CS229 Course Project: A new rival to Predator and ALIEN

Martin Raison
Stanford University
mraison@stanford.edu

Botao Hu
Stanford University
botaohu@stanford.edu

Abstract

This report documents how we improved the TLD framework for real-time object tracking [1] by using a new set of features and modifying the learning algorithm.

1 Introduction

The problem of real-time object tracking in a sequence of frames has gained much recognition in the Computer Vision community in recent years. The TLD framework (Kalal et.al. [1]), marketed as Predator, and the ALIEN tracker (Pernici et.al. [2]) are recent successful attempts to solve the problem. The TLD framework [1] improves the tracking performance by combining a detector and an optical-flow tracker. The purpose of the detector is to prevent the tracker from drifting away from the object, and recover tracking after an occlusion. Since the only prior knowledge about the object is a bounding box in the initial frame, the detector is trained online via semi-supervised learning. In order to build such a system, two challenges must be addressed:

1) finding a good set of features to be used by the detector for classifying image patches
2) using an efficient learning algorithm to train the detector with examples from previous frames

The solutions to these two problems are dependent on each other, and as such, they must be designed so as to fit into a single system.

Our goal was to investigate new approaches for 1) and 2) and try to find improvements in terms of robustness of tracking (good performance with a wide range of objects, tolerance to fast movements, camera blur, clutter, low resolution, etc) and efficiency (time and space complexity).

2 Motivation & Background Work

This section details the mode of operation of the TLD detector [1], and motivates our use of features based on compressive sensing for improving the system.

2.1 TLD detector operation

The main focus of this work is the detection component of the framework, and the associated learning process. The detector used in the TLD framework uses the following workflow:

1) Each frame is scanned using a sliding window, at multiple scales. About a hundred thousand windows are considered, depending on the size of the image and the size of the original object bounding box. The part of the image contained in a window is called a patch.
2) Each patch is flagged as positive or negative using a 3-step detection cascade:

- Variance filter: If the variance of the patch is less than half the variance of the object in its initial bounding box, the window is rejected
- Ensemble classifier: a confidence measure is obtained for the patch using random ferns. Several groups of features are extracted from the patch, and for each group, a probability is computed, based on the number of times the same combination of features appeared in previous frames as positive or negative examples. The final confidence measure is the average of the probabilities of each group of features.
- Nearest-Neighbor classifier: the Normalized Correlation Coefficient is used to evaluate the distance between the considered patch and two sets of patches: one set of positive patches, one set of negative patches (built from previous frames). These two sets of patches represent the object template, and are maintained by a P/N learning algorithm, introduced in [1].

Background substraction can also be used as a preliminary step to filter out windows.

2.2 Compressive sensing for image patch descriptors

The ensemble classifier is a critical part of the detection cascade. With the dataset used for the experiments, we

noticed that on average, it selects about 50 patches out of 25,000 patches on each frame. One of the main difficulties of training the ensemble classifier is that the size of the training set is small. The TLD framework tackles this issue by using random ferns, but in reality the independence assumption between groups of features is not verified. During the project, we tried to improve this part of the detection cascade by using alternative descriptors (i.e. sets of features) for the patches and modifying the classification algorithm accordingly.

Many descriptor extraction algorithms have been developed in the last few years, such as FREAK, BRISK, BRIEF or ORB. The recent Compressive Tracking (CT) method [3] introduces a way of computing descriptors based on compressive sensing. Given an image patch X of size $w \times h$, the patches $X_{i,j}$, $1 \leq i \leq w$, $1 \leq j \leq h$, are obtained by filtering X with a rectangle filter of size $i \times j$ whose elements are all equal to 1. Each pixel of a patch $X_{i,j}$ represents a rectangle feature (sum of all the elements in a rectangle). All the $X_{i,j}$ are then considered as column vectors, and concatenated to form a big column vector $\overline{X}$ of size $(wh)^2$. The many features contained in $\overline{X}$ are meant to account for a variety of scales and horizontal/vertical deformations of the object. The descriptor x of the patch is then obtained from $\overline{X}$ with a random projection. A very sparse random matrix R of size $l \times (wh)^2$ is used for the projection, where l is the size chosen for the descriptor. The matrix R is randomly defined by:

$$R_{i,j} = \begin{cases} -1 & \text{with probability } \frac{1}{2s} \\ 1 & \text{with probability } \frac{1}{2s} \\ 0 & \text{with probability } 1 - \frac{1}{s} \end{cases}$$

where s is a constant. Li et.al. showed in [4] that for s up to $wh/\log(wh)$, the matrix R is asymptotically normal (up to a constant). A consequence is that with high probability, $\overline{X}$ can be reconstructed from x with minimal error (Achlioptas [5]).

