

Multiple 3D Object Tracking for Augmented Reality

Youngmin Park*
GIST, U-VR Lab

Vincent Lepetit†
EPFL, CVLab

Woontack Woo‡
GIST, U-VR Lab

ABSTRACT

We present a method that is able to track several 3D objects simultaneously, robustly, and accurately in real-time. While many applications need to consider more than one object in practice, the existing methods for single object tracking do not scale well with the number of objects, and a proper way to deal with several objects is required. Our method combines object detection and tracking: Frame-to-frame tracking is less computationally demanding but is prone to fail, while detection is more robust but slower. We show how to combine them to take the advantages of the two approaches, and demonstrate our method on several real sequences.

Index Terms:

1 INTRODUCTION

Single 3D object tracking using natural features has now been well explored by the Computer Vision and Augmented Reality communities [4]. By contrast, the multiple object case, where several 3D objects must be considered simultaneously, has not really been addressed, while it is often needed for real applications, for example a tangible interface.

In this paper, we apply image feature recognition and keyframe techniques to several objects, because they were shown to be very robust for the single object case [8] and also for SLAM [3], by preventing loss of track and drift. Unfortunately, it is not possible to directly apply the method described in [8] in practice, because its complexity would grow with the number of objects.

We therefore propose an approach that is able to efficiently deal with several objects without concession to robustness. In addition, it is particularly stable. The key is an efficient combination of feature recognition and feature tracking from frame to frame. Because feature recognition is usually very time consuming, it is indeed very tempting to introduce feature tracking, which is generally cheaper. However, this has to be done carefully to avoid error propagation that eventually leads to drift.

Our approach is somehow related to [12] but is much more robust since ours exploits feature recognition. For each target object, we have its CAD 3D model plus a small set of reference images we will call “keyframes” of the object. We try to match each input frame against only a subset of the keyframes to keep processing at frame-rate, and track features lying on the visible objects over consecutive frames. The two sets of matches, those with the previous frame and those with the keyframes, are correctly combined by propagating errors to estimate the object 3D poses. Thus, robustness is achieved by regularly “injecting” features from the keyframes in the computation. Tracking features from frame-to-frame gives stability and relaxes the need for considering every keyframe at every frame, keeping the method efficient.

*e-mail: ypark@gist.ac.kr

†e-mail: vincent.lepetit@epfl.ch

‡e-mail: wwoo@gist.ac.kr

In the remainder of the paper, we first discuss related work on single and multiple object tracking. Then, we describe our method, and present our results.

2 RELATED WORK

Many multiple object 2D tracking algorithms have been developed [9, 10] however the 2D problem is only remotely related to the 3D case we address in this paper. By contrast, works on multiple object 3D tracking are almost inexistent, and we will focus here on recent single object 3D tracking.

The state-of-the-art is probably the tracking-by-detection approach of [8], which is the starting point of our method. It detects the object in each input frame independently by matching feature points extracted in the input frame against those extracted in some keyframes (or reference frames) representing the target objects. Other methods for this wide baseline matching, or recognition, of feature points are available [5, 1], but [8] is very simple to implement and very fast. The approach is robust to occlusion, light, scale and perspective changes. However, a naive extension to several objects would not scale, since the system would have to try all the objects for every single input frame, making the complexity grow linearly with the number of objects.

Our method is also related to frame-by-frame 3D tracking approaches, which tracks image features such as edges [2] or feature points [12] over the sequence. In [12], feature points are extracted from the current frame and matched against those from the previous frame, but also against those extracted in keyframes. We use a similar scheme, however the method of [12] to establish matches with the keyframes is very vulnerable to fast motion, contrary to ours. Another difference is the way we fuse the two sets of matches. Our approach relies on error propagation, which allows proper error handling.

3 MULTIPLE OBJECT TRACKING

In this section, we first give an overview of our multiple object detection and tracking algorithm, and then detail it step by step.

