Co-Tracking Using Semi-Supervised Support Vector Machines

Feng Tang, Shane Brennan, Qi Zhao and Hai Tao
UC Santa Cruz
Santa Cruz,CA,USA
{tang|shanerb|zhaoqi|tao}@soe.ucsc.edu

Abstract

This paper treats tracking as a foreground/background classification problem and proposes an online semi-supervised *learning framework. Initialized with a small number of labeled samples, semi-supervised learning treats each new sample as unlabeled data. Classification of new data and updating of the classifier are achieved simultaneously in a co-training framework. The object is represented using independent features and an online support vector machine (SVM) is built for each feature. The predictions from different features are fused by combining the confidence map from each classifier using a classifier weighting method which creates a final classifier that performs better than any classifier based on a single feature. The semi-supervised learning approach then uses the output of the combined confidence map to generate new samples and update the SVMs online. With this approach, the tracker gains increasing knowledge of the object and background and continually improves itself over time. Compared to other discriminative trackers, the online semi-supervised learning approach improves each individual classifier using the information from other features, thus leading to a more robust tracker. Experiments show that this framework performs better than state-of-the-art tracking algorithms on challenging sequences.*

1. Introduction

Object tracking is a critical and basic component in computer vision and has been an active research area for decades. Tracking is essentially the task of finding the object states (which can be position, scale, velocity and other parameters characterizing the object) from the observed image sequence. The task is challenging because of:

- difficulty in object re-acquisition, i.e., the capability to recover when the tracker drifts away;
- complicated object appearance changes which are difficult to model;
- background clutter which makes it difficult to distinguish the object from the background;
- complex non-linear dynamics, which makes it hard to predict the object state;
- occlusion which causes the observation to be noisy and incomplete.

Traditional tracking algorithms try to solve the above problems by building a generative model to describe the visual appearance of the object. The state estimation boils down to the problem of finding the state which has the most similar object appearance to the model in a maximum-likelihood or maximum-a-posterior formulation. Essentially, this type of algorithm is equivalent to finding the nearest neighbor in some high dimensional object representation space. However, it requires the use of a model to describe the properties of the object. To accommodate appearance changes the object model is often updated online. Unfortunately, object appearance changes are highly nonlinear and difficult to model. An incorrect update of the object model may lead to the "drifting" problem. Some examples of generative trackers are EigenTracking [3], *WSL* tracker [15], and Condensation [13]. In essence, the main problem for these generative trackers is that they rely on knowledge of the foreground only, completely ignoring the background, which is the main reason for the "drifting problem". The "layer tracker" proposed in [21] is also a generative model but it models both foreground and background, the per-pixel layer ownership is inferred by competing the foreground and background likelihoods.

Recently, discriminative methods have opened a promising new direction in the tracking literature by posing tracking as a classification problem. Instead of trying to build a complex model to describe the object, discriminative trackers seek a decision boundary that can best separate the object and the background. This naturally solves the background clutter and re-acquisition problem. Complex object dynamics are also not a problem since tracking becomes an online detection problem which makes no assumptions of where the object could be. To adapt to object appearance changes the discriminative methods update the decision boundary instead of the object appearance model.

Table 1. Comparison of different discriminative trackers

Tracker	no pre-train	online learning	collaborative training	easy to generalize	# of features
SVT	×	×	self-training	×	single
EST	√	√	self-training	√	multiple
OFS	√	√	self-training	√	single
Our approach	√	√	collaborative	√	multiple

The support vector tracker [1] (denoted as SVT afterwards) uses an offline-learned support vector machine as the classifier and embeds it into an optical-flow based tracker. Compared to later algorithms it is a discriminative tracker but the classifier never gets updated online. Collins et al. [7] were perhaps the first to treat tracking as a binary classification problem. In their approach the current frame is classified using a classifier learned in the previous frame. A variance ratio is used to measure feature discriminability and select the best color space feature from a feature pool for tracking, we denote their tracker as OFS in Table 1. Avidan's ensemble tracker [2] (denoted as EST) combines an ensemble of online learned weak classifiers using AdaBoost to label pixels in the next frame. After the data is labeled, the peak of the classification score map is defined to be the object. To handle the object appearance changes and maintain temporal coherence, in each frame a fixed number of classifiers that do not perform well or have existed longer than a fixed number of frames get removed or "pruned" from the tracker, and new classifiers are trained to replace them. This algorithm is elegant because the classifier can update online to accommodate the object appearance changes.

