Discriminatively Trained Particle Filters for Complex Multi-Object Tracking

Rob Hess and **Alan Fern**
School of EECS, Oregon State University, Corvallis, OR, 97331, USA
{hess,afern}@eecs.oregonstate.edu

Abstract

This work presents a discriminative training method for particle filters in the context of multi-object tracking. We are motivated by the difficulty of hand-tuning the many model parameters for such applications and also by results in many application domains indicating that discriminative training is often superior to generative training methods. Our learning approach is tightly integrated into the actual inference process of the filter and attempts to directly optimize the filter parameters in response to observed errors. We present experimental results in the challenging domain of American football where our filter is trained to track all 22 players throughout football plays. The training method is shown to significantly improve performance of the tracker and to significantly outperform two recent particle-based multi-object tracking methods.

1. Introduction

Particle filtering is a widely used framework for visual object tracking that is highly extensible and offers the flexibility to handle non-linearity and non-normality in the object models. In recent years, many new particle filter-based approaches have been proposed to solve difficult multi-object tracking problems [13, 9, 19]. However, most of this work has paid little attention to how to best tune the parameters of the proposed models and has instead relied on some combination of manual tuning and simple generative learning to set these parameters. These approaches, though, are often ineffective and/or extremely labor intensive.

In this work, we describe a conceptually simple particle filtering framework for multi-object tracking (Section 3). This framework is easy to extend and customize because it allows the user to define rich sets of features to capture essential properties of the tracking domain. Our main contribution is an error-driven, discriminative algorithm for training this model's (Section 4). Our training is tightly integrated into the filtering process and attempts to optimize parameters in response to observed tracking errors. In addition, we describe an important practical approximation to this algorithm (Section 5) that can significantly reduce training time for domains where many objects must be tracked.

Our decision to use a discriminative approach is motivated by several factors. First, there is a growing body of empirical evidence from a number of fields [11, 1, 4, 20] along with theoretical results [12] suggesting that discriminative training outperforms generative training when enough data is available. Second, unlike generative training methods, our discriminative approach does not make strong independence assumptions about the features, which would ignore important dependencies. Third, our discriminative approach is aimed at directly solving the problem we really care about: maximizing the accuracy of the learned filter. In contrast, generative training attempts to achieve this objective indirectly by maximizing the joint likelihood of the training data, which does not always relate well to filtering accuracy when the assumed model is wrong.

We apply our approach to tracking the 22 players and one referee in low-resolution American football video (Section 6). This problem is extremely challenging due to the erratic movement of players, the complexity of interactions that sometimes involve upwards of five or ten players, and the strong dependence of player behavior on the player's type and the stage of the play. To date, there has been very limited success in tracking for American football, and our experiments show that some state-of-the-art multi-object tracking methods do not work well in this domain. In comparison, we show that filters trained using our method substantially outperform untrained filters and these other methods and, to the best of our knowledge, represent the state-of-the-art in tracking in the American football domain.

2. Related Work

Here we discuss related work on particle filter-based multi-object tracking and discriminative filter training.

Particle Filter-Based Multi-Object Tracking: Since particle filter-based visual object tracking was first proposed, much progress has been made on tracking single objects using particle filters. Tracking multiple objects, however, poses the challenge of dealing with the high-dimensionality of the state space, which grows with the

number of objects. A naive solution is to use an independent single-object particle filter for each object, but this can break down when similar objects interact, leading to objects "hijacking" filters from other objects.

Some notable, but far from exhaustive, recent attempts to improve upon this naive approach include Okuma *et al*.'s boosted particle filter [13], which attempts to avoid hijacking by using a Haar-style object detector to terminate and resume tracks during and after an interaction; Khan *et al*.'s MCMC-based particle filter [9], which tracks objects in their joint state space and uses a Markov random field (MRF), built at each time step, that helps prevent hijacking by enforcing separation between nearby objects while allowing far apart objects to be tracked independently; and Yu and Wu's mean field Monte Carlo algorithm [18, 19], which also uses an MRF to enforce object separation but uses Monte Carlo variational inference.

