A Rapid Optimization Method For Visual Indirect SLAM Using a Subset of Feature Points

Ryosuke Kazami
Keio University
kazami@am.ics.keio.ac.jp

Hideharu Amano
Keio University
hunga@am.ics.keio.ac.jp

***Abstract*—**

In this paper, we propose a novel optimization method for indirect Simultaneous Localization And Mapping(SLAM). This method avoids error-intended map points using a random subset of extracted feature points. As a result, this method making the calculation time short at the cost of some accuracy.

The published code of ORB-SLAM2 has longer calculation time compared to other systems such as Semi-direct odometry(SVO). This is due to feature extraction time of an indirect method, which makes the indirect methods robust. However, considering CPU performance on edge devices like MAVs, a lightweight system is welcome.

ORB-SLAM2 is widely known as a robust and rapid visual Simultaneous-Localization-And-Mapping (vSLAM) method than before. It extracts FAST feature points from a given image and optimizes using all of these points for camera-posing estimation. However, some points disturb accurate estimation causing large reprojection error after optimization. Although these points are eliminated as outliers after optimization, they may be an obstacle for accurate pose estimation.

To make this system more robust and more rapid, We suggest an improved method to estimate camera-posing, where we use a subset of extracted points for estimation, avoiding mixing large-reprojection-error points in bundle adjustment.

The experimental results on the SLAM benchmark dataset KITTI odometry demonstrated that the proposed method outperformed the original implementation of ORB-SLAM2 from the viewpoint of calculation time.

***Index Terms*—SLAM, point cloud, bundle adjustment**

I. Introduction

Simultaneous Localization And Mapping (SLAM) technology is useful for Micro Aerial Vehicles (MAVs) and autonomous robots. In particular, in MAVs, it is highly necessary to operate the SLAM system on an edge device that does not have high computational power, because powerful computers are heavy and cannot be loaded onto MAVs.

At the same time, however, tracking accuracy is also so important that if accuracy decreases an autonomous robot easily loses its location.

There are two categories of visual SLAM algorithms. One is feature-based one whose merit is its speed and the other is direct algorithms whose strong point is robustness.

In this paper, we change one of the algorithms inside ORB-SLAM2 [1], which is well known as feature-based one. This also has difficulty in operating in edge devices.

In contrast, direct algorithm systems, such as SVO [2], are more rapid, but do not keep features so that it is difficult to operate loop closing. Hence in practical scenes, systems have less robustness.

In ORB-SLAM2 [1], calculating a large number of features, about 2000 points per frame, enables robust matching of points, thus achieving high accuracy. On the other hand, performing this feature extraction procedure requires a lot of calculation time for each frame, compared to direct methods. The problem is that the operating speed is slow [2]. As a result, when placed on edge devices, you will suffer from a trade-off between frames per second and accuracy.

By using multi-thread and robust ORB [3] feature, it presumes the location of the camera with the following steps.

1) Extract the features in an image frame and compare with them in the previous image frame.
2) If the point has the same feature, it can be assumed to move that position. Apply the non-linear optimization called the re-projection error optimization for 300-500 such moving traces, the position of the camera is presumed.

In the existing system ORB-SLAM2, all feature points in the current frame are used for building a pose graph for nonlinear optimization.

However, as a result of our experiments, we can say that there is no necessity of including all the points. Here, we propose a method to reduce the calculation time by reducing the number of points, without dropping tracking accuracy significantly.

Our experiment shows that even the Root Mean Squared Error (RMSE) improves to 1.24m when the original implementation was 1.28m. We succeeded in reducing the time taken for PoseOptimization every frame from 5.3msec to 2.8msec.

The rest of the paper is organized as follows:

2) How ORB-SLAM2 works
3) Implementation
4) Experimental Result
5) Conclusion

II. ORB-SLAM2 System Overview

The system overview of ORB-SLAM2 is shown in Fig. 1. ORB-SLAM2 is composed of three parallel threads.