This algorithm was used in [3] for the purpose of building a real-time object tracker - the "Compressive Tracker" (CT). This approach led to successful results, but we observed that the system itself has limitations. The CT method considers much fewer windows on each frame than TLD. As a consequence, while the computation time is significantly reduced, the CT tracker is not very robust to fast movements and occlusion. In addition, although the rectangle filters are intended to account for "scales" of the object, the scale itself is never explicitly determined. The size of the current bounding box always remains equal to the size of the initial bounding box, which is a significant drawback in scenes where the distance between the object of interest and the camera varies.

3 Methodology

We focused on transposing the CT method to the TLD framework, to improve the TLD detector while overcoming the limitations of the original CT approach.

3.1 Descriptor Computation

The main challenge for the descriptor computation was to scale up the algorithm. CT considers only windows near the current object location, whereas TLD scans the whole frame. So we needed to build descriptors that were easier to compute. Fortunately, another characteristic of CT descriptors is that they were intended to work for all scales of the object. This is not necessary in the case of the TLD framework, since each frame is scanned at multiple scales. Based on these observations, we slightly modified the descriptors, and designed an algorithm to compute them efficiently. This procedure was the object of the CS231A part of the project.

3.2 Online Naïve Bayes

Given the sparsity of the training data, and the necessity to train the classifier online, the Naïve Bayes algorithm was a natural choice. In addition, object tracking is specifically challenging because of appearance changes of the object of interest over the course of the video. In order to introduce decay in the model, we used a learning rate λ to do the relative weighting between past and present examples. During tracking, one model update is performed for each frame.

The input features are image patch descriptors. Each descriptor is denoted by $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, ..., x_n^{(i)})$, with a label $y^{(i)} = 1$ if the patch corresponds to the object of interest, $y^{(i)} = 0$ otherwise. The $x_j^{(i)}$'s for $j = 1, .., n$ are supposed to be independent given $y^{(i)}$, with

$$x_j^{(i)} \mid y^{(i)} = 1 \sim \mathcal{N}(\mu_j^{(1)}, \sigma_j^{(1)})$$
$$x_j^{(i)} \mid y^{(i)} = 0 \sim \mathcal{N}(\mu_j^{(0)}, \sigma_j^{(0)})$$

Also, for $k \in \{0, 1\}$, we denote by $\mu^{(k)*}, \sigma^{(k)*}$ the mean and variance of the positive ($k = 1$) and negative ($k = 0$) training examples drawn from the current frame. If we give a weight λ to the training examples from the previous frames, and a weight $1 - \lambda$ to the training examples from the current frame, we can derive the mean and variance of the resulting descriptor

distribution, and obtain the following update formulas:

$$\mu^{(k)} := \lambda\mu^{(k)} + (1-\lambda)\mu^{(k)*}$$

$$\sigma^{(k)} := \sqrt{\begin{array}{l}\lambda(\sigma^{(k)})^2 + (1-\lambda)(\sigma^{(k)*})^2 \\ \quad + \lambda(1-\lambda)(\mu^{(k)} - \mu^{(k)*})^2\end{array}}$$

Similar updates are used in [3].

4 Experiments

This section details how we tested the performance of our approach.

4.1 Dataset and evaluation

For evaluating our system, we used the videos from the TLD dataset [1] and other videos commonly used for evaluating trackers (Zhong et.al. [6]).

A typical measure for the performance of a tracking system is the PASCAL overlap measure [7]. A prediction is considered valid if the overlap ratio of the bounding box with the ground truth is greater than a threshold τ:

$$\frac{|B_{\text{prediction}} \cap B_{\text{ground truth}}|}{|B_{\text{prediction}} \cup B_{\text{ground truth}}|} > \tau$$

The value of τ chosen in [1] for comparing the TLD framework with other trackers is 0.25. We used this same threshold for our experiments.

We measured the performance of our system at two levels: we evaluated the performance of our new classifier alone (Section 4.3), and we evaluated the performance impact for the entire pipeline (Section 4.4). In both cases, we measured the performance in terms of precision, recall and f-score.