While each of the above methods has been shown to outperform the naive approach, there is still much room for further improvement. For example, Okuma *et al*.'s method does not explicitly reason about object interactions, but rather attempts to improve on the purely naive approach by using more powerful proposal and observation distributions. As such the approach is prone to losing object identities and locations when tracks are terminated. Khan *et al*.'s method tracks in the objects' joint state space and thus, in our experience, does not scale well when the number of objects is large and more uncertainty exists about objects' locations due, for example, to erratic object motion. Similarly, the joint inference approach employed in Yu and Wu's method suffers from quadratic complexity in the total number of particles used to track all of the objects. For both of these methods, it can be difficult to set the parameters of the interacting MRF model components to maximize accuracy, and they are currently set manually and/or learned generatively.

Discriminative Filter Training: There has been much recent work on discriminative training of sequential filters. For example, discriminatively trained conditional random fields (CRFs) for sequence data [10] have been shown to outperform generatively trained hidden Markov models (HMMs) in many domains. Collins has also proposed a generalized perceptron algorithm for sequential data [2] to train HMMs for natural language problems. This work has been further extended to large-margin discriminative training of HMM-style sequential models [15, 16]. Unfortunately, all of these methods assume small state spaces where exact, efficient filtering is possible, *e.g*. via dynamic programming, and hence are not directly applicable to object tracking.

In more recent work, discriminative training was used to set the noise parameters of continuous state extended Kalman filters (EKFs) for robot localization [1]. However, it is generally recognized that EKFs are not powerful enough for complex object tracking due to the normality assumptions they make. Our work can be seen as extending this approach to a more general class of process models.

In more closely related work, Limketkai *et al*. train large state-space CRFs for robot localization using Collins' perceptron algorithm within a particle filtering framework [11]. As far as we are aware, this is the only prior work to attempt any form of discriminative training of particle filters.

All of the above discriminative methods can be viewed as driving the learning process by iteratively using the current filter to perform inference on a set of training sequences and then updating parameters based on some measure of the disparity between the filter's output and the desired output. For example, Limketkai *et al*.'s approach computes a MAP state sequence for each training example using the current filter and then attempts to adjust filter parameters so that the ground truth sequences become more probable than the current MAP sequences. However because filters sometimes fail badly, often early in training, the filter parameter updates these approaches make can be dominated by the portion of the sequence that occurs after the failure rather than focused on correcting the original point of failure. Our experience in multi-agent tracking suggests that such parameter updates can be counterproductive.

To address this issue, the perceptron algorithm has been extended to focus training directly on points of filter failure. For example, recent work has proposed performing updates based only on the part of the predicted sequence up to and including the first failure [3]. Later work has extended this idea by updating at successive points of failure [5, 17]. The learning approach we describe in Section 4 can be viewed as extending these failure-driven approaches to the particle filtering framework.

Finally, we note that there has been work that uses discriminative learning to produce certain individual components of particle filters, for example by learning object-detection classifiers to be included as part of the particle filter's proposal distribution [13]. However, such indirect learning approaches never take into account the actual filter performance and hence offer no guidance to improve the filter further after the original components have been learned. Our approach is complimentary, as such learned components could be included in a filter and the filter then further improved based on actual filtering performance.

3. Psuedo-Independent Log-Linear Filters

Below we first review the standard Bayesian particle filter, and then describe our specific particle filter-based multi-object tracking framework, which uses pseudo-independent filters parameterized by log-linear models.

Particle Filters: Particle filtering is a Monte Carlo approximation to the optimal Bayesian filter [6], which monitors the posterior probability of a first-order Markov process

through the following formula:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \alpha p(\mathbf{y}_t|\mathbf{x}_t) \int_{\mathbf{x}_{t-1}} p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}). \quad (1)$$

Here, $\mathbf{x}_t$ is the process state at time t, $\mathbf{y}_t$ is the observation, $\mathbf{y}_{1:t}$ is all of the observations through time t, $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is the process dynamical distribution, $p(\mathbf{y}_t|\mathbf{x}_t)$ is the observation likelihood distribution, and α is a normalizing factor.

