Fast Simultaneous Tracking and Recognition Using Incremental Keypoint Matching

Jonathan Mooser, Quan Wang, Suya You, and Ulrich Neumann
CGIT Lab, Viterbi School of Engineering
University of Southern California
{mooser, quanwang, suyay, uneumann} @usc.edu

Abstract

This paper presents a unified approach to object recognition and object tracking, combining local feature matching with optical flow. Like many traditional recognition algorithms, the one described here implements recognition by matching detected image patches against a database of known objects. This algorithm, however, matches keypoints incrementally, meaning that it only tests a few keypoints at each frame until the complete object is identified. Recognition and tracking thus proceed in real-time, even with high dimensional features and an arbitrarily large database.

Central to this work is the system by which keypoint matching and optical flow mutually aid one another. Keypoint matching recognizes an object and estimates its pose in order to initialize tracking. Optical flow tracking, in turn, maintains the object pose over subsequent frames, discarding newly matched keypoints that do not fit with the current pose estimation. Experimental results demonstrate that this powerful combination provides robust, real-time recognition and tracking of multiple objects in the presence of scale and orientation changes as well as partial occlusion.

1. Introduction

Object recognition and object tracking are closely linked problems in computer vision. Both often rely on similar low level tasks such as keypoint matching and model fitting. Moreover, the two often play complementary roles within a larger system, with object recognition used to initialize and re-initialize tracking.

The ability to reliably recognize an object and then track its movements has numerous valuable applications. An autonomous navigation system, for example, might determine its location based on known objects within its field of view, then adjust its course by tracking those objects. An augmented reality application could display annotations based on the objects it sees, using the objects' positions and orientations for accurate spacial rendering. Traffic monitoring systems often seek to recognize and track vehicles. Security systems may attempt to do the same with people.

The goal of our work is to tightly integrate recognition and tracking into a unified system. We want to recognize an object in real time within the first few frames of a captured video, then track its pose through subsequent movements. Rather than implement recognition as a separate initialization step, however, we treat it as an ongoing process that both aids and benefits from tracking. Our strategy follows two key intuitions:

- If keypoints can be reliably tracked from one frame to the next, then matching results from multiple frames can be combined to produce a more confident object identification and a more robust pose estimation.
- Once we have enough keypoint matches to compute an object's pose, we can infer the locations of all other keypoint matches, and use those points to aid the tracking process.

These two principles lead to a process we call *incremental keypoint matching*. We begin with a database of learned keypoint descriptors, each stored with an object ID and an object-space location. At each frame, from a large collection of tracked keypoints, we match only a small subset against the database. We thus spread the processing time dedicated to keypoint matching evenly across all frames. If an object is, in fact, present we can positively identify it and compute its pose within a few frames.

Figure 1 illustrates this process. The matches found in the first frame are insufficient to recognize either object. The matches accumulate over subsequent frames and eventually enough keypoints are matched to positively identify the object (poses indicated by white frames). As the sequence continues, more points are matched against the database, providing an increasingly confident object identification and pose estimate. This way, should the system lose track of some of the originally matched keypoints, we can still track the overall object using newer matches.

Moreover, using the object's estimated pose, we can compute positions for keypoints that have never been matched to the database. Tracking these points along with

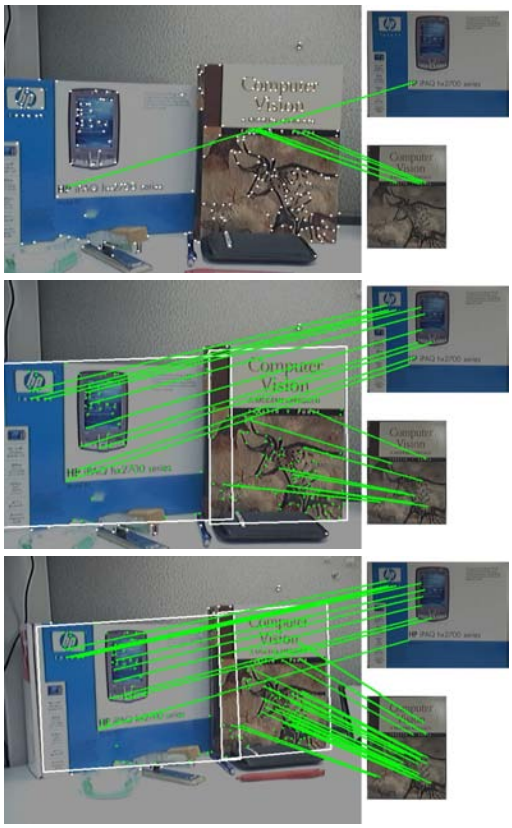

Figure 1. An example of Incremental Keypoint Recognition.

the keypoints that have been matched to the database helps to maintain an accurate pose through subsequent frames.

