Final Report CMSC426
Zaal Belihomji

To find the trajectory for the 377 images present in the dataset, we need to do the following steps.

3.1

First we need to compute the intrinsic matrix. We do this by calling the ReadCameraModel class and passing in the model file we retrieved from the Oxford dataset. This will return `fx, fy, cx, cy,_,` and `LUT.` With these values, `fx, fy, cx, cy,` we can compute the proper K matrix which is constructed like the following:

```
K = np.array([[fx, 0, cx],
[0, fy, cy],
[0, 0, 1]])
```

3.2

This will give us our `K` matrix which we will use to find out essential matrix
After this step, what I did was I read the images from the directory path and converted the images to colored format using `cv2.COLORBayerGR2BGR` with the use of the OS library to retrieve the images based on directory and cv2 library. This OS library doesn't add the images in chronological order, so we need to use the `.sort()` method which will sort the images in order. I reversed the sorted list to ensure the trajectories matched the expected result.

After using the `UndistortImage` class, we need to extract the key points and get the matches. The code I retrieved my keypoint and matcher code was from OpenCV:
https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html

3.3

This code extracts keypoints and finds the points that are matched between two images. I have saved this code in a method to extract matches for all of the 377 images with 376 iterations. I used `cv2.SIFT_create()` which is a class used to extract keypoints. Using the class SIFT, I extracted the keypoints and descriptors for image1 and image2. I used `cv2.BFMatcher()` which is a brute force matcher that matches descriptors between the two images. A distance threshold of 0.83 was used to get the corresponding valid matches. This method I used will return the key points for both images along with an array of matches that fall under our threshold distance.

3.4

This next step involves getting the final transformation matrix (4x4 matrix) for each iteration when iterating 376 to compare two images at a time chronologically. After getting image1 and image2 data, I pass these images into my `find_matches()` method to return the key points which are used to calculate our fundamental matrix. I use `cv2.findFundamentalMaT(points1, points2, cv2.FM_RANSAC)` using key points from the first image and second image to calculate the fundamental matrix. This matrix that is returned is the matrix that corresponds to the matching points between the two images. More so, it calculates the epipolar geometry from two different images based on their viewpoints. I use `cv2.FM_RANSAC` as the method to retrieve the fundamental matrix, `F`.

3.5

Once retrieving our `F` matrix, we can get our essential matrix by doing the following operation:
`E = K.T @ F @ K`
This method involves taking the transpose of `K` and multiplying it with `F`. This result will then get multiplied by `K` to get our essentially matrix, `E`.

3.6

Now that we have our `E` matrix, we call `cv2.recoverPose(E, points1, points2, K)`.
What this function inputs is our essential matrix, our two key points, and our camera intrinsic matrix. This will return 4 values, but we need to retrieve a rotation matrix `R` which is a 3x3 and a translation matrix `T` which is a 3x1 matrix. This is critical for creating our 4x4 transformation matrix. To construct that, we do the following:

```
T_matrix = np.array([[R_new[0][0], R_new[0][1], R_new[0][2], T_new[0][0]],
[R_new[1][0], R_new[1][1], R_new[1][2], T_new[1][0]],
[R_new[2][0], R_new[2][1], R_new[2][2], T_new[2][0]],
[0, 0, 0, 1]])
```

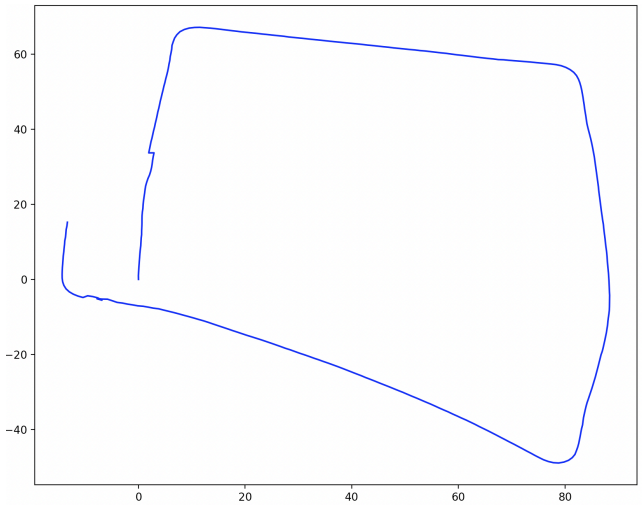
Where we take the rotation matrix in the top left of the 4x4. We concatenate the 3x1 translation next to it and we put the final row as [0, 0, 0, 1] as our homogenous row.
I added each of the T matrices for all 376 iterations in an array. With this array of T matrices, we can compute our final trajectory for 2D and 3D.

4: Reconstruction of Trajectory

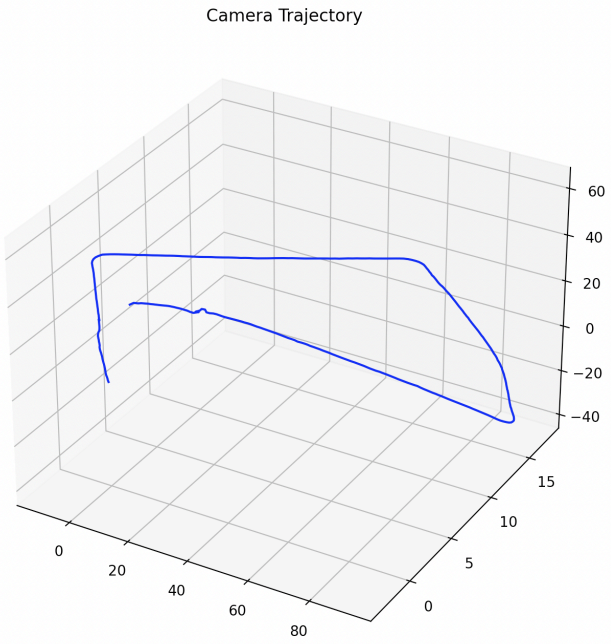
I set the camera position at 0, 0, 0 and have an initial transformation matrix which is set to a 4x4 identity matrix. What we want to do is add a new position to our camera position array to retrieve the trajectory. This can be done by concatenating the current world transformation matrix with the inverse of our current T in our loop. This will give us a camera position that you

can retrieve where we get a x, y, and z value. When plotting these values, I get the following plots:

2D:



3D:



Next page ->

Personal thoughts:

Overall, what I can say is that the use of RANSAC to get the key points as well as the use of calculating the fundamental matrix per image is very important to get the trajectory of the car through the frames. I could tell that the trajectory seemed to be very accurate based on the key points that were matched among the photos. I personally think this method is very important in computer vision for determining trajectories of anything as long as there is a sufficient amount of images that clearly shows a smooth transformation between each scene. On top of that, the use of the K matrix is extra important to utilize the camera's properties to get the most accurate trajectories based on the essential matrix that we calculated. With the use of rotation and translation that we receive from the essential matrix E and our K matrix, we can construct a proper trajectory.

Sources used:

- https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html
- https://docs.opencv.org/3.4/d7/d60/classcv_1_1SIFT.html
- https://amroamroamro.github.io/mexopencv/matlab/cv.recoverPose.html