The integral in (1) does not have a closed form solution, except in the most basic cases, so particle filters are used to approximate (1) using a set of weighted samples $\{\mathbf{x}_t^{(i)}, \pi_t^{(i)}\}_{i=1,\ldots,n}$, where each $\mathbf{x}_t^{(i)}$ is an instantiation of the process state, known as a particle, and the $\pi_t^{(i)}$'s are the corresponding particle weights. Under this representation, the approximation to the Bayesian filtering equation (1) is

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \alpha\, p(\mathbf{y}_t|\mathbf{x}_t) \sum_{i=1}^{n} \pi_{t-1}^{(i)}\, p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)}). \quad (2)$$

To implement a standard particle filter, one must choose a state representation $\mathbf{x}_t$, which, in the case of object tracking might include object locations, scales, etc. In addition, one must design three distributions: the process dynamical distribution, $p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})$, which, in object tracking, describes how objects move between time steps; the proposal distribution, $q(\mathbf{x}_t^{(i)}|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})$, which is sampled at each time step to update the particle distribution; and the observation likelihood distribution, $p(\mathbf{y}_t|\mathbf{x}_t^{(i)})$, which, in tracking, describes how objects appear within the video data, $\mathbf{y}_t$.

At each time step, given the previous particle set $\{\mathbf{x}_{t-1}^{(i)}, \pi_{t-1}^{(i)}\}$, a basic sequential importance resampling [6] particle filter updates the particles as follows:

1. Sample n particles $\mathbf{x}_{t-1}^{(i)}$ with replacement from current particle set according to probabilities $\pi_{t-1}^{(i)}$.
2. Generate an updated particle set by sampling from the proposal distribution, $\mathbf{x}_t^{(i)} \sim q(\mathbf{x}_t^{(i)}|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})$.
3. Reweight each particle according to the following formula and normalize so that the $\pi_t^{(i)}$ sum to 1:

$$\pi_t^{(i)} \propto \frac{p(\mathbf{y}_t|\mathbf{x}_t^{(i)}) p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(i)})}{q(\mathbf{x}_t^{(i)}|\mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})}. \quad (3)$$

In tracking and and many other applications, it is typical to estimate the process state at each time step as the sample mean of the particles $\hat{E}[\mathbf{x}_t] = \sum_{i=1}^{n} \pi_t^{(i)} \mathbf{x}_t^{(i)}$.

Pseudo-Independent Filters: When multiple interacting objects must be tracked, the joint state space of those objects is high dimensional and renders the straightforward application of particle filtering impractical. As discussed in Section 2, extensions to the basic particle filtering framework have been explored that utilize more complex MRF dynamic models to capture important object interactions in a more tractable way. In contrast to those methods, we use a simple particle filtering framework for multi-object tracking. Specifically, we assign each object of interest its own single-object particle filter. These filters are not completely independent, however, but are pseudo-independent, in the sense that each tracker estimates only a single object's state but has the previous state estimates of other trackers available as observations to use during inference.

Unlike the more sophisticated MRF approaches, our pseudo-independent approach does not increase the computational complexity of filtering over that of purely independent filters, but it still allows some amount of dependency between trackers through observations of one another's internal states from previous time steps. While the pseudo-independent approach is more restricted in the types of dependencies it can represent, we believe that it will be sufficient for many applications because the log-linear filtering framework described below enables straightforward representation of rich features of both individual objects and relations among objects, which the pseudo-independent filtering approach can exploit for improved performance.

Log-Linear Filters: To allow for maximum flexibility, we wish to allow the designer to devise arbitrary features that can capture joint properties of a tracked object's state, its observations, and the previous state estimates of other objects and to incorporate these features into the individual filters. For example, by including features that measure the distance between an object's proposed state and the predicted locations of other objects, it is possible to bias the filter against allowing two objects occupying the same space.

