

Interactive Offline Tracking for Color Objects

Yichen Wei Jian Sun Xiaoou Tang Heung-Yeung Shum
Microsoft Research Asia, Beijing, P.R. China
{yichenw, jiansun, xitang, hshum}@microsoft.com

Abstract

In this paper, we present an interactive offline tracking system for generic color objects. The system achieves 60-100 fps on a 320×240 video. The user can therefore easily refine the tracking result in an interactive way. To fully exploit user input and reduce user interaction, the tracking problem is addressed in a global optimization framework. The optimization is efficiently performed through three steps. First, from user's input we train a fast object detector that locates candidate objects in the video based on proposed features called boosted color bin. Second, we exploit the temporal coherence to generate multiple object trajectories based on a global best-first strategy. Last, an optimal object path is found by dynamic programming.

1. Introduction

Online visual tracking estimates the target state forward in time with no observations in the future. Traditional applications include surveillance and human computer interfaces. Most previous research on tracking have been performed under the online scenario [12, 7, 13]. By contrast, many tracking applications are *offline* - all video frames are available in advance. For example, a YouTube user edits the video before uploading, and a film maker adds special effects into an existing video. Other offline tracking applications include event analysis in surveillance, video annotation, object based video compression, video motion capture.

Perfect tracking is hard to enforce by fully automatic algorithms. When application requires high tracking accuracy, an interactive system that involves user inputs is often necessary. A small amount of user assistance can help a tracker to recover from failure. For example, the user can specify the object state (e.g., location and size) in a few frames, which we call *keyframes*.

The online tracking methods can be directly adopted for interactive tracking in a trial-and-error way: the system tracks the object from the first frame; the user inspects the tracking result until it goes wrong; then the user goes back several frames to correct the object state and restart

the tracker. This strategy has two major problems. First, the tracker may fail frequently due to continuous appearance changes, distractive objects, or background clutter. A lot of user interaction is required and this is unfavorable for an interactive system. Second, an online tracker does not guarantee to connect the tracks between two neighboring keyframes consistently and smoothly, which is however, usually required in applications. Online tracking is not designed to make full use of all available information.

On the contrary, offline tracking is more suitable for the interactive tracking task. Offline tracking [1, 10, 17, 5] is typically formulated as a global optimization problem to find an optimal path in the entire video. All keyframes are taken into account and the smoothness constraint can be explicitly enforced. The tracking result can quickly converge to what the user wants since the offline tracking works like an optimized "interpolation". The user can easily refine the result at any frame and restart the optimization. Good examples include interactive feature tracking [5] and contour tracking [1]. However, these systems are not designed for generic object tracking.

An interactive system should respond to user's input quickly, e.g., a few seconds. The biggest challenge for generic object tracking is the high computational cost required in the global optimization. In this paper, we present an interactive, offline tracking system for generic color objects. It follows the global optimization framework and is able to run at 60-100 fps on a 320×240 video. Experiments on various difficult real videos demonstrate the effectiveness and practicability of our system.

The major contribution of this paper is two-fold. First, we propose a three step optimization framework that addresses offline interactive tracking problem as well as an efficient implementation: 1) fast detection in sparse video frames; 2) multiple object trajectory tracking based on detection; 3) global path optimization based on multiple hypothesis. Second, we propose a novel color feature, called boosted color bin, that is flexible and effective for generic color objects.

1.1. Related work

Recent offline tracking works [1, 17, 5] formulate tracking as a global path optimization problem. Rotoscoping [1] is keyframe-based contour tracking designed for graphics applications. The contour tracking is cast as a spacetime optimization problem that solves for time-varying curve shapes based on input video and user inputs. It is difficult to be applied to other tracking tasks. In [17], 3D curved object trajectory segments are extracted from the video using spectral clustering. Then, the occlusion and final object path are inferred based on these trajectory segments. The trajectory segment extraction and analysis are too slow (e.g., 1 fps) for interactivity. The interactive feature tracking system in [5] tracks a 20×20 image patch. A k-d tree is used to quickly detect the feature and produce multiple hypotheses in every frame. The detection can run at 100-200 fps for a 320×240 video. The patch based representation has limited effectiveness on generic objects that can change pose, viewpoint, scale and aspect ratio. The k-d tree will be very expensive to support the patches with varying size and shape. Before the user interaction, the k-d tree cannot be pre-computed since the size and shape of the object is unknown. Other offline tracking work includes: multiple hypothesis tracking [16], and joint trajectories optimization for multiple objects [10].

Detection has been widely used in tracking, such as SVM tracking [2], relevance vector tracking [19], and subspace tracking [11]. In this paper, we also use a detector to quickly locate the candidate objects and reject a large majority of non-object regions. Our proposed boosted color bins is inspired by recent online feature selection [6, 3, 9]. The works in [6, 3] classify individual pixels by selecting the most discriminative colors using foreground/background variance ratio, or compute a combination of color and local orientation histogram using Adaboosting. On-line boosting [9] trains a small fixed number of selectors instead of a large number of weak classifiers to obtain the real-time performance.

