Monocular Visual Odometry

Hao-Chih Lin, Mayank Mittal, Chen-Yen Kao
hlin@ethz.ch, mittalma@ethz.ch, chen-yen.kao@uzh.ch

***Abstract*—This report provides a brief overview of a visual odometery algorithm for monocular camera. The algorithm, implemented on MATLAB, relies on Harris features with keypoint tracking between two consecutive frames and pose estimation following a Markov model. We test our code on several publicly available datasets.**

***Multimedia Link*— https://youtu.be/trbBh8Rjc4s**

I. Introduction

With the recent advances in robotics, a common problem, often referred to as the localization problem, is to estimate a robots pose. Odometry is one such method to do so. In odometry, the pose of the robot is estimated incrementally relative to the changes in its surrounding. Traditional systems used sensors like rotary encoders to find out position by measuring the angle rotated by the wheels shaft. However, this method is futile for non-wheeled locomotion systems, such as legged robots, and aerial vehicles. Additionally, rotary encoders are erroneous as wheels tend to slip and drift which cannot be measured from the encoders. This error is compounded when the vehicle operates over non-smooth surfaces.

Hence, a better way to perform odometry is to rely on visual systems like cameras, and LiDARs. Visual odometry is considered a subpart of a bigger structure from motion (sfm) problem in computer vision. Although it can be used in both outdoor and indoor environments, it is considered accurate only in feature rich scenes as opposed to a textureless environments

Our visual odometry (VO) pipeline can be divided into two parts. The initialization phase, also called bootstrapping, provides the initial set of landmarks and camera pose. The continuous operation phase, on the other hand, processes each subsequent frames to estimate pose and regularly triangulate new landmarks.

II. Building Blocks

A. Feature Detection

Features in an image are often locations which differ from the immediate surroundings. These may be corners, edges,or even blobs. For a naive corner detection method, we can apply a window around each pixel and shift it to compute Sum of Squared Difference (SSD) scores. The score would be much larger than zero at the corner points. However, doing so is computationally intensive. Instead we use Harris corner detection method [2] in our implementation which is a more efficient way to detect corners in an image.

Advisor: Prof. Dr. Davide Scaramuzza, Instructor for "Vision Algorithm for Mobile Robotics", ETH Zürich, WS 2018/2019.

In Harris corner detection method, the gradients of the image I are calculated using Sobel filters. The SSD score at a pixel (x, y) is computed by:

$$SSD(\Delta x, \Delta y) = \sum_{x,y \in P} (I(x, y) - I(x + \Delta x, y + \Delta y))^2$$

Using Taylor expansion, we can approximate the above equation as:

$$\begin{aligned} SSD(\Delta x, \Delta y) &\approx \sum_{x,y \in P} (I_x(x, y)\Delta x + I_y(x, y)\Delta y)^2 \\ &\approx \begin{bmatrix} \Delta x & \Delta y \end{bmatrix} M \begin{bmatrix} \Delta x \\ \Delta y \end{bmatrix} \end{aligned}$$

where $M = \begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix}$.

The Harris scores for each pixel is then calculated as $R = det(M) - \kappa \text{ trace}(M)$, where κ is a parameter that needs to be tuned. Using non-maximum suppression, we pick only the strongest features in each block and use them as keypoints in our pipeline.

B. Feature Tracking

Lucas-Kanade Tracking (KLT) [3] is similar to template tracking and is used to find correspondences between keypoints in two input frames. In this, a box-sized window is created around each keypoint in the first and second frames to compute a descriptor. The descriptor for the keypoint in the first frame is warped using the warp function $W(x, p)$ where x and p are the pixel points and the warp parameters. Then, the SSD score is calculated between the warped descriptor and the descriptor of second like the following formula

$$E = \sum_{x \in T} [I(W(x, p)) - T(x))]^2$$

where E is the SSD score, T(x) is the template image and I(W(x,p)) is the image after warped

In order to minimize the error we know that the minimization form can be written in the following form

$$E = \sum_{x \in T} [I(W(x, p)) + \nabla I \frac{\partial W}{\partial \boldsymbol{p}} \Delta p - T(x)]^2$$

To solve the minimization form, we solve for the equation $\partial E / \partial \Delta \boldsymbol{p} = 0$ we can get the close form solution

$$\Delta \boldsymbol{p} = H^{-1} \sum_{x \in T} [\nabla I \frac{\partial W}{\partial \boldsymbol{p}}]^T [T(x) - I(W(x, p))]$$

where

$$H = \sum_{x \in T} [\nabla I \frac{\partial W}{\partial \boldsymbol{p}}]^T [\nabla I \frac{\partial W}{\partial \boldsymbol{p}}]$$

after we compute the Δp we then update it into $p \leftarrow p + \Delta p$ and repeat it until the $\Delta p < \epsilon$. In this VO pipeline, the Matlab built-in function, named $vision.PointTracker$ was chosen for the KLT implementations.