While in principle one can attempt to define the dynamic and observation distributions in terms of such features, doing so is quite difficult in practice. In particular, one must decide how to weight the potentially many features against one another and/or make the assumption that features are independent in order to support traditional generative learning. In this work, we instead utilize a more flexible combination of features based on log-linear modeling.

From an algorithmic perspective, the main difference between our particle filtering framework and the traditional framework is in the way we compute particle weights. The traditional reweighting computation in equation (3) is the result of trying to account for dynamics, observations, and proposal bias all at once by combining their associated generative model terms. In contrast, we attempt to learn a single reweighting function defined in terms of arbitrary features, which can encompass all of those terms but is not restricted to the specific form of equation (3).

In particular, our particle weighting function takes the form of a log-linear combination of weighted features:

$$\pi_t^{(i)} \propto \exp\Big(\sum_j w_j f_j(\mathbf{y}_t, \mathbf{x}_t^{(i)})\Big). \quad (4)$$

Here the $f_j(\cdot)$'s are user-defined feature functions and the w_j's are the weights of the features, which parameterize the model. By including features that correspond to the logarithms of the process dynamics, observation likelihood, and proposal distribution terms in equation (3), the log-linear model can be made strictly more expressive than the traditional formulation. Note that, for our pseudo-independent filters, each $\mathbf{y}_t$ contains traditional observation data as well as information about the previous states of other filters to allow the features to model interactions between objects. In Section 6, we describe a number of feature functions for multi-object tracking in the American football domain.

There are two ways to view this log-linear filtering model. From an algorithmic perspective, it can be seen as a more flexible parameterization of the standard particle filtering algorithm. From a modeling perspective, it can be viewed as an undirected, log-linear probabilistic model over sequences—as in the work on CRF-filters [11]—for which a particle filtering inference procedure based on nonparametric belief propagation can be derived. In either case, the ultimate goal is to learn feature weights that optimize filter performance as described next.

4. Error-Driven Discriminative Filter Training

We take a supervised approach to training our individual log-linear particle filters. The training set contains examples of the form $(\mathbf{x}^*_{0:T}, \mathbf{y}_{1:T})$, where $\mathbf{y}_{1:T}$ is an observation sequence, for example, a video of a football play, and $\mathbf{x}^*_{0:T}$ is the ground-truth or target state sequence, for example, the trajectory of a particular player in the football play. The goal is to optimize the filter weights so that the filter output is as close to the target state sequences as possible.

The basic training approach iterates through the training examples and, for each one, calls Algorithm 1, which produces updated filter weights that are used as the initial weights for the next call. The iteration continues until a maximum number of steps is reached or performance no longer improves. In practice, we average the weights learned across iterations to arrive at the final weight vector, which is a common technique that has been shown to improve performance of online learning [2].

Algorithm 1 is identical to the log-linear filter of Section 3 with the addition of training mechanisms in lines 6 through 12. Intuitively, each call to this algorithm monitors the filter's performance on the current example and tunes the feature weights each time the filter fails. Specifically, at each time step, a new particle set is proposed and reweighted according to the log-linear model. After particle reweighting, the filter's current state estimate $\hat{\mathbf{x}}_t$ is compared to the ground truth state $\mathbf{x}^*_t$, and, if the distance between the two is above a user-specified threshold λ_u, a weight update is performed (see below). Additionally, if the distance between $\hat{\mathbf{x}}_t$ and $\mathbf{x}^*_t$ is greater than a second threshold λ_r, the filter is reset to the current ground truth state.