2. System Overview

The system takes the input set of (state, observation) pairs $\mathcal{I} = \{\tilde{\mathbf{x}}_k, \mathbf{y}(\tilde{\mathbf{x}}_k)\}_{k \in \mathcal{K}}$ specified by the user in keyframes and outputs the object states $\mathcal{X} = \{\mathbf{x}_i\}$ in all frames, by minimizing a global cost function:

$$E(\mathcal{X}) = \sum_{i=1}^N d(\mathbf{x}_i, \mathcal{I}) + \lambda \sum_{i=1}^{N-1} s(\mathbf{x}_i, \mathbf{x}_{i+1}), \quad (1)$$

subject to the hard constraints $\mathbf{x}_i = \tilde{\mathbf{x}}_i$ for $i \in \mathcal{K}$. The cost function includes a data term $d(\cdot)$ that measures the evidence of the object state given user inputs, and a smoothness term $s(\cdot)$ that measures the smoothness of object motion.

In this paper, we represent the object as a rectangle $\mathbf{x} = \{c, w, h\}$, where c is the center point, w and h are

the object width and height. The observation is a RGB color histogram $\mathbf{y}(\mathbf{x}) = \{y_1, \dots, y_B\}$ of the object, with B (typically $8 \times 8 \times 8$ in RGB color space) bins. The color histogram has been proven robust to drastic changes of the object, e.g., viewpoint change, pose variation, and non-rigid motion. Based on this observation model, we use the Bhattacharyya distance [7] of histograms as the data term, $d(\mathbf{x}_i, \mathcal{I}) = BD(\mathbf{y}(\mathbf{x}_i), \tilde{\mathbf{y}}) = 1 - \sum_{j=1}^B \sqrt{y_j \cdot \tilde{y}_j}$, where $\tilde{\mathbf{y}}$ can be the closest observation, or linearly interpolated from observations in the keyframes. For the smoothness term, we simply use the Euclidean distance function. Handling of occlusion is discussed in Section 4.3.

The user starts the tracking by specifying the target object in two keyframes, i.e., the first and the last. The system then minimizes (1) and outputs the tracking result. The user then adds or modifies keyframes to correct or refine the result. Therefore, the optimization cost of (1) is critical to make the whole interaction process efficient.

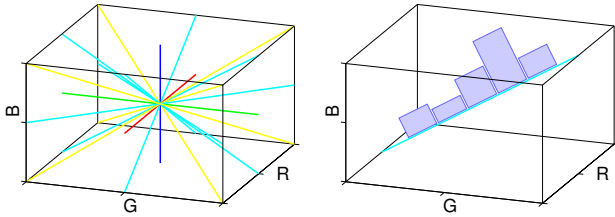
The bottleneck in (1) is the computation of data terms in all frames. Even ignoring scale variations, exhaustive search within the whole state space needs to extract the 3D color histograms and compute the histogram distances for all positions. The complexity in one frame is $O(WH(wh + B))$, where (W, H) are the image dimensions. For a 80×80 object in a 320×240 image, the computation takes about 20 seconds, that is far below our interactive requirement.

Our system perform the optimization efficiently by three steps that consecutively discard useless computation as soon as possible and retain useful information. First, a detector is used to quickly reject a large majority (about 98%) of non-object regions and locate a number of candidate objects in the video. The detector is based on proposed boosted color bin features (Section 4) and trained from the positive and negative examples in the keyframes. Inevitably, the “true” object may be missed in a few frames due to appearance change or partial occlusion.

Second, we exploit temporal coherence of object motion to find the missed object by a trajectory growing algorithm (Section 5). Based on a best-first strategy, the algorithm generates multiple object trajectories from the detected objects and maintains a few best object candidates in each frame. The missed objects have a chance to be tracked from other frames.

Finally, dynamic programming is used to optimize the cost function (1) using the most reliable object candidates, as similar in [5].

Our key observation is that, exhaustive global optimization is too time consuming. Exhaustive detection on all frames is very fast but cannot be made reliable enough. By exploiting the temporal coherence, combining sparse detection and multiple trajectory tracking, we are able to develop an efficient and effective interactive tracking system.



(a) 13 lines in color space (b) a 1D histogram on a line

Figure 1. Color bin feature. It is the bin of a 1D histogram computed on colors projected on a line in RGB color space. A color bin feature represents the percentage of a quantized color within the object. It is a global statistic that is rotation and scale invariant.

3. Fast Detection using Boosted Color Bin

We present a fast detector based on simple color features. It locates the object candidates and rejects most background regions within the video volume.

3.1. Color bin as feature

A lot of commonly used object features depend on spatial configuration of the object, e.g., template, Haar wavelet, oriented histogram. A tracker or detector based on such features could easily fail when the object exhibits drastic changes of spatial structure caused by pose change, partial occlusion, non-rigid motion, etc. These types of problems, however, usually arise in offline video manipulation.