The main problem with SVT is that it is hard to generalize it to arbitrary object types. This is because the SVM uses supervised learning to build a classifier off-line and therefore a large hand-labeled dataset is required. This sometimes is a problem since manually labeling a large dataset is very expensive or potentially impossible. The main problem with OFS and EST is that they use the classification results to update the classifier itself. This is referred to as "self-training" in the machine learning literature. The problem with self-training is that the classification mistakes reinforce themselves and the algorithm is not robust to outliers [23], and the machine learning community still has some argument about this self-learning approach [6].

To solve these problems we treat object tracking as an online semi-supervised classification problem. Semi-supervised learning learns from a combination of both labeled and unlabeled data. Semi-supervised learning provides a general framework to learn a classifier for different types of objects which may not have enough labeled data. Some examples of semi-supervised learning algorithms include: the Expectation-Maximization (EM) algorithm [10], co-training [4], tri-training [22], and transductive support vector machine [16]. One typical algorithm is the co-training method which has two classifiers that train each other using unlabeled data. Inspired by the co-training idea, we propose a semi-supervised learning method which uses multiple independent features for training a set of classifiers online. The classifiers then collaboratively classify the unlabeled data and use this newly labeled data to update each other. In our algorithm two relatively independent features are used: color histograms and histograms of oriented gradients (HoG) [9] as the object representation. Of course other types of features can be easily incorporated. Each feature is used to train an online support vector machine, and their outputs are combined to give the final classification results. The main advantages of this scheme are:

- It is a collaborative approach that uses the strength of different views of the object to help improve each other, hence a more robust tracker. In addition, it is also a fusion framework to combine disparate feature in a principled way.
- It can handle arbitrary object types without the need of any prior knowledge of the object type.
- The framework naturally combines the generative framework and discriminative framework in the sense that it keeps the object model in the form of support vectors and at the same time it seeks a decision boundary to separate foreground and background regions.

A detailed comparison of the proposed approach to popular discriminative trackers is given in Table 1.

There are also some related papers on online detector learning. Levin et al. [17] build boosting classifiers for gray-image and background difference features which co-train each other to improve the overall detection performance. Nair et al. [18] proposed an unsupervised learning approach for human detection which uses motion information as an "automatic labeler" to supply labeled training examples. This kind of algorithm only works under restricted conditions. More recently, Javed [14] proposed to improve an off-line learned object detector using co-training. They use the PCA coefficients as the features to be used for each classifier (online boosting). The downside of this approach is that it needs a pre-trained detector which limits its capability to generalize to arbitrary object types. Grabner et al. [12] propose online boosting to adapt the classifier to object appearance changes, but they rely on a single feature and still fall into the self-training framework.

The rest of the paper is organized as follows: section 2 gives a brief introduction to the two key components of our approach - online support vector machines and co-training. The proposed semi-supervised learning framework is presented in section 3. Experiments and comparisons are shown in section 4. We make conclusions and discuss future work in section 5.

2. Background

2.1. Online Support Vector Machines

Using support vector machines for classification provides tools for learning models that generalize well even in sparse high dimensional settings. SVM classifiers of the form $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) + b$ can be learned from the training data $(\mathbf{x_i}, y_i); \mathbf{x} \in \mathbb{R}, y \in \{-1, +1\}, i \in 1, ..., N$ by maximizing the margin, or equivalently minimizing:

$$\min_{\mathbf{w},b,\xi} \frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{N} \xi_i \tag{1}$$

subject to the constraints:

$$y_i(\mathbf{w} \cdot \Phi(\mathbf{x}) + b) \geq 1 - \xi, \xi \geq 0, i \in 1, ..., N \tag{2}$$

where ξ are the slack variables, C is the tradeoff parameter between allowed error in the samples and the margin. By taking the Lagrangian of Eq.1 and setting it to zero, one can solve and express the problem in its dual form:

$$\min_{0\leq\alpha\leq C} W = \frac{1}{2}\sum_{i,j=1}^{N} \alpha_i Q_{i,j} \alpha_j - \sum_{i=1}^{N} \alpha_i + b\sum_{i=1}^{N} \alpha_i \tag{3}$$

where $Q_{i,j} = y_i y_j \Phi(\mathbf{x}_i)\Phi(\mathbf{x}_j)$.

