A Novel Method for Camera Pose Tracking Using Visual Complementary Filtering

Xiangkai Lin, and Ronggang Wang *

Shenzhen Graduate School, Peking University
*Corresponding author: rgwang@pkusz.edu.cn

Abstract. Camera pose tracking is one of the major problems of Augmented Reality (AR) system. While tracking technologies based on fiducial markers have dominated the development of AR applications for almost a decade, various real-time approaches to Natural-Image-Marker tracking have recently been presented. However, most existing approaches do not yet achieve sufficient frame rates for AR on mobile phones or at least require an extensive training phase in advance. In this paper, we propose a novel method for fast camera pose tracking using visual complementary filtering. Our method can be used in Augmented Reality System with a stored natural image. The implementation is very fast and robust, allowing running even on mobile phones at highly interactive rates and accuracy. Our implementation can immediately track features captured from a photo without any training.

Keywords: Augmented Reality (AR); Visual Complementary Filtering; Mobile Computing; Natural Image Marker.

1 Introduction

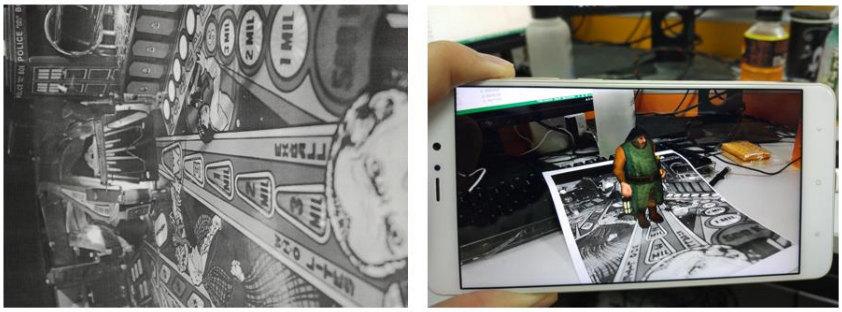
In a typical AR game application scenarios, we demand that users can interact with a certain object, such as a marker of a nature image (c.f. **Fig. 1**(a)). The user can observe the marker through the camera, while AR algorithm will draw a virtual object on the corresponding marker position (c.f. **Fig. 1** (b)). To ensure that the virtual object remain stationary as the camera moves, we need to track the precise camera pose from the marker. Such camera pose tracking algorithms face great challenges in tracking accuracy and running speed.

Existing nature image marker tracking AR systems can be divided into two categories: the first kind of methods first detect the marker, then perform tracking based on its salient feature points (e.g. [1]). The second kind of methods perform detection and tracking simultaneously, then fuse their results using optimization (e.g. [2]). The former is easy to be lost due to its simplicity, while the latter is too complex to achieve real-time on mobile devices.

To deal with above problem, we employ visual complementary filtering to fuse the results of detection and tracking, avoiding complex optimizing process while significant reduce the running time and improve the robustness.

The main contributions of this paper are as follows.

- We propose a method based on visual complementary filtering to fuse the results of detection and tracking.
- We design a fast and robust camera pose tracking system, which can be used in Augmented Reality.



(a) (b)

Fig. 1. (a) Nature image marker. (b) Draw virtual objects on the marker.

2 System Overview

Our system consists of four modules. A Rigid Detector which is used for initialize our system through a rigid marker detection. Then we use a Tracker to track significantly feature corners in main thread. At the same time a Fast Detector is running in the background thread preparing to update the Tracker. And we design a Fusion using a visual complementary filtering to fuse the results of detection and tracking. The framework is shown in **Fig. 2**.

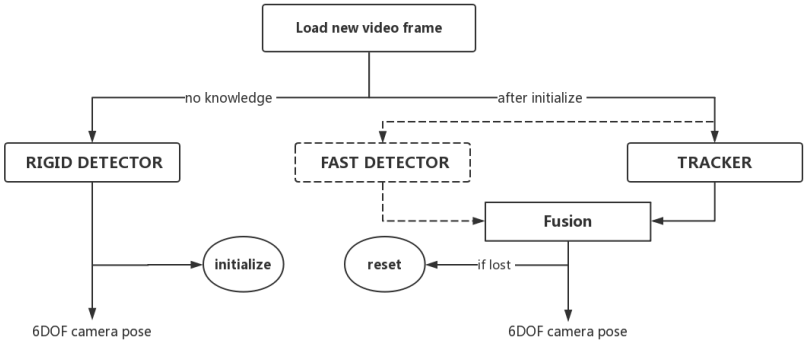


Fig. 2. Framework of our camera pose tracking system.