1) Tracking thread; this performs feature extraction of the current frame, stereo matching, and camera pose estimation by optimizing reprojection error for all the incoming frames.

978-1-7281-5268-4/19/$31.00 ©2019 IEEE
DOI 10.1109/CANDARW.2019.00055

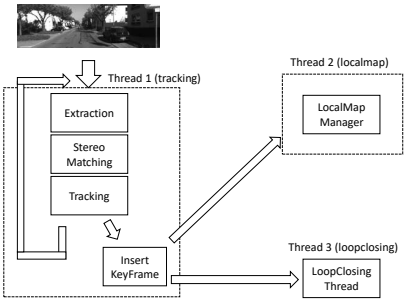


Fig. 1. The system overview of ORB-SLAM2.

2) LocalMapping thread; this manages local maps. Local maps have several keyframes and points near the current location. This runs bundle adjustment for camera poses and maps points location.
3) LoopClosing thread; this detects a large loop when revisiting the already seen scene by inquiring the scene database. This removes piled drift error with the bundle adjustment.

Thread 2 creates a map point after confirming a new keyframe (one in 2-3 frames). A map point location is calculated with the current camera pose and the depth of an extracted feature point. If the map point produces large reprojection error in pose graph optimization or not used, this will be removed from the map.

Thread 3 gets a feature pattern of the current frame, and stores it to the database. This thread always searches the database, and if it finds the same scene in the database, it detects a large loop and performs loop closing. DBoW2 [4] helps to get and search for a feature pattern.

Thread 1 first performs stereo matching between left and right images. By matching, the depth of each point is estimated. Allowing a certain vertical shift, the stereo matcher finds similar points around the left one. Then this thread compares the previous features with the current features, matches and optimizes so that the reprojection error becomes small. In this optimization procedure, the initial value of pose transition is as the transition from those two frames before to the previous frame. From here, we can perform an iteration using Jacobians to approximately solve the nonlinear equation like gradient descent. This algorithm is realized with the Gauss-Newton method or improved version called the Levenberg method which uses a damping factor. This procedure is called PoseOptimization. As a result, it can estimate the camera pose transition of 6 dimensions.

The method proposed in this paper is to speed up this function, PoseOptimization. It is performed for such camera pose estimation and also used to optimize the local map near the current location.

The algorithm of PoseOptimization is as Algorithm. 1.

In this paper, we do not intrude on Levenberg method.

Data: The pairs of matched points between the current 2 frames
Result: Improved camera pose

```
CurrentFrame.pose = Velocity *
LastFrame.pose;
for 10 times do
    for i := [0..N − 1] do
        jacobian :=
         points[$i].calculateJacobian;
        CurrentFrame.pose.aggregate(jacobian)
    end
    CurrentFrame.pose.apply();
end
```

Algorithm 1: PoseOptimization

Instead, we changed the number of points used to build the input to the optimization method to improve the speed of the total SLAM system.

III. Proposed method and implementation

This section describes the proposed method and its specific implementation.

First, we reconfirm the procedures performed in ORB-SLAM2 [1].

It matches Feature points extracted in the previous frame and those of extracted in the current frame. Specifically, you set the camera pose transition between the two-earlier frame and the previous frame as the initial pose estimation, and then project the previous points into the current frame. Now you can search for points which have a similar feature around each projected feature point. For the most similar feature point, if the similarity exceeds the threshold, matching process succeeds.

There are usually about 300-500 points matched. The existing method uses all the matched pairs as input for PoseOptimization.

In PoseOptimization, when creating a Hessian matrix approximated with Jacobian [5], differential values for reprojection errors are analytically calculated for each point and aggregated. This algorithm requires O(N) time for the number N of pairs.

Therefore, when the number of pairs is decreased, the time required for this PoseOptimization is proportionally shortened.

PoseOptimization is a process executed in the same main thread that becomes a critical path for dealing with incoming frames. If you can reduce the time used in this thread, the system will achieve more frames per second.

