Tracking as Online Decision-Making:
Learning a Policy from Streaming Videos with Reinforcement Learning

James Supančič, III
University of California, Irvine
jsupanci@uci.edu

Deva Ramanan
Carnegie Mellon University
deva@cs.cmu.edu

Abstract

We formulate tracking as an online decision-making process, where a tracking agent must follow an object despite ambiguous image frames and a limited computational budget. Crucially, the agent must decide where to look in the upcoming frames, when to reinitialize because it believes the target has been lost, and when to update its appearance model for the tracked object. Such decisions are typically made heuristically. Instead, we propose to learn an optimal decision-making policy by formulating tracking as a partially observable decision-making process (POMDP). We learn policies with deep reinforcement learning algorithms that need supervision (a reward signal) only when the track has gone awry. We demonstrate that sparse rewards allow us to quickly train on massive datasets, several orders of magnitude more than past work. Interestingly, by treating the data source of Internet videos as unlimited streams, we both learn and evaluate our trackers in a single, unified computational stream.

1. Introduction

Object tracking is one of the basic computational building blocks of video analysis, relevant for tasks such as general scene understanding and perception-for-robotics. A particularly popular formalism is that of model-free tracking, where a tracker is provided with a bounding-box initialization of an unknown object. Much of the recent state-of-the-art advances make heavy use of machine learning [23, 54, 40, 56, 20], often producing impressive results by improving core components such as appearance descriptors or motion modeling.

Challenges: We see two significant challenges that limit further progress. First, the limited quantity of annotated video data impedes both training and evaluation. While image datasets involve millions of images for training and testing, tracking datasets have hundreds of videos. Lack of data seems to arise from the difficulty of annotating videos, as opposed to images. Second, as vision (re)-integrates with robotics, video processing must be done in an online, streaming fashion. This requires a tracker to make on-the-fly decisions such as when to re-initialize itself [51, 16, 30, 47, 21] or update its appearance model (the so-called template-update problem [54, 21]). Such decisions are known to be crucial in terms of final performance, but are typically hand-designed rather than learned.

Contribution 1 (interactive video processing): We show that reinforcement learning (RL) can be used to address both challenges in distinct ways. In terms of data, rather than requiring videos to be labeled with detailed bounding-boxes at each frame, we interactively train trackers with far more limited supervision (specifying binary rewards/penalties only when a tracker fails). This allows us to train on massive streaming datasets, 100X larger than prior work (Table 1).
prior work. Interestingly, RL also naturally lends itself to streaming “open-world” evaluation: when running a tracker on a never-before-seen video, the video can be used for both evaluation of the current tracker and for training (or refining) the tracker for future use (Fig. 1). This streaming evaluation allows us to train and evaluate models in an integrated fashion seamlessly. For completeness, we also evaluate our learned models on standard tracking benchmarks.

Contribution 2 (tracking as decision-making): In terms of tracking, we model the tracker itself as an active agent that must make online decisions to maximize its reward, which is (as above) the correctness of a track. Decisions ultimately specify where to devote finite computational resources at any point of time: should the agent process only a limited region around the currently predicted location (e.g., “track”), or should it globally search over the entire frame (“reinitialize”)? Should the agent use the predicted image region to update its appearance model for the object being tracked (“update”), or should it be “ignored”? Such decisions are notoriously complicated when image evidence is ambiguous (due to say, partial occlusions): the agent may continue tracking an object but perhaps decide not to update its model of the object’s appearance. Rather than defining these decisions heuristically, we will ultimately use data-driven techniques to learn good policies for active decision-making (Fig. 2).

Contribution 3 (deep POMDPs): We learn tracker decision policies using reinforcement learning. Much recent work in this space assumes a Markov Decision Process (MDP), where the agent observes the true state of the world [34, 56], which is the true (possibly 3D) location and unoccluded appearance of the object being tracked. In contrast, our tracker only assumes that it receives partial image observations about the world state. The resulting partially-observable MDP (POMDP) violates Markov independence assumptions: actions depend on the entire history of observations rather than just the current one [19, 43]. As in [15, 22], we account for this partial observability by maintaining a memory that captures beliefs about the world, which we update over time (Sec. 3). In our case, beliefs capture object location and appearance, and action policies specify how and when to update those beliefs (e.g., how and when should the tracker update its appearance model) (Sec. 4). However, policy-learning is notoriously challenging because actions can have long-term effects on future beliefs. To efficiently learn policies, we introduce frame-based heuristics that provide strong clues as to the long-term effects of taking a particular action (Sec. 5).