The Karush-Kuhn-Tucker (KKT) conditions uniquely define the solution of dual parameters by:

$$g_i = \frac{\partial W}{\partial \alpha_i} = \sum_{i,j=1}^{N} Q_{i,j}\alpha_j + y_i b - 1 \begin{cases} > 0 & \alpha_i = 0 \\ = 0 & 0 \leq \alpha_i \leq C \\ < 0 & \alpha_i = C \end{cases} \tag{4}$$

$$h = \frac{\partial W}{\partial b} = \sum_{i=1}^{N} y_i \alpha_i = 0 \tag{5}$$

Those samples with $g_i = 0$ are usually called support vectors, samples with $g_i < 0$ are called error vectors, the rest are called reserve vectors and exceed the margin ($g_i > 0$). Note that this "sparse" representation can be viewed as data compression as in the construction of a K-Nearest-Neighbor (KNN) classifier.

In [5] an online method is proposed to incorporate or remove samples from an SVM one vector at a time. It gives the exact solution for $N \pm 1$ samples in terms of the N old samples and the sample to be added or removed instead of an approximate solution as in [19]. The basic idea is to progressively perturb the new sample coefficients with the goal of maintaining the KKT conditions for all the previous training data. The algorithm is initialized with the previous solution to equations 4 and 5. Each incremental/decremental perturbation leads to a category change for at least one sample. Finally, all the samples fall into one of the three categories: support vector, error vector or reserve vector, and both new and old data satisfy the KKT conditions (Eq. 4-5). Details of this algorithm can be found in [5].

2.2. Co-training

A typical algorithm in semi-supervised learning is co-training [4]. It allows one to start with only a few labeled examples to produce an initial weak classifier and then use unlabeled data to improve the performance. The basic intuition is that sometimes features describing the data are redundant and could be split into two sets, each of which on its own is sufficient for correct classification. Then one can build a classifier for each set. These classifiers then go through unlabeled examples, label them, and add the most confident predictions to the labeled set of the other classifier. In other words, the classifiers train each other using the unlabeled data.

3. The Co-Tracking Algorithm

Our algorithm combines independent features using a co-training framework and applies the resulting classifier to the task of object tracking. We refer to this approach as the Co-Tracking algorithm. An overview and pseudo-code for our algorithm can be seen in the table below. In this section we proceed to discuss each part of the algorithm in more detail, and provide results of our approach in section 4.

Overview of our proposed Co-Tracking algorithm.

Initialization:

1. Acquire N labeled frames either manually or by another tracking algorithm.
2. Acquire positive and negative samples for each feature type.
3. Train an SVM classifier for each feature type using the labeled samples.
4. Obtain a weight for each classifier.

Online Co-Tracking:

```
for i = N + 1 to the end of the sequence
    1. Build a confidence map for each classifier.
    2. Combine the confidence maps using the classifier weights.
    3. Find the object in current frame using the combined confidence map.
    4. Obtain new samples for each classifier using the co-training framework.
    5. Update the SVM classifiers with the new samples.
    6. Obtain new weights for each classifier.
    7. Remove old samples from the SVMs
end
```

3.1. Initialization

3.1.1 Acquiring Labeled Samples

Most tracking algorithms begin with a single labeled frame that is labeled either manually or by an accurate detection algorithm. Since our algorithm relies on support vector machines which must be trained on labeled data our algorithm relies on labeled data as well. This requires a method of obtaining labeled frames for training purposes. We have found that our co-tracking algorithm can perform well with as few as 10 labeled frames for the sequences shown in the experiments. These labels do not need to be done manually, instead they can be the result of another, simpler tracking algorithm such as the mean-shift tracker [8] or change based tracker [20]. Therefore, our algorithm does have an Achilles heel of relying on another tracking algorithm to accurately track the object for 10 frames. However, most objects do not change appearance or move a significant distance in such a short time (333 milliseconds in NTSC video) and even basic tracking algorithms can track an object for such a small number of frames.

Given labeled frames positive samples can be obtained by computing feature vectors for the area of the frames which contain the object as given by the label. Negative samples can also be obtained by computing feature vectors for areas of the frames that do not overlap with the object. With these labeled samples the SVMs can be trained and the co-tracking algorithm can then take over responsibility from the simpler tracking algorithm.