Algorithm 1 Error-driven particle filter training

Input: $(\mathbf{x}^*_{0:T}, \mathbf{y}_{1:T})$ – Training sequence
$\mathbf{w}$ – Input feature weight vector
γ – Learning rate
λ_u – Update threshold
λ_r – Restart threshold ($\geq \lambda_u$)

1: Initialize particles: $\mathbf{x}^{(i)}_0 = \mathbf{x}^*_0$ and $\pi^{(i)}_0 = \frac{1}{n}, i = 1, \ldots, n$
2: **for** $t = 1$ to T **do**
3: Sample n particles $\mathbf{x}^{(i)}_{t-1}$ from current particle set
4: Sample new particle set from proposal distribution: $\mathbf{x}^{(i)}_t \sim q(\mathbf{x}^{(i)}_t | \mathbf{x}^{(i)}_{0:t-1}, \mathbf{y}_{1:t})$
5: Reweight each particle using current weights: $\pi^{(i)}_t \propto \exp\left(\sum_j w_j f_j(\mathbf{y}_t, \mathbf{x}^{(i)}_t)\right)$
6: Compute $\epsilon = ||\mathbf{x}^*_t - \hat{\mathbf{x}}_t||$, where $\hat{\mathbf{x}}_t$ is the current state estimate
7: **if** $\epsilon > \lambda_u$ **then**
8: Update feature weights $\mathbf{w}$ (one of equations 5, 6, or 8)
9: **end if**
10: **if** $\epsilon > \lambda_r$ **then**
11: Reset particle filter to ground truth state at time t by setting all particle states to $\mathbf{x}^*_t$
12: **end if**
13: **end for**

By updating the feature weights when an error is made during inference and by restarting the filter when a large enough error is made, the filter is allowed to focus its attention on meaningful points of failure, resulting in more purposeful weight updates. In addition, performing successive restarts can be seen as reducing training time by allowing the filter to encounter a variety of errors within a training iteration. If instead the filter is never reset for large errors, which is essentially Limketkai *et al.*'s approach [11], many weight updates would be performed on wildly diverging state estimates due to cascading errors. We have found this approach to work poorly in our tracking domain where filters often diverge badly in earlier training iterations.

Using the generic error-driven approach described above, we can optimize a number of discriminative objective functions by specifying appropriate weight update equations (line 8), along with the type of state estimate $\hat{\mathbf{x}}_t$ to be used (line 6). In what follows, we specify these choices for three different objective functions.

Perceptron Updates: Collins' perceptron update [2] can be viewed as an approximation to the gradient of the conditional log-likelihood of the training data $\log p(\mathbf{x}^*_{0:T} | \mathbf{y}_{1:T})$ with respect to the feature weights, and thus, the perceptron algorithm can be thought of as attempting to maximize this probability via approximate gradient ascent.

The batch perceptron update of Collins can easily be modified into an iterative update for use in our error-driven

training approach. The update to weight w_j takes the form:

$$w_j \;=\; w_j + \gamma \left(f_j(\mathbf{x}_t^*, \mathbf{y}_t) - f_j(\hat{\mathbf{x}}_t, \mathbf{y}_t) \right), \qquad (5)$$

where γ is the learning rate. In this case, the state estimate $\hat{\mathbf{x}}_t$ used by the filter is taken to be the particle with the highest particle weight (*i.e*. the MAP state estimate). Thus, filters learned with the perceptron update aim at making the filter's MAP state close to ground truth at every time step.

Minimizing the Squared Residual of Mean: In the case of object tracking, the perceptron update is somewhat unsatisfactory because it places emphasis on the MAP particle instead of on the sample mean $\hat{E}[\mathbf{x}_t]$, which is typically used as the state estimate in particle filter-based object tracking and in our experience typically leads to more robust tracking. To remedy this, we can use an error-driven learning instantiation where the sample mean is taken as the state estimate (*i.e*. $\hat{\mathbf{x}}_t = \hat{E}[\mathbf{x}_t]$), and the objective function is to minimize the squared residual of the sample mean $||\hat{E}[\mathbf{x}_t] - \mathbf{x}_t^*||^2$. We update the weights upon each error in the gradient direction of this objective, which for weight w_j yields the following update:

$$w_j \;=\; w_j + \gamma \left(\hat{E}[\mathbf{x}_t] - \mathbf{x}_t^* \right)^{\mathsf{T}} \left(\hat{E}[\mathbf{x}_t f_j(\mathbf{x}_t, \mathbf{y}_t)] - \hat{E}[\mathbf{x}_t] \hat{E}[f_j(\mathbf{x}_t, \mathbf{y}_t)] \right). \qquad (6)$$