An interactive tracking system should use more flexible feature and avoid such frequent failures which require more user actions. For generic color objects, the color histogram is rotation and scale invariant, and has been shown very robust for object tracking [7, 6, 17]¹. However, as mentioned before, computation of 3D color histograms is too slow. Instead, we propose simple and fast color features - color bins of multiple 1D color histograms (Figure 1).

For a given object window, we project all pixel colors on a set of one dimensional lines in RGB color space. These lines have different directions and pass through the point (128, 128, 128). In our implementation, we evenly sample the direction by 13 lines, as shown in Figure 1 (a). Then, a 1D (normalized) histogram of the projected values is calculated on each line, as shown in Figure 1 (b). We use 8 bins for each histogram through an empirical comparison and treat all $13 \times 8 = 104$ color bins as our features. They are extracted in a short constant time for any image rectangle using integral histogram data structure [15]. In the offline tracking framework, since all integral histograms can be pre-computed, the feature extraction is extremely fast.

¹Of course, if there is a certain spatial configuration of the target object, a histogram considering spatial information will be better [14, 4].

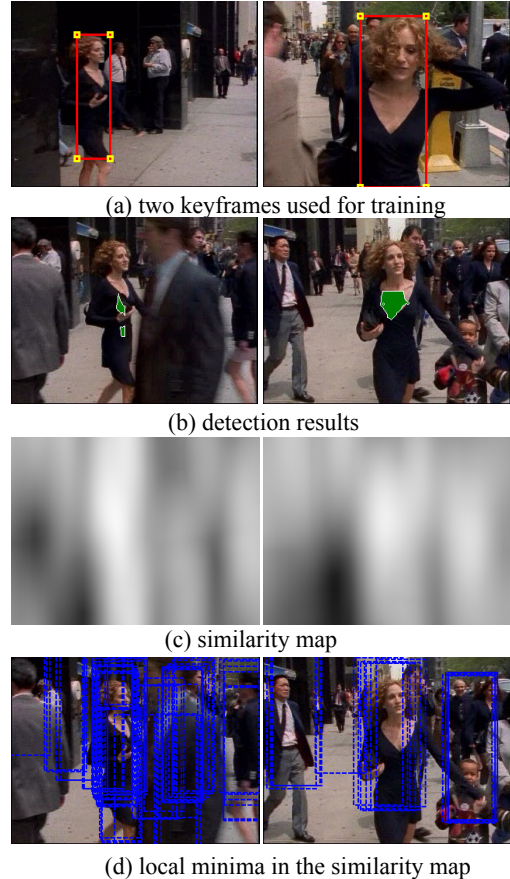


Figure 2. Detection in cluttered background. (a) two user-specified red rectangles in two keyframes. (b) detected object centers (green regions). (c) similarity maps using 3D color histogram distance. Background clutter results in large ambiguous regions. (d) all local minima (blue rectangles) of the similarity maps in (c).

3.2. Boosted color bins

Although a single color bin feature seems weak, the combination of multiple bins can be powerful and highly discriminative. To effectively combine these weak features, we use Adaboosting [8, 18] to train a strong object detector

$$h^{strong}(x) = \text{sign}\left(\sum_i \alpha_i h_i(x) - \beta\right), \quad (2)$$

where $\{\alpha_i\}$ are linear weights and β is a constant threshold. We use the basic stump decision weak classifier

$$h(x) = \begin{cases} 1 & \text{if } sx < s\theta \\ 0 & \text{otherwise} \end{cases}, \quad (3)$$

where x is a single color bin feature, $s \in \{-1, 1\}$ and the threshold θ best separates the training examples.

A weak classifier imposes a constraint on the percentage of certain colors of an object, e.g., the pink color in the object is no less than 30%. The colors of common objects

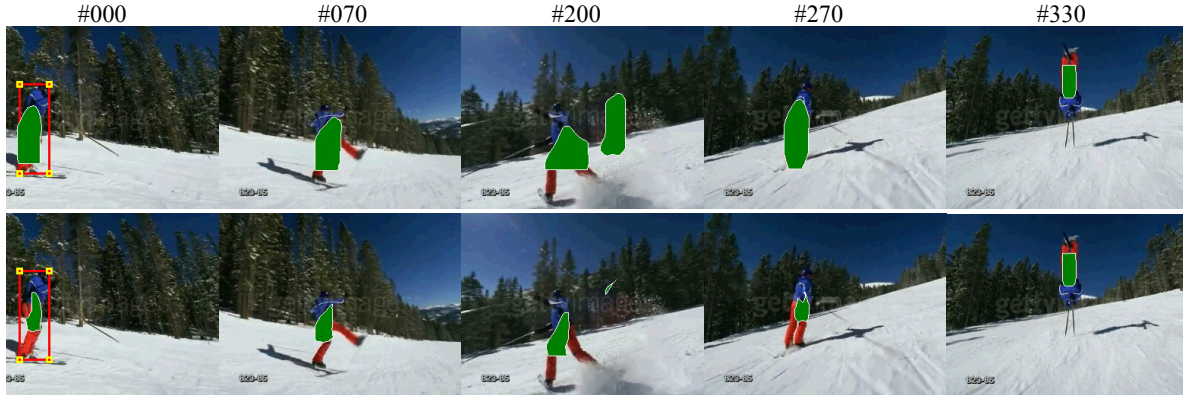


Figure 3. Frame #000 is a keyframe for training. Green regions are detected object centers. Top and bottom rows are results using the first and the first two selected weak classifier(s). The precision is significantly improved and the object is better localized. Very large pose variation in frame #330 is correctly handled.

usually fall in a small number of bins, and they provide the most informative information to identify the object. The Adaboosting algorithm sequentially selects most distinctive color features to separate the object from the background. We call those features “boosted color bins”.