3.1.2 Assigning Classifier Weights

In order to combine trained classifiers into a final classifier we must assign a weight to each of them. Logically, this weight should be based on the accuracy of each classifier. We therefore adapt the concept from AdaBoost [11] of determining the weight of a classifier based on its error on a labeled validation set. During initialization all of the samples are labeled and therefore these samples can be used as a validation set. We then evaluate the classifiers for every sample in this set and define the error as:

$$\epsilon = \frac{(N - Y_{+}) - (M - Y_{-})}{2} \tag{6}$$

$$Y_{+} = \Sigma_{i=1}^{N} sign(S(V_{i+})) \qquad Y_{-} = \Sigma_{j=1}^{M} sign(S(V_{j-}))$$

where $S()$ is trained the classifier, V_{i+} is the i^{th} positive sample, V_{j-} is the j^{th} negative sample, N is the number of positive samples (equivalent to the number of labeled frames) and M is the number of negative samples. This error places equal importance on the positive and negative samples which is necessary due to the fact that the negative samples can greatly outnumber the positive samples. Once these errors have been computed they can be combined to obtain the following weights:

$$W_{color} = 1 - (\epsilon_{color} + \tau)/(\epsilon_{color} + \epsilon_{hog} + \tau) \tag{7}$$

$$W_{hog} = 1 - (\epsilon_{hog} + \tau)/(\epsilon_{color} + \epsilon_{hog} + \tau) \tag{8}$$

where τ is some small constant used to avoid the divide by 0 issue that is raised when ϵ_{color} and ϵ_{hog} are both 0. Even for an extreme case when one feature fails to distinguish the object from background, i.e., one of the weights is 0, (the) other feature(s) might still be used to obtain correct tracking.

3.2. Locating the Object

Given a set of classifiers and an unlabeled frame the algorithm must determine the location of the object within the frame. To do this, each classifier is applied to each location in the unlabeled frame using a sliding window technique. This operation builds a confidence map for each feature on the unlabeled frame. Given the confidence maps built by the classifiers and a weight for each classifier the next task is to combine the confidence maps into a final combined confidence map. This combination can be visualized in figure 1.

Using the combined confidence map the algorithm needs to determine the location of the object in the image. One method of accomplishing this task would be to use a gradient ascent algorithm that was initialized at the last known location of the object. This would in effect be adding a spatial constraint to the algorithm, telling the algorithm “the object is more likely to be at this location.” However, this approach is not robust against rapid changes in object position due to either large object motion or camera motion. We therefore remove this spatial constraint and choose the location of the

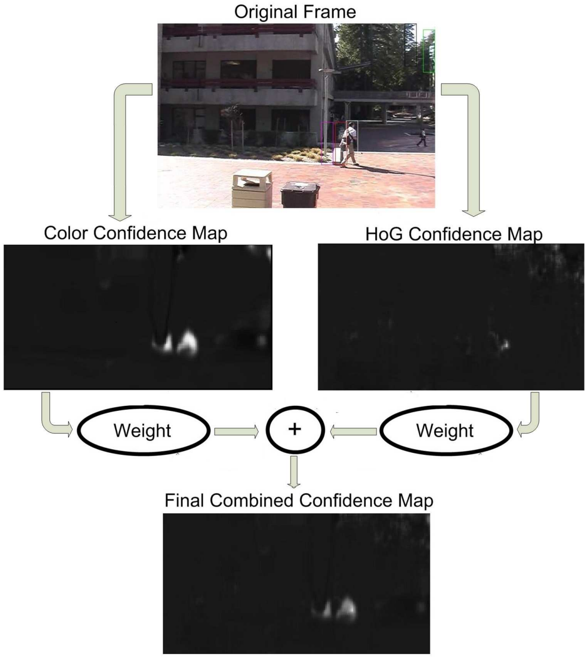


Figure 1. Combining the confidence maps from individual features into a final confidence map. Each feature has an associated weight which represents the contribution of its associated confidence map to the final confidence map. As can be seen, the final confidence map has a more distinct peak than either of the individual confidence maps, leading to more stable tracking. The corresponding frame of the sequence can be seen in at the top of the figure. The white bounding box shows the location of the global peak in the combined confidence map. The colored boxes show the locations of the next 4 highest peaks.

object in the current frame to simply be the global maximum in the combined confidence map. There could also be a tradeoff between the extent of spatial constraint and the computation cost. With a larger spatial constraint, we have better tolerance to large object motion, but also have the chance to incorporate confusing objects. This tradeoff may depend on the application and a rough estimate of the object motion. In our work, to make the algorithm general, we remove the spatial constraint and the experiments show that the tracker can still keep track without getting confused by background.

