Name: Jose Julian Sanchez Alarcon

Technical Report

In this project we will try to reconstruct and visualize the 3D trajectory of the camera from the given sequence of frames by performing a sequence of steps explained in this report.

3.1 Compute Intrinsic Matrix

First we computed the intrinsic matrix by calling the given function **ReadCameraModel()** with the camera model directory as a parameter. From the function we will get **fx** and **fy**(the focal lengths of the camera expressed in pixel units), also **cx** and **cy** (or the intersection of the camera's optical axis with the image plane). We need these values to calculate the **intrinsic matrix K**, by forming the matrix in this form **[[fx, 0, cx], [0, fy, cy], [0, 0, 1]].**

3.2 Load and Demosaic Images

The next step is to load and demosaic the images in the given oxford dataset. In this step, we use **cv2.imread** to read and return the pixel values of the image. Then we use the **cv2.cvtColor** function to the images by converting every image pixel values from the Bayer format with a GR (green-red) alignment to BGR to produce a full color image from the raw sensor data. It also uses **UndistortImage()** function to undistort the images using the **Look Up Table (LUT)** provided as one of the return values from the ReadCameraModel function.

3.3 Keypoint Correspondences

To detect and compute the keypoint correspondences, we opted to use the **SIFT** algorithm. Finding the keypoints and descriptors of each image is an important step since even if an image is transformed in some way, its keypoints will remain the same, potentially allowing us to identify the image or match it with others, and the descriptor contains information on the keypoint's neighborhood. The combination of keypoints and descriptors will allow us to match different images efficiently for the next steps.

3.4 Estimate Fundamental Matrix

After the previous steps, we can continue to try to estimate the fundamental matrix of the sequence of images, and then recovering the essential matrix for each pair of successive images using the keypoints found in the previous step. For this we initiate a matcher with **cv2.BFMatcher()**, that compares each descriptor in the first set with each descriptor in the second set using distance calculations and start to find matched in the sequence of images in our list, from their corresponding descriptors found in the previous step. We only try to keep the good matched that we deem good by implementing a threshold of distance, in this case I opted for making the threshold 0.75. The we will be able to recover the estimate the fundamental matrix by using the function **cv2.findFundamentalMat()** , that calculates the fundamental matrix from two sets of points in two images , from the keypoints from the good descriptors in the images. We decided to use RANSAC algorithm to estimate the parameters to compute the matrix. From this function we will get a variable F representing the fundamental matrix for the 2 current images.

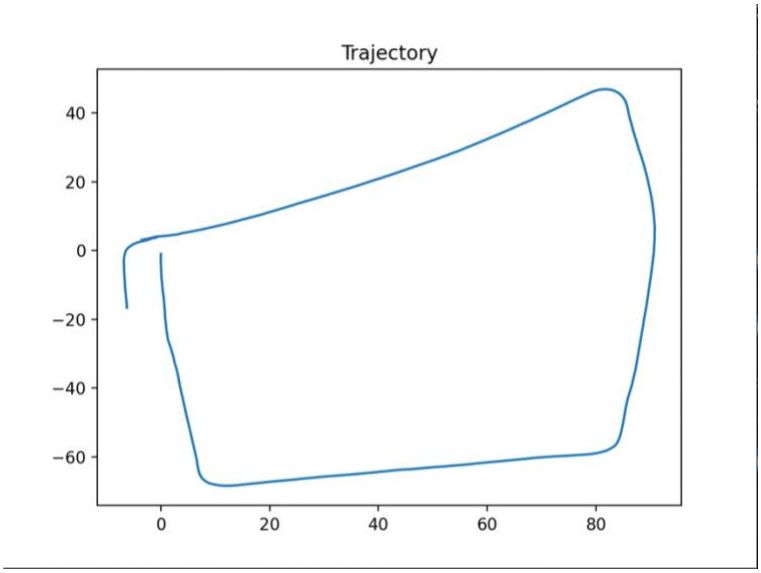
3.5 Recover the Essential Matrix

Then the essential matrix E is computed from the Fundamental Matrix F by computing the the calibration parameters in this form: **E(essential matrix) = K.T * F * K.**

3.6 Reconstruct Rotation and Translation Parameters from E

Finally, we can use the **cv2.recoverPose** function to decompose the Essential matrix into rotation R and translation T matrices. We can use these matrices to reconstruct the camera trajectory based on the translation and rotation parameters computed by the recoverPose fucntion. We then calculate the trajectory points and plot the trajectory.