C. Feature Matching

Feature matching comprises of efficiently searching for likely matching candidates in two images. For feature matching on Harris keypoints, descriptors are created based on the patch intensities around the keypoints in both the images. Sum of Squared Distances (SSD) scores are computed between each *query* keypoint in one image with all the *database* keypoints in the other image. The keypoint matched to the *query* keypoint is the one corresponding to the minimum SSD score.

D. Perspective-3-Point (P3P)

For given 2D keypoint $[u, v, 1]^T$ in image frame and corresponding 3D landmark $[X, Y, Z, 1]^T$ in world frame, the relation between these two vector is as follows:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K[R|T] \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

where s is a scale factor for the image point, K is camera intrinsic matrix, which is given. The goal of $P3P$ is to compute a camera extrinsic matrix $[R|T]$ which satisfies the above formulation. Based on Carnots theorem [1], using three non-collinear 2D keypoints, the $P3P$ will output four valid solutions. The additional fourth keypoint can disambiguate the solutions. In our VO pipeline, the $P3P$ algorithm is adopting the open-source implementation provided in [1]. Generally, the usage of $P3P$ will be combined with RANSAC to filter out the outliers in the set of point correspondences.

E. Essential Matrix Formulation

For given keypoints in both frames, we have the pixel point[u,v]. For the frame 1 we have the formula

$$\lambda_1 \begin{bmatrix} u_1^i \\ v_1^i \\ 1 \end{bmatrix} = \lambda_1 p_1^i = K_1[I|0] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix}$$

For the second frame we have another formula

$$\lambda_2 \begin{bmatrix} u_2^i \\ v_2^i \\ 1 \end{bmatrix} = \lambda_1 p_2^i = K_2[R|T] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix}$$

where K_1 and K_2 are the intrinsic parameter

By using the epipolar geometry we can derive the epipolar constraint equation

$$p_2^T E p_1 = 0$$

where $E = [T]_x R$ is the essential matrix. [4]
For uncalibrated cameras we can derive the fundamental matrix F

$$p_2^T K_2^{-T} E K_1^{-1} p_1 = 0$$

where $F = K_2^{-T}[T]_x R K_1^{-1} p_1$ In order to reduce the error we can use the normalized points instead.

$$\hat{p}_i = \begin{bmatrix} \frac{\sqrt{2}}{\sigma} & 0 & -\frac{\sqrt{2}}{\sigma}(\mu_x, \mu_y) \\ 0 & \frac{\sqrt{2}}{\sigma} & -\frac{\sqrt{2}}{\sigma}\mu_y \\ 0 & 0 & 1 \end{bmatrix} p_i$$

where μ = mean of the points and σ is the mean standard deviation.
Now we have the equation

$$\hat{p}_2^T F \hat{p}_1 = 0$$

To solve the essential matrix F we can use the SVD method.

$$F = U \Sigma V^T$$

and to make sure that the determine on the essential matrix [4] to be zero we let the last element in the matrix Σ be zero, then we can recover the essential matrix E, then we decompose the essential matrix by using SVD and get the R and T

$$E = U \Sigma V^T$$

where

$$\hat{T} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \Sigma V^T$$

$$\hat{R} = U \begin{bmatrix} 0 & \mp 1 & 0 \\ \pm 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T$$

and the final R and T are

$$T = K_2 \hat{T}$$

$$R = K_2 \hat{R} K_1^{-1}$$

there are four different R and T that are possible, and we pick the one with all the points or most of the points are in front of both cameras.