Figure 2 shows detection results in a real video. A detector is trained using examples in two keyframes, as shown in Figure 2 (a). The detected objects in two other frames are shown in Figure 2 (b). In Figure 2 (c), we compute dense similarity maps of Bhattacharyya similarity of 3D color histograms by sweeping the object window in those frames in Figure 2 (b). Due to background clutter, there are large ambiguous regions and many local minima as shown in Figure 2 (d). Even in these difficult cases for a 3D color histogram, the boosted color bins work well.

Figure 3 is an example of non-rigid motion. The top and bottom rows are detection results using the first and the first two weak classifier(s). It could be shown that the first color bin essentially separates the red trousers from all remaining regions, and the second color bin separates the blue clothes. The precision is substantially improved after adding the second feature. This demonstrates the effectiveness of combining multiple color bins. Note that the skier upside down in #330 is also detected. Such large pose variation will be problematic for features dependent on the spatial configuration of the object.

3.3. Training

Training samples are generated from the user specified keyframes. For robustness to possible object variations and to generate more positive samples, we perturb the position and dimensions of the user specified object rectangle by a few pixels. We also scale the object colors by a small amount (factors from 0.8 to 1.2), accounting for possible appearance change. We find this appearance perturbation improves the detector performance obviously. Negative

samples are evenly sampled from the background by only varying the position of the rectangle. On average, we generate about 1000 positive samples and 2000-3000 negative samples per keyframe so that the training can be done in fractions of a second.

Due to very limited training data, robustness issue must be considered. Some color bin features are not stable due to appearance changes, pose variation, or partial occlusion. As mentioned above, we use a coarse discretion (8 bins) to increase robustness. Also, a very small threshold in a weak classifier implies an unstable bin value and we only accept a weak classifier whose threshold is large enough (above 20% of the average bin value). The appropriate number of selected weak classifiers depends on the separability between the object and the background and varies from 2 (for the skier example shown in Figure 3) to 10. In all videos we have tested, using at most 10 color bins are good enough to achieve reasonable training errors. Although sometimes using more features can further decrease the training error, it may cause over-fitting since the background could contain similar distracting clutters. To avoid over-fitting, training is terminated when the detection rate on training data is high enough ($> 98\%$).

4. Temporal Trajectory Growing

The detector cannot guarantee to find the true object in all frames due to limited training data. There could also be false detections. To handle these problems, best object candidates are firstly found in sparse video frames based on the detector. By exploiting the temporal coherence, object trajectories are then grown from the initial candidates within the video volume. During the growing process, more object candidates are found and there are better chances to recover the missed true objects and defeat the false detections.

Input: initial object candidates S , candidate number K
Output: best K object candidates $\{M_i\}$ on all frames

Initialize $\{M_i\}$.
while fetch a best score state \mathbf{x} from S until empty
 for $j = \text{Frame_No}(\mathbf{x}) + 1$ to N (**forward**)
 $\mathbf{x} = \text{MeanShiftTracker}(\mathbf{x})$
 if \mathbf{x} is very close to an existing one in M_j
 break.
 else if $|M_j| < K$
 add \mathbf{x} into the M_j .
 else if the score of $\mathbf{x} <$ the worst score in M_j
 break.
 else replace the worst one in M_j with \mathbf{x} .
 end for
 for $j = \text{Frame_No}(\mathbf{x}) - 1$ to 1 (**backward**)
 ...
end while

Figure 4. Trajectory growing algorithm. *MeanShiftTracker* seeks the nearest mode starting from the initial state \mathbf{x} . *Frame_No*(\mathbf{x}) returns the frame number of the state \mathbf{x} .

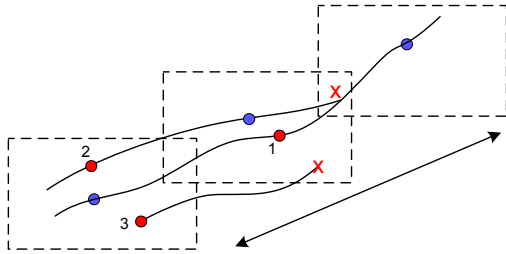


Figure 5. Trajectory growing. The red dots are initial object candidates in the I-frames. The blue dots are the “grown” objects. From candidate 1, a long and smooth trajectory is grown forward and backward. The trajectory from candidate 2 is merged in the middle of trajectory 1 (at the red cross). Candidate 3 is a false detection in background. Its trajectory terminates in the frame which has enough better candidates.

