VAMR Project Report

Rupal Saxena, Shreedhar Govil, Jaswin

Introduction

Our solution consists of two parts, an initialization and a continuous pipeline. A bash script was created in order to assist code execution. Details are included in the readme file. We use OpenCV for basic tracking, feature detection, and other vision algorithms.

We obtained a working visual odometry pipeline, and have uploaded the appropriate videos. It is important to note: 1) We did not record the entire dataset for Malaga and KITTI. The resulting videos would be prohibitively large. Instead, we present the full path in the 'result' section of this report. 2) The video we upload is sped-up for compression. The actual pipeline is slower (about 1 frame per second). 3) Not all triangulated landmark can be shown in the frame of trajectory plot, as some landmarks are very far away. Instead, the number of landmarks is plotted separately.

In the videos we upload, we also present the current image frames. In it, candidate keypoints are plotted in yellow, and odometry keypoints in green.

Initialization

In the beginning of the pipeline, an initialization algorithm is selects frame[0] and frame[i], where i is a small number. Good features are selected using OpenCV function cv2.goodFeaturesToTrack. Internally, it uses Shi-Tomasi corner detection. Between the two selected images, keypoint correspondences are captured using KLT.

Then, Essential matrix is obtained from the keypoint correspondences. This is implemented using cv2.findFundamentalMat in OpenCV. The transformation matrices obtained from this essential matrix are then disambiguated, finding one correct transformation matrix from the four possible ones. The disambiguation is done by triangulating 3D landmarks from the keypoint correspondences and transformation matrices. The transformation matrix with the largest number of positive-z landmarks is chosen. We also calculate the angle of rotation (using angle-axis representation) of the two possible rotations. Any rotation corresponding to impossibly large angle (close to $\pm\pi$) is discarded.

The outputs of initialization pipeline are: keypoints and corresponding landmarks. These results are passed to the continuous pipeline.

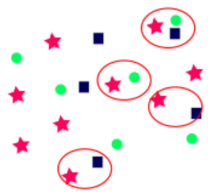
Continuous pipeline

After initialization, we obtain a number of landmarks and keypoints. At each iteration, a continuous vision algorithm pipeline selects the next images and calculates the camera pose. It consists of two parts, a localization algorithm and new keypoints addition.

Localization is done using perspective-N-point (PnP) algorithm. PnP gets rotation and orientation of the camera as a function of landmark-keypoint correspondences. Then, the result is passed through a RANSAC algorithm, in order to eliminate noise and outlier data. This is implemented using OpenCV function cv2.solvePnPRansac. The output is a transformation from the camera frame to the world frame. In order to get the camera position and orientation, this transformation is inverted.

Addition of new candidate keypoints is more complicated. At each iteration, cv2.goodFeaturesToTrack is used to detect new candidates. Then, the pixel distance between those new candidate keypoints and odometry/previous candidate keypoints are measured. New candidate keypoints close to odometry/previous candidate keypoints are discarded. This is done to prevent redundant/overlapping keypoints to be selected as candidates. The figure below describes this process:

1. New candidates chosen using OpenCV goodFeaturesToTrack



2. Discard candidates close to keypoints (green) and previous candidates (blue)

3. Keep new candidates far from other keypoints

4. Add them to the candidates

Then, KLT is used on previous candidate keypoints. If any candidate from previous iterations is not detected in KLT, it is discarded. The logic being this candidate do not have good quality. After this step, the new candidate keypoints are added to the previous candidates.

A candidate keypoint is added to odometry keypoints after it has spanned a certain amount of bearing angle. This bearing angle is called the threshold. In our implementation, threshold is chosen as 1 deg when there are fewer than 20 keypoints, and 5 deg elsewhere. This is done so that the number of odometry keypoints never drop too low. Suppose that in the first appearance of a candidate, it has pixel coordinate p1. In the current frame, it has pixel coordinate p2. The change of bearing angle is obtained from:

$$p_1 = [u_1, v_1, 1]^T \ , p_2 = [u_2, v_2, 1]^T$$

$$p_{norm\_1} = K^{-1} p_1 \ , p_{norm\_2} = K^{-1} p_2$$

$$\alpha = \angle\left(p_{norm_1}, R_{12} * p_{norm\_2}\right)$$

Incidentally, we find that the bearing-angle is a good indication for new candidate assessment. The reason is that it guards against pure rotation, because bearing angle only changes with translation. Other assessment method, such as counting the number of frames between first appearance of candidates and the current frame, is prone to pure rotation.