F. RANdom SAmple Consensus (RANSAC)

RANSAC [5] algorithm stands for Random Sample Consensus algorithm which can help us remove outliers in a very efficient way. First of all, we estimate the fraction of inliers in the dataset(w) and also set the probability of success(p). Since we are using the 8-point algorithm applied to Structure from Motion, we extract 8 points every time. With fraction of inliers(w), probability of success(p), and 8 points, we have the number of iteration that we have to run.

$$k = \frac{log(1-p)}{log(1-w^s)}$$

During each iteration and for the eight points we extracted, we compute the essential matrix and find out the R and T. With our guess of R and T we can do the triangulation of all the points and reproject them and compute the reprojection error,

and if the error is smaller than a threshold we consider it as a inlier, after run through all the iterations we can select the best R and T with the most inliers. We then takes these inliers to do the further step to compute the triangulation.

G. Triangulation

In the linear triangulation [6] we use the keypoints and the R and T to compute the landmark, but here we have to consider the baseline, if the baseline is too small there are large depth error, but if the baseline is too large, the minimum measurable distance increase and it is difficult to search problem for close objects. Here we use 2 keyframes as our baseline. For the first frame we have the equation

$$\lambda_1 \begin{bmatrix} u_1^i \\ v_1^i \\ 1 \end{bmatrix} = K_1[I|0] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix} = \lambda_1 p_1 = M_1 P$$

For the second frame we have another equation

$$\lambda_2 \begin{bmatrix} u_2^i \\ v_2^i \\ 1 \end{bmatrix} = K_2[R|T] \begin{bmatrix} X_w^i \\ Y_w^i \\ Z_w^i \\ 1 \end{bmatrix} = \lambda_2 p_2 = M_2 P$$

we then can derive these two equation into the form

$$p_1 \times M_1 \cdot P = [p_1]_x \cdot P = 0$$
$$p_2 \times M_2 \cdot P = [p_2]_x \cdot P = 0$$

we than can solve for the P by Matlab built-in SVD function, and take the last column of V to get the result.

III. Visual Odemetry Pipleline

A. Bootstrapping

In the bootstrapping part, we have the following works to do. First of all, we find the keypoints in our input frames, and here we adopt the Harris corner detection method. Subsequently, we track the keypoints that we extracted from the first frames to the keypoints we extracted from the second frame by Lucas-Kanade tracking method. With the tracked keypoints, we can save the corresponding ones and remove the irrelevant ones. With the remaining tracked keypoints we can find out the corresponding essential matrix and compute the rotation matrix (R) and the translation (T). However, we are not sure about that all the remain keypoints are inliers, so we take the RANSAC approach to find the best R and T that contains the most inliers. With all the inliers in hand, we compute the transformation matrix and use these parameters and the keypoints to triangulate our 3D landmarks.

B. Continuous Operation

In this part, the data flow of continuous processing follows the assumptions of Markov property, which means that for each time step(i), the updating state S^i and camera pose T_{WC}^i only depends on previous frame I^{i-1}, previous state S^{i-1} and current frame I^i as follows:

$$[S^i, T_{WC}^i] = processFrame(I^i, I^{i-1}, S^{i-1})$$

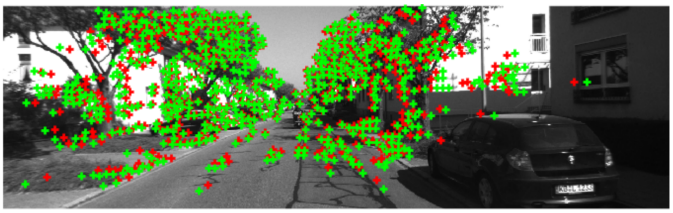

Fig. 1. Keypoints in the current frame (KITTI dataset). The green and red points are tracked keypoints pair in the current and previous frames.

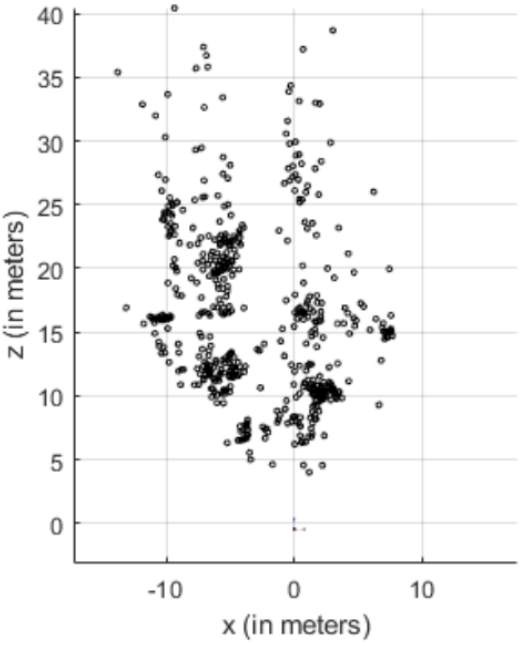

Fig. 2. Initialized landmarks and pose after bootstrapping (KITTI dataset)