Our algorithm makes use of four basic low-level components: a keypoint detector, a keypoint descriptor, an optical flow tracker, and a RANSAC pose estimator. Section 3 briefly describes each of these. The primary contribution of this work is the overall algorithm that fits these components together, as detailed in section 4.

Our Experiments focus on 2D objects. The database therefore specifies the position of each keypoint as (x, y) coordinates. Sections 5 and 6 describe our experiments and their results respectively. We show that incremental keypoint matching can robustly recognize objects in a cluttered scene, usually within ten to twenty frames, and that the subsequent tracking is reliable.

2 Related Work

In recent years a wide array of techniques have been developed for tracking complex objects using either global templates [3, 4, 13, 14] or collections of local features [1, 5, 7, 12]. In any practical implementation, all of these require some sort of recognition or detection to initially define what is being tracked. The approach to this requirement tends to fall into one of two extremes.

On the one hand, one may implement tracking by simply using a recognition algorithm fast enough to operate in real-time. This is the approach taken, for example, by Lepetit, *et al.* [7], who use randomized trees to quickly match local image patches to one of several pre-computed views of an object. At each frame, the whole object is recognized from scratch in order to recover its pose. A similar procedure is followed by Wang, *et al.* [17].

This approach is effective as long as the recognition is guaranteed to work in real-time. Two problems are likely to arise, however. The first is that for any frame in which recognition fails, we have no ability to locate the object until recognition succeeds again. One of our main goals is thus to maintain a reliable pose estimate even when recognition briefly fails. The second problem is that even very fast recognition algorithms will become slower as the size of the database increases. An algorithm that works in real-time for a small database may therefore become unacceptably slow when the database is much larger.

At the other extreme, one may recognize an object in the first frame of a sequence, then use a separate tracking algorithm to follow its position as the sequence proceeds [1, 3, 4, 5, 9, 10, 12, 13]. These are often presented without reference to specific recognition techniques under the assumption that any recognition process will suffice so long as it returns an initial object position. This provides a certain degree of flexibility, because one recognition algorithm can be replaced with another as needed without affecting the tracking algorithm. For experimental purposes, tracking can be initialized manually.

The drawback to treating tracking and recognition as completely separate processes is that recognition can be slow. A disproportionately long time needs to be applied to the first frame, after which the object may have moved too far to be tracked. An equivalent problem will arise should tracking fail and recognition be called for re-initialization. It is precisely this problem that we address with incremental recognition. By matching a few features against the database at each frame, we spread processing time evenly across an entire sequence, resulting in a system more appropriate for real-time applications.

Sakagaito, *et al.* [14] describe a form of simultaneous tracking and recognition based on fast high dimensional nearest neighbor searching, in which the matching results from one frame are used to initialize the search for the next frame. The primary difference between their work and our own is that they perform matching by global object template rather than local features, leaving no way to handle occlusion. Moreover, they only estimate object pose by embedding a pre-computed pose for every object template. Using image keypoints with known object space correspondences, we can achieve accurate pose estimation even for previously unseen orientations and scales.

An incremental approach to tracking was presented by Welch and Bishop [18] in which under-constrained prob-

lems are solved by incorporating sensor observations over time. Their work focused on artificial landmarks and did not address recognition.

We also draw an important distinction between our approach and those that track the motion of generic moving objects within the camera's field of view. For example, Song and Nevatia [16] and later Perera, *et al.* [11] demonstrate multiple vehicle tracking in urban environments. Taking any sufficiently large region of moving pixels to be either a car or a group of cars, both apply probabilistic motion models to infer the most likely series of positions over a sequence of frames. Our work, by contrast, recognizes and tracks specific object instances based on a trained appearance model. Unrecognized objects, moving or stationary, are ignored.

