Branch and Track

Steve Gu
Duke University
Durham, NC 27708, USA
steve@cs.duke.edu

Carlo Tomasi
Duke University
Durham, NC 27705, USA
tomasi@cs.duke.edu

Abstract

We present a new paradigm for tracking objects in video in the presence of other similar objects. This branch-and-track paradigm is also useful in the absence of motion, for the discovery of repetitive patterns in images. The object of interest is the lead *object and the distracters are* extras*. The lead tracker branches out trackers for extras when they are detected, and all trackers share a common set of features. Sometimes, extras are tracked because they are of interest in their own right. In other cases, and perhaps more importantly, tracking extras makes tracking the lead nimbler and more robust, both because shared features provide a richer object model, and because tracking extras accounts for sources of confusion explicitly. Sharing features also makes joint tracking less expensive, and coordinating tracking across lead and extras allows optimizing window positions jointly rather than separately, for better results. The joint tracking of both lead and extras can be solved optimally by dynamic programming and branching is quickly determined by efficient subwindow search. Matlab experiments show near real time performance at 5-30 frames per second on a single-core laptop for 240 by 320 images.*

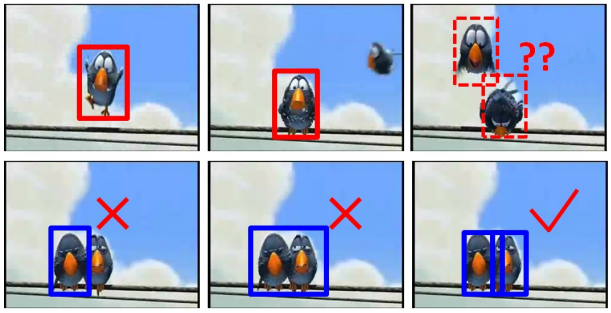

Figure 1. Top row: from left to right are the 1^{st}, 100^{th} and 150^{th} frame of a video sequence. The target to be tracked is bounded by a red rectangle in the first frame. Bottom row: Three possible tracking outcomes for the 200^{st} frame. The appearance change of the birds are dramatic and the confusion is most significant in the 150^{th} frame: The second bird resembles the first bird in frame 100 more than the first bird in frame 150 resembles its own image in frame 100. As a result of this confusion, a typical tracker can easily follow the wrong target (bottom left), or encompass both birds (bottom center) if the window is allowed to change shape. In contrast, our tracker avoids this confusion (bottom right) and tracks each bird correctly. In addition, each tracker adapts better to change, because trackers share features. (All demo videos used in this paper are downloaded from YouTube.)

1. Introduction

The important problem of tracking objects in video presents many challenges. The object itself may change in appearance and configuration; illumination and viewpoint may vary over time; occlusions may hide the object completely or in part; the background may be cluttered with other items similar to the object being tracked.

The rich literature aimed at these challenges describes a wealth of tracking methods. A majority of these track rectangular windows around the object of interest. Conceptually, the key step in these methods is the search of the one window whose appearance is most similar to that of a model derived from the object window in the previous frame:

$$\hat{W}_k = \arg\min_{W_k \in \mathcal{R}} \mu\left(W_k; W_{k-1}, I_k, \mathcal{F}_{k-1}\right) \qquad (1)$$

In this equation, W_k is the window geometry (its position, size, aspect ratio,...) in the k^{th} frame I_k, and $\mathcal{F}_{k-1}$ is the appearance model computed from frames up to I_{k-1}. The set $\mathcal{R}$ is the space of all windows to be searched in I_k, and μ is the matching cost that measures dissimilarity of appearance. For instance, in the classical Lucas-Kanade tracker [17, 21], $\mathcal{F}_{k-1}$ collects the pixel values in W_{k-1} and μ is the Sum of Squared Differences (SSD) measure. In a more recent example [7], $\mathcal{F}_{k-1}$ is a bag of SIFT feature descriptors in W_{k-1} and μ accounts for both matching costs and a measure of geometric closeness between W_k and W_{k-1}.