4.1. Extraction of initial object candidates

The video frames are sparsely sampled (every 10 frame in our implementation) and we call the sampled frames *I-frames*. We run the detector on I-frames by sampling the state space (the object location is sampled at an interval of 1/4 of object size, and a few discrete scales are used when necessary). All passing states with too low Bhattacharyya similarity score (< 0.6) are discarded.

The remaining states are then moved to nearby modes (local minima in the state space) by mean shift [7] algorithm, which is very efficient for seeking the nearest mode in a gradient ascent way. After convergence, too close modes are merged to generate a sparse set of initial object candidates.

4.2. Trajectory growing

Based on a best first strategy, our trajectory growing algorithm efficiently maintains K best object candidates on each frame. The algorithm encourages the propagation of better object candidates as soon as possible, therefore unreliable object candidates can be discarded quickly.

The algorithm always grows a trajectory by tracking the currently best object candidate in the initial set forward and backward using mean shift algorithm. The growth is stopped in two cases: the tracked state is very close to an existing one; or there are already K better hypothesis on that frame.

Formally, we denote S as all initial object candidates and M_i as the set of object candidates on *i*th frame. In I-frames, M_i is initialized using S . In other frames, M_i is set as empty. Figure 4 describes the trajectory growing algorithm and Figure 5 is an illustration.

4.3. Global path optimization

Finally, only the best K object candidates in each frame are considered and dynamic programming is used to minimize the cost function (1) in $O(NK^2)$ time. We take K as a small constant so this step is very fast. We follow Buchanan and Fitzgibbon’s approach [5] to handle occlusion via adding two state transition costs into the smoothness term in (1):

$$s'(\mathbf{x}_i, \mathbf{x}_{i+1}) = \begin{cases} s(\mathbf{x}_i, \mathbf{x}_{i+1}) & \mathbf{x}_i \text{ and } \mathbf{x}_{i+1} \text{ are visible} \\ \lambda_r/\lambda & \mathbf{x}_i \text{ and } \mathbf{x}_{i+1} \text{ are occluded} \\ \lambda_o/\lambda & \text{otherwise} \end{cases},$$

where λ_o and λ_r are penalties paid for an object entering/exiting the occlusion state and retaining the occlusion state. We set $\lambda_r = 0.4\lambda_o$ as suggested in [5], leaving only λ_o adjustable for the user. The “Girl” example in Figure 6 contains multiple occlusions. After two parameter adjustments (final $\lambda_o = 0.18$), we obtain good result using only two keyframes.

5. Experimental Results

In all tracking results (Figure 6, 7, 8, 9), red rectangles represent user specified objects in keyframes, yellow solid rectangles represent finally optimized object states, and blue dashed rectangles represents the final best K object candidates.

Parameter setting and user interaction. Most parameters are fixed in all experiments. The candidate number K is 10 by default. Two parameters are adjustable for user interaction: $\lambda \in [0.001, 0.02]$ for the smoothness term, $\lambda_o \in [0.1, 0.5]$ for occlusion. Since the global path optimization is very fast ($> 5,000$ fps), the user can interactively adjust these two parameters to tune the final track path.

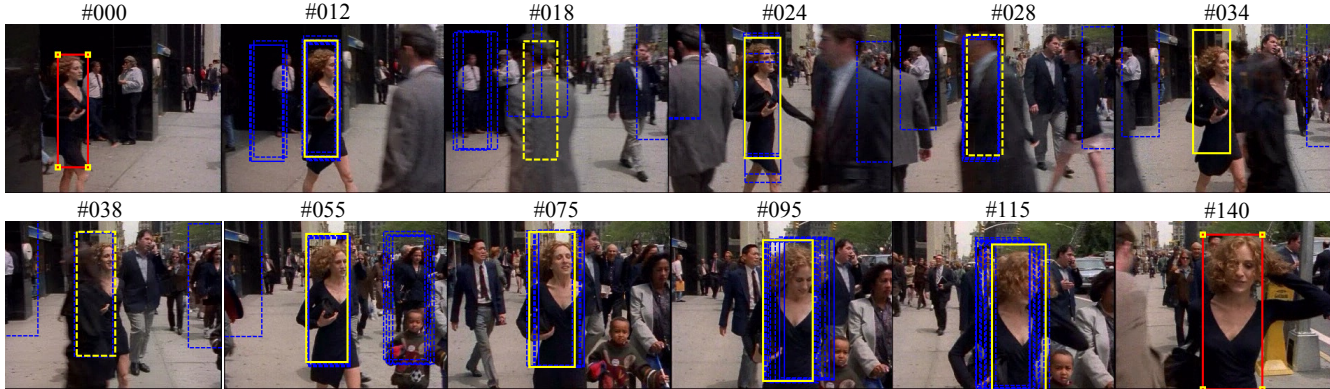


Figure 6. “Girl” example. Two red rectangles are specified by the user. The yellow rectangles are the final tracked object and the dashed blue rectangles are the local mode after trajectory growing. The optimization with occlusion handling correctly labels occlusions around the frames #018, #028, and #038. The dashed yellow rectangles are interpolated using the results entering/exiting the occlusion.

	frames/key	object size	grow(s)	all(s)
Girl	141/2	85 × 220	2.3	2.7
Boy	864/3	33 × 129	7.5	8.7
Skiers	250/2	19 × 53	1.5	1.9
Man	284/5	108 × 197	4.5	5.2

Table 1. Timing. “key.” is the number of keyframes. “grow” is the time for trajectory growing. “all” is the overall time.