3 System Components

The keypoint detector, keypoint descriptor, optical flow tracker, and RANSAC pose estimator make up the low-level components of our algorithm. Each is based mostly on well-established computer vision techniques. We describe these components briefly in the following subsections.

3.1 Keypoint Detector

From the first captured frame, I_0, we detect an initial set of keypoints, K_0. Our keypoint detector follows the basic framework of Shi and Tomasi [15], who show that a pixel, i, for which the gradient covariation matrix, $M(i)$, has two large eigenvalues is particularly effective for tracking. We thus scan an entire image and accept i as a candidate keypoint if the smaller eigenvalue of $M(i)$ is above a predefined threshold and is a local maximum within a 5x5 pixel neighborhood.

We then apply two additional filters to the set of candidate keypoints. First, all keypoints within 20 pixels of the frame boundary are discarded, as tracking frequently becomes unreliable near an image's edge. Then we enforce a limit on the total number of keypoints. When more than 250 keypoints exist in any frame, we retain only the top 250 having the largest eigenvalues.

As subsequent frames, $I_1, I_2, ..., I_t$, are captured we generate corresponding keypoint sets, $K_1, K_2, ..., K_t$. In most cases K_t is generated by tracking the points from K_{t-1}, as will be described in section 3.3. Over time, however, we lose keypoints either because they drift out of the camera's field of view or due to tracking failures. When fewer than 50 keypoints remain, we add more by repeating the detection process.

3.2 Keypoint Matching Using Walsh-Hadamard Kernel Projections

In order to match a keypoint against a database of known objects, we use a low dimensional descriptor vector for the 32x32 pixel image patch surrounding it. By itself, each image patch effectively represents a 1024-dimensional vector. Matching in this form would be unnecessarily inefficient, as nearest neighbor searches in very high dimensions are computationally intensive. The descriptor thus serves to reduce the dimensionality of the patches, while preserving their most distinctive features.

Our descriptors use Walsh-Hadamard kernel projections as described by Hel-Or and Hel-Or [6]. The WH descriptors take rectangular image patches as input and reduce them to low dimensional vectors, with lower dimensions encoding low frequency information and higher dimensions encoding higher frequency information. Given an image patch $\mathbf{p}$ each element of the kernel projection, $\hat{p}$, is given by the inner product, $\hat{p}_i = \mathbf{u}_i^T \mathbf{p}$, where $\mathbf{u}_i$ is the i^{th} WH kernel.

Before applying the kernel projections, each patch is normalized so that its pixel intensities range from 0 to 255. Two patches that differ by a constant scale or offset will become equivalent after normalization, providing a degree of illumination invariance.

We found, through trial and error, that twenty dimensions are sufficient to retain the most characteristic features of each patch and provide reliable matching results. The first WH kernel simply computes a sum of the patch's intensity values, which contains no discriminative information after normalization. We thus discard the first kernel and build our 20-dimensional descriptor vector using WH kernels u_2 through u_{21}.

Given the 20-dimensional descriptor vector for a keypoint, we find its match in a database of learned keypoints by Euclidean nearest neighbor search. We also apply a distance ratio filter, only accepting those matches where the distance to the nearest neighbor is less than 0.75 of the distance to the second nearest neighbor.

3.3 Bidirectional Optical Flow

The goal of the optical flow process is to take a set of keypoints, K_{t-1}, and use the frames I_{t-1} and I_t to estimate their new locations, producing a new set of keypoints, K_t. We use a variation of the Lucas-Kanade optical flow tracker [8] based on image pyramids [2]. The pyramidal optical flow algorithm takes two images and a previous point location as input and returns a current point location.

$$k_{curr} := OpticalFlow(I_{prev}, I_{curr}, k_{prev}) \qquad (1)$$

The function sometimes returns an error, in which case the point p_{prev} is deemed untrackable and discarded. Even after accounting for these errors, however, the optical flow results may be unreliable. We thus incorporate an additional filter, similar to the one originally reported in [9], referred to as *bidirectional optical flow filtering*.

