Final Project Report

1 Estimating Rotations and Translations Between Frames

Computing Intrinsic Matrix

From the outputs of `ReadCameraModel('./Oxford_dataset_reduced/model')`, we obtained the horizontal and vertical focal lengths as well as the horizontal and vertical principal points. Using these, we derive the intrinsic camera matrix:

$$K = \begin{bmatrix} 964.828979 & 0 & 643.788025 \\ 0 & 964.828979 & 484.40799 \\ 0 & 0 & 1 \end{bmatrix}$$

This reveals that the camera's x and y focal lengths are approximately 964.83 pixels, and the principal point is approximately at $(x, y) = (643.79, 484.41)$ on the image sensor. Expectedly, the focal lengths are the same vertically and horizontally, which suggests that the aspect ratio is one to one.

Loading and Demosaicing Images

Before proceeding with further calculations, we first load the images using `cv2.imread` in sorted order to ensure chronological arrangement. Each image is then demosaiced using `cv2.cvtColor` with `cv2.COLOR_BayerGR2BGR` color conversion. This will interpolate the missing colors for each pixel and reconstruct the full-color image.

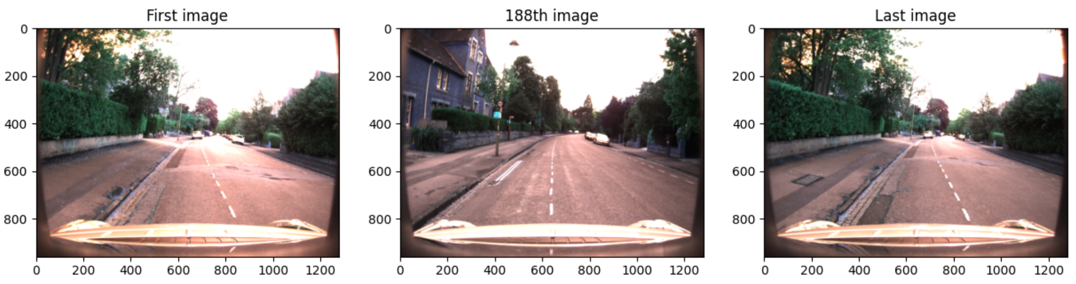

Figure 1: Demosaic images with GBRG alignment

We see in **Figure 1** that we now have demosaiced images, which will facilitate the rest of the image processing processes.

Key point Correspondences

We will use the SIFT algorithm, initialized using `SIFT_create`, to detect key points and extract descriptors from each image with `sift.detectAndCompute`. The images are converted into grayscale before applying SIFT as it reduces computational complexity and the influences of changes in lighting. SIFT is chosen for its invariance to scale and rotation, which is important for our purposes. To match key points across images, we employ the Fast Library for Approximate Nearest Neighbors (FLANN) matcher along with a KDTree index. We specify five KDTree as the algorithm used and 50 as the number of times the trees are recursively traversed. After tuning, I found that 50 checks offers a good balance between speed and accuracy in the final results. Work Cited

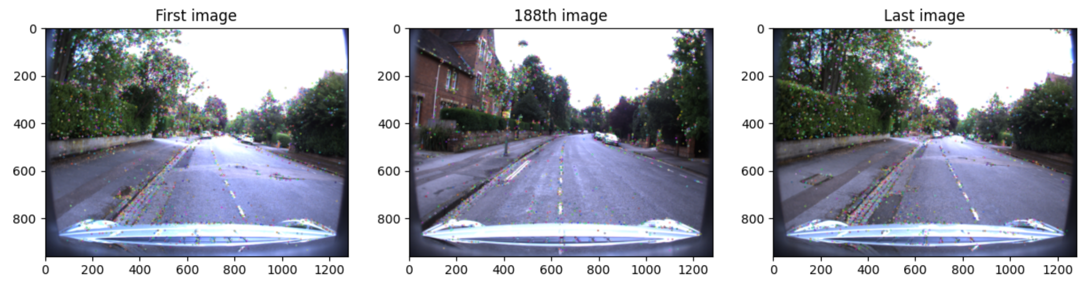

Figure 2: Images with key points

Figure 2 depicts a couple of frames with their corresponding key points. These key points represent the distinctive features in each image while the descriptors are vectors that encapsulate the local image region around each key point.

Once the key points have been identified for each sequential image pairs, they are matched using the FLANN matcher. After matching, we employ a ratio test to filter out more ambiguous matches. A match is considered good if the distance to the nearest match is significantly smaller than the distance to the second-nearest match ($< 70\%$).