In any online algorithm it is important to monitor the confidence of the tracking result at each step in order to determine whether or not to add that current instance to its “database” of features and update the classifier. In our approach this monitoring is achieved with the use of a hard threshold on the confidence of the final classification result. If the value of the confidence map at the global maximum is lower than this threshold the classifiers are not updated for the current frame and the algorithm immediately moves on to the next frame. This threshold must be manually defined and represents how conservative one wants to be in their updating scheme. In frames where the object is not present due to occlusion the confidence of the global peak is typically around 0.3-0.4 and we therefore chose a threshold value of 0.5. An alternative to this thresholding approach is to provide a weight to each sample based on the value of the global maximum in the confidence map. This allows a softer update where the classifiers are always updated but are updated conservatively during times when the algorithm is unsure of its result, and updated strongly when the algorithm is more sure of its result.

3.3. Updating the Classifiers Through Co-Training

To update the classifiers we must find new samples from the processed frame and add them to the SVMs. Adding every background region as a new negative sample is costly. Instead, we want to add negative samples that are close to the classifier hyperplane such that the classifier will learn to better distinguish these samples. This would normally be difficult since it requires an evaluation of every background sample in order to find its distance to the hyperplane but this information has already been computed when building the confidence maps used previously in the algorithm. To find the most significant negative samples we find the K highest peaks in the confidence map that do not overlap with the object. We refer to these K peaks as modes. We can then add each mode to the classifiers as a negative sample as well as the positive sample which is collected based on the tracking result. To ensure a classifier does not have a bias towards negative samples we give the new positive sample a weight equivalent to the sum of the weights of the negative samples. If the positive sample has a high confidence we add it to the validation dataset as positive samples. We also add the mode with the lowest confidence to the validation dataset as negative samples. This ensures that samples in the validation dataset are very likely labeled correctly.

The heart of the co-training framework lies in the fact that the new samples generated by one feature are passed onto the classifier for the other feature. Essentially, the color classifier is telling the HoG classifier “I find these regions difficult to classify, learn to discriminate against them so that in future frames I can rely on you to reject these regions.” The HoG classifier likewise has the color classifier learn to discriminate well in the regions the HoG classifier performs poorly. In future frames a given classifier will not perform any better on those regions than it did previously and will still give that background region a high confidence, but the other classifier will perform better on the region than it did previously and will assign that region a low confidence. After updating the classifiers using the new samples, the classifier weights are updated according to the equations in section 3.1.2.

3.4. Pruning Samples

The number of samples in the SVM classifiers can grow quickly over time. In addition, new samples have less and less weight in the classifier as the total number of samples grows, causing the algorithm to learn changes in the appearance of the object more and more slowly. To overcome this issue, samples that are more than T frames old are removed from the SVM, which is possible due to the incremental/decremental SVM algorithm we are using. We used values of T between 10 and 35 and saw little difference in our test sequences, but having T be very large led to a degradation in tracking results, and a value of T that was too small led to classifiers which were trained on very few data points, which led to unreliable classification.

4. Experiments and Results

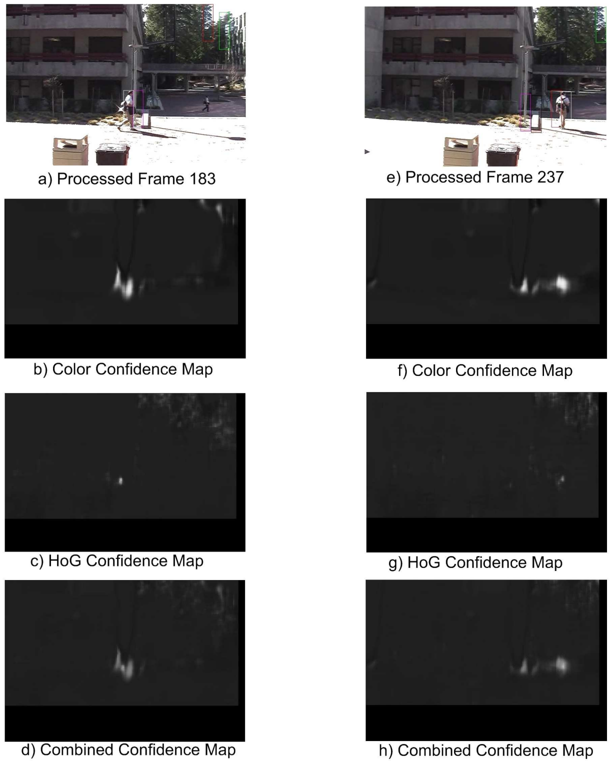
We implemented the proposed approach in MATLAB and tested it on several challenging sequences. Results are compared with both Ensemble Tracking [2] and mean-shift tracking [8]. The Mean-Shift tracker used for comparison was implemented by us in C++.