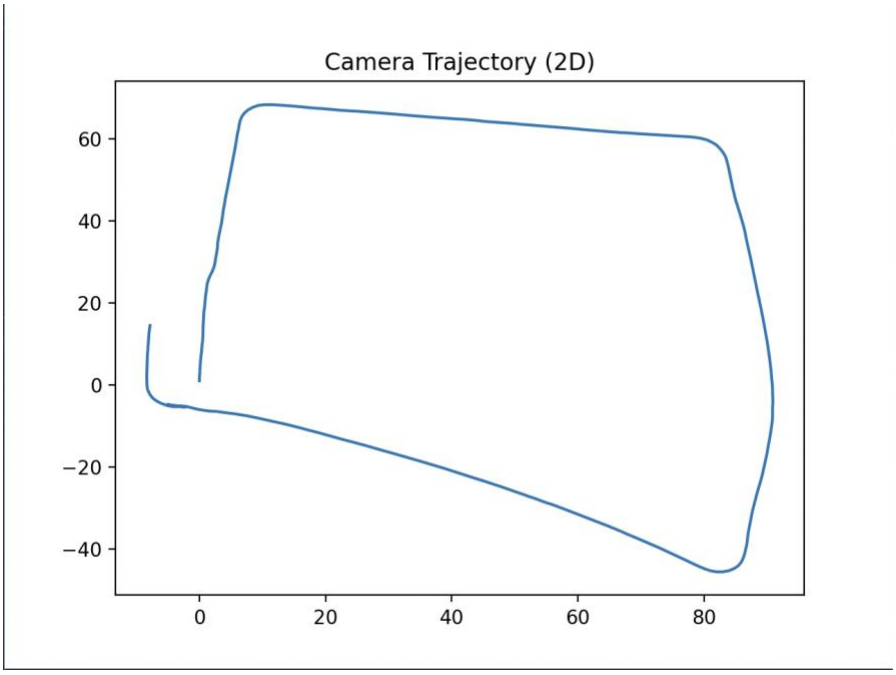
figure of trajectory reconstruction from 3.6



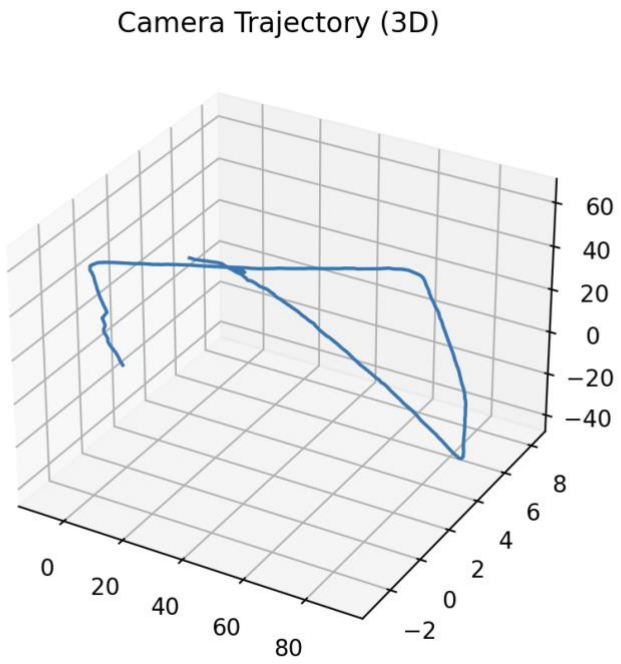
4. Reconstruct the trajectory

In his step we are trying to reconstruct the trajectory of the camera through the scene based on the estimated rotations and translations between the successive frames. We first initialize the total homogeneous transformation matrix as an identity matrix of size 4x4, this matrix represents the combined transformation (rotation and translation) of the camera from the initial position to the current position. Then we iterate over each pair of estimated rotations and translations. A 4x4 identity homogeneous matrix is initialized to start. The top left 3x3 submatrix of T is replaced by the rotation matrix for the current frame from the rotations list, and the top right 3x1 submatrix of T is replaced by the translation vector for the current frame. Then we update the total transformation matrix by pre-multiplying the inverse of the current transformation matrix. This effectively applies the inverse of the current transformation to the total transformation, essentially 'undoing' the movement from the current frame to the next. This is done because the translations and rotations obtained from cv2.recoverPose are from the perspective of the second camera frame to the first. Therefore, to get the transformation of the camera in the world coordinate system, we need to apply the inverse of these transformations. Once we obtain all our trajectory points from the loop of all rotations, we plot the reconstructed trajectory points in 2 dimensions an d 3 dimensions using pyplot.

2D reconstruction of trajectory



3D reconstruction of trajectory



6. Reconstruct Rotation and Translation Parameters Yourself

Now, to reconstruct the rotation and translation parameters, we implemented our own version of **cv2.recoverPose** function. We first created a function to decompose the essential matrix into 4 distinct combinations of rotations and translations. Using the SVD function from scipy, we **calculated U and V.T**, as well as creating the **W** matrix. Then we calculated the possible rotations using U, W and V.T and translation from U, and ensured the determinants of the rotations was positive for a valid rotation. Finally, we returned the 4 possible combinations of rotation and translations. For the main function recover_pose that received the same values as the CV2 function, we calculated the rotations and translations with the previous described function. For each points calculated using the rotations and translations, we triangulated the points in the scene 4 times, once per camera matrix, and returned the translation and rotation pair associated with the camera matrix, and using a count variable, we returned the rotation and translation pair which most matched points in front of both cameras.