Due: 16th May 2023 1:30 PM Total points: **100**

Visual odometry is an important concept in robotic perception where it is used to estimate the trajectory of a robot (or more precisely the trajectory of a robot's camera). Visual odometry is conceptually similar to structure from motion, but instead reconstructs motion from motion.
In this project you are given frames of a driving sequence taken by a camera in a car and a script to extract the camera's intrinsic parameters. Using this sequence of frames, you will perform a series of steps (identify shared keypoints, estimate the fundamental and essential matrices between frames, decompose the essential matrices into translation and rotation parameters, plot the camera center) to reconstruct and visualize the 3D trajectory of the camera.
The driving sequence dataset can be found in this folder. Note: It is almost 500MB and you will need to use your UMD login for access.

1 Extra Credit for Early Submissions

You will receive 1 point of extra credit for every full 24 hour period before the deadline you submit your assignment, up to 10 points.
Note: The extra credit points on the final project will only count towards the final project grade and not the overall grade. In other words, if you miss any points on the project itself, the extra credit will help you make that up.

2 Restricted Functions and Installation

So long as you follow each of the steps in the instructions (i.e., don't call a function that performs the entire pipeline for you) you may use cv2 functions to complete the main part of the assignment. In part6 you will receive extra credit for implementing some of these functions yourself. Use only numpy in the extra credit sections.
Matplotlib, os, and/or glob can be used for loading data, plotting, etc.

3 Estimate Rotations and Translations Between Frames

55 points total

Estimate the 3D motion (translation and rotation) between successive frames in the sequence by performing the following steps. You will perform these steps 376 times, starting from the first image and ending at the second to last image. Be sure to store the rotations and translations for use later.
This could take tens of minutes to run, so be sure to give yourself adequate time!

3.1 Compute Intrinsic Matrix

5 points

Extract the camera parameters using `ReadCameraModel.py` as follows:
`fx, fy, cx, cy, _, LUT = ReadCameraModel('./Oxford_dataset_reduced/model')`.
Using `fx,fy,cx,` and `cy` defined above, compute the camera's intrinsic matrix K.

3.2 Load and Demosaic Images

5 points total

The input images are in Bayer format from which you can recover the color images using the demosaic function with GBRG alignment. That is, load in the Bayer pattern encoded image `img` and convert it into a `color_image` using:
`img = cv2.imread(filename,flags=-1)`
and
`color_image = cv2.cvtColor(img, cv2.COLOR_BayerGR2BGR)`
Optionally undistort the current frame and next frame using `UndistortImage.py` as follows:
`undistorted_image = UndistortImage(color_image,LUT)`

3.3 Keypoint Correspondences

15 points

Find point correspondences between successive frames using a keypoint algorithm of your choice. You are welcome to use code from online, just be sure to cite your source.
Hint: Don't keep bad matches.

3.4 Estimate Fundamental Matrix

10 points

Using the matched keypoints you just identified, estimate the fundamental matrix between the two frames.
Hint: See `cv2.findFundamentalMat`.

3.5 Recover Essential Matrix

10 points

Estimate the Essential Matrix E from the Fundamental Matrix F by accounting for the calibration parameters.

3.6 Reconstruct Rotation and Translation Parameters from E

10 points total

Decompose E into a physically realizable translation T and rotation R. That is, among the four possible decompositions, chose the one that satisfies the depth positivity constraint.
Hint: See `cv2.recoverPose`.

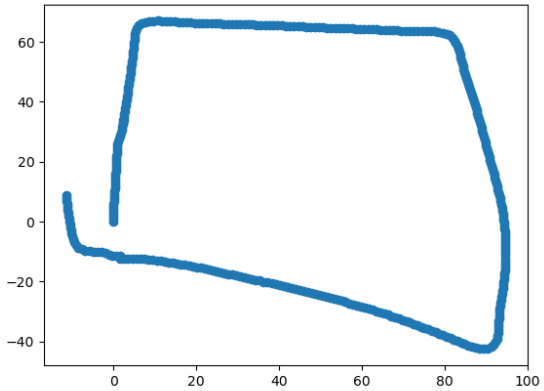

Figure 1: My reconstruction of the trajectory (projected onto 2 dimensions).

4 Reconstruct the Trajectory

20 points

In the previous section, you should have computed 376 distinct rotations R_i and translations t_i. Starting from $i = 0$, these matrices and vectors tell you how to translate and rotate a point in camera i's coordinate system to map it into camera $i + 1$'s coordinate system.
Assuming that the first video frame started at the origin, compute and plot the positions of the camera centers (for each frame) based on the rotation and translation parameters between successive frames. My reconstruction, projected onto 2 dimensions, is illustrated in Figure 1. Include a 3-D reconstruction of the trajectory as well.
Hint: If the 4×4 matrix T_{12} takes a point in camera 1's coordinate system and puts it in camera 2's coordinate system then $T_{1,2}^{-1}$ takes a point in camera 2's coordinate system and places it in camera 1's coordinate system. Similarly, if the matrix product $T_{23}T_{12}$ takes a point in camera 1's coordinate system and maps it to camera 3's coordinate system, then $(T_{23}T_{12})^{-1} = T_{12}^{-1}T_{23}^{-1}$ takes a point in camera 3's coordinate system and maps it to camera 1's coordinate system.

5 Technical Report and Discussion

25 points

Clearly and cogently document your methods and results. From your PDF report, it should be clear what you did, how/why you did it, and how well it worked, without needing to run code or sift through 300 figures. If you used a cv2 function, explain how that function works.

6 Reconstruct Rotation and Translation Parameters Yourself

20 points extra credit

Write a function to replace the `cv2.recoverPose` function in the pipeline.
Your function should first decompose the essential matrix into 4 distinct combinations of rotations and translations; each of these combinations represents a potential camera matrix from frame i+1. It should then triangulate all the points in the scene 4 times, once per camera matrix (assuming frame i's camera used the world coordinate system). You should return the translation/rotation pair associated with the camera matrix for which most matched points are in front of both cameras.

7 Useful Resources

- A detailed description of a subset of the individual steps can be found at this page.
- A video lecture on the Estimation of the Fundamental and the Essential Matrix.

Submission Instructions

Your canvas submission should consist of a zip file named **YourDirectoryID_FinalProject.zip**, for example xyz123_FinalProject.zip. The file must contain the following:

- FinalProject.py, *not .ipynb*
- report.pdf

Use relative pathing assuming that `FinalProject.py` and the `Oxford_dataset_reduced` directory have the same parent directory.

Collaboration Policy

You are encouraged to discuss ideas with your peers. However, the code should be your own and should represent your understanding of the assignment. **Code should not be shared or copied.** If you reference anyone else's code in writing your project, you must properly cite it in your code (in comments) and in your report.

Please list any individuals you collaborated with at the end of your report.

Plagiarism

Plagiarism of any form will not be tolerated. You are expected to credit all sources explicitly. If you have any doubts regarding what is and is not plagiarism, talk to me.

Credit

Thanks to Ashok Veeraraghavan, Ioannis Gkioulekas, Mahammed Charifa, Cornelia Fermuller, and Kanishka Ganguli for sharing their course resources.
The dataset used is by courtesy of Oxford's Robotics Institute.