4.1. Implementation details

The color feature used to characterize a sample is a 64 $(4 \times 4 \times 4)$ dimensional histogram equably discretized in $\{$R,G,B$\}$ space. To compute the HoG feature, the sample image is divided into $b \times b$ sub-patches of equal size ($b = 4$ in our implementation). A HoG with 9 orientation bins is computed for each sub-patch and the final feature vector is a concatenation of these 9-dimensional vectors.

4.2. Results

The first comparison we make is of a sequence which contains a pedestrian walking through a courtyard. The camera is moving in the opposite direction as the pedestrian and the background is changing over time. Obviously, background subtraction cannot be used here. The sequence is challenging because the background is also very cluttered with bushes, buildings, and other pedestrians. In addition, this person changes his pose significantly while walking. Tracking results and the corresponding confidence maps for this sequence are shown in Figure 2. The white boxes are the detected object location and colored boxes are the next 4 highest peaks in the confidence map. As can be observed in some frames, such as shown in figure 2(b-c), the color confidence map is confusing. There are several peaks which have almost the same height as the global peak and it is difficult to make the correct classification without other information. However, the HoG confidence map is reliable in the regions the color classifier is unreliable and by combining the features together we can still obtain a clear peak at the correct location in the final confidence map, as shown in Figure 2(d). Similarly, in Figure 2(f-h), the HoG confidence map is confusing but the color feature is reliable, and the final confidence map has a clear global peak. In this instance the algorithm can still keep track of the object.


a) Processed Frame 183

e) Processed Frame 237

b) Color Confidence Map

f) Color Confidence Map

c) HoG Confidence Map

g) HoG Confidence Map

d) Combined Confidence Map

h) Combined Confidence Map

Figure 2. The results of our algorithm on a video of a pedestrian walking across a courtyard. As can be seen, the combined confidence map obtained through combining HoG and color information through our co-training framework is more reliable than the confidence map from either feature individually. a-d show the results from frame 183 of the sequence, and e-h show the results from frame 237 of the sequence.

Results from a more challenging sequence in which two pedestrians are walking together along a crowded street with an extremely cluttered background demonstrates the discriminative power of our algorithm. In this sequence the hand-held camera is extremely unstable. The shaky nature of the sequence makes it all the more difficult to accurately track the pedestrians. Despite this, our algorithm is able to track the pedestrians throughout the entire 140 frames of the sequence. Shai Avidan mentions in [2] that the Ensemble Tracker is able to track for the first 80 frames of the sequence but does not mention the performance for the remaining 60 frames. Confidence maps generated by the Ensemble Tracker and by our algorithm are shown in Figure 3. As can be observed, our algorithm produces cleaner confidence maps that are smoother and have a clearn peak at the object center which leads to more stable tracking.

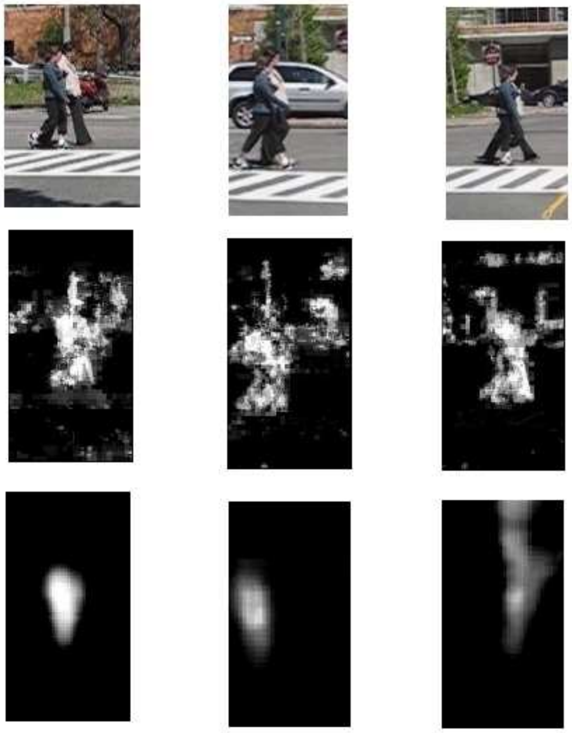


Figure 3. A comparison of our confidence maps with those of the Ensemble Tracker. Original frames are given in the top row. The middle row contains the confidence maps generated by the Ensemble Tracking algorithm. The bottom row shows the confidence maps generated by our algorithm.