Algorithm. 2 indicates the original ORB-SLAM2 pose optimization algorithm. In this method, the edges of the pose graph are created from all points whose map points are not empty.

In this paper, we propose a method to reduce the number of matched pairs input to PoseOptimization. Here, Algorithm. 3 shows the algorithm of this proposed method that generates a shuffled subset of pairs.

Data: The pairs of matched points between the current 2 frames
Result: An optimizer which is created from shuffled subset of the pairs

```
N := # of feature points in the
 current frame;
Array := [0 .. N-1] ;
for  i = Array[ 0 .. N-1] do
    if points[$i] != null then
     | optimizer.createEdge(points[$i]);
    end
end
return optimizer
```

Algorithm 2: The Original Method

Data: The pairs of matched points between the current 2 frames
Result: An optimizer which is created from shuffled subset of the pairs

```
N := #  of feature points in the current frame ;
Array := [0 .. N-1] ;
Array = Shuffle(Array);
for  i = Array[ 0 .. ratio * N] do
    if points[$i] != null then
     | optimizer.createEdge(points[$i]);
    end
end
return optimizer
```

Algorithm 3: The Proposed Method

This implementation reduces processing time because it reduces the data size needed to be processed by PoseOptimization. Specific experimental results are shown in the next chapter.

IV. Experimental Results

A. Experimental Setup

As the target images, KITTI odometry benchmark [6] sequence 00 is adopted, and the time for PoseOptimization is evaluated. The accuracy of tracking trajectory after the whole sequence is finished is also evaluated.

A stereo camera is used. After inserting the timing measurement code into PoseOptimization in ORB-SLAM2 [7], experiments are done for the following two cases.

1) The original case using all pairs for PoseOptimization.
2) The proposed method which selects $N \times ratio$ feature points in N feature points, and uses only possible pairs for PoseOptimization.

The accuracy evaluation of the tracking trajectory is done by accumulating Absolute Pose Error (APE) with Root Mean Squared Error (RMSE) [8]. For comparison with ground truth, evo [9] is adopted. The tool enables to compare tracking trajectory after the SE(3) Umeyama alignment so that the APE can be computed.

TABLE I
Calculation time of PoseOptimization

	Time of PoseOpt / msec			Accuracy/m		
Ratio	total	stddev	max	mean	median	rmse
0.5	2.8	2.1	7.9	1.13	1.12	1.24
0.6	3.3	2.5	9.2	1.19	1.06	1.34
0.7	3.9	2.7	9.9	1.19	1.15	1.33
0.8	4.7	3.2	11.5	1.17	1.12	1.32
0.9	5.1	3.7	12.4	1.17	1.04	1.32
1.0	5.3			1.16	1.09	1.28

B. Results

Table I shows the experimental results. We changed the ratio of using feature points for PoseOptimization from 50% to 100% by 10%. Figure 2 shows the value of RMSE under each condition, and Figure 3 shows the computation time of RoseOptimization of each condition. They are corresponding to Table I.

