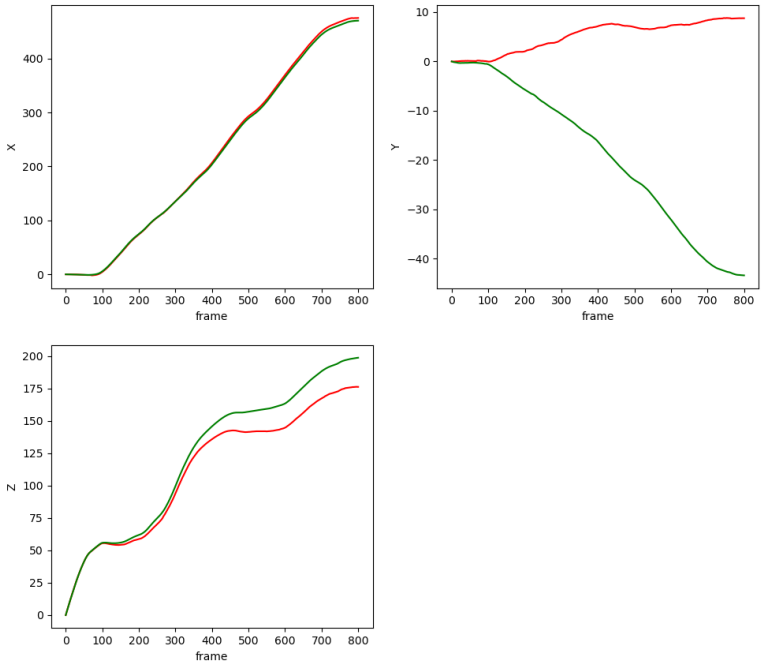
Cameron Cipriano
Challenge Final
CS585: Image and Video Computing

I. Method

To complete this challenge of implementing two-frame odometry, I conceptually followed the algorithm pipeline defined in the Visual Odometry Wikipedia page linked in the README file. The first two steps were already completed, comprising image acquisition and image pre-processing for lens distortion, enabling linear-based algorithms to function properly. In the third step, a feature detection process is required in order to find locations in an image that are distinct, relatively small in size, and reliably and repeatably found in the presence of geometric and photometric transformations, such as rotation, scale, and illumination changes between consecutive frames. Out of many possible feature descriptor choices, I chose to use the Oriented FAST and Rotated BRIEF (ORB) feature descriptor due to its efficient computation, equivalent/superior performance compared to other commonly used descriptors like the Scale-Invariant Feature Transform (SIFT) or Speeded-Up Robust Features (SURF), and ease of implementation. Once these ORB features were computed for a single frame, they can either be *tracked* from frame to frame, or *matched* given an additional set of ORB features from a consecutive frame. In my implementation, I chose to track the initial ORB features using an iterative, pyramid-based Lucas-Kanade (LK) feature tracker. This iterative, multiscale tracker was used due to its robustness for detecting apparent motion in areas of an image with high frequencies such as foliage and complex textures, as well as a faster moving camera where pixels between frames can travel farther than one pixel. Without this, the motion vectors would be seemingly random and could severely disrupt the estimation of the camera's pose. Once the ORB features detected in a previous frame are tracked in the current frame, Nister's 5-Point algorithm is applied to five or more keypoints in order to estimate the essential matrix between two image pairs, thus giving us the necessary information to extract the transformation from points in the previous frame to points in the current frame. RANSAC is applied to this process so that I could find the best possible transformation from one frame to the next, overall reducing errors. Now, the essential matrix is decomposed into a rotation matrix and translation vector, defining a new position from the previous one.

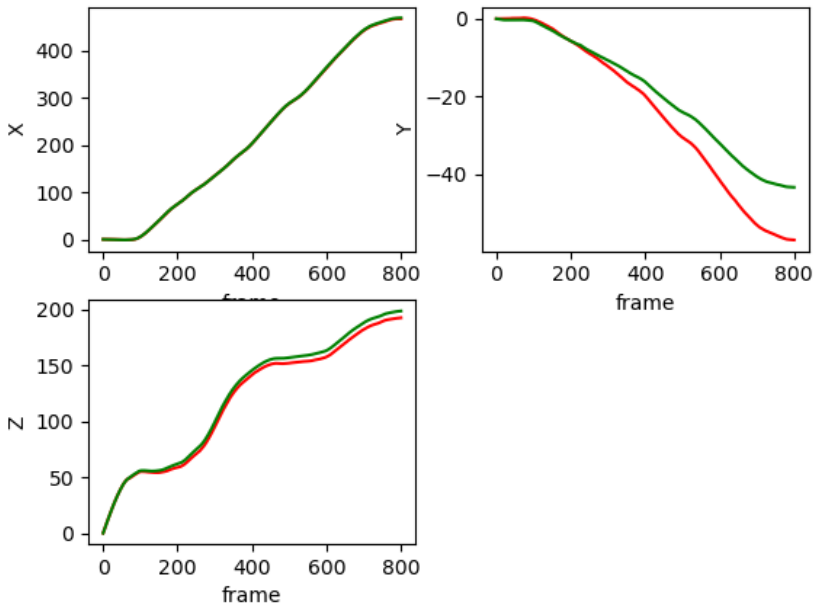
II. Changes, Tuning, and Possible Improvements

In terms of tuning this algorithm to achieve the lowest possible MSE, the main two elements that had the largest effect on the loss score was the number of initial keypoints generated, and the frequency with which ORB features were resampled. In my initial testing, I was finding the top 4,000 ORB features during a particular sampling and tracking each keypoint until the number of trackable features dwindled to 2,000 due to occlusions or traveling past the camera, out of view. This produced decent results, with a locally obtained MSE of around 24. However, I noticed that it was particularly inept at resolving the Y-coordinate, as demonstrated in the figure below, top-right plot.



(Green Line = Ground Truth, Red Line = Predicted)

I then increased the number of detected ORB features to 10,000 at the cost of increased computation time, as well as increased the resampling threshold to 3,000 tracked keypoints, resulting in roughly a 3-fold reduction in MSE. In the final iteration, I settled with 8,000 detected ORB features and a 3,000 keypoint resampling threshold, giving me an MSE of 6.8266. The paths can be seen below.



(Green Line = Ground Truth, Red Line = Predicted)

Some additional tuning that slightly affected the performance was the patchSize and edgeThreshold of the ORB features. I initially reduced these two parameters to be a 15x15 patch, in hopes of achieving greater locality of detected points, but it seemed to reduce the quality of the overall detected feature. I tried smaller positive increments from 15x15, however, I settled on the 31x31 default as it seemed to be optimized for this.

III. Changes/Improvements

I initially chose to use tracking and optical flow computation based on the algorithm pipeline, however, I did not wholeheartedly consider a matching-based approach, computing ORB features in a previous and current frame to find identical pairs. Interestingly, ORB features are known to perform well with Nearest Neighbor-type matching. As previously mentioned, the iterative, pyramid-based LK tracker is more susceptible to false matches in high-frequency image patches, where a matching-based approach may not fail given sufficient saliency and repeatability of detected points. Therefore, instead of ruling this out immediately, a change/potential improvement would be to use this approach, despite its additional computational time and complexity.