Finally, since the goal was to build a real-time system, we evaluated the speed of our algorithm in terms of frames per second.

4.2 Preliminary experiments

Before adopting the approach detailed in Section 3, we did some early experiments with popular keypoint detectors and descriptor extraction algorithms (FREAK, BRISK, SURF, ORB etc). The pipeline was:

1. On each frame, run the keypoint detector
2. Compute a descriptor for each keypoint
3. Classify the descriptors with a Naïve Bayes algorithm

Figure 1: Early experiments. Keypoints are detected on each frame, and then classified. The red points are negative, the blue points are positive. The green box is the ground truth bounding box for the object.

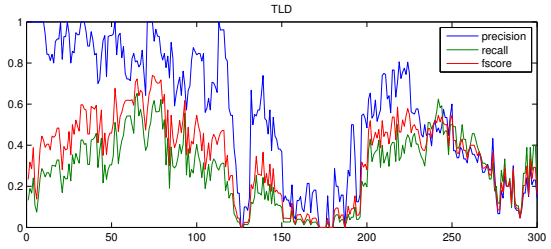
This method allowed us to perform detection without using a sliding window mechanism. However, there were limitations. First, keypoint detectors do not uniformly detect keypoints on the frame, and sometimes do not even detect any keypoint on the object of interest, making further detection impossible. Then, using available implementation of descriptors, we could not easily tune parameters such as the number of features. Finally, these descriptors are more appropriate for detecting characteristic points on an object, rather than full objects. For these reasons, we moved on to the previously described method, keeping the sliding-window mechanism but using CT-like descriptors for classification.

4.3 Classifier performance

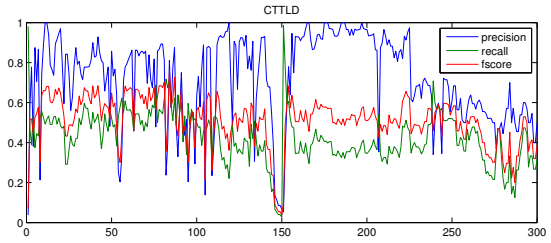
To compare our new classifier with the original ensemble classifier, we modified the TLD pipeline. On each frame, before the learning step, we replaced the predicted bounding box with the ground truth. This ensured that the training examples selected afterwards were similar. Otherwise the training sets for different classifiers would diverge, and the comparison would become irrelevant.

On each frame, we assigned a label to all the image patch descriptors using the PASCAL measure (Section 4.1), and compared it with the output of the classifier. Figure 2 shows an example of precision, recall and f-score as a function of time, using a descriptor length $n = 100$ and a learning rate $\lambda = 0.75$. With these settings, we achieved an f-score of 36.9% on average on the whole dataset (precision=47.8%, recall=42.2%), while the TLD framework achieved 30.8% (precision=48.2%, recall=25.8%). Our system significantly improves the classification recall, for an almost equivalent precision.

The introduction of decay (through the parameter λ) makes the classification performance much more stable over time. The example on Figure 2 shows that the original ensemble classifier fails to adapt quickly when the appearance of the object changes, because



(a) Original TLD



(b) Improved TLD

Figure 2: Precision (blue), recall (green) and f-score (red) as a function of time for the "panda" video from the TLD dataset (the frame number is shown on the x-axis). Our classifier produces a more stable f-score.

the weight of the first training examples remains too high. On the other hand, the f-score of our system is almost always above 0.2. This is important because the detector is most useful when the object is difficult to track, i.e. when the optical-flow tracker trajectory is most likely to drift from the object.

We tested several values of λ (Figure 3). Increasing λ reduces the vulnerability to temporary appearance changes of the object (blur, occlusion, etc), but the system adapts more slowly to long-term appearance changes (orientation change, shape change, etc). We obtained the best performance with $\lambda = 0.75$.

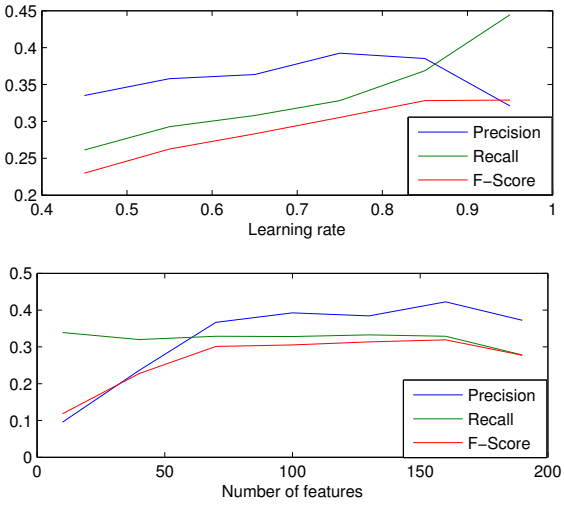


Figure 3: Top: Average precision/recall/f-score on the dataset as a function of the learning rate λ (number of features $n = 100$). Bottom: Average precision/recall/f-score as a function of n ($\lambda = 0.75$).