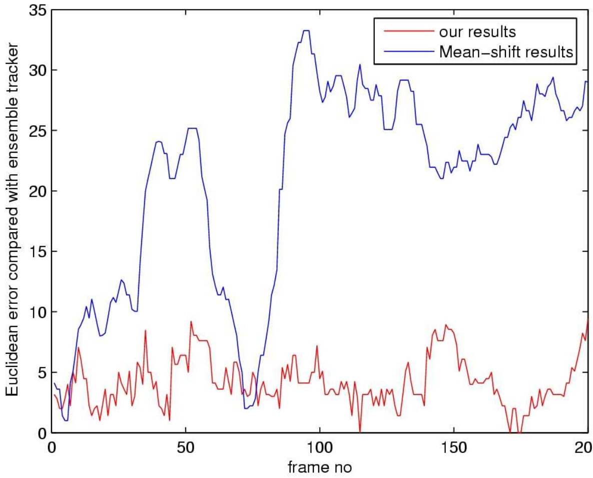


Figure 4. A comparison of our algorithm with the mean-shift tracking algorithm on a PETS2004 sequence.

Figure 4 shows the results of our algorithm compared with the mean-shift algorithm on a PETS sequence. As can be seen, our algorithm is able to track the object more stably, and hence closer to the ground truth, than mean-shift. In most of our test sequences mean-shift is unable to track the object through the entire sequence whereas our algorithm is able to track the object throughout the sequence. When failures do occur in our algorithm it is usually a result of large scale changes in the object. Figure 5 shows additional frames processed by our algorithm.

5. Conclusions

In this paper we pose tracking as a semi-supervised learning problem and propose a robust tracker using online support vector machines and the co-training framework. Our algorithm only requires a small amount of labeled data during initialization after which the algorithm can continually improve itself using co-training. As a result, an increasingly robust tracker is achieved. However, our tracker uses a fixed object scale and orientation representation and as a result is unable to handle large changes in object scale and orientation. In the future we would like to make our tracker adapt to these changes in a principled manner. The proposed algorithm also lends itself easily to extension; additional features can easily be incorporated into the co-tracking framework. The only requirement is that the additional features not be strongly correlated with other features. These features also need not be classified with an SVM, they can be of any classifier type so long as it can generate a confidence map. We thereby provide a cohesive and principled framework for combining, or “fusing”, disparate feature types.

6. Acknowledgement

This research has been supported by research grants US-NSF (National Science Foundation), IIS-0348020.

References

[1] S. Avidan. Support vector tracking. *PAMI*, 26(8):1064–1072, 2004. 2

[2] S. Avidan. Ensemble tracking. *PAMI*, 29(2):261–271, February 2007. 2, 6

[3] M. J. Black and A. Jepson. Eigentracking: Robust matching and tracking of articulated objects using a view-based representation. In *ECCV*, pages 329–342, 1996. 1

[4] A. Blum and T. Mitchell. Combining labeled and unlabeled data with co-training. In *COLT*, 1998. 2, 3

[5] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *NIPS*, pages 409–415, 2000. 3

[6] O. Chapelle, B. Scholkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, 2006. 2

[7] R. Collins, Y. Liu, and M. Leordeanu. Online selection of discriminative tracking features. *PAMI*, 27(10):1631–1643, October 2005. 2

[8] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, pages 142–151, 2000. 4, 6