We depart from this tradition in order to address a particularly insidious type of background clutter, that is, the presence of additional objects that are similar to the one being tracked. Borrowing motion picture terminology, we call the

main object of interest the *lead* object, and the other similar, distracting objects the *extras*. For instance, a person in a crowd is selected as the lead – perhaps by a user or by an automatic person detector – and the other people around the lead are the extras. Similarly, we can think of a lead car in a traffic of extras, or a lead fish in a school of extras.

We contend that the best way to track the lead is to also track the extras, for several reasons. First, in some cases we are interested in all objects, such as when we want to determine traffic patterns or flow to popular destinations. Second, tracking extras makes each tracker aware of all the others, even if the lead is the only object of interest. Tracking the extras shows how similar they are to the lead, and this lets the lead tracker determine more appropriate thresholds of similarity. It also informs each tracker of where all similar objects are, so that the choice of motions that fit all objects can be made collectively and globally optimal for all, rather than greedily for each single object. This particularly important point is illustrated in Figure 1.

A third reason for tracking both lead and extras is that trackers of similar objects can share appearance models. If one tracker observes a person from many viewpoints, another person tracker can use the richer appearance model from the first tracker to adapt more nimbly and flexibly to changes in pose of its own target.

Conceptually, the fact that there is typically a lead object suggests a *branch and track* paradigm. The lead object is described first, and an automatic process then determines whether the tracker needs to *branch out*, that is, to spawn a new tracker that has initially the same appearance descriptor as the lead tracker but for a different image window.

This paradigm is surprisingly useful. First, literal branching – one object that becomes many over time – is pervasive in nature. Cells divide into multiple copies (Figure 2); bacteria multiply (*ibidem*); clouds shred into cloudlets in the wind; rivers branch into multiple streams around obstacles; plant twigs spawn more twigs.

Another source of branching is just visual and relates to image projection, with its tendency to occlude objects from view and then reveal them again. For instance, a queue of people can appear as a single person if the camera is facing the person in front. As the camera travels around the queue, more people come into view.

The branching from lead to extras need not occur over time, but can be merely conceptual, and arise even when motion is minimal or nonexistent. For instance, a user or an automatic detector selects a single flower, and the "tracker" branches out to find other flowers in the meadow (Figure 3).

In summary, the branch and track paradigm addresses in a unified way at least three seemingly different tasks: tracking genuine branching processes (Figure 2), tracking a lead object surrounded by extras, and filling in a region with repetitive patterns and identifying similar objects.

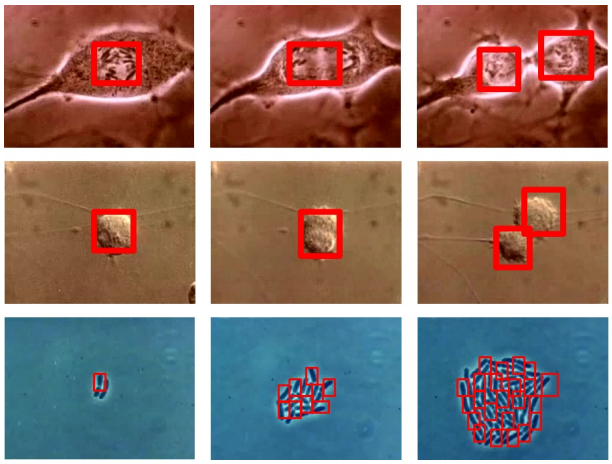

Figure 2. Cell division (top and middle row) and bacterial growth (bottom row). Red rectangles are the targets to be tracked. Our tracker handles these natural branching processes well.