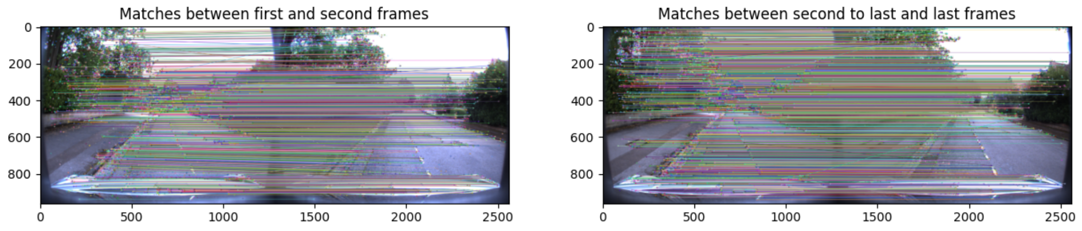
Matches between first and second frames
Matches between second to last and last frames

Figure 3: Visualization of matches between the first two frames and last two frames

We notice in **Figure 3** that the most correspondences seem reasonable as they are uniformly distributed, indicating a high degree of correspondence accuracy.

Estimating Fundamental Matrix

From the previous step, we have identified the necessary key points to estimate the fundamental matrices for each image pair. We utilize the `cv2.findFundamentalMat` function to estimate the fundamental matrix F, based on the point correspondences between the source points from one image and the destination points from the following image. We are using the RANSAC algorithm to identify the inliers. In each iteration of RANSAC, eight points are randomly sampled to estimate the fundamental matrix. As such, the estimated model with the most inliers is the optimal model. The function also conveniently returns a mask that represents the inliers that conform to the estimated model. These inliers are more likely to be accurate matches and will facilitate subsequent processes, including estimating the camera pose. We can somewhat visualize the estimated fundamental matrix through epilines. Work Cited

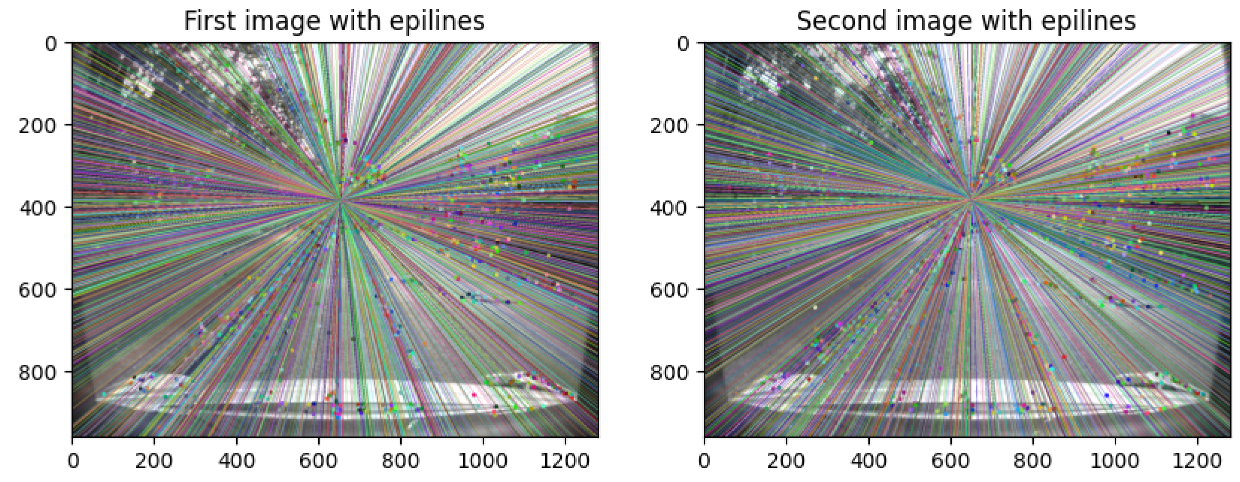

Figure 4: Visualization of epipolar lines

We see that all the epipolar lines intersect at the epipole, which seems to be very close to the center of the image. This is to be expected since the change camera position between each frame is very minimal.

Recovering Essential Matrix

The essential matrix E can be calculated from the fundamental matrix F and the intrinsic parameters K, as represented by the equation $E = K_1^T F K_2$. Here, K_2 represents the intrinsic matrix for the first camera while K_1 for the second. In this case, $K_1 = K_2 = K$ since we only have one camera. Having the Essential matrix will help use calculate the camera's pose (i.e. rotation and translation) between frames up to scale.

Reconstructing Rotation and Translation Parameters from E

We can reconstruct the rotation and translation parameters from E using `cv2.recoverPose`. The rotation and translation outputs outline the camera's movement between image pairs. This function breaks apart the essential matrix to retrieve the relative rotation R and translation t between the camera poses. It employs the camera's intrinsic parameters (i.e. fx, cx, cy) and the inlier points from both images derived in previous steps.