2. Related Work

Tracking datasets: Several established benchmarks exist for evaluating trackers [55, 23]. Interestingly, there is evidence to suggest that many methods tend to overfit due to aggressive tuning [40]. Withholding test data annotation and providing an evaluation server addresses this to some extent [26, 42]. Alternatively, we propose to evaluate on an open-world stream of Internet videos, making over-fitting impossible by design. It is well-known that algorithms trained on “closed-world” datasets (say, with centered objects against clean backgrounds [39, 4]) are difficult to generalize to “in-the-wild” footage [48]. We invite the reader to compare our videos in the supplementary material to contemporary video benchmarks for tracking.

Interactive tracking: Several works have explored interactive methods that use trackers to help annotate data. The computer first proposes a track. Then a human corrects major errors and retrains the tracker using the corrections [8, 2, 50]. Our approach is closely inspired by such active learning formalisms but differs in that we make use of minimal supervision in the form of a binary reward (rather than a bounding box annotation).

Learning-to-track: Many tracking benchmarks tend to focus on short-term tracking (< 2000 frames per video) [55, 23]. In this setting, a central issue appears to be modeling the appearance of the target. Methods that use deep features learned from large-scale training data perform particularly well [51, 52, 29, 27, 36]. Our focus on tracking over longer time frames poses additional challenges - namely, how to reinitialize after cuts, occlusions and failures, despite changes in target appearance [14, 54]. Several trackers address these challenges with hand-designed policies for model updating and reini-
tialization - TLD [21], ALIAN [41] and SPL [47] explicitly do so in the context of long-term tracking. On the other hand, our method takes a data-driven approach and learns policies for model-updating and reinitialization. Interestingly, such an “end-to-end” learning philosophy is often embraced by the multi-object tracking community, where strategies for online reinitialization and data association are learned from data [24, 56, 28]. Most related to us are [56], who use an MDP for multi-object tracking, and [22], who use RL for single target tracking. Both works use heuristics to reduce policy learning to a supervised learning task, aligning such an “end-to-end” learning philosophy with the multi-object tracking community, where strategies for online reinitialization and data association are learned from data. In our framework, the tracking, reinitialization, and appearance-update modules can be treated as black-boxes.

### 3. POMDP Tracking

We now describe our POMDP tracker, using standard notation where possible [43]. For our purposes, a POMDP is defined by a tuple of states ($\Omega$, $A$, $B$): At each frame $i$, the world state $\omega_i \in \Omega$ generates a noisy observation $o_i \in O$ that is mapped by an agent into an action $a_i \in A$, which in turn generates a reward.

In our case, the state $\omega_i$ captures the true location and appearance of the object being tracked in frame $i$. To help build intuition, one can think of the location as 2D pixel coordinates and appearance as a 2D visual template. Instead of directly observing this world state, the tracking agent maintains a belief over world states, written as

$$b_i = (\theta_i, h_i), \quad \text{where} \quad \theta_i \in \mathbb{R}^{h \times w \times f}, h_i \in \mathbb{R}^{H \times W}$$

where $\theta_i$ is a distribution over appearances (we use a point-mass distribution encoded by a single $h \times w$ filter defined on $f$ convolutional features), and $h_i$ is a distribution over pixel positions (encoded as a spatial heatmap of size $H \times W$). Given the previous belief $b_{i-1}$ and current observed video frame $o_i$, the tracking agent updates its beliefs about the current frame $b_i$. Crucially, tracker actions $a_i$ specify how to update beliefs, that is, whether to update the appearance model and whether to reinitialize by disregarding previous heatmaps. From this perspective, our POMDP tracker is a memory-based agent that learns a policy for when and how to update its own memory (Fig. 3).