3.1 Overview

As discussed in the introduction, we want to rely on object detection because it makes tracking applications more robust, but it has to be done carefully to keep processing time consistent with real-time constraints. An overview of our approach is given in Figure 2. Our starting idea is to distribute the time complexity of detecting multiple objects over consecutive frames. We do not try to detect every object in every input frame but only as many as possible while maintaining camera frame-rate. The objects present in the frame but not detected will be detected in one of the few next frames. That results in a small delay that is kept under a second in practice, so it is not really perceptible by the user.

When a new object appears, it is therefore quickly detected by the system which initializes a frame-by-frame tracking for this object. To do so, we rely on what we call “temporal keypoints”, which are feature points detected on the object surface and that matched over consecutive frames. From them, we can continue to estimate the object pose accurately even without object detection because it

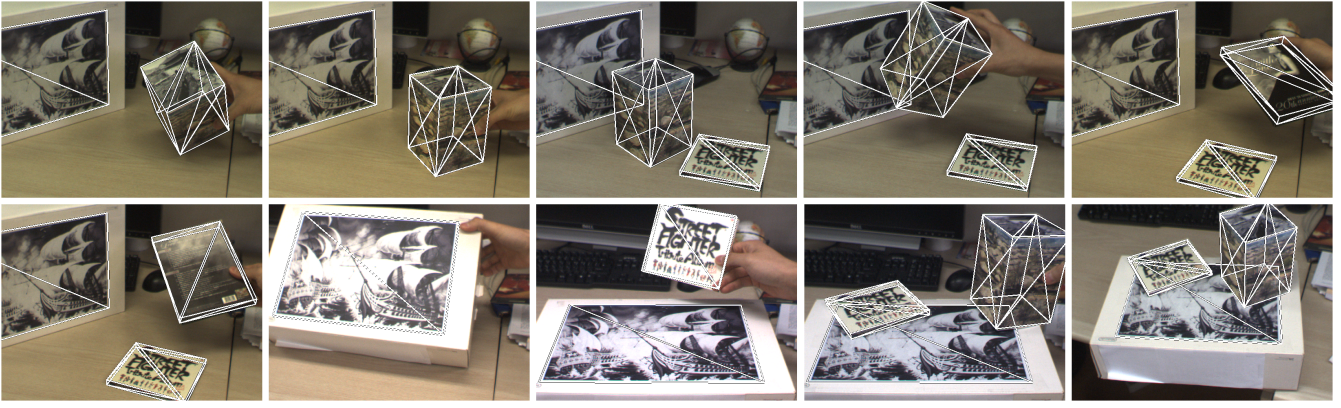


Figure 1: Multiple 3D object tracking: Our method can automatically recognize and track several objects simultaneously under partial occlusions and illumination changes, at about 15 Hz.

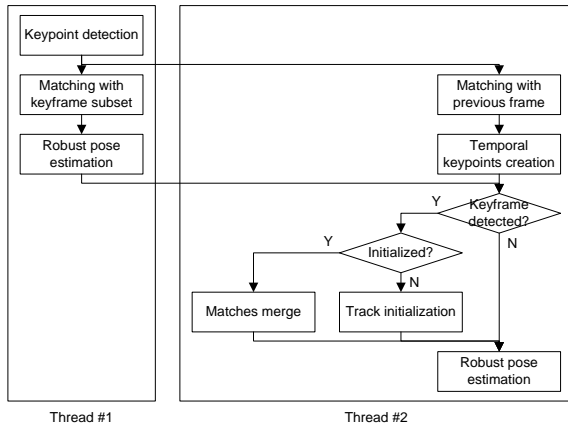


Figure 2: Overview of the processing for one input frame. Detection and frame-by-frame tracking are performed on two different cores in parallel.

is less time consuming method compared to object detection. The other advantage is that it is usually more accurate.

In the meantime, we keep trying to detect the application objects, even when a frame-by-frame tracking process is already running for them. That way, we prevent common problems such as loss of track due to fast motion or occlusions, and drift. Note that detection and tracking are clearly separated and most of the computations can be done on different cores of the processor in parallel.