The state S in this phase is defined as S^i = (P^i, X^i), where P^i is denoted as keypoints in i-th frame $\{p_k^i\}$, $k \in [1, K]$, K being the keypoints count. The 3D landmarks are denoted with $X^i = \{x(p) \forall p \in P^i\}$. The state S will be augmented in the next $Triangulating$ phase.

To estimate the camera pose T_{WC}^i, the $processFrame()$ function could be divided into three procedures:

1) Associating keypoints to existing landmarks: Given keypoints $S.P^{i-1}$ at previous time with previous frame I^{i-1}, the corresponding keypoints $S.P^i$ at current time can be tracked by KLT, mentioned in $II.B$, as feeding the current frame I^i. After applying KLT tracker, the tracked corresponded 3D landmarks will be saved into $S.X^i$. The column size of $S.X^i$ should be the same as $S.P^i$. With these keypoints-to-landmarks associations, the camera pose T_{WC}^i can be estimated by $P3P$ in the next procedure.

2) Estimating the current pose: With keypoints $S.P^i$, associated 3D landmarks $S.X^i$ and camera intrinsic matrix K, the camera pose from camera frame to world frame T_{CW}^i could be computed by $P3P$ algorithms mentioned in $II.D$. Because of KLT tracking errors, such as image blur, moving objects in the scene and also the inherited accumulated position error of $S.X^i$, the $P3P$ needs to be combined with RANSAC to filter

out the outliers. The camera pose from world frame to camera frame T^i_{WC} could be obtained by applying the inverse of T^i_{CW} as follows:

$$T^i_{WC} = (T^i_{CW})^{-1} = [R^T_{CW} | - R^T_{CW} T_{CW}]$$

3) Refining the pose: Theoretically, the outputs of $P3P$ is enough for continuous operation, however, the $P3P$ algorithm only takes three pairs of keypoints-to-landmarks associations among inliers set after RANSAC step for pose estimation. The accuracy of estimated pose will be directly influenced by inherited errors of randomly selected three pairs of associations. Therefore, to obtain more robust and better accurate of pose estimates, all association pairs among inliers set should be used for refine the estimated pose. After some practical experiments, instead of using DLT algorithm, a nonlinear least square method with minimizing the reprojection error between $S.P^i$ and $S.X^i$ is adopt to refine the pose estimated by $P3P$.

Before applying the refinement, the RANSAC outliers of both $S.P^i$ and $S.X^i$ have to be removed from the array. Then, computing the reprojected keypoints from $S.X^i$ with camera intrinsic matrix K and T^i_{CW} from $P3P$ outputs. The refined pose ${T^i_{CW}}'$ would be found by taking the argument of minimizing the sum of errors between original keypoints and reporojected keypoints with respect to given $P3P$ output pose T^i_{CW} as follows:

$${T^i_{CW}}' = \arg\min_{{T^i_{CW}}'} \sum_k [p^i_k - (K \cdot {T^i_{CW}}' \cdot x^i_k)]^2$$

where p^i_k is k-th of $S.P^i$, x^i_k is k-th of $S.X^i$ and $k \in [1, K]$, K being the keypoints count. Without the refinement, the accumulated error caused by inaccurate camera pose estimates and positions of new adding landmarks will diverge so fast that the number of inlier inside $P3P$ with RANSAC will be lower than given threshold, the continuous operation will be stuck.

C. Triangulating new landmarks

The initial set of landmarks obtained form bootstrapping may not be visible to the camera after the camera has moved far enough. Thus, new landmarks need to be triangulated at each frame for pose estimation through continuous operation. In order to preserve the Markovian property of our pipeline, we follow the procedure as described below:

1) Adding new landmarks: Using the candidate points from the previous state $S^{t-1}.C$, we apply KLT tracking to find matches in the current frame. We set the matched keypoints pair from $S^{t-1}.C$ and the current frame into $S^t.C$ and $S^t.F$ respectively. Similar to what is done bootstrapping, we perform triangulation on these matched keypoints and check that the candidate landmark should lie in-front of the current camera frame. We then calculate the bearing angle between landmark and the two frames in which it has been observed. If the angle is above a threshold, we add the candidate keypoint and its corresponding landmark to the existing lists $S^t.P$ and $S^t.X$ respectively, and remove the associated entries from $S^t.C$, $S^t.F$ and $S^t.T$.