Specifically, beliefs are updated as follows:

$$h_i = \begin{cases} TRACK(h_{i-1}, \theta_{i-1}, o_i) & \text{if } a_i^{(1)} = 1 \\ REINIT(\theta_{i-1}, o_i) & \text{otherwise} \end{cases}$$

$$\theta_i = \begin{cases} UPDATE(\theta_{i-1}, h_i, o_i) & \text{if } a_i^{(2)} = 1 \\ \theta_{i-1} & \text{otherwise} \end{cases}$$

where $a_i = (a_i^{(1)}, a_i^{(2)})$. Object heatmaps $h_i$ are updated by running the current appearance model $\theta_{i-1}$ on image regions $o_i$ near the target location previously predicted using $h_{i-1}$ (“tracking”). Alternatively, if the agent believes it has lost track, it may globally evaluate its appearance model (“reinit”). The appearance model is then “updated” with the currently-predicted image region or possibly unchanged. In our framework, the tracking, reinitialization, and appearance-update modules can be treated as black-
boxes given the above functional form. We will discuss particular implementations shortly (with copious details in the supplementary material), but first, we focus on the heart of our RL approach: a principled framework for learning to take appropriate actions. To do so, we begin by reviewing standard approaches for decision-making.

**Online heuristics:** The simplest approach to picking actions might be to pre-define heuristics functions which estimate the correct action. For example, many trackers reinitialize whenever the maximum confidence of the heatmap is below a threshold. Let us summarize the information available to the tracker at frame $i$ as a “state” $s_i$, which includes the previous belief $b_{i-1}$ (the previous heatmap and appearance model) and current image observation $o_i$.

$$a^*_i = \text{Heur}_on(s_i), \quad \text{where} \quad s_i = (b_{i-1}, o_i) \quad (2)$$

**Offline heuristics:** A generalization of the above is to use offline training data to build better heuristics. Crucially, one can now make use of ground-truth training annotations as well as future frames to better gauge the impact of possible actions. We can write this knowledge as the true world as well as future frames to better gauge the impact of possible actions in that next frame. This intuition is formalized through the well-known Bellman equations that recursively define Q-functions to return a goodness score (the expected future reward) for each putative action $a$:

$$Q(s_i, a_i) = R(s_i) + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1}), \quad (4)$$

where $s_i$ includes both the tracker belief state and image observation, and $R(s_i)$ is the reward associated with the reporting the estimated object heatmap $h_i$. We let $R(s_i) = 1$ for a correct prediction and 0 otherwise. Finally, $\gamma \in [0, 1]$ is a discount factor that trades off immediate vs future per-frame rewards. Given a tracker state and image observation $s_i$, the optimal action is readily computed from the Q-function:

$$a^* = \arg \max_a Q(s_i, a)$$

**Q-learning:** Traditionally, Q-functions are iteratively learned with Q-learning [43]:

$$Q(s_i, a_i) = \epsilon Q(s_i, a_i) + \ldots$$

$$\alpha \left( R(s_i) + \gamma \max_{a_{i+1}} Q(s_{i+1}, a_{i+1}) - Q(s_i, a_i) \right) \quad (5)$$

where $\alpha$ is a learning rate. To handle continuous belief states, we approximate the Q-function with a CNNs:

$$Q(s_i, a_i) \approx CNN(s_i, a_i)$$

that processes states $s_i$ and binary actions $a_i$ to return a scalar value capturing the expected future reward for taking that action. Recall that a state $s_i$ encodes a heatmap and an appearance model from previous frames and an image observation from the current frame.

### 4. Interactive Training and Evaluation

In this section, we describe our procedure for interactively learning CNN parameters $w$ (that encode tracker action policies) from streaming video datasets. To do so, we gradually build up a database of experience replay memories [1, 34], which are a collection of state-action-reward-nextstate tuples $D = \{(s_i, a_i, r_i, s_{i+1})\}$:

$$L(w, D) = \left( r_i + \max_{a_{i+1}} Q_{w}(s_{i+1}, a_{i+1}) - Q_{w}(s_i, a_i) \right)^2 \quad (6)$$

Gradient descent on the above objective is performed as follows: given a training sample $(s_i, a_i, r_i, s_{i+1})$, first perform a forward pass to compute the current estimate $Q_{w}(s_i, a_i)$ and the target: $r_i + \max_{a_{i+1}} Q_{w}(s_{i+1}, a_{i+1})$. Then backprop through the weights $w$ to reduce $L(w)$.