If a candidate keypoint is selected, its corresponding landmark is triangulated. This triangulation uses: first pixel coordinate p_1, current pixel coordinate p_2, first projection matrix M_1, and current projection matrix M_2. The result is landmarks coordinate in world coordinate.

$$norm\_p_i = K^{-1} p_i$$

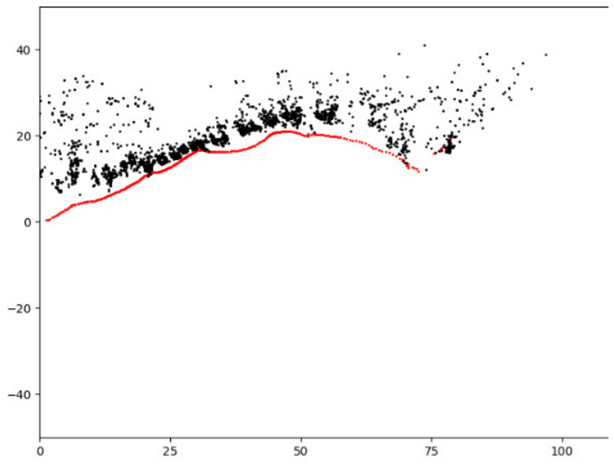
$$Landmark = triangulate(M_1, M_2, norm\_p_1 \ , norm\_p_2)$$

This landmark and its corresponding keypoint is only added to the odometry keypoints if it is projected in front of the camera ($Z_c >= 0$).

Result

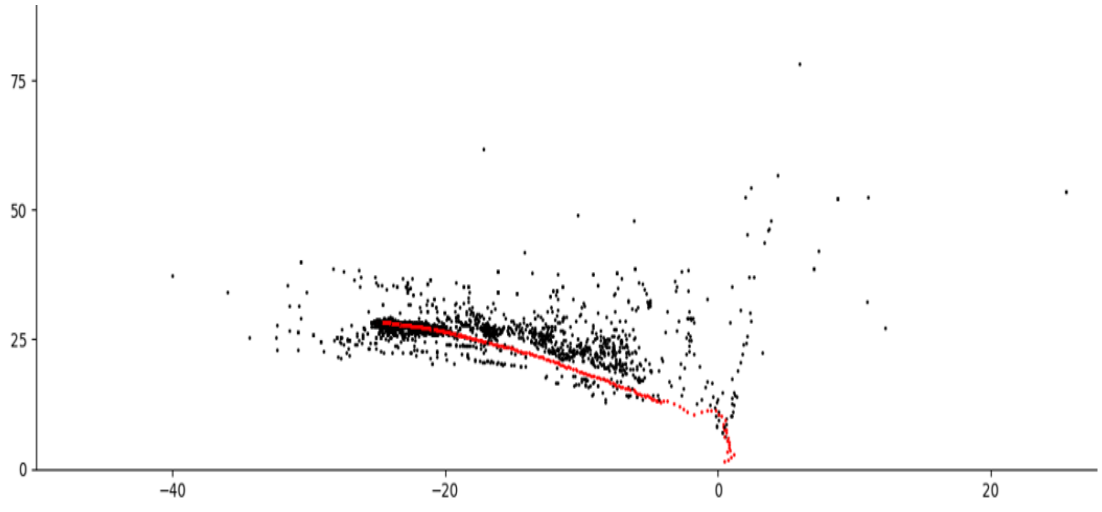
We present the result as a plot of camera trajectory (shown in red) and landmarks (shown in black).

The result of Parking dataset is given below:



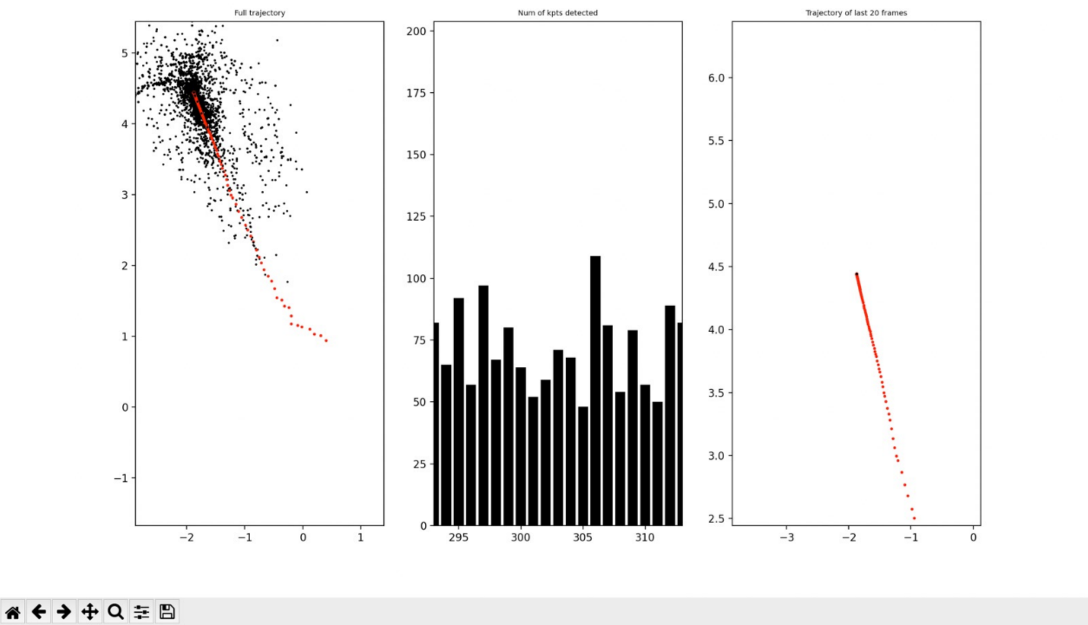
It is possible to make out the outlines of the cars in the landmarks. However, our pipeline performs poorly on this dataset. There is an error of initial pose, causing the entire trajectory to be rotated. Near the end, the scale drift is large, as can be seen in the red scatter plot becoming further apart. In the actual ground truth, the car only moves in the X direction. This is quite different from the trajectory we obtained.

The result of Malaga dataset is given below:



The Malaga dataset gives the best result. The shape of the road can clearly be seen from the landmarks. The good performance of the Malaga dataset can be attributed to the abundance of good, distinct features in the images.

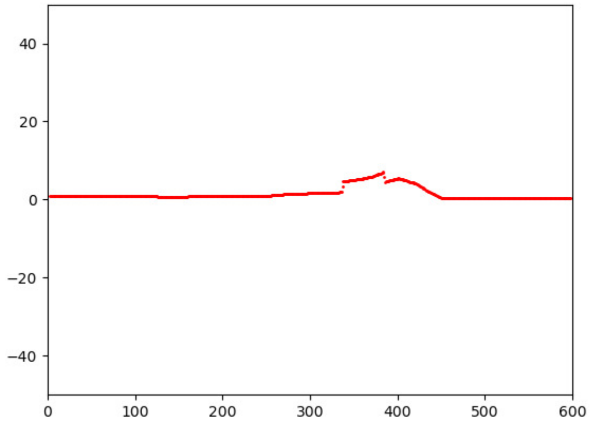
Result of KITTI dataset is shown in the next image, together with number of landmarks at each iteration (middle) and local trajectory of the last frames (right).



It can be seen from these images that the odometry pipeline works as expected.

Challenges

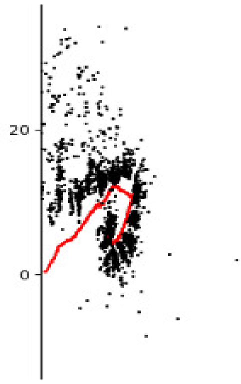
We tried to implement a scale drift correction algorithm. However, it did not work. The idea is: the height of the car and camera is constant. Therefore, the lowest landmark must correspond to the ground. In every iteration, a scale is defined as the ratio between the y value of the lowest landmarks and the lowest landmarks in the initial frames. In order to guard against outliers in the scale value, (e.g. no ground is detected in landmark) a moving average filter of 50 frames is maintained. The plot of scale as a function of frame index for the Parking dataset is shown below:



After we obtain the scale, we could not manage to correct the odometry pipeline. We tried the following method:

$$T_{now} = T_{prev} + (T_{now} - T_{prev})/scale$$

The reasoning is that this will correct the scale of triangulation, and keep it consistent. However, the algorithm did not work. This is the result for Parking dataset:



Even qualitatively, the trajectory is obviously wrong.

We also tried:

$$P_{new} = T_{cam} + (P_{new} - T_{cam})/scale$$

Here, P_new is the newly triangulated landmarks. The result is also bad, with the code terminating because it runs out of good landmarks.

Acknowledgments

We would like to thank Professor Scaramuzza, Manasi Muglikar and Nico Messikommer for providing an interesting and insightful course.