2) Adding new candidate keypoints: As candidate keypoints may decrease with time, we need to replenish their count as well. To do so, we detect new Harris corners in the current frame and add only those keypoints which are not matched with already existing keypoints in $S^t.C$ and $S^t.P$. To further prevent redundancy, we check that the new keypoint should not be within a particular radius to the keypoints in $S^t.C$ and $S^t.P$. The filtered keypoints are then appended to $S^t.C$ and the current camera pose is added $S^t.T$ marking the camera pose when the feature points are first observed.

3) Bootstrapping after regular intervals: To counter the scale drift issue in monocular odometry, we perform bootstrapping again after regular intervals and discard the previous state. The landmarks that are obtained from the bootstrapping stage are then transformed back into the global frame using the camera pose. This prevents the re-initialization of the global frame. In our experiments, we found that this improved our result a lot, and thus decided to keep it as a part of our implementation.

IV. User Interface

The User Interface as shown in Figure 3 plots the status of our Markovian state. We show the tracked keypoints in the frame $S^t.P$ in green and the candidate keypoints $S^t.C$ in red. Besides that, we plot the number of landmarks observed in the last 20 frames to keep track that sufficient number of landmarks are being added at each instant. Along with that we plot the entire global trajectory and a local trajectory of the latest 20 frames.

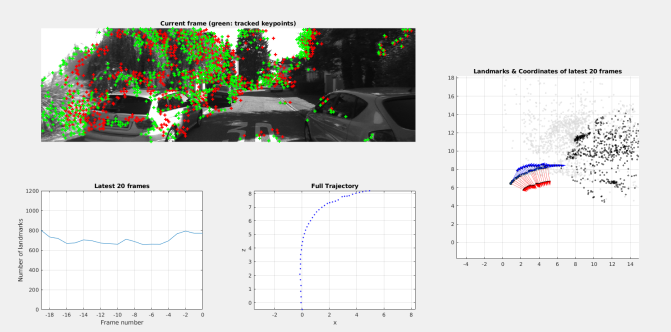

Fig. 3. MATLAB Figure showing the trajectory on KITTI Sequence 00

V. Results

In all the three datasets, we carefully tuned our parameters. The odometry estimated using our VO pipleline for the three publicly available datasets– KITTI Dataset, Malaga Urban Dataset, and Parking Dataset– are shown in Figures 4, 5 and 6 respectively.

VI. Remarks

The first version of our VO pipeline only implemented the $P3P$-RANSAC without pose refinements (the method mentioned in $III.B.3$), the performance was highly unstable and it was hard to reproduce the similar result over several