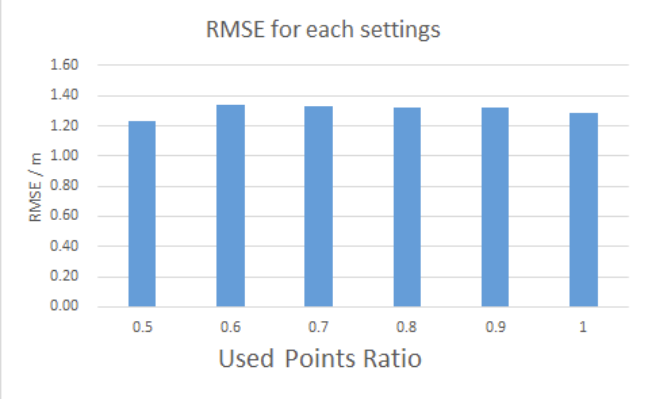


Fig. 2. Used Points Ratio and RMSE value

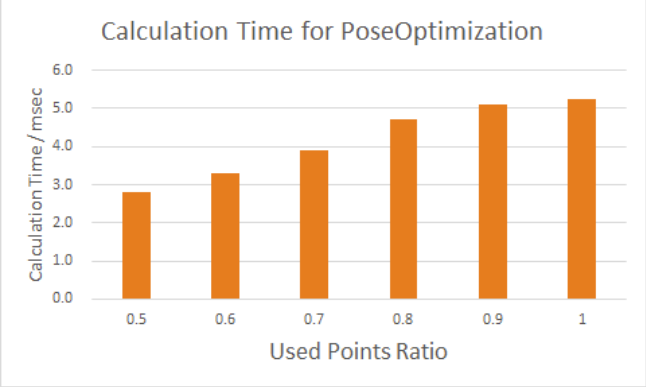


Fig. 3. Used Points Ratio and calculation time

Figure 3 shows that the computation time is reduced when the ratio of using points is reduced. The traditional case using all feature points took 5.3 msec, while the proposed method using 50% points reduced to 2.8msec. Moreover, Table I shows

that RMSE is improved even if the ratio of using feature points is reduced to 50%. The influence of the proposed method to the accuracy is considered in the next subsection.

C. Consideration

Figure 4 shows the distribution of re-production error of sorted points in PoseOptimization from the 203-th frame to the 211-th frame of KITTI odometry benchmark. The results show that a few points with large error give a large impact on the total error.

In the step of PoseOptimization, when the re-production error is larger than a threshold in any iteration, it is processed as the Outlier. Since the iteration of the optimization is done only four times, recording a large re-production error even once, it gives a large influence to the accuracy of the camera position.

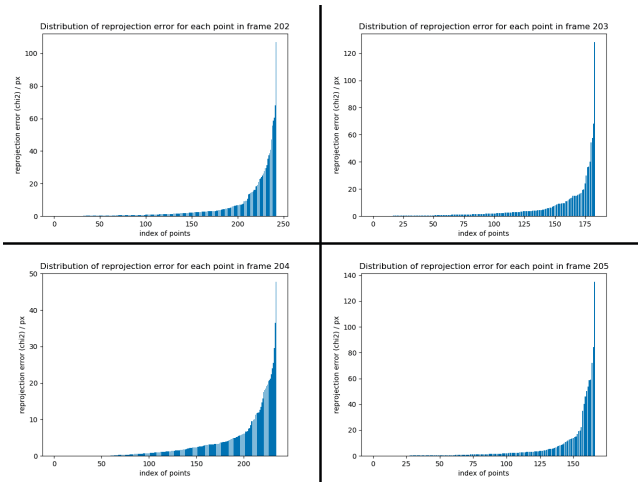


Fig. 4. Distribution of re-projection error in frames

By shuffling and using only a part of points, some cases could improve the evaluation of tracking trajectory since points with the large error were not used circumstantially.

To investigate it, we changed the seeds of random number used in PoseOptimization, and executed KITTI odometry benchmark sequence 00 multiple times to investigate the accuracy of tracking trajectory. Here, 50% of the feature points in the current frame (ration = 0.5) was used and the pose graph is drawn as shown in Figure 5.

This graph shows that the accuracy varies with the random numbers; in some cases, the accuracy is better than the traditional methods, while it is worse in other cases.

Here, the problem is what types of points have a large re-production error. Figure 6a shows an image in KITTI odometry benchmark sequence 00, and the colored squares show the location of points for feature extraction. On the other hand, Figure6b shows points whose re-production error in PoseOptimization is large (chi2 $>$ 15) with their error.