Bidirectional optical flow first applies the function call in (1) then calls

$$k'_{prev} := OpticalFlow(I_{curr}, I_{prev}, k_{curr}) \qquad (2)$$

with the parameters swapped as if the frames were being captured in reverse. Under ideal circumstances, k'_{prev} will be identical to k_{prev}, and even with image noise and other factors the two points should be very close. So if k'_{prev} and k_{prev} are very different, we know that the optical flow results are unreliable and the point should be discarded. Figure 2 shows an example of bidirectional optical flow filtering.

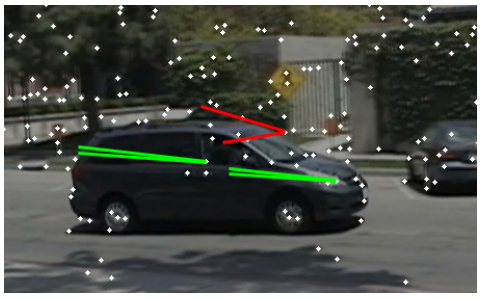

Figure 2. Bidirectional Optical Flow. For three detected keypoints we show the optical flow vector in both directions. For two of those keypoints (green vectors), the vectors match and are deemed valid. For a third, however, they do not match (red vectors) and the point is discarded.

3.4 RANSAC Pose Estimation

Every keypoint record in the database includes the keypoint's location on the object, so that every match generates a correspondence between image coordinates and object coordinates. As keypoints are tracked by optical flow, we maintain their match results, so that the set of correspondences carries over from one frame to the next.

Given a set of correspondences, we can find the object's pose with respect to the camera by least-squares fit. The keypoint matches, however, will generally contain a few gross errors, substantially degrading the pose estimation. To remove these outliers, we use RANSAC. When a sufficiently large set of matches fits a pose hypothesis, we consider the object positively identified and the pose estimation correct. Both recognition and pose estimation are thus successful if and only if RANSAC is successful. Matches that do not fit the estimated pose are assumed to be erroneous and discarded.

4 Simultaneous Tracking and Recognition

Our algorithm combines the components described in section 3 into a unified tracking and recognition system. It first uses the keypoint detector and descriptor to incrementally recognize the object and estimate its pose. Then, all unmatched keypoints are back-projected using the computed pose, generating an additional set of correspondences between image and object. These processes are described in the next two subsections.

4.1 Incremental Keypoint Matching

In the first captured frame we match a fixed small number of detected keypoints (typically 10) against the database. Although every successful match produces a correspondence between image coordinates and object coordinates, the few matches found within a single frame will seldom suffice to positively identify an object or compute its pose. This is especially true considering that some matches will be erroneous.

We track individual keypoints from frame to frame using optical flow, however, so that the set of matches for each visible object continually grows. We apply RANSAC to the largest sets of matches at each frame, attempting to fit a pose estimation.

Along with incremental keypoint matching, we also apply a kind of "incremental RANSAC." Normally, RANSAC iterates over a large number of possible poses, trying to find the one to which the greatest number of keypoints fits. A larger number of iterations means a greater chance of success but also slower performance. We intentionally run very few RANSAC iterations each frame (usually ten to twenty). Those few iterations may not return a successful pose hypothesis, but over the course of several frames RANSAC will eventually succeed. As with keypoint matching, we take the iterations that are traditionally applied at a single frame and spread them over multiple frames.

The advantages of incremental keypoint matching are thus twofold. The first is one of performance. High dimensional nearest neighbor searches are costly, so to match all detected keypoints would put too much processing time in a few initial frames (see the results and discussion of figure 5). By matching points incrementally, we spread the computation evenly over multiple frames and maintain a consistent frame rate. The second advantage is that combining matches from multiple frames produces a more reliable pose estimation. If the object is partially occluded, there may be too few visible key points to produce a pose estimation, even if all of them are tested against the database. Incremental keypoint matching can handle cases where a partial occlusion reveals different parts of the object at different frames.