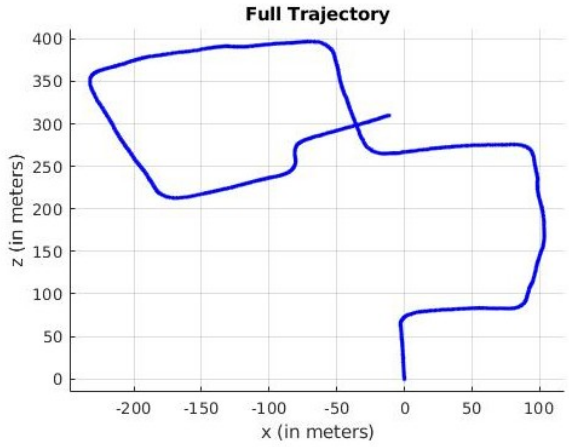

Fig. 4. Estimated Odometry in KITTI Sequence 00

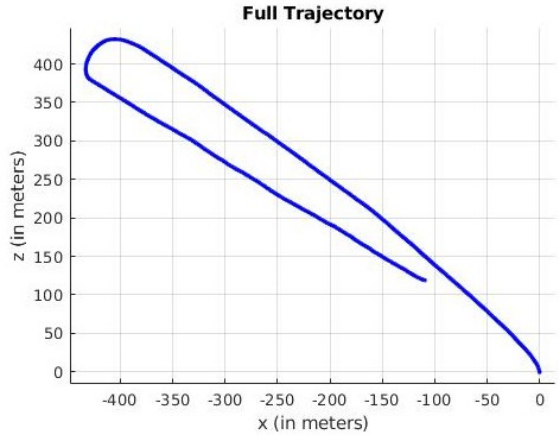

Fig. 5. Estimated Odometry in Malaga Urban Sequence 07

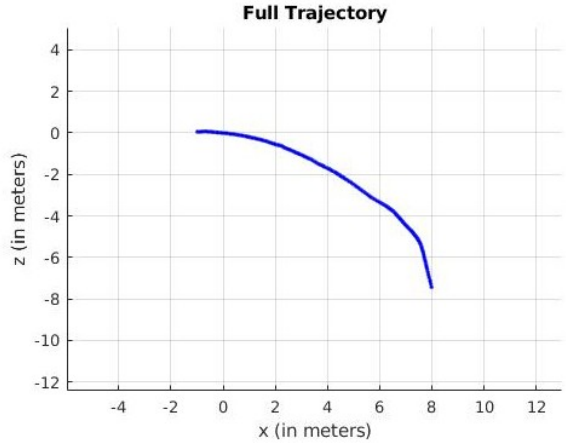

Fig. 6. Estimated Odometry in Parking Dataset

testings even with the same parameters. Sometimes the outputs trajectory could successfully pass the first corner, but most of time it can not draw a simple straight line as the camera moving forward. To tackle this issue, we have tested dozens of parameters sets, rewritten or exchanged the implementations of each sub-functions, but still could not improve the performance. In order to address the exact reasons, we tried to reviewed all data inside state cell S^i at each time step i in paused debugging mode at run time. Finally, we figured out that camera pose T^i_{WC} estimated by $P3P$ was not stable, in other words, under same image dataset with same parameters, the pose output at specific frame i at testing(k) could somehow be significant different to testing($k+1$). This phenomenon make us to notice the main issue of $P3P$-RANSAC is to randomly select three pairs of keypoint-to-landmarks association from the inlier set. The error of pose estimate did not be minimized at each continuous processing step, in addition, as stated in $III.B$, our VO pipeline follows the assumptions of Markov property, which means that the error of pose estimation will be accumulated continuously. These accumulated error will also influence the accuracy of position for the new adding landmarks, which will be taken as inputs of $P3P$ in the next iteration. In the end, when $P3P$-RANSAC can not find enough inlier properly under given thresholds, the pose outputs will stuck.

There are several potential methods to solve this problem, one way is to refine the estimated pose by all associations pairs in inlier set, like the method presented in $III.B.3$ to minimize reprojected errors. After applying this mechanism, our VO system is able to reproduce whole trajectory from image dataset (Figure 4). However, the refinements described in this paper only minimize the pose in consecutive frames, means that the error is still slowly accumulating over frames. This limitations cause the performance of our VO pipeline is quite stable in short period of time and independence of parameters tuning, but quite sensitive for long time operation which needs parameters fine tuning depending on specified image dataset, such as how many of new keypoints will be detected in the new coming frame, what is the minimization or maximization thresholds for new adding landmarks (e.g. in forward motion, the max number should be low. But in turning motion, it should be high, otherwise, most of new landmarks will be discarded and there will not be enough landmarks for $P3P$ in the next iteration.), etc. To overcome this challenge, the typical solutions are adding global Pose Graph Optimization or sliding window based Bundle Adjustment (BA), these algorithms could be one side of our future works.

REFERENCES

[1] Kneip, Laurent, Davide Scaramuzza, and Roland Siegwart. "A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation.", 2011
[2] Davide Scaramuzza "Lecture Note 05 Point Feature Detection and Matching" pp. 37–64, 18.10.2018.
[3] Davide Scaramuzza "Lecture Note 11 Tracking" pp. 49–69, 28.11.2018.
[4] Davide Scaramuzza "Lecture Note 08 Multiple View Geometry 2" pp. 16–39, 09.11.2018.
[5] Davide Scaramuzza "Lecture Note 08 Multiple View Geometry 2" pp. 40–71, 09.11.2018.
[6] Davide Scaramuzza "Lecture Note 07 Multiple View Geometry 1" pp. 18–30, 07.11.2018.