The user interactions are invoked in two cases: 1) the correct object is found but not on the final path. This is typically caused by background clutter. The user selects the found object as a hard constraint and the path optimization runs again; 2) the object is not found or not accurately located. The user adds a keyframe where the error is largest, and the whole algorithm restarts.

By default, object scale is linearly interpolated from two neighboring keyframes. However, the user can enable a multi-scale search in the detection stage (5 evenly divided scales between two keyframes are used) if the object scale change cannot be approximated linearly. The “Boy” example in Figure 7 uses multi-scale detection. We set $K = 20$ to increase the chance that the correct scale is found.

Pre-computation Integral histogram is very fast for feature extraction but memory consuming. For a 320×240 image, storing all 13 one dimensional integral histograms with 8 bins costs about 30MB. We pre-compute the integral histograms in I-frames and store them on disk. Fortunately, we do not need to load all of them into memory. In the training stage, only integral histograms for several keyframes are loaded. In the detection stage, only integral images corresponding to the selected color bins (no more than 10) in the I-frames are loaded. For a 320×240 image, the memory cost is 0.7MB/I-frame.

	rejection rate	recall	precision
Girl	92.5%	82.6%	82.7%
Boy	99.4%	95.1%	71.2%
Man	94.3%	95.7%	62.3%
Skiers	98.9%	47.1%	18.8%

Table 2. Detection evaluations.

Timing The algorithm consists of four steps: training, detection, trajectory growing, and path optimization. Both training and path optimization takes less than one second. The detector runs at 2000 ~ 4000 fps (200 ~ 400 fps on I-frames), dependent on the number of boosted color bins.

The trajectory growing is the most expensive part due to the computation of 3D color histograms in the mean shift tracking algorithm. Its running time is proportional to the object size and the maximum candidate number K . For a 80×80 object and $K = 10$, our system can achieve 60-100 fps. For a larger size object, a multi-scale approach is adopted. We first obtain an object path in the low resolution video and then refine the path by mean-shift mode seeking in the original resolution video. The timings of all examples in the paper are summarized in Table 1.

Detection evaluation We evaluate our detector on video frames (every 10) with manually labeled ground truth. A detected rectangle is a true positive if it overlaps with the ground truth by more than 50%. Table 2 reports rejection rates, precision, and recall for all examples. The rejection rate is the percentage of rejected states. It indicates the saved time with respect to an exhaustive search. The precision is the percentage of correctly detected objects among all detected objects. The recall is the percentage of frames in which the true object is detected.

As observed, all examples have very high rejection rates, which is the key to the interactive system. The example

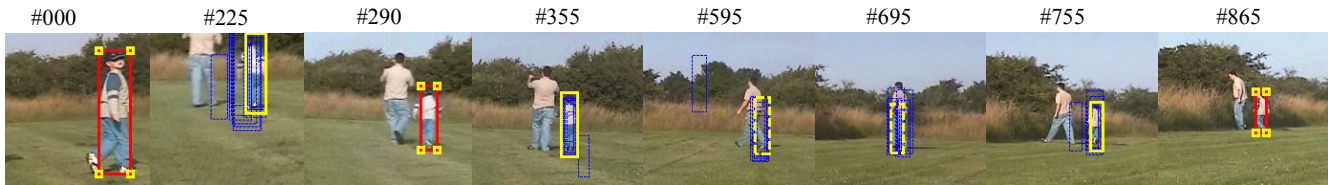


Figure 7. “Boy” example. The scale change of the boy is large and nonlinear. Using five scale levels and three keyframes, our approach produces good tracking result, with occlusion in #595 and #695 correctly handled.