4.2 Back-Projecting Unmatched Keypoints

One of our algorithm's most powerful features is its incorporation of unmatched features. Keypoints that cannot be matched against the database are matched by back-projecting the current pose estimate. Given any keypoint in the image, we test whether the current pose estimate implies that it should fall on the surface of the object. If it does, we can compute its location in object coordinates and thus generate a match. If it turns out that the keypoint actually does

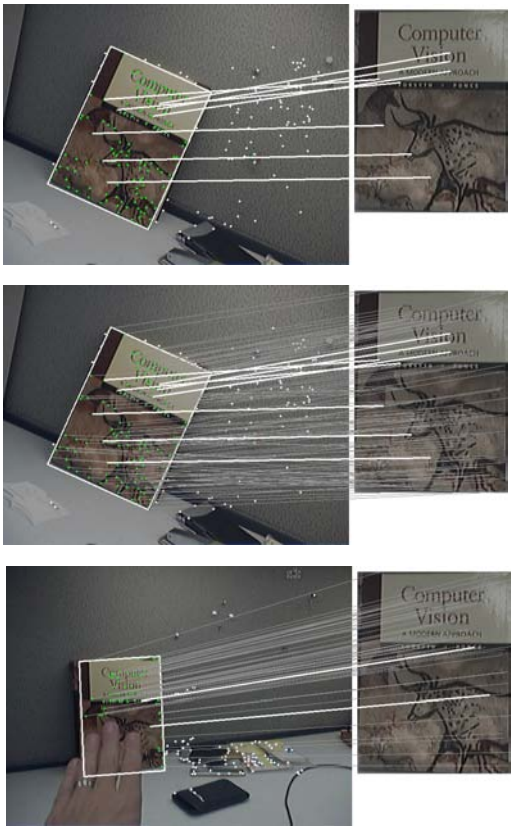

Figure 3. Database matches and projection matches. Thick lines indicate database matches; thin lines indicate projection matches.

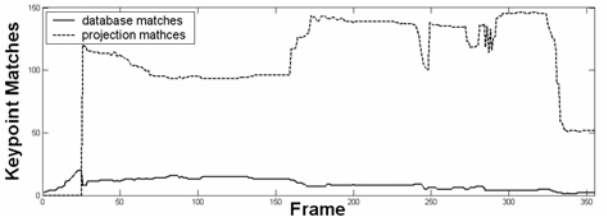

Figure 4. The number of database matches and projection matches over the course of the sequence in figure 3.

not belong to the object, perhaps because it belongs to an occluding object, then it won't fit the pose estimates in future frames and the point can be discarded.

The system thus maintains two sets of matches, those returned from database searches ("database matches") and those found by back projection ("projection matches"). The set of projection matches is usually much larger than the set of database matches, which is, in part, why they are so useful. Even in cases where all of the database matches are hidden, the projection matches offer a reliable pose estimate.

Figure 3 illustrates an example. After the book is recognized, there are eight database matches (thick lines). Using the resulting pose estimate, projection matches are computed for every other keypoint on the cover of the book (thin lines). As the sequence continues, the points are tracked to produce a pose at each frame. During a period of substantial occlusion, the number of database matches falls as low as 2 (too few to compute a pose). Combined with the tracked projection matches, however, the systems still maintains an accurate pose estimation.

Figure 4 plots the number of database matches and projection matches over the entire sequence. With the first successful pose estimation (frame 26), RANSAC outliers are removed, and the number of database matches falls to eight. While the number of database matches fluctuates, there are consistently at least 50 projection matches, enough to maintain an accurate pose.

Algorithm 1 details the complete process applied to each frame for each object. It takes the current image and previous image as input, along with three sets of keypoints - unmatched keypoints, the database-matched keypoints, and the projection-matched keypoints. It will update these three sets and return the object's pose, which will be NULL if RANSAC is unsuccessful. Note that RANSAC outliers are removed from all of the sets, because those keypoints are generally either part of another object or simply do not track well. Also note that once a keypoint is matched as a database match, its object location does not change as long as it remains a RANSAC inlier. This is not the case for projection matches, whose object location is re-computed every frame using the most recent pose estimate.

5 Experimental Setup

Although the basic framework of algorithm 1 could, in principle, apply to any 3D object, our experiments focus on 2D rectangular objects to simplify the training process. Each object can be represented as an image, with points on the object's surface represented by (x, y) coordinates. The transformation from object space to image space is estimated as an affine transformation, which, in most cases, provides a very reasonable pose estimate.