Using this update, w_j converges when the state $\mathbf{x}_t$ is uncorrelated with feature $f_j(\cdot)$ according to the sample distribution. Using this update in our error-driven framework can be viewed as performing stochastic gradient descent on the truncated squared residual, which is set equal to zero when the residual is less than our update threshold λ_u.

Minimizing Mean Squared Error: The above update has the unfortunate property of ignoring the sample variance $\widehat{\mathrm{var}}(\mathbf{x}_t)$ of the state $\mathbf{x}_t$. In particular, we have

$$||\hat{E}[\mathbf{x}_t] - \mathbf{x}_t^*||^2 \;=\; \hat{E}\left[||\mathbf{x}_t - \mathbf{x}_t^*||^2 \right] - \widehat{\mathrm{var}}(\mathbf{x}_t). \qquad (7)$$

This shows that it is possible to minimize the mean's residual while both the mean squared error (MSE) and the variance take on very large values, which is quite undesirable. For example, in object tracking this would be akin to minimizing the residual to a ground truth location at the center of the video frame by placing an equal number of equally weighted particles at each of the four corners of the frame.

Thus, we here consider minimizing the MSE $\hat{E}\left[||\mathbf{x}_t - \mathbf{x}_t^*||^2 \right]$ as a potentially more robust objective, which from (7), corresponds to minimizing the sum of the mean's squared residual and the sample variance. In tracking, minimizing this objective corresponds to having all of the particles bunched tightly around the ground truth location. Differentiating $\hat{E}\left[||\mathbf{x}_t - \mathbf{x}_t^*||^2 \right]$ with respect to w_j yields the following gradient descending weight update:

$$w_j \;=\; w_j + \gamma \left(\hat{E}\left[||\mathbf{x}_t - \mathbf{x}_t^*||^2 f_j(\mathbf{x}_t, \mathbf{y}_t) \right] - \hat{E}\left[||\mathbf{x}_t - \mathbf{x}_t^*||^2 \right] \hat{E}\left[f_j(\mathbf{x}_t, \mathbf{y}_t) \right] \right). \qquad (8)$$

This update has the intuitively satisfying property of converging when the MSE is uncorrelated with feature $f_j(\cdot)$. Again, using this update, our error-driven training process can be viewed as stochastic gradient descent on a truncated version of MSE.

5. Improving Training Computation Time

Since each iteration of error-driven training involves running the particle filter once for each training example, this approach can become computationally expensive when each run of the particle filter has a non-trivial computational cost. This is particularly true in tracking domains in which a large number of objects must be tracked at once, necessitating many calls to feature functions involving expensive pixel-level operations on sizable regions of the video frame, which form a computational bottleneck.

In order to overcome the computational burden associated with performing many of these calculations repeatedly, we use an approximate method that operates on strategically drawn state samples with pre-computed values for these features. Specifically, we pre-process each training example $(\mathbf{x}_{0:T}^*, \mathbf{y}_{1:T})$ by drawing a large set of samples $\{\tilde{\mathbf{x}}_t^{(k)}\}_{k=1,\ldots,N}$ normally distributed around each of the ground truth states $\mathbf{x}_t^*$ with large variance and, for each of these samples, pre-computing the values of all features involving expensive pixel-level operations.

Then, during training, the particle filter is modified by only allowing the proposal distribution to propose states at time t that are represented in $\{\tilde{\mathbf{x}}_t^{(k)}\}_{k=1,\ldots,N}$. This can be implemented by turning the continuous proposal distribution into a discrete distribution in a straightforward way. In this way, during the particle re-weighting step, the available pre-computed feature values are used for all of the particles, while all other features are computed online.