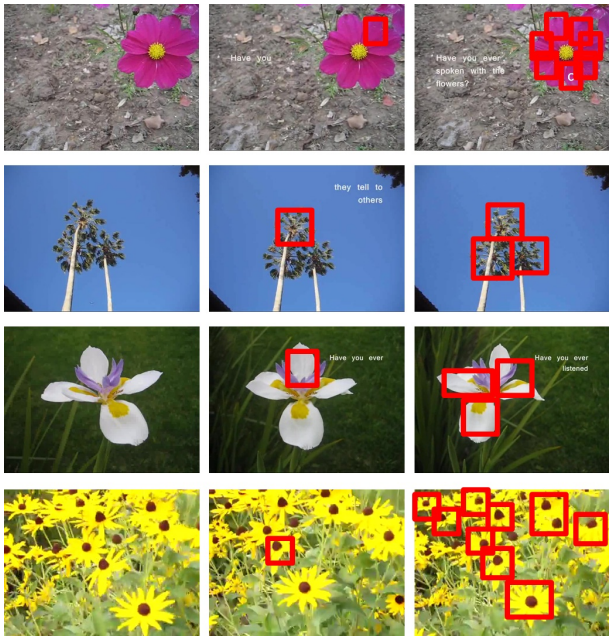

Figure 3. Each row exhibits a video sequence and the left column shows a sample frame from each video. The initial window is placed on the frame shown in the middle column and similar objects are found and tracked in successive frames – or, if desired, in the same frame, fed repeatedly to the algorithm. The shape of the windows can vary, and the filling process takes from several frames to several hundred frames to finish (right column), depending on the extent of appearance change.

The rest of the paper is organized as follows: Section 2 compares our work in previous literature, and clarifies in particular the key differences from methods for multiple object tracking. The problem formulation and algorithms are

presented in Section 3. Experimental results are presented in Section 4 and Section 5 concludes.

2. Previous Work

The state of the art of visual tracking has advanced significantly in the past 30 years [17, 21, 9, 3, 22, 1]. On one hand, feature detectors and descriptors [21, 16, 4, 20, 6] have been developed for capturing visual appearance in an invariant way. On the other hand, online learning algorithms such as Random Forests [2], Boosting [23], Multiple Instance Learning [1] and Nearest Neighbor [7] have been proposed to enhance the trackers' adaptivity to changes of illumination, scale, cluttered background and occlusion.

It is important to distinguish our work from that on multi-object tracking [14, 19, 10, 12, 8, 18, 13], which has focused mostly on vehicles and pedestrians. While people or vehicles do appear as split or merged in the videos considered in this type of work [18], these methods are mainly concerned with trajectory analysis and either data association or identity management [14, 12]. In these methods, each tracker has a separate state. The goal is then to maintain the trajectories of individual trackers and avoid confusion between different people or vehicles.

While our work of course benefits from these insights, our trackers share appearance models and coordinate their processes of window selection, primarily for tracking a single window in a more reliable way! The introduction of a branching process also naturally defines a dependency relation among the lead and the extras and between extras and extras. We can use this dependency to optimize the tracking result in a global way and improve data association by asking for an optimal configuration across trackers, similar to the matching of pictorial structures [5].

Our contributions First, to the best of our knowledge, ours is the first work to instantiate model sharing through an explicit branching process with the goal of eliminating target confusion and to enrich models by feature sharing. Second, we exploit the branching relation for both lead and extras to jointly track multiple windows. The branching process uses Efficient Subwindow Search (ESS)[11] to handle scale change and a coordinated, structural configuration is tracked efficiently by dynamic programming [5].

3. Branch and Track

For ease of exposition, we start with the the description of our basic tracking module for a single window. We then describe the branching conditions and the notion of branching tree, and show how to use branching trees to cast the tracking of both lead and extras as a global optimization. We describe a feature sharing mechanism that helps individual trackers adapt to appearance change. We also give an efficient algorithm to do the branch and track.

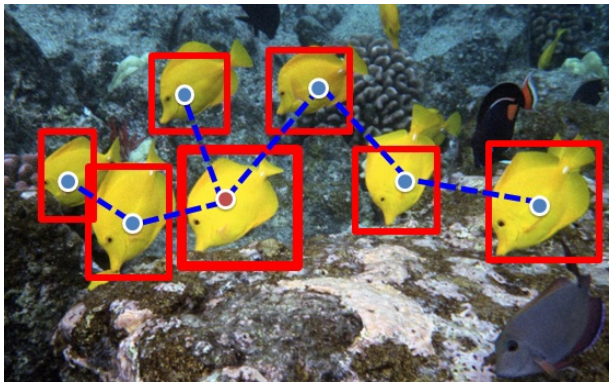

Figure 4. A school of yellow tang. The fish with the red dots and the maximal rectangle is the lead and the rest are the extras. The blue dashed lines among the trackers are the arcs in the branching tree and encode pairwise spatial constraints.

