CS 231A Project Proposal

**Title:** Implementation of TLD Long-term tracking framework.
**Group Members:** Jamie Ray, Jason Jong, Paul Chen

**Introduction:**

We are interested in the long term tracking problem in computer vision in which an object in a video stream is tracked over time. The object may change in scale, rotation, illumination, or even be partially occluded; where the detector is to recognize and track the object when it is present in the video stream. Preferably, a detector can process the video in real time and reliably track the object for an indefinite amount of time.

A working solution for the long-term tracking problem has many interesting applications. If an object of interest appears several times in a video, one could avoid having to manually scan the entire video and instead generate a timeline of when it can be found so the user can focus on those 'interesting' times. A wildlife camera looking for pandas could be processed to flag or report panda sightings for panda fans. We might imagine a multiple-tracking system that can tag each object so that users could easily find their favorite actor, player, car in a video, etc. Other potential applications include aerial surveillance by autonomous drones, video object stabilization, or automated scientific data collection.

In 2010, Kalal et al. proposed the Tracking-Learning-Detection (TLD) framework that synthesizes recent advances in both trajectory trackers and object detectors in a novel semi-supervised learning framework. In essence, given an initial bounding box of the object to track, TLD continually augments the detector with the new information from the incoming video with the help of the tracker.

The TLD framework achieved impressive results, but still has several limitations such as out-of-plane rotations, articulated objects, and single object tracking. We are interested in reproducing TLD's results, and in the process we hope to gain further insights for potential improvements. There exists an open source implementation, but by building as much as we can ourselves we can learn about several methods in computer vision and use the open source version as a benchmark.

**Implementation Details:**

As a group of three, it is convenient for us that this algorithm is defined by three subtasks: tracking, learning, and detection. There is of course interaction between them, but the components are distinct and can be built somewhat independently. We will plan to develop the system using MATLAB for its convenience in prototyping and visualization. Although one strength of the algorithm is a complexity footprint small enough to work in real time and/or on mobile devices, our initial focus will be on successful output rather than efficiency considerations.

The authors model the object as a set of size-normalized patches of constant aspect ratio. Some of these patches truly represent the object, while others are patches known to be background. By comparing patches in a new frame with this model, the system can predict locations for the object and compare them with tracking predictions. It is easy to update using good or bad patches found in future frames.

The tracker performs the task of estimating the motion of an object between two frames. There exists a significant literature on this problem with varying degrees of complexity and learning involved. The authors choose to estimate motion of points within the object's bounding box using a Lucas-Kanade tracker and then choose the median translation vector as their prediction (this is known as Median Flow). One must first find interest points to track between frames (the structure tensor can be useful to pick distinctive points in the object bounding box). Then similar points in the succeeding frame are searched for - various methods of comparison are possible, such as cross-correlation or the authors' pyramidal matching method, and should account for the hypothesis that the point won't have moved too far in the frame. Finally, while each point's motion is estimated independently, we have to use all these predictions to generate a transformed bounding box in the new frame. In this approach, the old box is essentially translated by the median of the per-point motion vectors (unless this motion is too large, in which case the tracker claims to have failed). It is unclear how other transformations of the patch are addressed - the paper suggests that scale change is possible, but a mechanism to estimate scale isn't discussed. Building a similar Median Flow tracker should be feasible, although we may be interested in allowing for more complex tracks involving e.g. affinities rather than just translation. In addition, we can easily replace this component with other tracking algorithms to understand the system's sensitivity to the tracker output.

The learning component is defined by two distinct goals of the detector (which are addressed by separate 'agents') - to correctly localize all appearances of the desired object, and to avoid predicting a location at which it doesn't appear (recall and precision). The PN learning framework leverages structure in the video stream to harvest examples that can help with each of these goals. Namely, this structure is the spatiotemporal continuity of object locations under conditions on the reliability of a track. New positive (P) examples come from the motion of the object through frames, which is estimated by the tracker and can thus find views that may not be classified correctly by the detector. Similarly, new negatives (N) come from the notion that the object is unique and can't appear in a drastically different position if the track is reliable, so patches far from the track location are taken as negatives. Thus, it is important to note that the PN agents may help the detector discriminate a unique object from other objects of the same class that appear elsewhere in the frame. A track is termed 'reliable' if it has ever contained a patch that was well-matched (at some threshold) with an early part (e.g. first half) of the object model (set of positive patches collected over time). Thus, it leverages the output of the tracker to choose potential patches for training the detector, and the output of the detector to decide whether a track is 'reliable' enough to learn from. It might be interesting to try to modify the learners. For example, we could try to help the agents realize when other objects of the same class appear and flag them as positive instead of negative. We could add different ways to measure the reliability, or have a parametrized tracker that could be updated via detector output.

The detector takes a cascaded approach to quickly eliminate the most unlikely swaths of the frame. The first stage examines patches of the same aspect ratio as the object in a pyramid of locations and scales throughout the image. It computes their variances, and discards any patch whose grayscale variance is significantly different from the model patches (computing the variance with integral images). The next stage is an ensemble classifier that averages predictions of base classifiers. Each base classifier is built of a set of binary pixel comparisons

within the patch (15x15 patch, 13 comparisons per classifier in the paper), and the bit vector encoded by the comparisons is used to index into a posterior distribution of detection probability. These distributions are stored in a table and updated with new examples when they are added by the PN agents. However, it only learns from failures - if the ensemble correctly predicts the new examples, the posteriors aren't changed. The ensemble score is thresholded, and high scores are passed to the final stage, a nearest-neighbor approach using the model patches. The distance is measured by normalized correlation coefficient between the query patch and the model patches, and the output score is a 'relative similarity' which compares the correlation with the best-matching positive patch to the best-matching negative patch. The patch with highest relative similarity is then the detector's prediction of the new object location.

We summarize subtasks of implementation here:

Tracker:

- Choose points within object to track
- Estimate motion vector for each point
- Combine estimates into global object transformation (location in new frame)
- Recognize tracking failures

Detector:

- Compute pyramid of possible patches
- Filter patches of incorrect variance
- Train ensemble classifier
- Filter patches with poor pixel-comparison codes
- Update ensemble with PN-harvested patches
- Update model with PN-harvested patches
- Compute correlations with model

Learners:

- Find 'reliable' tracks
- Choose negative examples away from track
- Choose positive examples close to track
- Warp positive examples for more views
- Pass to detector to update ensemble, model patches
- 'Forget' patches as necessary

**Weekly Milestones:**

1/31: Read and understand the TLD paper
2/7: Start implementing each of the three components.
2/13: Midterm Report, all three components should be done.
2/21: Testing of individual components
2/28: Integration into complete system
3/7: Testing and Evaluation
3/11: Project Presentation
3/19: Final Report

**Extensions:** There are also many ways we could work to modify the TLD algorithm. Each of the three tasks can be adjusted, and we would like to examine weaknesses of the system and be able to trace those problems back to a specific task in order to improve them.

**References:**

http://epubs.surrey.ac.uk/713800/1/Kalal-PAMI-2011%281%29.pdf
http://www.vision.ee.ethz.ch/boostingTrackers/
http://cmp.felk.cvut.cz/tld (code and data)