2.1 Rigid Detector

The framework of the Rigid Detector is shown in **Fig. 3**. The Rigid Detector is awakened only when the system starts for the first time or the Tracker has been interrupted in the previous frame, and it will sleep other time.

The ORB features [3] are detected in the coming frame. Afterwards these features are matched against a reference feature map. This feature map is generated when given a new marker image, which detected features from the marker image in multiscale. Then, the 2D feature matches are taken to calculate the homography from marker image to the current frame by applying several RANSAC iterations [4]. Considering all the corresponding 3D features in the marker are on the same plane, we generate a truncated extrinsic matrix from homography [5], which can be taken to decompose the exact camera 6-DOF pose [5]. Followed by a pose refinement step with a non-linear-least-square algorithm [6], we can get the accurate 6-DOF camera pose.

If a sufficient number of feature correspondences have been detected and validated by RANSAC iterations, we initialize our Tracker and Fast detector. We use the camera pose to find the marker area in current frame and then update the homography from marker image to current frame.

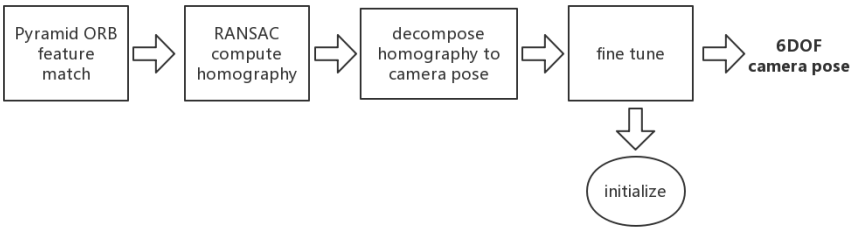


Fig. 3. Framework of the Rigid Detector.

2.2 Tracker

The framework of the Tracker is shown in **Fig. 4**. While initializing the Tracker, we detect some Shi-Tomasi corners [7] in the marker area and store the initial homography H_{lm}. When a new frame comes, we use the Lucas-Kanade algorithm [7] to find the matching points between the two frames and then compute the homography H_{cl}. We update the homography iteratively to get the final homography H_{cm} using:

$$H_{cm} = H_{cl} \times H_{lm} \tag{1}$$

where H_{cm} is the homography from marker image to current frame, H_{cl} is the homography from current frame to latest frame and H_{lm} is the homography from current frame to marker image.

After that, we decompose H_{cm} to get the camera pose from marker to current frame. Finally, we update the H_{lm} to H_{cm} and wait the next cycle.

We can use the result to find the current marker area and update the tracking Shi-Tomasi corners. It's no need to run it every cycle, so we put it in background.

Once Tracker is initialized, it can run without detector. Since we only track very few corners, it can be very fast. However, the error may accumulate with the iteration, resulting in drift.If Tracker lost, we will reset our system and call the Rigid Detector.

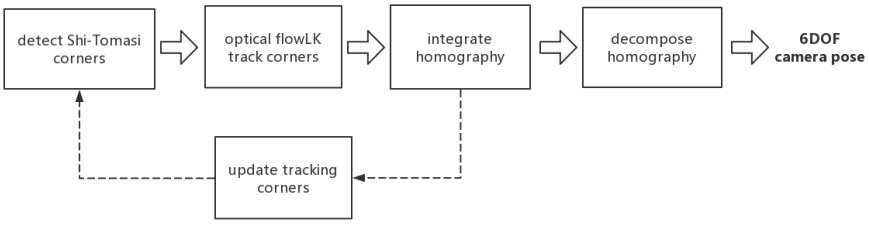


Fig. 4. Framework of the Tracker.

2.3 Fast Detector

As we can see in **Fig. 5**, compared with the Rigid Detector, the Fast Detector only detect features in the marker area and relax the feature extracted parameters. In matching process, it takes into account the latest camera pose to speed up matching. And it won't do the time-consuming fine tune.