For each of the objects used in our experiments, we build a database of keypoints in an off-line training process. In order to generate descriptors that could be recognized from multiple viewpoints, we transformed each object to simulate a variety of possible views. This is similar to the training process described in [7], except that rather than generating views randomly, we used evenly distributed values of scale, in-plane rotation and out-of-plane rotation. In all, we use five scales, seven in-plane-rotations, and three out-of-plane rotations for a total of 105 total views. Out of those views, we preserve the detected keypoints that appear in at least 25% of the images. A WH kernel projection, as described in section 3.2, is computed for each of those key-

Algorithm 1 Simultaneous Tracking and Recognition

structure Keypoint {
  Point $imgLoc$
  Point $objLoc$
  Time $lastSearched$
  }

K_U: set of unmatched Keypoint
K_D: set of database matched Keypoint
K_P: set of projection matched Keypoint
I, I_{prev}: current and previous frames

procedure ProcessFrame
  /* Update all keypoint positions */
  for $k \in (K_U \cup K_D \cup K_D)$
    $k.imgLoc$:= OpticalFlow($I, I_{prev}, k.imgLoc$)
  end for

  /*Add new features if needed*/
  if $K_U \cup K_D \cup K_P$ too small
    $K_U := K_U \cup$ DetectKeypoints(I)
  endif

  /* Match a small subset of points */
  S := elements of $K_U \cup K_P$
    having oldest $lastSearched$ time
  for $k \in S$
    $k.lastSearched := currentTime$
    Point p := DatabaseMatch(I, k)
    if($p \neq NULL$)
      $k.objectLoc := p$
      move k to K_D
    endif
  end for

  /* Find the new pose, remove outliers, and*/
  /* compute new projection matches */
  $(Inliers, Outliers, ObjPose)$:= Ransac($K_P \cup K_D$)
  if $RanSacSuccess$
    for $k \in Outliers$
      discard k
    end for
    for $k \in K_U \cup K_P$
      $k.objLoc :=$ Project($k.imgLoc, ObjPose$)
      if $k.objLoc$ within object boundary
        move k to K_P
      else
        move k to K_U
      endif
    end for
  endif

points for each of the views on which it appears. Those kernel projections are the descriptors that make up the database for each object.

All experiments used a Pentium D 2.8 GHz machine running Windows XP.

6 Results

Figure 6 illustrates some trained objects along with a video sequence in which they were successfully recognized and tracked. All of the examples involve significant background clutter and, in some cases, substantial occlusion as well.

The truck in sequence (b) becomes completely occluded by a passing bus and then re-recognized. To illustrate the advantage of incremental keypoint matching, we processed the same sequence using a traditional approach. The traditional implementation is identical except that it attempts to match every keypoint from every frame until the object is recognized. Figure 5 shows a frame-by-frame timing comparison. Once the truck has been recognized, the standard approach is slightly faster, requiring, on average, 27.33 ms per frame. Incremental keypoint matching averages 34.75 ms per frame over the same period. As the truck becomes completely occluded, however, the traditional approach performs numerous database searches every frame, dramatically degrading performance. Over this period, it requires an average of 183.89 ms, the slowest frame taking 255.05 ms. The incremental version averages 36.66 ms per frame while the truck was occluded, or about 4.5 times faster.

The important difference between the two approaches is not their absolute processing times but their performance ratio. We use a brute force nearest neighbor search, and could likely improve performance with a more sophisticated search algorithm. Standard matching is slower, however, due to the total number of searches performed, which does not depend on any particular search algorithm.

Because we use an affine transformation instead of a homography, out-of-plane rotations introduce some errors in the pose estimation. To measure the extent of these errors, we created an artificial sequence using the book shown in figure 6(a). The sequence consisted of 200 views of the the object rendered against a black background, with the in-plane and out-of-plane rotations each varying by 50°. We compared the locations of the object corners at each frame to a pre-computed ground truth. The average difference between the estimated locations and ground truth locations was 9.01 pixels with a standard deviation of 5.30. The maximum error for any one corner location was 21.93 pixels

Both plots show periodic spikes in processing time. These correspond to the frames in which the keypoint detector searched for new keypoints. This approximately doubles the processing time for a frame, but occurs less than once every ten frames.