In the next sections, we first detail the data structure we use to represent the objects tracked by the system, then we discuss the detection process, and describe how temporal keypoints are used and fused to the object detection results to provide a stable track of the visible objects.

3.2 Object Model

Our object model contains both the geometrical information and appearance of a target object. The geometrical information is a standard CAD 3D model stored as a list of triangles. Many software now exist to easily build such 3D model from a set of pictures.

The appearance part is made of a small set of keyframes that are registered images of the object shot from various viewpoints so that they cover most of the object. Usually, 3 to 4 keyframes are sufficient to cover an object along 360 degrees with some parts

taken more than once. In each keyframe, we extract feature points we call keypoints. It is easy to estimate the 3D locations \mathbf{M}_i of these keypoints by back-projecting them on the object 3D model. These keypoints and their 3D locations are stored as well. They will be used during the detection phase.

3.3 Sequential Multiple Object Detection

3.3.1 Keyframe Selection

We divide all the keyframes from all the objects into smaller subsets and match one subset with each camera image that is we build subsets S_i such that:

$$\begin{aligned}
 S_1 &= \{ \kappa_1, \kappa_2, \dots, \kappa_f \} \\
 S_2 &= \{ \kappa_{f+1}, \kappa_{f+2}, \dots, \kappa_{2f} \} \\
 &\vdots \\
 S_{N/f} &= \{ \kappa_{f \cdot \text{floor}(\frac{N}{f}) + 1}, \dots, \kappa_N \},
 \end{aligned}$$

where the κ_j are the keyframes, f is the number of keyframes that can be handled at frame-rate, and N is the total number of keyframes.

Each input frame is matched against the keyframes of one of these subsets S_i using the method described in the next section. The subsets are simply considered iteratively one after the other: after N/f frames, we restart from S_1 .

3.3.2 Matching with Keyframes and Pose Estimation

We use the method described in [8] to match the input frame against keyframes because it is fast and robust under perspective distortion and different viewing conditions. It gives a number of initial correspondences between feature points extracted in the input frame and the keypoints of each of the keyframes in S_i for which the 3D locations \mathbf{M}_i are known. Since some of these correspondences can be erroneous, the object pose is estimated using RANSAC and a non-iterative P-n-P algorithm [6].

The detection is considered successful when the number of inliers found by RANSAC is larger than a threshold. If the object was no present before, we perform a non-linear optimization to refine the object pose computed by RANSAC to initialize the frame-by-frame tracking. If the object was already present, the inlier matches are added to the frame-by-frame matching to estimate the object pose as it will be described below in Section 3.4.

3.4 Stable Tracking with Temporal Keypoints

We now explain our stable tracking algorithm that combines object detection and frame-by-frame tracking. Frame-by-frame tracking has two purposes: It completes detection to track the object whenever it is present, and it stabilizes the estimated pose, removing the jittering effect that detection alone would produce.

We extract feature points in each input frame, and match them against those extracted in the previous frame using standard techniques based on cross-correlation and local search. Compared to the KLT tracker [11], it allows us to easily handle appearing and disappearing feature points, and to prevent drift of the tracked locations. We call these feature points “temporal keypoints”.

We fuse temporal keypoints with keypoints matched during object detection to estimate the object poses. For a given object, its pose parameters at time t $\mathbf{p}^{(t)}$ is estimated by minimizing:

$$\mathbf{p}^{(t)} = \underset{\mathbf{p}}{\operatorname{argmin}} \sum_i \rho \left(\|\phi(\mathbf{p}, \mathbf{M}_i) - \mathbf{m}_i\|^2 \right) + \sum_j w_j \rho \left(\|\phi(\mathbf{p}, \mathbf{X}_j) - \mathbf{x}_j\|^2 \right) \quad (1)$$

where \mathbf{p} is a 6-vector that contains the rotation and translation values, the \mathbf{M}_i are the 3D locations of the keypoints extracted from the matched keyframe and the \mathbf{m}_i , their 2D locations, found by the object detection process explained Section 3.3.2, the \mathbf{X}_j and the \mathbf{x}_j are respectively the 3D and 2D locations found for the j^{th} temporal keypoint. $\phi(\mathbf{p}, \mathbf{M})$ is a function that returns the 2D projection of \mathbf{M} under camera pose \mathbf{p} .