Fast Detector will only run in background thread. Once have a result, we use it to correct our Tracker's bias.

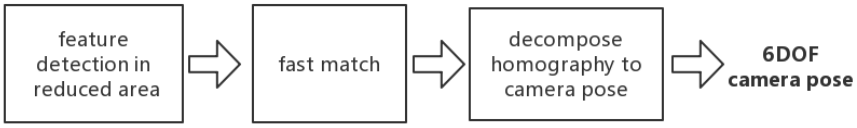


Fig. 5. Framework of the Fast Detector.

2.4 Fusion with Visual Complementary Filtering

The framework of the Fusion is shown in **Fig. 6**. The camera pose result of the Tracker T_t is accurate and smooth, but it will accumulate error and gradually leads to a drift. The Detector's result T_t has a high frequency error but the mean is stable. The two modules are complementary in frequency characteristic [8]. So, we use a low-pass filter F_l to filtering T_d, and a high-pass filter F_h to filtering T_t , and use the complementary filtering method to combine the two results and then get the accurate camera pose T_f .

$$T_f = F_l \times T_d + F_h \times T_t \tag{2}$$

As in Eq. (2), $F_l = \frac{c}{s+c}$, $F_h = 1 - F_l$, where s is a constant number and c is an all-pass filter. In our method, we choose the parameter c as a constant number 9, while parameter s as 1.

We can directly use Eq. (2) to filter the translation. While dealing with rotation, we should firstly change it into quaternions and then filter with a SLERP interpolation. The final result T_f will be used to update the Tracker.

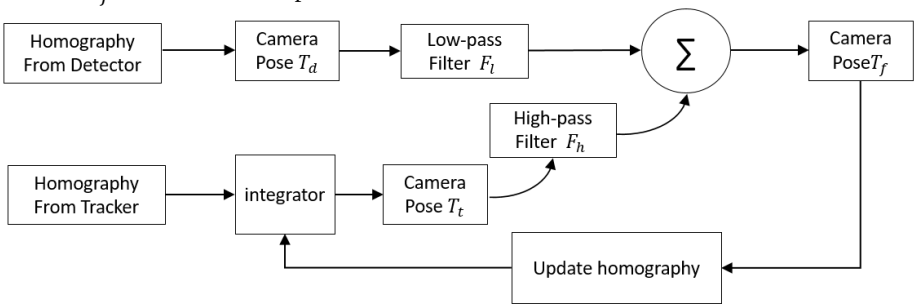


Fig. 6. Framework of the Fusion with Visual Complementary Filtering.

3 Experiments and Analysis

We test our system in terms of running speed and tracking accuracy. All the input frames are resized into size 640 × 480.

3.1 Tracking Precision Test

We compare the performance of our system with that of ARToolkit, which is the currently best open-source AR system. We use the same marker image as ARToolkit.

To evaluate the tracking accuracy of the camera pose, we use a ray tracer to generate images with known ground truth (c.f. **Fig. 7**). We can use a Rendering Engine like OPENGL to generate it.

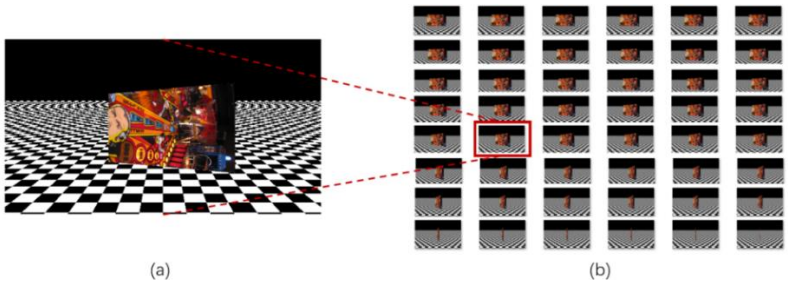


Fig. 7. (a) A synthetic image. (b) A small part of our synthetic image data set.