3.1. The Single Tracking Model

Although many tracking methods can form a basis for branch and track, we use the nearest-neighbor based tracker in [7] for its simplicity and straightforward implementation. The tracker accumulates object and background models over frames and each model is composed of SIFT features associated to the object and background respectively. The equation for tracking a single optimal window $\hat{W}_k$ is:

$$\hat{W}_k = \arg\min_{W_k \in \mathcal{R}} \left\{ \underbrace{E(W_k)}_{\text{visual inconsistency}} + \lambda \underbrace{\kappa(W_k, W_{k-1})}_{\text{motion discontinuity}} \right\} \tag{2}$$

The term $E(W_k)$ sums the cost of each feature within that window and the cost of a feature is based on its relative closeness to both object model and background model. $\kappa(W_k, W_{k-1})$ penalizes window geometry change including window position, width, height and aspect ratio variation between W_k and W_{k-1}. The regularization parameter λ balances visual inconsistency and motion discontinuity. $\mathcal{R}$ contains all possible rectangular shapes within the image domain. In [7], searching for the optimal window is fast thanks to the efficient subwindow search method [11].

3.2. Branching Conditions

How should one decide if the current tracker is to spawn another tracker for extra targets? Our basic idea is as follows: For each frame and each individual tracker, we first determine a single, optimal window using Equation (2). Then, recursively, we look for a second-best window in the rest of the search space, if there is any that is similar enough to the best window and – to avoid branching to the parent itself – in a different enough position in the image. In

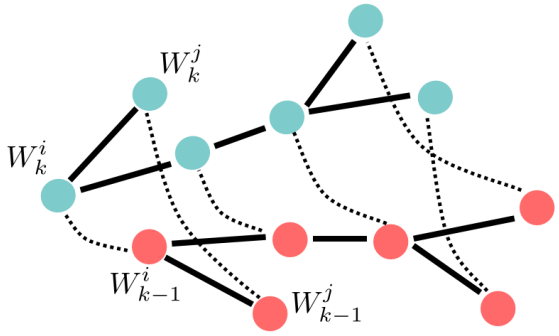

Figure 5. Matching of two branching trees in consecutive frames.

addition, branching is discouraged if the appearance of the current parent window has changed dramatically since its inception. This last condition is introduced to prevent trivial branching, in cases where the current parent window has lost most of its features as a result of occlusion or dramatic changes of appearance. Thus, a parent tracker can spawn a child tracker if the parent has

a) small overlap with the window of the child tracker;

b) sufficient similarity to the child window;

c) sufficient similarity to its own appearance at inception.

These conditions can be formalized as follows:

Definition 1 (Branching condition) *Let W_k^i be the window of the i^{th} tracker at frame k and let $\alpha(i)$ be the index of the first frame in which tracker i is born. A test window W_k^t is spawned from W_k^i if:*

$$\frac{1}{\zeta}E(W_{\alpha(i)}^i) \leq E(W_k^i) \leq \zeta E(W_k^t \setminus W_k^i) \tag{3}$$

where $\zeta > 1$ controls the stability of the branching process and $E(W_k^t \setminus W_k^i)$ evaluates the visual consistency of window W_k^t excluding the content belonging to W_k^i. This exclusion copes with condition a). The first inequality in Equation (3) addresses condition c), and uses the evaluation score of the appearance of the i^{th} tracker at its inception to decide if the tracker in its current state is sufficiently reliable. The second inequality complies with condition b) by comparing the similarity of the child and parent window.

3.3. Branching Tree and Joint Tracking

The fact that a tracker is branched from its parent tracker naturally defines a dependency relation:

Definition 2 *The branching tree associated to the k^{th} image I_k is a directed tree $\mathcal{T}_k = \langle \mathcal{V}_k, \mathcal{E}_k \rangle$ with $\mathcal{V}_k$ the set of trackers and $\mathcal{E}_k$ the arc set encoding branching relations.*

In other words, $(W_k^i, W_k^j) \in \mathcal{E}_k$ implies that the tracker W_k^j was branched out from tracker W_k^i at some frame before the k^{th} frame. Figure 4 shows the tree among a school of fishes. Note that the branching tree is a dynamic tree with structure updated online frame by frame. The joint tracking is formulated as finding the optimal tree:

$$\hat{\mathcal{T}}_k = \arg\min_{\mathcal{T}_k} \left\{ \underbrace{E(\mathcal{T}_k)}_{\text{visual inconsistency}} + \lambda \underbrace{\kappa(\mathcal{T}_k, \mathcal{T}_{k-1})}_{\text{structural difference}} \right\} \tag{4}$$

