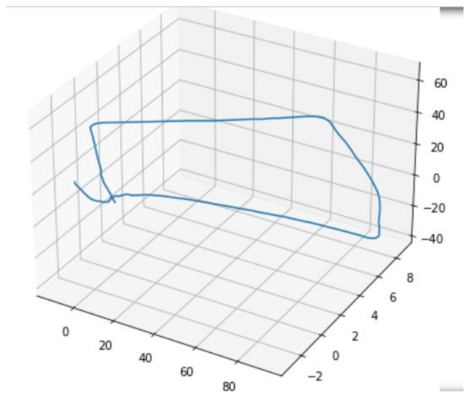
4 – Final Progression on Trajectory:

The images to the left correspond to the results before fixing the trajectory using the chain rule for the rotation and translation matrices. Results are improved are using the fundamental mask and removing ambiguous points.









References:

Brute-Force Matching with SIFT Descriptors and Ratio Test Implementation Ideas:
https://www.codestudyblog.com/sfb20b2/0305190325.html

'Some Notes on Visual Odometry' Provided by Christopher A. Meltzer

Recover Pose documentation: https://www.kite.com/python/docs/cv2.recoverPose

findFundamentalMat examples:
https://www.programcreek.com/python/example/89336/cv2.findFundamentalMat

Find the Essential Matrix using Fundamental reference:
https://en.wikipedia.org/wiki/Fundamental_matrix_(computer_vision)

https://en.wikipedia.org/wiki/Essential_matrix