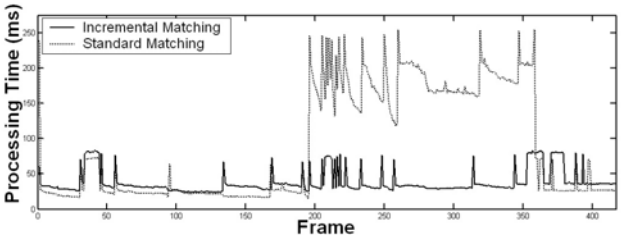

Figure 5. Timing results for the sequence shown in figure 6 (b) using both incremental matching and a traditional approach.

7 Conclusion

In this paper we have demonstrated a valuable technique for recognizing and tracking objects. Our experiments indicate that it performs in real-time, and that it handles difficult cases such as occlusion and varying scales and orientations. The key features of our algorithm are incremental recognition, which allows us to utilize database matches from multiple frames, and the incorporation of unmatched keypoints, which provides added robustness against occlusion and tracking failures.

While our current experiments are limited to 2D objects, future work will attempt to recognize 3D objects as well. While this will involve a more complex training process, the basic algorithm should, in theory, work in its present form.

Acknowledgements

This study was funded by the Center of Excellence for Research and Academic Training on Interactive Smart Oilfield Technologies (CiSoft); CiSoft is a joint University of Southern California-Chevron initiative.

This work made use of Integrated Media Systems Center Shared Facilities supported by the National Science Foundation under Cooperative Agreement No. EEC-9529152. Any Opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect those of the National Science Foundation.

References

[1] S. Avidan. Ensemble tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(2):261–271, 2007.

[2] J. Y. Bouguet. Pyramidal implementation of the Lucas-Kanade feature tracker. OpenCV library http://sourceforge.net/projects/opencvlibrary, 2001.

[3] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–575, 2003.

[4] M. Grabner, H. Grabner, and H. Bischof. Real-time tracking via online boosting. In *Proccedings of the 17th British Machine Vision Conference*, pages 47–56, 2006.

[5] M. Grabner, H. Grabner, and H. Bischof. Learning features for tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2007.

[6] Y. Hel-Or and H. Hel-Or. Real-time pattern matching using projection kernels. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(9):1430–1445, 2005.

[7] V. Lepetit, P. Lagger, and P. Fua. Randomized trees for real-time keypoint recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 775–781, 2005.

[8] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[9] J. Mooser, S. You, and U. Neumann. Real-time object tracking for augmented reality combining graph cuts and optical flow. In *Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 145 – 152, Nov. 2007.

[10] V. Parameswaran, V. Ramesh, and I. Zoghlami. Tunable kernels for tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2179–2186, Jun. 2006.

[11] A. G. A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 666–673, Jun. 2006.

[12] J. Platonov, H. Heibel, P. Meier, and B. Grollmann. A mobile markerless AR system for maintenance and repair. In *Proceedings of the IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 105 – 108, 2006.

[13] X. Ren and J. Malik. Tracking as repeated figure/ground segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2007.

[14] J. Sakagaito and T. Wada. Nearest first traversing graph for simultaneous object tracking and recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2007.

[15] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 593–600, Jun 1994.

[16] X. Song and R. Nevatia. Detection and tracking of moving vehicles in crowded scenes. In *Proceedings of the IEEE Workshop on Motion and Video Computing, 2007*, page 4, Feb. 2007.

[17] Q. Wang and S. You. Real-time image matching based on multiple view kernel projection,. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, Jun. 2007.

[18] G. Welch and G. Bishop. SCAAT: incremental tracking with incomplete information. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 333–344, 1997.