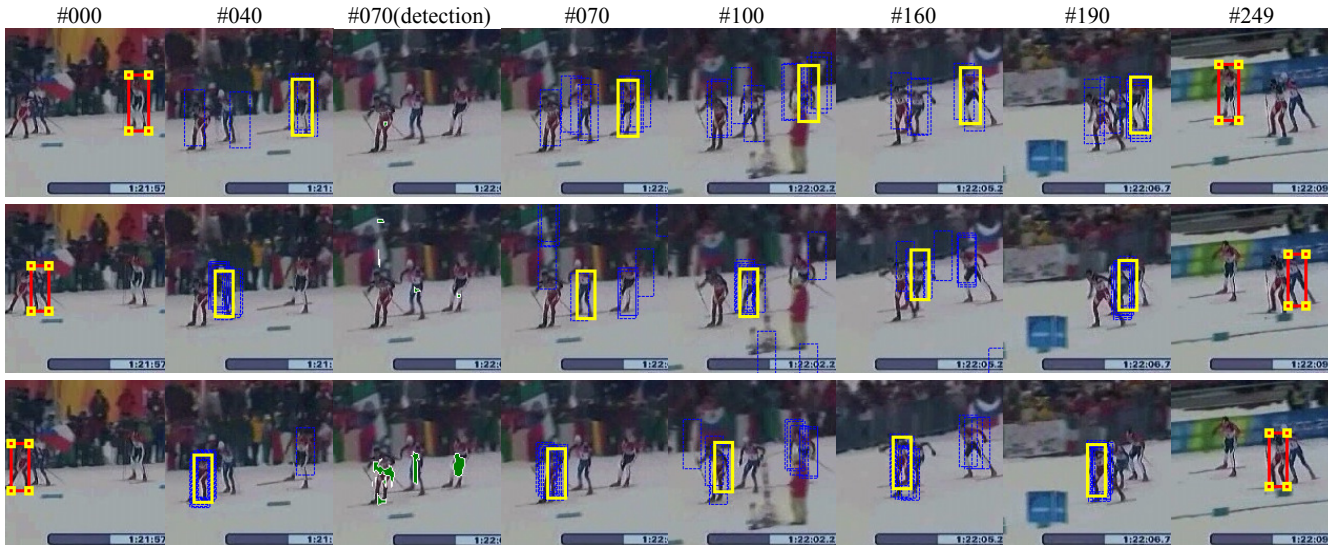


Figure 8. “Skiers” example. We independently track three skiers in each row. The first and last frames are keyframes. Three skiers with similar appearances move and occlude each other, and the background contains many similar colors and patterns. Although our detector outputs many false positives and the true object is not detected in some frames, e.g., frame #070 in the third column (green regions are detected object centers), the trajectory growing and global optimization successfully track the skiers.

“Skiers” shown in Figure 3 is difficult due to similar objects and background clutter. In fact, we have trained a detector using all frames with ground truth. The detector’s precision is only 35% (of course, the recall is 100%). This indicates the intrinsic difficulty of this example. Although only half of the true objects are detected, our system works well thanks to the trajectory growing which recovers missed true objects from other correctly detected ones.

Comparison In the “Man” example in Figure 9, the appearance of the target object (the man with a white suit) changes dramatically due to lighting and pose variations. There are also objects (woman and stones) with similar color in the background. We compare our interactive tracker and an online mean-shift tracker [7] on this difficult example.

The online mean shift tracker quickly drifts after tracking a few frames. After a large drift is observed, we step back a few frames, correct the result, and restart the tracker. The top row of Figure 9 shows several frames with large drifts during tracking. Even when we restart the tracker more than ten times, the tracking results are still jaggy, because the online tracker cannot guarantee a smooth track transition

between neighboring keyframes.

In middle and bottom rows of Figure 9, we show our offline tracking results using 3 and 5 keyframes. Usually, the keyframe is added where the error is largest. As we can see, adding a small number of keyframes results in substantial improvements as a result of the global optimization. The results are consistently tight and smooth. The whole process takes about one minute. To achieve the same quality results using the online tracker, we need to restart the online tracker 20-30 times, which takes about five minutes.

We have also applied bi-directional tracking [17] on this example. Using 5 keyframes, bi-directional tracking can produce a result of the same quality. But the trajectory segment extraction and analysis requires many 3D histogram extractions and spectral clustering, which are computationally intensive - about 0.5 fps in our implementation. The interactive feature tracking [5] cannot be directly applied since we are not able to pre-compute the k-d tree before the user interaction. One can imagine using multiple small fixed size patches to represent an object. However, this representation is sensitive to the change of appearance structures and slows down the detection by several times.