The complete training algorithm is written in Alg. 1. We choose random videos from the Internet by sampling phrases using WordNet [32]. Given the sampled phrase and video, an annotator provides an initialization bounding box and begins running the existing tracker. After tracking, the annotator marks those frames (in strides of 50) where the tracker was incorrect using a standard 50% intersection-over-union threshold. Such binary annotation (“correct” or “failed”) requires far less time per frame than bounding-box annotation: we design a real-time interface that simply requires a user to depress a button during tracker failures. By playing back videos at a (user-selected) sped-up frame rate, users annotate 1200 frames per minute on average (versus 34 for bounding boxes). Annotating our entire dataset of 10
Algorithm 1: Our final learning algorithm interactively labels a streaming dataset of videos while learning a tracker action policy \(Q_w\). Given a video, steps 5 through 11 run a tracker according to the current policy. An annotator then assesses the binary reward \(r_i\) (correctness) for the highest-scoring bounding box extracted from the heatmap \(h_i\) by using an intersection-over-union threshold. Annotated frames (and their associated state-action-reward tuples) are added to our experience replay database \(D\). We then sample a minibatch of replay memories and update the action policy \(w\) with backprop (Eq. 6).

5. Implementation

**TRACK/REINIT:** Our TRACK function (Eq. 2) takes as input the previous heatmap \(h_{i-1}\), appearance model \(\theta_{i-1}\), image observation \(o_i\), and produces a new heatmap for the current frame. We make use of the state-of-the-art fully-convolutional tracker FCNT [51] and refer to the reader to that work for precise implementation details, but summarize them here: TRACK crops the current image \(o_i\) to a region of interest (ROI) around the most-likely object location from the previous frame (the argmax of \(h_{i-1}\)). This ROI is resized to a canonical image size (e.g., \(224 \times 224\)) and processed with a CNN (VGG16) to produce a convolutional feature map. The object appearance model \(\theta_{i-1}\) is represented as a filter on this feature map, allowing one to compute a new heatmap with a convolution. When the tracker believes it has lost track, the REINIT model simply processes a random ROI.

**Q-function CNN:** Recall that our Q-functions process a tracker state \(s_i = ((h_{i-1}, \theta_{i-1}), o_i)\) and a candidate action \(a_i\), to return a scalar representing the expected future reward of taking that action (Fig. 4 and Eq. 6). In practice, we define two separate Q-functions for our two binary decisions (TRACK/REINIT and UPDATE/IGNORE). To plug into standard learning approaches, we formally define a single Q function as the sum of the two functions, implying that the optimal decisions can be computed independently for each. We found it sufficed to condition on the heatmap \(h_{i-1}\) and implemented each function as a CNN, where the first two hidden layers are shared across the two functions. Each shared hidden layer consists of \(16 \times 16 \times 4\) convolution followed by ReLU and \(4 \times 4\) max pooling. For each decision, an independent fully-connected layer ultimately predicts the expected future reward. When training the Q-function using experience-replay, we use \(\gamma = .95\), a learning rate of 1e-4, a momentum of .9 and 1e-8 weight decay.

**Offline heuristics:** Deep Q-learning is known to be unstable, and we found good initialization was important for reliable convergence. We initialize the Q-functions in Eq. 6 (which specify the goodness of particular actions) with the offline heuristics from Eq. 3. Specifically, the heuristic action \(a_i^*\) has a goodness of 1 (scaled by future discount rewards), while other actions have a goodness of 0:

\[
Q_{\text{init}}(s_i, a_i) \leftarrow \mathbf{1}[a_i = a_i^*] \sum_{j \geq i} \gamma^{j-i} \tag{7}
\]

where \(\mathbf{1}\) denotes the identity function. In practice, we found it useful to minimize a weighted average of the true
Online Tracking Benchmark [55]

Figure 5. OTB-2013: Our p-tracker (solid line) compared to FCNT [51], MUSTer [16], LTCT [30], TLD [21] and SPL [47] (dashed lines). In general, many videos are easy for modern trackers, implying that a method’s rank is determined by a few challenging videos (such as the confetti celebration and fireworks on the bottom left). Our tracker learns an UPDATE and REINIT policy that does well on such videos.

In general, many videos are easy for modern trackers, implying that a method’s rank is determined by a few challenging videos (such as the confetti celebration and fireworks on the bottom left). Our tracker learns an UPDATE and REINIT policy that does well on such videos.

loss in Eq. 6 and a supervised loss $\left(Q_w - Q_{init}\right)^2$, an approach related to heuristically-guided Q-learning [20, 7]. Defining a heuristic for TRACK vs REINIT is straightforward: $a^*_i$ should TRACK whenever the peak of the reported heatmap overlaps the ground-truth object on frame $i$. Defining a heuristic for UPDATE vs IGNORE is more subtle. Intuitively, $a^*_i$ should UPDATE appearance with frame $i$ whenever doing so improves the confidence of future ground-truth object locations in that video. To operationalize this, we update the current appearance model $\theta_i$ on samples from frame $i$ and compute $\Delta_+$, the number of future frames where the updated appearance increases confidence of ground-truth locations (and similarly $\Delta_-$, the number of frames where the update decreases confidence of track errors). We set $a^*_i$ to update when $\Delta_+ + \Delta_- > .5N$, where N is the total number of future frames.