The terms of the sums in Eq. (1) are weighted differently: The w_j s are weights that will allow to handle the fact that the \mathbf{X}_j are not known exactly, as it will be detailed below. Since the 3D locations \mathbf{M}_i come from the keyframes registered offline, we assume they are error-free and we do not use such a weight for the corresponding terms. $\rho(\cdot)$ is a robust estimator; we use an approximation of the Tukey estimator:

$$\rho(x^2) = \begin{cases} x^2 & \text{when } x^2 < c^2 \\ c^2 & \text{otherwise} \end{cases}, \quad (2)$$

where c is a thresholding value. When no correspondences are available from the detection, only the second term of the sum is used. In our implementation, we use the Levenberg-Marquardt algorithm for minimization.

The 3D locations of the temporal keypoints are estimated by back-projecting their 2D location in the previous image using the estimated pose $\mathbf{p}^{(t-1)}$. Back-propagation can be performed efficiently using OpenGL rendering to quickly find the triangle on which the keypoint lies [12]. Since $\mathbf{p}^{(t-1)}$ is only an approximation of the true pose, we use the w_j s to weight the contributions of the different temporal keypoints, according to their error estimates. We take the w_j as:

$$w_j = \rho(\operatorname{tr}(\Sigma_{\mathbf{X}_j})), \quad (3)$$

where $\operatorname{tr}(\cdot)$ is the trace function, and $\Sigma_{\mathbf{X}_j}$ is the covariance on the 3D location \mathbf{X}_j . $\Sigma_{\mathbf{X}_j}$ is obtained in two steps: First we estimate the covariance $\Sigma_{\mathbf{p}^{(t-1)}}$ of the pose estimate for the previous frame. Then we propagate this error to \mathbf{X}_j .

Under the standard assumption that the correspondences are independent, $\Sigma_{\mathbf{p}^{(t-1)}}$ can be computed by back-propagation as:

$$\Sigma_{\mathbf{p}^{(t-1)}} = \left(\left(\frac{\partial f_k}{\partial \mathbf{p}} \right)_{\mathbf{p}^{(t-1)}}^\top \Sigma_k \left(\frac{\partial f_k}{\partial \mathbf{p}} \right)_{\mathbf{p}^{(t-1)}} \right)^{-1}, \quad (4)$$

where Σ_k is the 2×2 covariance on the extracted keypoints, and the f_k are functions that can be expressed as:

$$f_k(\mathbf{p}) = \phi(\mathbf{p}, \mathbf{X}_k^{(t-1)}). \quad (5)$$



Figure 3: Keyframes used in our experiments.

In this expression, $\mathbf{X}_k^{(t-1)}$ denotes the k^{th} temporal keypoint used to estimate pose $\mathbf{p}^{(t-1)}$. Let’s now consider a temporal keypoint that comes from a match $\mathbf{x}_j^{(t-1)} \leftrightarrow \mathbf{x}_j$. The covariance $\Sigma_{\mathbf{X}_j}$ of its 3D location \mathbf{X}_j can be estimated by forward propagation of $\Sigma_{\mathbf{p}^{(t-1)}}$ as:

$$\Sigma_{\mathbf{X}_j^{(t)}} = \begin{pmatrix} \frac{\partial g_k}{\partial \mathbf{p}} \end{pmatrix}_{\mathbf{p}^{(t-1)}, \mathbf{x}_j^{(t-1)}} \Sigma_{\mathbf{p}^{(t-1)}} \begin{pmatrix} \frac{\partial g_k}{\partial \mathbf{p}} \end{pmatrix}_{\mathbf{p}^{(t-1)}, \mathbf{x}_j^{(t-1)}}^\top + \begin{pmatrix} \frac{\partial g_k}{\partial \mathbf{x}} \end{pmatrix}_{\mathbf{p}^{(t-1)}, \mathbf{x}_j^{(t-1)}} \Sigma_{\mathbf{p}^{(t-1)}} \begin{pmatrix} \frac{\partial g_k}{\partial \mathbf{x}} \end{pmatrix}_{\mathbf{p}^{(t-1)}, \mathbf{x}_j^{(t-1)}}^\top, \quad (6)$$