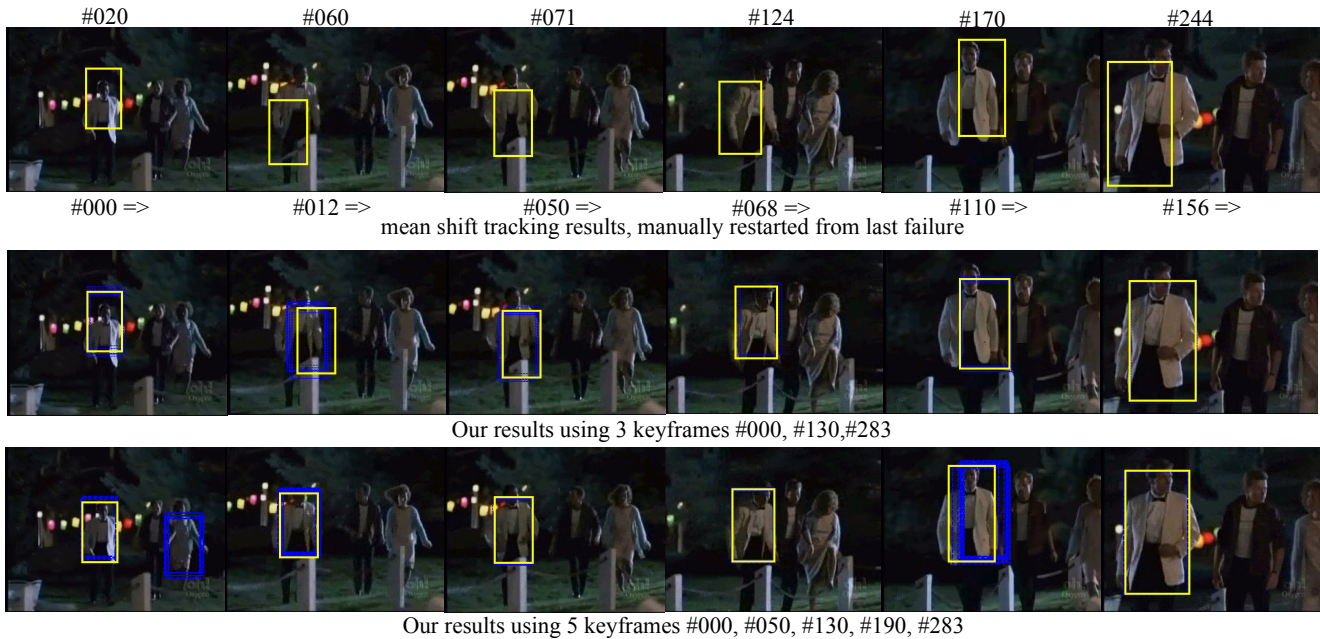


Figure 9. “Man” example. The top row is the intermediate tracking results by an online mean shift tracker. To successfully track the man with the white suit, the user has to restart the tracker when the tracking drift is large. Here we show several drifted results after restarting the tracker. For example, the 4th column is the drifted result in frame #124 using the restarted tracker from the frame #068. The user needs to go back a number of frames to restart the tracker when a large drift is observed. This video requires more than ten restarts and the final results are still jaggy. The middle and bottom rows are our tracking results using 3 and 5 keyframes. The offline tracking produces better results and requires a smaller amount of user effort.

6. Conclusion

We have presented an interactive offline tracking system for generic color objects. The tracking is performed via an efficient global optimization framework. The system can obtain high quality results for difficult videos in a short time and is useful for offline tracking applications.

We focused on color feature in this paper. It is interesting to investigate whether combining other features, such as Haar wavelet or oriented histograms, can improve the detector performance. Another future work is interactive multiple object tracking.

References

- [1] A. Agarwala, A. Hertzmann, D. Salesin, and S. Seitz. Keyframe-based tracking for rotoscoping and animation. In *SIGGRAPH*, 2004.
- [2] S. Avidan. Support vector tracking. In *CVPR*, 2001.
- [3] S. Avidan. Ensemble tracking. In *CVPR*, 2005.
- [4] S. T. Birchfield and S. Rangarajan. Spatiograms versus histograms for region-based tracking. In *CVPR*, 2005.
- [5] A. M. Buchanan and A. W. Fitzgibbon. Interactive feature tracking using k-d trees and dynamic programming. In *CVPR*, 2006.
- [6] R. T. Collins and Y. X. Liu. On-line selection of discriminative tracking features. In *ICCV*, 2003.
- [7] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000.
- [8] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *In Computational Learning Theory: Eurocolt 95*, 1995.
- [9] H. Grabner and H. Bischof. On-line boosting and vision. In *CVPR*, 2006.
- [10] M. Han, W. Xu, H. Tao, and Y. H. Gong. An algorithm for multiple object trajectory tracking. In *CVPR*, 2004.
- [11] J. Ho, L. K., M. Yang, and D. Kriegman. Visual tracking using learned linear subspaces. In *CVPR*, 2004.
- [12] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *ECCV*, 1996.
- [13] A. D. Jepson, D. J. Fleet, and T. El-Maraghi. Robust, on-line appearance models for vision tracking. In *CVPR*, 2001.
- [14] P. Pérez, C. Hue, J. Vermaak, and M. Gangnet. Color-based probabilistic tracking. In *ECCV*, 2002.
- [15] F. Porikli. Integral histogram: A fast way to extract histograms in cartesian spaces. In *CVPR*, 2005.
- [16] D. Reid. An algorithm for tracking multiple targets. *IEEE Tran. on Automatic Control*, 24(6):843–854, 1979.
- [17] J. Sun, W. Zhang, X. Tang, and H.-Y. Shum. Bi-directional tracking using trajectory segment analysis. In *ICCV*, 2005.
- [18] P. Viola and M. Jones. Robust real-time face detection. *IJCV*, 57(2):137154, 2004.
- [19] O. Williams, A. Blake, and R. Cipolla. A sparse probabilistic learning algorithm for real-time tracking. In *ICCV*, 2003.