This approach allows training to proceed significantly faster than when using the full particle filter while still providing a close approximation to the way the full filter performs. Performance may suffer in tracking domains where most errors are made due to background clutter and other factors rather than to hijacking by other objects. However, in such domains, it is possible to use a slightly modified version of this method that generates sample sets during actual runs of the tracker and iteratively augments those sets after they have been used for several iterations of training. We are currently investigating such an approach.

6. Experiments and Results

We test our approach by tracking the 22 players and one referee in low-resolution videos of American football. Each video contains footage of a single football play shot from a panning, tilting, and zooming camera with a sideline view

Figure 1: *A typical video frame from our dataset.*

high above the center of the field. Figure 1 depicts a typical view from this camera. Every video is pre-processed to register frames to an overhead model of the football field using the method described in [7], thereby enabling us to determine players' locations in football field coordinates.

The football domain is an extremely challenging testbed for any multi-object tracking algorithm. The players being tracked in this domain move very erratically, and the characteristics of players' motion change substantially depending on the player's type and the time stage of the video. In the first several seconds of each video, for example, nearly all of the players stand virtually still, while perhaps one or two move only gradually. However, after the ball is snapped (a one-time event that occurs in every football play), all of the players begin to move very quickly, and the tracker must be able to adapt accordingly. In addition, football players interact in complex ways, frequently in very large groups. For example, in every play, a group of five linemen on the offense stand shoulder to shoulder in a line and attempt to block a group of about three to five defensive players who, in turn, attempt to break through the offensive line. Many other complicated interactions take place between smaller groups of players throughout the course of a play. In what follows, we first describe the filter models and feature functions we use to capture these interactions and other salient aspects of the football domain. We then move on to present the results of our experiments.

6.1. Football domain modeling

In the football domain, we represent each player as a rectangular region defined by his location in football field coordinates and the scale of the region (*i.e.* $\mathbf{x} = \{x, y, s_x, s_y\}$). Players' motion is modeled using the second-order autoregressive dynamical model that is common in the tracking literature, in which $\mathbf{x}_t \sim \mathcal{N}(g(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}), \Sigma_d)$, where $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ is the normal distribution with mean $\boldsymbol{\mu}$ and covariance Σ,

$$g(\mathbf{x}_{t-1}, \mathbf{x}_{t-2}) = A_1\mathbf{x}_{t-1} + A_2\mathbf{x}_{t-2}, \qquad (9)$$

and A_1 and A_2 define a constant acceleration model.

The proposal distribution we use is slightly non-standard. Specifically, it takes the form of the process dynamical distribution above but uses the previous sample means in place of the state values $\mathbf{x}_{t-1}$ and $\mathbf{x}_{t-2}$. In other words, under our proposal distribution, $\mathbf{x}_t \sim \mathcal{N}(g(\hat{E}[\mathbf{x}_{t-1}], \hat{E}[\mathbf{x}_{t-2}]), \Sigma_d)$, where $g(\cdot)$ is as in (9). We have found that this form of proposal distribution works well in practice and is more stable than the motion model.

We use a variety of feature functions to describe different aspects of players' appearances, their motion, and the interactions between players. In order to model motion patterns that change depending on the time stage of the football play, each feature we describe below is replicated once for each possible time stage. All features that do not correspond to the current time stage take on zero values. In all, each filter uses 15 base features, described below, and 4 time stages for a total of 60 features.

Player motion features: We use 11 different features to describe player motion. The first of these is the logarithm of the Gaussian probability density value of the player state $\mathbf{x}_t$ evaluated under the proposal distribution. The second is the negative squared distance between the player's current state and the state estimate from the previous time step, *i.e.* $-||\mathbf{x}_t - \hat{E}[\mathbf{x}_{t-1}]||^2$. Intuitively, these features allow for a trade-off between rewarding a particle for being close to the proposal's prediction and not straying too far from the player's previous location.