where $g_k(\mathbf{p}, \mathbf{x})$ is a function that back-projects keypoint $\mathbf{x}_j^{(t-1)}$ on the object under pose \mathbf{p} . Its Jacobian matrices in Eq. (6) are computed at pose $\mathbf{p}^{(t-1)}$ and for keypoint $\mathbf{x}_j^{(t-1)}$.

4 EXPERIMENTS

For our experiments, we considered the seven box-shaped objects and the two planar objects displayed Figure 3. The objects were registered in the keyframes using manual 3D-2D correspondences. More complex shaped objects could also be considered if their 3D model is available. We used only one keyframe per subset.

We tested our implementation on a 3.2 GHz multi-core CPU PC. The total keyframe set includes 9 keyframes shown in Figure 3. It works at 15 to 20 frames per second, which is less than the detection only [8] because of the combined tracking steps. However, it compensates with multiple objects and improves stability, and more importantly, the tracking frame rate does not decreased drastically with the number of considered objects.

In Figure 1, up to three objects are in the point of view of the camera and tracked simultaneously. The pose of the objects are accurately estimated despite the partial occlusions between objects. Figure 4 shows another example where one object is completely occluded by another one but tracked again when it is visible again.

Figure 5 demonstrates the accuracy of the proposed tracking algorithm. We fixed a camera and tracked the movement of an object which was constrained to lie on a planar surface as shown on the first row. The recovered trajectory for the object centroid is shown on the second row and exhibits only very limited jitter and lies on a 3D plane as expected. This validates the accuracy of the system.

Figure 6 plots the evolution of frame rate for a typical sequence. While the number of tracked objects varies over the sequence, the frame rate is almost constant around 16Hz, showing the approach is suitable for interactive applications and does not degrade with the number of tracked objects as long as this number remains reasonable.

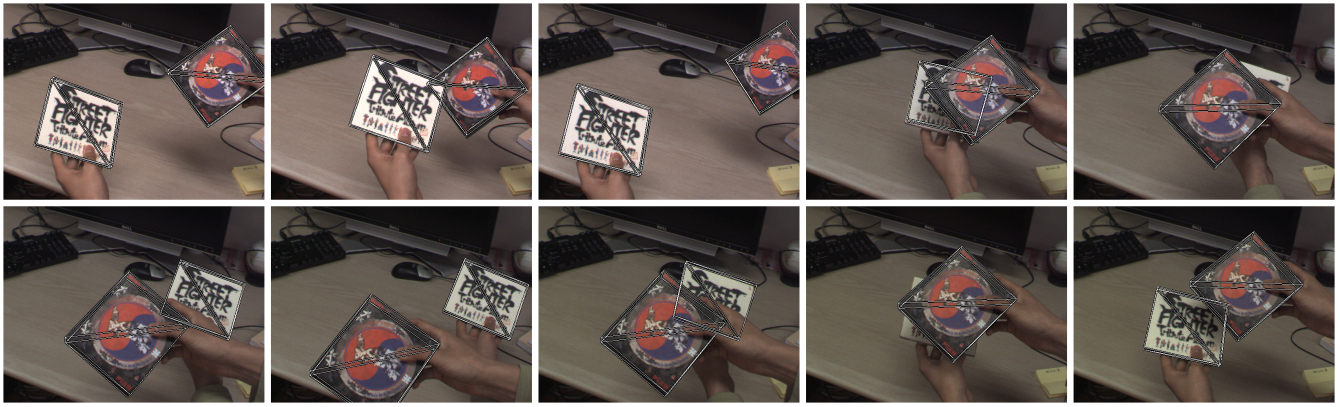


Figure 4: Tracking several objects under difficult conditions: The objects are moving fast, one object completely occludes the other one in the middle of the sequence, but the occluded object is re-initialized when it is visible again.