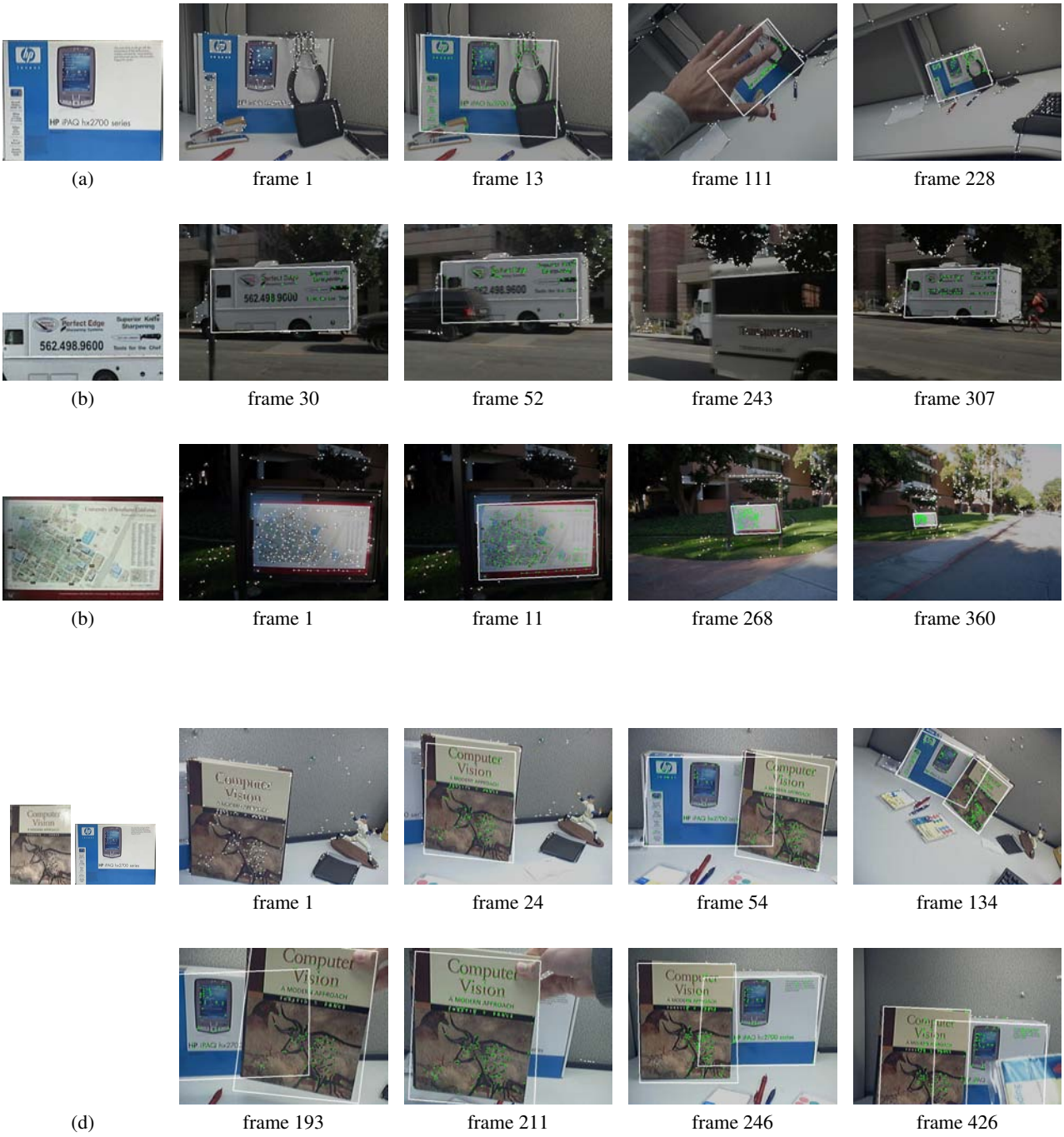

Figure 6. Several examples of simultaneous tracking and recognition. (a) A heavily occluded box in a cluttered setting is recognized after 13 frames. The occluding hand covers all but a few keypoints, but these are enough to continue robust tracking. When the occlusion is removed more keypoints are detected and they, too, are incorporated for tracking. (b) A truck tracked amongst passing cars and other occluding objects. After complete occlusion it takes 64 frames to re-initialize, which at the measured frame rate is about two seconds. (c) An outdoor map recognized within ten frames and tracked over varying distances. By the end of the sequence, although the object is too far to be recognized by keypoint matching, robust tracking is still possible. (d) Tracking multiple objects. Once both the book and the box have been recognized, they can be tracked together through substantial scale and orientation changes. The system tracks the book while we move it to completely occlude the box. Once the box is revealed again, it is re-recognized within about 30 frames.