Finally, we observed how the performance varied with the number of features (Figure 3). If n is too low, the model is highly biased, and the descriptors don't capture enough information about the patches. We couldn't observe any clear sign of overfitting when increasing the number of features. However, high values of n require more computation, which is critical for a real-time system. We found $n = 100$ to be a good compromise between accuracy and speed.

4.4 Overall system performance

For evaluating the complete pipeline, we measured the precision, recall and f-score of the bounding box prediction for each video. This is different from the classifier evaluation, where we measured the precision, recall and f-score of the image patch classification for each frame. The precision P is the rate of valid bounding boxes among all the predictions, the recall R is the rate of predicted bounding boxes among all those that should have been predicted, and as usual, the f-score is defined as $F = \frac{2PR}{P+R}$. We obtained one value of P, R, and F for each video. A comparison of the two trackers is shown on figure 4.

Our system did slightly better overall than the original TLD framework, but since the average numbers are very close, tests on a more extensive dataset would be required for confirming the progress. In addition, both systems have their strengths and weaknesses. To understand the results, we did a qualitative analysis of the performance for each video. We observed that our system is more resistant to image blur, clutter and occlusion, whereas the original TLD framework is better for discriminating between objects with small variation of intensity, and more robust to illumination changes. Examples are given on Figure 5. A general observation

Sequence	TLD (Measured)	CT-TLD
david	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
jumping	1.00 / 0.87 / 0.93	1.00 / 0.98 / 0.99
pedestrian1	1.00 / 0.64 / 0.78	**1.00 / 0.83 / 0.91**
pedestrian2	0.77 / 0.70 / 0.73	**0.73 / 0.88 / 0.80**
pedestrian3	**0.88 / 1.00 / 0.94**	0.83 / 0.97 / 0.90
car	**0.95 / 1.00 / 0.97**	0.93 / 0.98 / 0.95
panda	0.52 / 0.46 / 0.49	**0.49 / 0.49 / 0.49**
animal	1.00 / 0.79 / 0.88	**1.00 / 0.82 / 0.90**
board	0.83 / 0.84 / 0.84	**0.99 / 0.86 / 0.92**
car11	0.93 / 0.94 / 0.94	**0.99 / 1.00 / 1.00**
caviar	**0.27 / 0.27 / 0.27**	0.71 / 0.16 / 0.26
faceocc2	0.99 / 0.99 / 0.99	**1.00 / 1.00 / 1.00**
girl	1.00 / 0.91 / 0.95	**0.96 / 0.95 / 0.95**
panda2	1.00 / 0.43 / 0.60	**1.00 / 0.59 / 0.74**
shaking	1.00 / 0.15 / 0.27	**0.82 / 0.35 / 0.49**
stone	**0.99 / 0.99 / 0.99**	1.00 / 0.88 / 0.94
singer1	**1.00 / 1.00 / 1.00**	**1.00 / 1.00 / 1.00**
mean	0.79 / 0.72 / 0.74	**0.81 / 0.74 / 0.76**

Figure 4: Comparison of our system (CT-TLD) with the original TLD. The numbers in each column are the precision, recall and f-score.

is that our system can deal with a wider range of appearance changes and movements, whereas the original TLD framework tends to be more precise (higher overlap when the tracking is successful). This can probably be explained by the nature of the descriptors: our descriptor corresponds to a summation of intensities over rectangles, whereas TLD uses more localized features (intensity difference between pairs of pixels).

Finally, we measured the speed of our system. Initially, the frame rate was low, but writing C code instead of MATLAB code increased the average frame rate to 11 fps on a 2.7 Ghz Intel i7 processor on a 640x480 video, with an initial bounding box of size 100x70. The initial TLD framework is faster: during our tests, we achieved 20 fps on average. Our system's performance still has the right order of magnitude for real-time operation, and we plan on optimizing it to get smoother tracking.