The first experiment measured the orientation accuracy of targets while fixing the distance. We measure rotation accuracy by term:

$$E_r = \sqrt{\left(g(f(R_t) - f(R_g))\right)^2} \tag{3}$$

where R_t is the current rotation matrix and R_g is the ground-truth rotation matrix. The function f(.) is to changes the rotation matrix to a quaternion vector, and the function g(.) is to calculate inner product.

We set the marker on the screen as the initial pose, and rotate marker around an axis from degrees 0 to 90. At the same time, we generate the rotation-test data set.

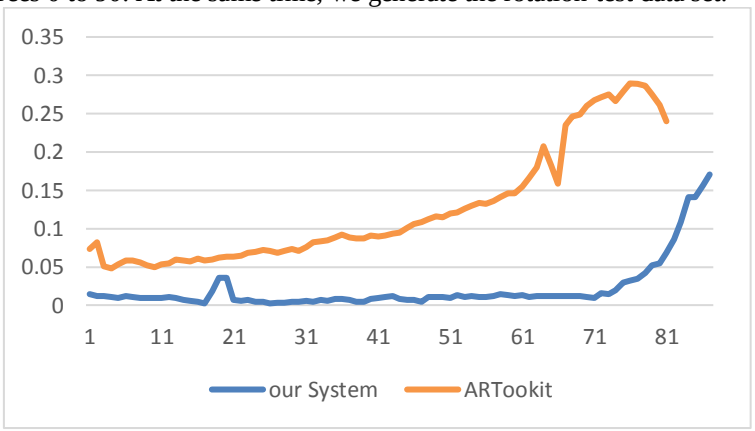


Fig. 8. Comparison of rotation accuracy. The abscissa is the rotation angle.

Test result is shown in **Fig. 8**. we can see that error is increasing with the increase of rotation angle. The reason is through rotation, image will suffer a significant affine transformation, which will destroy the features in tracking. Compared with ARtookit, our system has smaller error while rotating. Besides, our system can work in a larger range of vision rotation.

The complementary experiment is to hold angle and vary the distance. We move the marker away from the screen in a consistent distance and create our translation-test data set. We can measure translation accuracy by the translation data variance. Since different systems have different scale factors, we should firstly normalize their results.

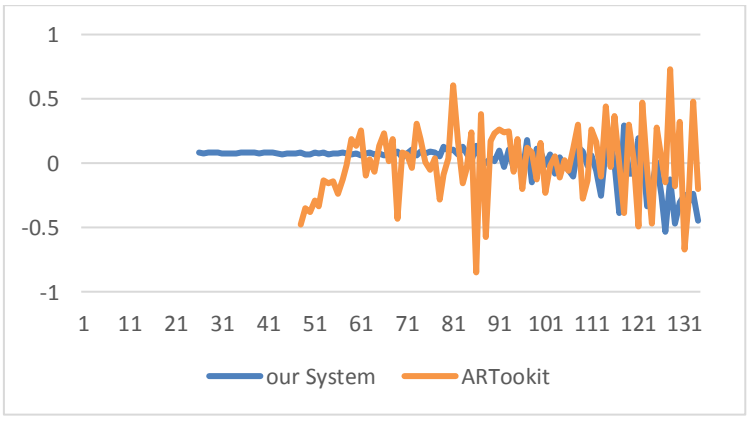


Fig. 9. Comparison of translation accuracy. The abscissa is normalized distance.

Test result is shown in **Fig. 9**. When the marker is close, we can only see a part of it, so it is hard to initialize. Once initialized, a closer image will provide more clear features to track, which makes the result more accuracy. As distance increases, the accuracy will decrease. Compared with ARtookit, our system can be initialized in a closer distance. Besides, our system provides significantly more robust results.

We also compare the performance of our whole system with only use the Tracker and the Detector. We compare their rotation accuracy using the rotation-test data set.

From **Fig. 10**, the Detector work in a smaller range of angle range with a larger deviation. The Tracker will accumulate error over time. Finally, through our method of visual complementary filtering, we can get the best results with least error.