6. Experiments

Evaluation metrics: Following established protocols for long-term tracking [21, 47], we evaluate $F1 = \frac{2pr}{p+r}$, where precision ($p$) is the fraction of predicted locations that are correct and recall ($r$) is the fraction of ground-
and Fig. 7 for discussion.

Comparing to two classic long-term trackers: TLD [2013] (Figs. 5 and 6) performs competitively (Fig. 7), but tends to over-predict occlusions (which rarely occur in short-term videos). But fortunately, we can learn a dedicated policy for short-term tracking (p-track-short) by applying reinforcement learning (Alg. 1) on short-term training videos. For each test video in OTB-2013, we learn a policy using the 40 most dissimilar videos in VOT-2016 (and vice-versa). We define similarity between videos to be the correlation between the average (ground-truth) object image in RGB space. This ensures that, for example, we do not train using Tiger1 when testing on Tiger2. Even under this controlled scenario, p-track-short significantly outperforms prior work on both OTB-2013 (Figs. 5 and 7) and VOT-2016 (Fig. 8).

Long-term baselines: For the long-term setting, we compare to two classic long-term trackers: TLD [21] and SPL [47]. Additionally, we also compare against short-term trackers with public code that we were able to adapt: FCNT [51], MUSTer [16], and LTCT [30]. Notably, all these baselines use hand-designed heuristics for deciding when to appearance update and reinitialize.

Long-term videos: Qualitatively speaking, long term videos from the internet are much more difficult than standard benchmarks (c.f. Fig 5 and Fig. 6). First, many standard benchmarks tend to contain videos that are easy for most modern tracking approaches, implying that a method’s rank is largely determined by performance on a small number of challenging videos. The easy videos focus on iconic [39, 4] views with slow motion and stable light-video sequences. In contrast, Internet videos vary widely in difficulty, we supplement averages with boxplots to better visualize performance skew. When evaluating results on standard benchmarks, we use the default evaluation criteria for that benchmark.

6.1. Comparative Evaluation

Short-term benchmarks: While our focus is long-term tracking, we begin by presenting results on existing benchmarks that tend to focus on the short-term setting – the Online Tracker Benchmark (OTB-2013) [55] and Visual Object Tracking Benchmark (VOT-2016) [23]. Our policy tracker (p-track-long), trained on Internet videos, performs competitively (Fig. 7), but tends to over-predict occlusions (which rarely occur in short-term videos). But fortunately, we can learn a dedicated policy for short-term tracking (p-track-short) by applying reinforcement learning (Alg. 1) on short-term training videos. For each test video in OTB-2013, we learn a policy using the 40 most dissimilar videos in VOT-2016 (and vice-versa). We define similarity between videos to be the correlation between the average (ground-truth) object image in RGB space. This ensures that, for example, we do not train using Tiger1 when testing on Tiger2. Even under this controlled scenario, p-track-short significantly outperforms prior work on both OTB-2013 (Figs. 5 and 7) and VOT-2016 (Fig. 8).

Long-term baselines: For the long-term setting, we compare to two classic long-term trackers: TLD [21] and SPL [47]. Additionally, we also compare against short-term trackers with public code that we were able to adapt: FCNT [51], MUSTer [16], and LTCT [30]. Notably, all these baselines use hand-designed heuristics for deciding when to appearance update and reinitialize.

Long-term videos: Qualitatively speaking, long term videos from the internet are much more difficult than standard benchmarks (c.f. Fig 5 and Fig. 6). First, many standard benchmarks tend to contain videos that are easy for most modern tracking approaches, implying that a method’s rank is largely determined by performance on a small number of challenging videos. The easy videos focus on iconic [39, 4] views with slow motion and stable light-video sequences. In contrast, Internet videos vary widely in difficulty, we supplement averages with boxplots to better visualize performance skew. When evaluating results on standard benchmarks, we use the default evaluation criteria for that benchmark.