The formulation is clearly a generalization of Equation (2). In particular, we express $E(\mathcal{T}_k)$ as the summation of the visual inconsistency of each tracker belonging to $\mathcal{V}_k$:

$$E(\mathcal{T}_k) = \sum_{i=1}^{|\mathcal{V}_{k-1}|} E(W_k^i) \tag{5}$$

The structural difference term $\kappa(\mathcal{T}_k, \mathcal{T}_{k-1})$ compares all corresponding pairs of windows of $\mathcal{T}_k$ and $\mathcal{T}_{k-1}$ in terms of their spatial geometry. Let $e_{k-1}^{i,j} = (W_{k-1}^i, W_{k-1}^j)$ be a window pair in $\mathcal{E}_{k-1}$ and its corresponding candidate window pair at frame k be (W_k^i, W_k^j) (Figure 5). We specify:

$$\kappa(\mathcal{T}_k, \mathcal{T}_{k-1}) = \sum_{e_{k-1}^{i,j} \in \mathcal{E}_{k-1}} \|(W_k^i - W_k^j) - (W_{k-1}^i - W_{k-1}^j)\|^2 \tag{6}$$

We represent W_k^i as a two dimensional vector that encodes the spatial (x, y) coordinate of the center of the window, and $\|u\|^2$ is the squared Euclidean norm of the two dimensional vector u. Note that the size of each window is assumed to be fixed for all the frames after its birth. But different trackers can have different window sizes depending on how each window is initialized in branching.

3.4. The Algorithm

We give a practically efficient algorithm for branch and track. The general framework is fairly simple. The input is the branching tree $\mathcal{T}_{k-1}$ in the previous frame, the accumulated object and background models, and the current frame I_k. The output is the updated tree in the current frame, together with the updated object and background models.

Step.1: Process image I_k and evaluate $E(W_k^i)$ for each possible window location and tracker index i. Then compute the optimal tree $\hat{\mathcal{T}}_k$ using Equation (4);

Step.2: Apply branching tests of Equation (3) to each node tracker of $\hat{\mathcal{T}}_k$ and recursively spawn new trackers using Equation (2) if they pass the branching tests;

Step.3: Update the child-parent relations of $\hat{\mathcal{T}}_k$ and update both object and background models accordingly. Set the frame counter $k = k + 1$.

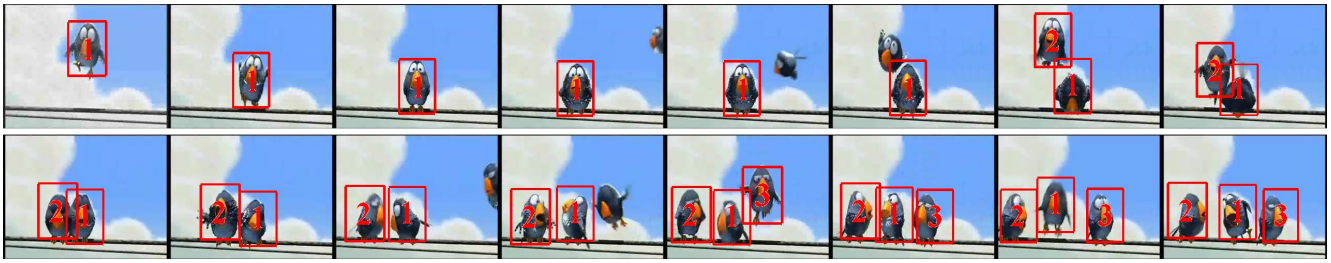

Figure 6. Branch and track on a video sequence where birds with similar appearance appear one after another. Note that the appearance change of the birds is sometimes large and the background clutter caused by the extras makes a single tracker difficult to follow the target correctly. Our framework handles this case well and each bird gets tracked correctly in all the 500 frames.

We analyze the complexity of each step. In Step.1, after SIFT detection, evaluating the consistency for each possible window position is fast using the integral image representation. The optimal tree configuration by Equation (4) can be found efficiently by dynamic programming, similar to the matching of pictorial structures [5].