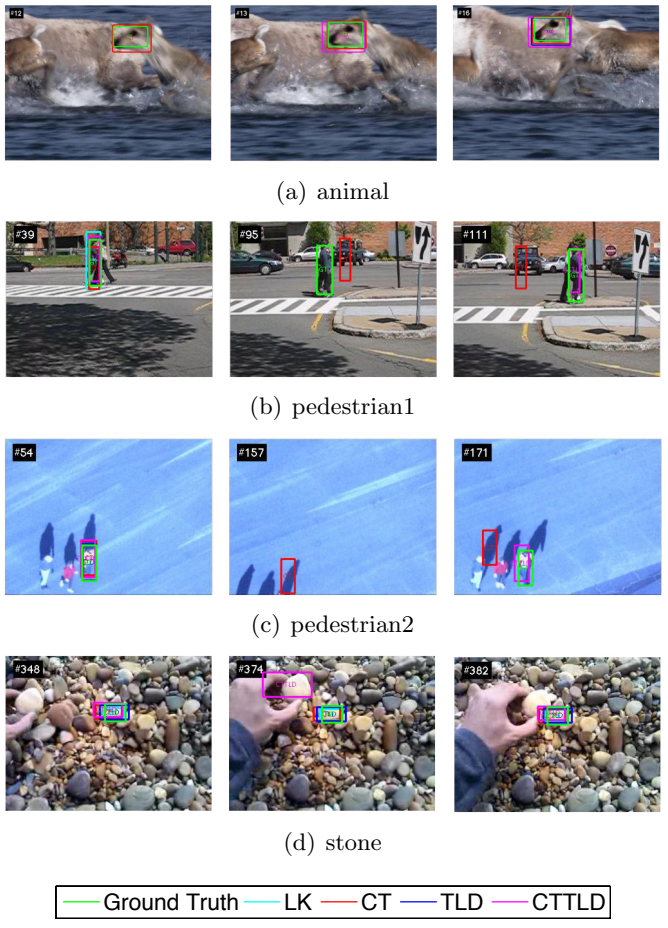


Figure 5: Superposition of the bounding boxes output by the optical-flow tracker alone (LK), the original CT system, the original TLD system, and our system (CTTLD). On (a) (blur example) and (b) (clutter example), our system recovers faster. On (c), TLD never recovers after the occlusion. On (d), our tracker jumps to another similar object.

5 Conclusion

Our classifier improves the tracking performance of the TLD framework in a large range of common situations. In some cases, such as when the intensity variance over the object is low, the original TLD system still remains better. Our learning algorithm does not seem to be the bottleneck. The introduction of a decay parameter makes the classification performance significantly more stable over time. On the other hand, our features do not capture the same kind of information as the small scale features used in the original TLD system. In the future, we intend to improve our descriptors by using a retinal topology inspired from FREAK keypoint descriptors [8], in order to capture both local and larger scale information about the object.

6 Acknowledgements

We would like to express our gratitude to Alexandre Alahi (Post-doc at the Stanford Computer Vision lab), who accepted to mentor our project.

7 Appendix

This project is done jointly with the CS231A class project for all members of the team.

References

[1] Z. Kalal, K. Mikolajczyk, and J. Matas. "Tracking-learning-detection". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 34.7 (2012), pp. 1409–1422.

[2] F. Pernici. "FaceHugger: The ALIEN Tracker Applied to Faces". In: *European Conference on Computer Vision (ECCV)* (2012).

[3] K. Zhang, L. Zhang, and M.H. Yang. "Real-time Compressive Tracking". In: *ECCV* (2012).

[4] P. Li, T.J. Hastie, and K.W. Church. "Very sparse random projections". In: *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2006, pp. 287–296.

[5] D. Achlioptas. "Database-friendly random projections: Johnson-Lindenstrauss with binary coins". In: *Journal of computer and System Sciences* 66.4 (2003), pp. 671–687.

[6] W. Zhong, H. Lu, and M.H. Yang. "Robust object tracking via sparsity-based collaborative model". In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1838–1845.

[7] M. Everingham et al. "The pascal visual object classes (voc) challenge". In: *International journal of computer vision* 88.2 (2010), pp. 303–338.

[8] P. Vandergheynst, R. Ortiz, and A. Alahi. "FREAK: Fast Retina Keypoint". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. IEEE. 2012, pp. 510–517.