Points with large errors are for a lot of textures; for example, leafy recess, leaves, and bricks sidewalk whose brightness is

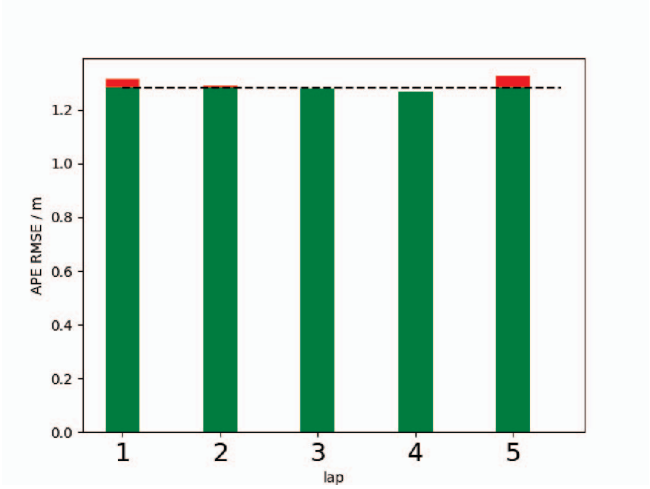
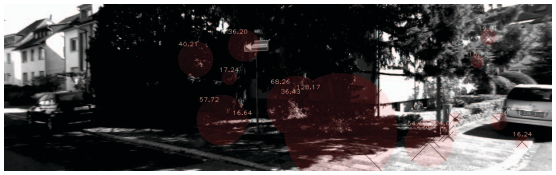


Fig. 5. Influence of random seed (lower is better) (ratio = 0.5)

(a) extracted feature points (frameno=203)



(b) Points whose re-projection error is large (frameno=203)

Fig. 6. (a) shows extracted and stereo matched points; (b) shows points whose re-projection error is larger than 15, and the error is shown on the left top of red circles, whose size means chi2 error in bundle adjustment;

changed drastically. The problems around textures of SLAM which uses feature points have been noticed as the lack of textures. That is, the region which does not have enough textures, the edge corner does not exist, and so it is difficult to find feature points. It means that the information around the area cannot be used.

However, on the other hand, the region with too many textures also causes the failure of SLAM or accuracy degradation. For example, in images with too many textures like road surface, a pattern is iteratively aligned. It tends to cause the alignment of points with similar features, and matching error to the neighboring points may happen. One of the solutions to the problem is introducing the mechanism to suppress the number of feature points by using the edge extraction and reducing the feature points in the high-density edge region.

V. Conclusion

We have presented a new method for pose graph optimization used in ORB-SLAM2. This method realizes the advantage of being less computationally expensive making the PoseOptimization function twice rapider without suffering little decrease in tracking trajectory accuracy.

The comparison to the existing research ORB-SLAM2 shows that the proposed method achieves the higher efficiency.

The reason this method does not suffer from accuracuy decrease can be explained in the viewpoint of texture. Both too low and too high texture areas are harmful to tracking accuracy, and in the section IV-C, we confirm points whose reprojection error is large from high texture area.

Investigating a new method to avoid these extreme texture regions is our future work.

References

[1] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, Vol. 33, No. 5, pp. 1255–1262, 2017.

[2] C. Forster, M. Pizzoli, and D. Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 15–22, May 2014.

[3] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to sift or surf. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pp. 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[4] Dorian Gálvez-López and J. D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, Vol. 28, No. 5, pp. 1188–1197, October 2012.

[5] T.Okatani. Bundle adjustment. https://ipsj.ixsq.nii.ac.jp/ej/?action=repository_action_common_download&item_id=62864&item_no=1&attribute_id=1&file_no=1, 2009.

[6] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[7] Raúl Mur-Artal and Juan D Tardós. Github: raulmur/ORB_SLAM2. https://github.com/raulmur/ORB_SLAM2, 2017.

[8] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *IROS*, pp. 573–580. IEEE, 2012.

[9] Michael Grupp. evo: Python package for the evaluation of odometry and slam. https://github.com/MichaelGrupp/evo, 2017.