The branching process in Step.2 uses the Efficient Subwidnow Search method [11], which in practice reduces the typical running time for searching for an optimal window drastically to sub-linear time with respect to the image size, instead of the otherwise quadratic complexity. Of course the actual running time depends on the size of the tree and also on how many new trackers are spawned.

The last step is efficient as well. The model updating scheme is the same as in [7] and we basically include all the matched features into the current object model and discard old features in a First-In-First-Out order. The model created by the first frame is always kept for reference.

3.5. Feature Sharing

Different from single object tracking, the branch and track paradigm provides a natural way to learn and share features among both lead and extras. Let $\mathcal{F}_k^i$ be the individual object feature set (i.e. the set of SIFT features) for tracker W_k^i at the k^{th} frame. We use all the features accumulated by each single tracker to probe new windows in the next frame. The shared feature set, denoted $\mathcal{F}_k$, is therefore:

$$\mathcal{F}_k = \bigcup_{i=1}^{|\mathcal{V}_k|} \mathcal{F}_k^i \tag{7}$$

This simple feature sharing scheme is surprisingly useful as the shared feature set actively gathers appearance information in a more global way and hence enables each individual window to be matched more reliably than what local methods can do. The additional advantage of feature sharing is the speedup in matching features. Instead of matching individual trackers one by one, we can now match all trackers in a batch processing mode.

4. Experiments

Our experiments are mainly proof-of-concept. In the first experiment, we test our trackers on both natural and visual branching process. Sample results are shown in Figure 2, Figure 3, and Figure 6.

In the second experiment, we show in Figure 7 that branch and track can also be useful in discovering repetitive patterns and filling a coherent region if we treat a static image as a fake video sequence.

The proposed branch and track algorithm is efficient in practice. Excluding the time for SIFT detection, the computation can be performed in $5 - 30$ frames per second on a single-core laptop for video sequences of resolution 320×240, which contain 2 to 20 similar objects. The program is written in MATLAB and modules of Efficient Subwindow Search and dynamic programming are implemented in C++/Mex. More video demos and the code are available at `www.cs.duke.edu/~steve/branch_and_track.html`.

4.1. Limitations

The limitations of the current framework are threefold: First, we assume that the branching tree never shrinks for simplicity. This is not realistic in practice and we expect one can also handle shrinking cases by determining in addition whether a tracker is lost or not.

Second, we find that branching is sensitive to the shape of the window. For instance, if the initial window already covers many similar objects and leaves few similar objects outside, the branching seems unlikely to happen. The improvement could be to automatically find the pattern in images and adjust the window to fit the pattern scale.

Third, the SIFT features seem not very reliable in places of motion blur and uniform texture. Therefore, our tracking result should benefit from better feature descriptors.

5. Conclusion and Future Work

Branch-and-track is a simple and effective method for tracking a lead object in the presence of multiple distracters

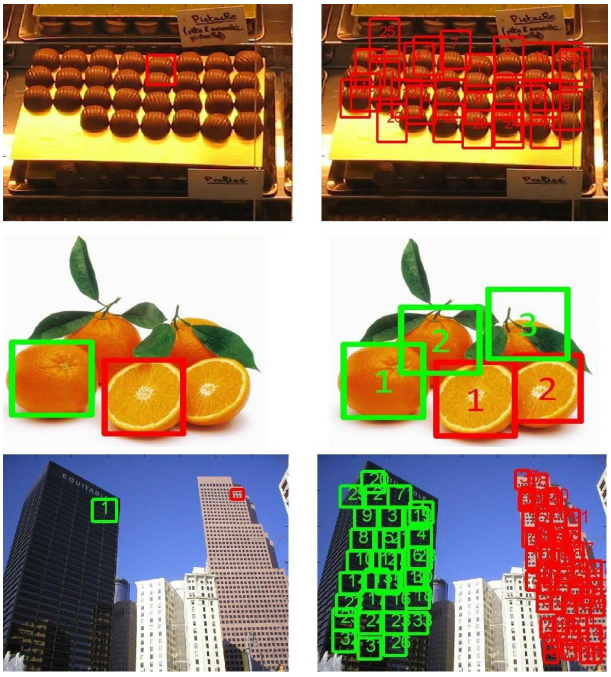

Figure 7. Branch and track on static images from [15]. The left column shows the image with one or two windows specified by a user. The right column shows the results of spatial tracking.