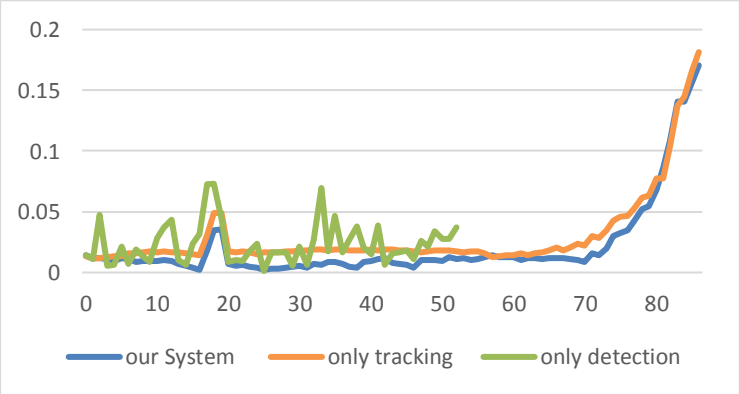


Fig. 10. Internal comparison results of rotation accuracy.

3.2 Running Speed Test

We test the running speed of our system both on desktop hardware and mobile phones. We calculate the time from the acquisition of a new frame to the calculation of a new camera pose result. All performance values are averaged over 30 iterations. And we separately test the Rigid Detector, the Fast Detector, the Tracker and the main thread.

As for desktop hardware, we use a Lenovo y720 with an Inter i5-6300HQ CPU@2.3Ghz, an ASUS n56 with an Inter i5-3230M CPU@2.6Ghz, and a Surface 3 with an Inter Atom x7-Z8700 CPU@1.6Ghz. The testing results are shown in **Table 1**

Table 1. Average running time in desktop hardware. All values are given in milliseconds.

PC	Lenovo r720	ASUS n56	Surface 3
Rigid Detector	11.38	20.71	32.21
Fast Detector	8.14	18.49	28.45
Tracker	2.52	3.42	6.65
Main Thread	3.12	4.16	7.99

As for mobile phones, we choose Samsung s7, Xiaomi 5, and Meizu note3 as our test platform. The testing results are shown in **Table 2** .

Table 2. Average running time on mobile phone. All values are given in milliseconds.

PHONE	Samsung S7	Xiaomi 5	Meizu note3
Rigid Detector	22.47	25.21	57.13
Fast Detector	18.35	21.28	51.36
Tracker	6.36	8.57	18.92
Main Thread	7.39	9.56	19.18

As is shown **Table 1** in and **Table 2**, both on the desktop hardware and mobile phones, the most time-consuming part is the Rigid Detector. However, it is an one-time run only happen in initialization. The second time-consuming module is the Fast Detector, which runs in the background thread and won't block the main thread. Tracker is very fast, which make our system run in no more than 20 milliseconds per frame even in low-end mobile phones.

4 Conclusion

In this paper, we present a novel camera pose tracking method using visual complementary filtering. Our system is broadly practical since it only needs a picture of nature image as the marker to track camera pose. The experiments show that our system can achieve higher speed and higher accuracy compared with other implementations. Our system can be easily applied to AR applications in mobile phones.

References

1. Wagner, D., Reitmayr, G., Mulloni, A., Drummond, T., Schmalstieg, D.: Pose tracking from natural features on mobile phones. In: ISMAR '08 Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, pp. 125–134. IEEE/ACM (2008)
2. Engedal T. Camera pose estimation apparatus and method for augmented reality imaging: U.S. Patent 8,452,080[P]. 2013-5-28
3. Rublee, E., Rabaud, V., Konolige, K., Bradski, G.: ORB: an efficient alternative to SIFT or SURF. In: IEEE International Conference on Computer Vision (ICCV), pp. 2564–2571. IEEE (2011)
4. Herling, J., Broll, W.,: An adaptive training-free feature tracker for mobile phones. In ACM Symposium on Virtual Reality Software and Technology. pp. 35-42. IEEE(2010)
5. Olson E.: AprilTag: A robust and flexible visual fiducial system. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), 19(6), 3400-3407. IEEE (2011)
6. Hartley, R., Zisserman, A.: Multiple View Geometry in Computer Vision Second Edition. Cambridge University Press. (2004)
7. Lucas, B. D., Kanade, T.: An iterative image registration technique with an application to stereo vision. 674-679. (1981)
8. Chang-Siu E., Tomizuka M., Kong K.: Time-varying complementary filtering for attitude estimation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). pp. 2474-2480. IEEE (2011)