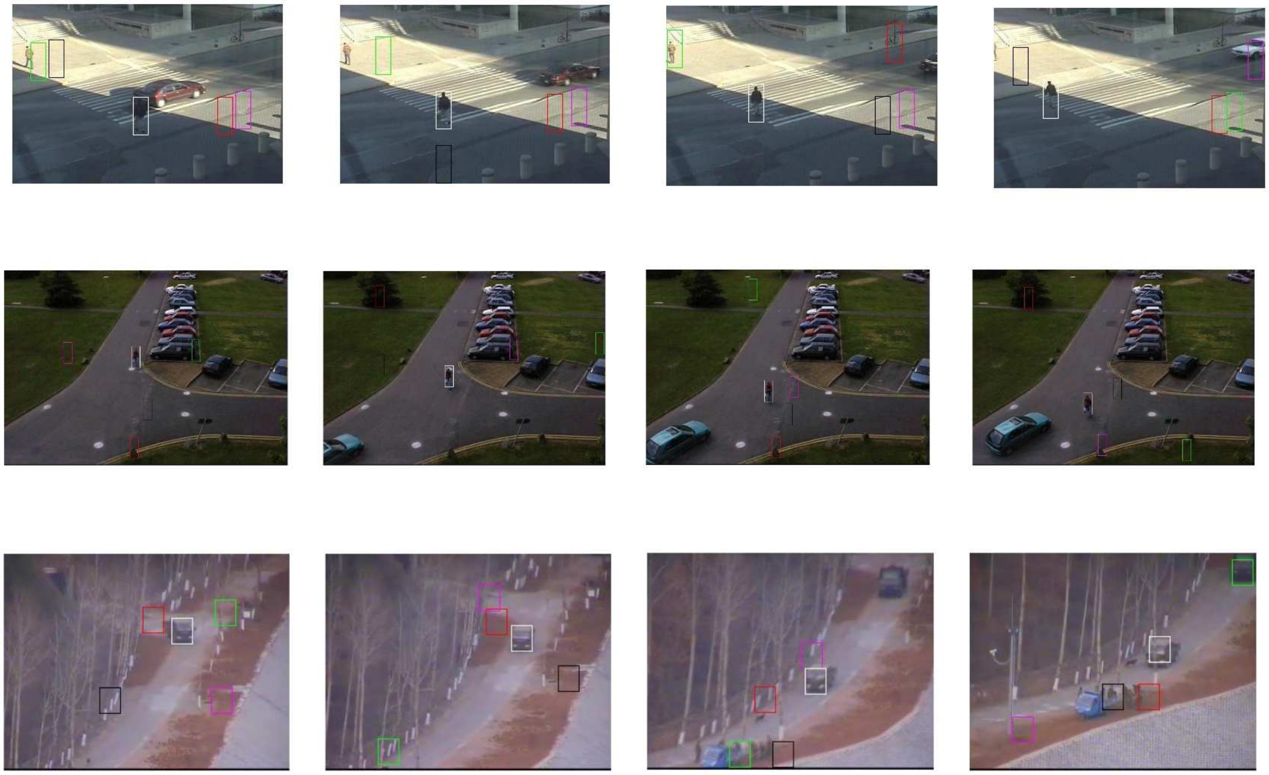


Figure 5. Additional results from our algorithm. The white bounding box is in the location of the object. Colored boxes represent the next 4 highest peaks in the confidence map. The top row shows the results of our algorithm on frames 45, 62, 82, and 110 from a sequence used in the Ensemble Tracking paper. The middle row shows the esults of our algorithm on frames 441, 502, 531, and 554 from a PETS2001 sequence. The bottom row shows the results of our algorithm on frames 83, 179, 331, and 392 from a vehicle sequence.

[9] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, pages I: 886–893, 2005. 2

[10] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, Series B., 1977. 2

[11] Y. Freund. Boosting a weak learning algorithm by majority. In *COLT*, pages 202–216, 1990. 4

[12] H. Grabner and H. Bischof. On-line boosting and vision. In *CVPR*, pages I: 260– 267, 2006. 2

[13] M. Isard and A. Blake. Condensation - conditional density propagation for visual tracking. *IJCV*, 29(1):5–28, 1998. 1

[14] O. Javed, S. Ali, and M. Shah. Online detection and classification of moving objects using progressively improving detectors. In *CVPR*, pages I: 696–701, 2005. 2

[15] A. Jepson, D. Fleet, and T. El-Maraghi. Robust online appearance models for visual tracking. *PAMI*, 25(10):1296–1311, October 2003. 1

[16] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML*, pages 200–209, 1999. 2

[17] A. Levin, P. Viola, and Y. Freund. Unsupervised improvement of visual detectors using co-training. In *ICCV*, pages 626–633, 2003. 2

[18] V. Nair and J. Clark. An unsupervised, online learning framework for moving object detection. In *CVPR*, pages II: 317–324, 2004. 2

[19] J. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods—Support Vector Learning*, 1999. 3

[20] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *CVPR*, pages II: 246–252, 1999. 4

[21] H. Tao, H. Sawhney, and R. Kumar. Object tracking with bayesian estimation of dynamic layer representations. *PAMI*, 24(1):75–89, January 2002. 1

[22] Z.-H. Zhou and M. Li. Tri-training: exploiting unlabeled data using three classifiers. *IEEE Transactions on Knowledge and Data Engineering*, 17:1529–1541, 2005. 2

[23] X. Zhu. Semi-supervised classification, 2002. CVonline: Semi-Supervised Learning. Available: http://homepages.inf.ed.ac.uk/cgi/rbf/ CVONLINE/entries.pl?TAG1180. In CVonline: On-Line Compendium of Computer Vision [Online]. R. Fisher (ed). Available: "http://homepages.inf.ed.ac.uk/rbf/CVonline/". [accessed on April.5, 2007]. 2