The rotation and translation are then adjusted to conform to the coordinate system. Specifically, the rotation matrix R is transposed, and the translation vector t is negated and rotated. This adjustment is necessary because `cv2.recoverPose` returns a transformation from the second image to the first, but we want the transformation from the first image to the second.

Lastly, we compute the cumulative rotation and translation matrices. The cumulative rotation matrix is derived by multiplying the previous rotation with the current rotation. The cumulative translation matrix is derived by the global rotating the current translation with the previous rotation and adding the previous translation. The rotation of the translation vector will adjust it to the coordinate system and the addition of the previous translation will provide the total translation from the beginning to now.

Trajectory Reconstruction

The accumulated translation matrices will allow us to see the camera's motion over time. We visualize the camera trajectory by initially reshaping the list of trajectories into a 3D array with columns corresponding to the X, Y, and Z coordinates of the camera's positions. The 2D plot of the trajectory is then obtained by plotting the X and Z coordinates of the trajectory. The scale of the axes are set equal so that the unit of distance is represented accurately.

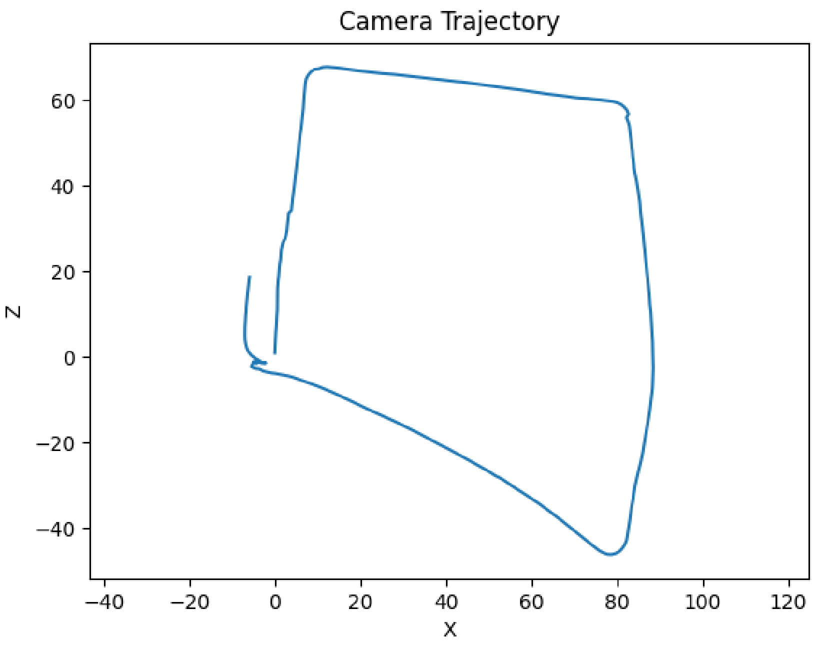

Figure 5: 2D reconstruction of the trajectory

In **Figure 5**, we see that the trajectory looks reasonable but there are still some jagged areas. This could be the result of unfiltered noise in previous steps and would benefit from further tuning.

We can apply similar steps to visualize the camera's motion within the 3D space. Notably, the Y and Z axes will be swapped in this plot so that the Y-axis points upwards. We also maintain an equal aspect ratio of the 3D plot. This is accomplished by calculating the maximum range in each direction and accordingly setting the limits for each axis. Similar to the 2D plot, this ensures that the scales on all axes are equal, thereby providing an accurate depiction of the camera's movement within the space.

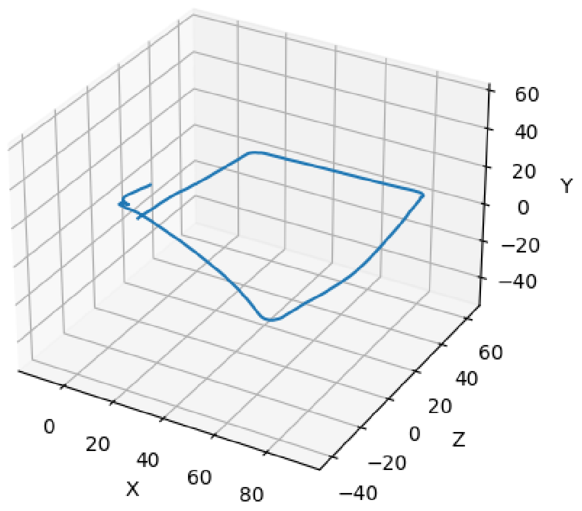

Figure 6

Again, this plot appears to accurately represent a car's trajectory through the streets, which demonstrates that the overall performance of the visual odometry system is effectively capturing the movement and positioning of the camera.