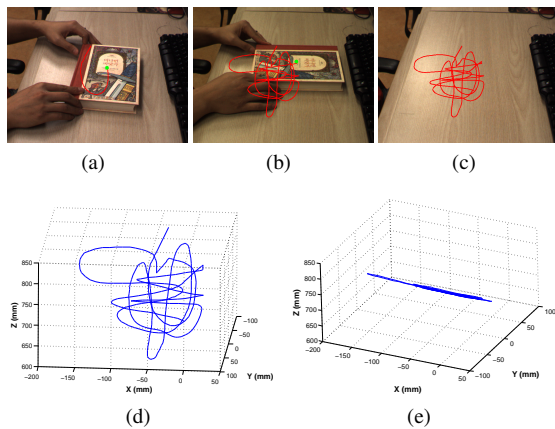


Figure 5: Accuracy of the proposed tracking algorithm: (Top row:) An object is moved while constrained to lie on a planar surface. (Bottom row:) The recovered trajectory for the object centroid exhibits only small jitter and lies on a 3D plane as expected.

5 CONCLUSION & FUTURE WORK

We presented a method for tracking simultaneously multiple 3D objects using a monocular camera. From the experiments, it is shown that multiple objects are successfully considered in a reasonable frame rate. Typical applications are table-top AR and tangible interaction applications.

However, our approach still not properly scales with the number of objects and is currently limited to a database with a few tens of objects. To handle larger databases, it may be interesting to first limit the number of possible present objects with a scalable recognition method such as [7] which does not provide any exact pose but can provide the list of the visible objects. This will be addressed in future work.

ACKNOWLEDGEMENTS

This research is supported by the Foundation of UCN Projects, the MKE, and the 21C Frontier R&D Program in Korea as a result of subproject UCN 08B3-O1-20S

REFERENCES

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. In *9th European Conference on Computer Vision*, May 2006.

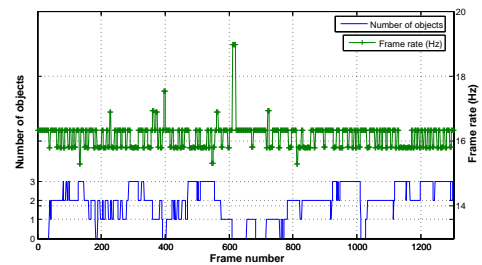


Figure 6: Tracking frame rate: The green curve represents the evolution of the frame rate over a period of time, the blue curve represents the number of objects tracked by the system. While the number of objects varies, the frame rate remains almost constant around 16 Hz.

[2] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(7):932–946, July 2002.

[3] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *International Symposium on Mixed and Augmented Reality*, Nara, Japan, November 2007.

[4] V. Lepetit and P. Fua. Monocular model-based 3D tracking of rigid objects. *Foundations and Trends in Computer Graphics and Vision*, 1(1):1–89, 2005.

[5] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[6] F. Moreno-Noguer, V. Lepetit, and P. Fua. Accurate non-iterative $O(n)$ solution to the PnP problem. In *IEEE International Conference on Computer Vision*, Rio de Janeiro, Brazil, October 2007.

[7] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Conference on Computer Vision and Pattern Recognition*, volume 2, pages 2161–2168, October 2006.

[8] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Conference on Computer Vision and Pattern Recognition*, Minneapolis, MI, June 2007.

[9] D. Reid. An Algorithm for Tracking Multiple Targets. *IEEE Trans. on Automatic Control*, 24(6):843–854, Dec 1979.

[10] S. Oh, S. Russell, and S. Sastry. Markov Chain Monte Carlo Data Association for General Multiple-Target Tracking Problems. In *Proc. of the IEEE Conference on Decision and Control*, Bahamas, Dec. 2004.

[11] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.

[12] L. Vacchetti, V. Lepetit, and P. Fua. Stable real-time 3D tracking using online and offline information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10):1391–1402, 2004.