– the extras – and can also be used for the detection of repetitive patterns in still images. Tracking extras makes tracking the lead nimbler and more robust, both because shared features provide a richer object model, and because tracking extras accounts for sources of confusion explicitly.

Coordinating tracking across lead and extras allows optimizing window positions jointly rather than separately, for better results. The optimization is highly efficient thanks to dynamic programming and efficient sub-window search, which can handle large motions and changes in object size.

Future work includes the design of feature descriptors to better capture motion blur and appearance change. We also favor an algorithm that automatically detects spatial temporal patterns among similar objects.

Acknowledgement: This work is supported by the National Science Foundation under Grant No. IIS-1017017 and by the Army Research Office under Grant No. W911NF-10-1-0387.

References

[1] B. Babenko, M. Yang, and S. Belongie. Visual tracking with online multiple instance learning. In *IEEE CVPR*, pages 983–990, 2009.

[2] L. Breiman. Random forests. *Mach. Learning*, 45(1):5–32, 2001.

[3] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE CVPR*, pages 2142–, 2000.

[4] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *IEEE CVPR*, pages 886–893, 2005.

[5] P. Felzenszwalb and D. Huttenlocher. Pictorial structures for object recognition. *IJCV*, 61(1):55–79, 2005.

[6] S. Gu, Y. Zheng, and C. Tomasi. Critical nets and beta-stable features for image matching. In *ECCV*, pages 663–676, 2010.

[7] S. Gu, Y. Zheng, and C. Tomasi. Efficient visual object tracking with online nearest neighbor classifier. In *ACCV*, 2010.

[8] R. Hess and A. Fern. Discriminatively trained particle filters for complex multi-object tracking. In *IEEE CVPR*, pages 240–247, 2009.

[9] M. Isard and A. Blake. A smoothing filter for condensation. In *ECCV*, pages 767–781, 1998.

[10] Z. Khan, T. Balch, and F. Dellaert. Mcmc-based particle filtering for tracking a variable number of interacting targets. *IEEE PAMI*, pages 1805–1918, 2005.

[11] C. Lampert, M. Blaschko, and T. Hofmann. Efficient subwindow search: A branch and bound framework for object localization. *IEEE PAMI*, 31(12):2129–2142, 2009.

[12] B. Leibe, K. Schindler, and L. V. Gool. Coupled detection and trajectory estimation for multi-object tracking. In *ICCV*, pages 1–8, 2007.

[13] K. Li, E. Miller, M. Chen, T. Kanade, L. Weiss, and P. Campbell. Computer vision tracking of stemness. In *ISBI*, pages 847–850, 2008.

[14] Y. Li, C. Huang, and R. Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. In *IEEE CVPR*, pages 2953–2960, 2009.

[15] Y. Liu, H. Hel-Or, C. S. Kaplan, and L. V. Gool. Computational symmetry in computer vision and computer graphics. *Foundations and Trends in Computer Graphics and Vision*, 5(1-2):1–195, 2010.

[16] D. Lowe. Object recognition from local scale-invariant features. In *ICCV*, pages 1150–1157, 1999.

[17] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI*, pages 674–679, 1981.

[18] A. Perera, C. Srinivas, A. Hoogs, G. Brooksby, and W. Hu. Multi-object tracking through simultaneous long occlusions and split-merge conditions. In *IEEE CVPR*, pages 666–673, 2006.

[19] V. Reilly, H. Idrees, and M. Shah. Detection and tracking of large number of targets in wide area surveillance. In *ECCV*, pages 186–199, 2010.

[20] E. Shechtman and M. Irani. Matching local self-similarities across images and videos. In *IEEE CVPR*, 2007.

[21] J. Shi and C. Tomasi. Good features to track. In *IEEE CVPR*, pages 593 – 600, 1994.

[22] C. Tomasi, S. Petrov, and A. Sastry. 3d tracking = classification + interpolation. In *ICCV*, pages 1441–1448, 2003.

[23] P. Viola, J. Platt, and C. Zhang. Multiple instance boosting for object detection. In *NIPS*, 2005.