6.2. System Diagnostics

We now provide a diagnostic analysis of various components of our system. We begin by examining several alternative strategies for making sequential decisions (Fig. 10).

Online vs offline heuristics: We begin by analyzing the online heuristic actions of our baseline tracker, FCNT. FCNT updates an appearance model when the predicted heatmap location is above a threshold, and always tracks without reinitialization. This produces a F1 score of .09. Next, we use offline heuristics to learn the best action to
We compare the final p-trackers built after seeing 16,000 videos. In Fig. 11, our p-tracker’s performance increases as it learns its policy using additional Internet videos. Above, we plot the distribution of F1 scores on our hold-out test data, at various stages of training. At initialization, the average F1 score was 0.08. After seeing 16,000 videos, it achieves an average F1 score of 0.30. Our results suggest that large-scale training, made possible through interactive annotation, is crucial for learning good decision policies.

Figure 11. Our p-tracker’s performance increases as it learns its policy using additional Internet videos. Above, we plot the distribution of F1 scores on our hold-out test data, at various stages of training. At initialization, the average F1 score was 0.08. After seeing 16,000 videos, it achieves an average F1 score of 0.30. Our results suggest that large-scale training, made possible through interactive annotation, is crucial for learning good decision policies.

take. These correspond to tracking when the predicted object location is correct, and updating if the appearance model trained on the new patch produces higher scores for ground-truth locations. We train a classifier to predict these actions using the current heatmap. When this offline trained classifier is run at test-time, F1 improves to .13 with the track heuristic and .14 with the update heuristic, and .20 if both are used.

**FCNT vs CCOT:** We use FCNT for our ablative analysis; initializing p-track using CCOT’s more complex online heuristics proved difficult. However, our final system uses only our proposed offline heuristics, so we can nonetheless train it using CCOT’s TRACK, REINIT, and UPDATE functions. In Fig. 9 we compare the final p-trackers built using FCNT’s functions against those built using CCOT’s functions. As consistent with prior work, we find that CCOT improves overall performance (from .30 to .36).

**Q-learning:** Finally, we use Q-learning to refine our heuristics (Eq. 6), noticeably improving the F1 score to .30. Learning the appearance update action seems to have the most significant effect on performance, producing an F1 score of .28 by itself. During partial occlusions, the tracker learns to delicately balance between appearance update and drift while accepting a few failures to avoid the cost and risk of reinitialization. Overall, the learned policy dramatically outperforms the default online heuristics, tripling the F1 score from 9% to 30%!

**Training iterations:** In theory, our tracker can be interactively trained on a never-ending stream. However, in our experiments, Q-learning appeared to converge after seeing between 8,000 and 12,000 videos. Thus, we choose to stop training after seeing 16,000 videos. In Fig. 11, we plot performance vs training iteration.

**Computation:** As mentioned previously, comparatively slower trackers typically perform better [23]. On a Tesla K40 GPU, our tracker runs at approximately 10 fps. While computationally similar to [51], we add the ability to recover from tracking failures by reinitializing through detection. To do so, we learn an attention policy that efficiently balances tracking vs reinitialization. Tracking is fast because only a small region of interest (ROI) need be searched. Rather than searching over the whole image during reinitialization, we select a random ROI (which ensures that our trackers operate at a fixed frame rate). In practice, we find that target is typically found in \( \approx 15 \) frames.

**Conclusions:** We formulate tracking as a sequential decision-making problem, where a tracker must update its beliefs about the target, given noisy observations and a limited computational budget. While such decisions are typically made heuristically, we bring to bear tools from POMDPs and reinforcement learning to learn decision-making strategies in a data-driven way. Our framework allows trackers to learn action policies appropriate for different scenarios, including short-term and long-term tracking. One practical observation is that offline heuristics are an effective and efficient way to learn tracking policies, both by themselves and as a regularizer for Q-learning. Finally, we demonstrate that reinforcement learning can be used to leverage massive training datasets, which will likely be needed for further progress in data-driven tracking.

**Acknowledgments:** This research was supported in part by the National Science Foundation (NSF) under grants CNS-1518865 and IIS-1618903, and the Defense Advanced Research Projects Agency (DARPA) under Contract No. HR001117C0051. Additional support was provided by the Intel Science and Technology Center for Visual Cloud Systems (ISTC-VCS). Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the view(s) of their employers or the above-mentioned funding sources.
References