We also use a set of binary features that indicate in which of the eight compass directions a player is moving. These are calculated by quantizing the player's motion vector $(\mathbf{x}_t - \hat{E}[\mathbf{x}_{t-1}])$. These features allow us to model motion tendencies that occur regularly across all football plays. For example, the quarterback nearly always moves backwards immediately after the ball is snapped.

Appearance features: We use two features to describe players' appearances. The first of these is an RGB histogram-based feature, which is calculated based on the Bhattacharyya coefficient-based histogram distance (as in [14]) between a reference histogram and the histogram of the region defined by $\mathbf{x}_t$. The second appearance feature is based on background modeling and is computed by using a pre-computed background color model to count the number of foreground pixels inside the player region defined by $\mathbf{x}_t$ to the total area of that region.

Player interaction features: The final two features describe interactions between players. The first of these is similar to the MRF edge potential used in [9], which penalizes overlap between player regions in the video frame. The second is similar to the MRF edge potential used in [18], which penalizes proximity between players in state space. Intuitively, these features cause trackers to "repel" one another, thereby helping them to to better cope with interactions between players by avoiding hijacking.

6.2. Results

Our dataset contains 20 videos of different football plays, each around 400 frames long, along with hand-labeled ground truth data for every player in every frame[1].

We used a four-fold cross validation approach to evaluate our training method on this dataset. Specifically, we divided the entire set into four folds of five videos each and, for each fold, trained trackers on the other three folds over 100 iterations of the method described in Section 5 using each of the weight updates in equations (5), (6), and (8). To account for the different characteristics of various player types, we learned 13 separate sets of weights, one for each individual player type, with filters for players of the same type sharing the same set of weights in both tracking and learning.

We evaluated each fold after every 10 training iterations using the full pseudo-independent log-linear filtering model with averaged feature weights for each player type. In addition, we also evaluated each video using Khan *et al.*'s MCMC-based particle filter [9] and Yu and Wu's mean field Monte Carlo algorithm [18], two state-of-the-art multi-object particle-filter trackers. For all of these methods, player types and initial locations are computed automatically using mixture-of-parts pictorial structures [8]. We also tried to use the CRF-Filter perceptron algorithm of Limketkai *et al.* [11] on our data set and found that it did not converge well in training. We believe that this is due to the non-incremental nature of the updates, which are based in large part on extremely erroneous filtering runs.

Figures 2 and 3 compare the performance of pseudo-independent log-linear filters trained using the weight updates in equations (5), (6), and (8) and of Khan *et al.*'s and Yu and Wu's methods. Figure 2 depicts the mean squared per-frame pixel error $(\hat{\mathbf{x}}_t - \mathbf{x}_t^*)^2$, while Figure 3 depicts the failure rate, which is the proportion of frames in which a player was considered to be lost (*i.e.* in which the tracker is at least 5 yards in error). Results are given averaged over all players for both the training and testing folds.

First, these results make it clear that the training algorithm is able to significantly reduce the training error, indicating that the optimization approach is effective. Further, by both error measures, pseudo-independent log-linear filters trained using each of the weight updates achieve substantially better results than untrained pseudo-independent log-linear filters (iteration 0) and the previous state-of-the-art methods on the test set. It is especially important to note that the significantly lower lost-player rates attained by our filters indicate that our discriminative error-driven training approach is effective in achieving its goal of improving filter performance by learning to overcome failures. This also suggests that the approximate method described in Section 5 is a valid approximation to the full tracking process.

[1]Our entire hand-labeled tracking dataset is available at `http://eecs.oregonstate.edu/football/tracking/dataset`.

We also see that the MSE and residual updates perform better than the perceptron updates on this data, with the MSE update having an edge in peak accuracy and the performance of perceptron-trained trackers deteriorating quickly after achieving their peak performance. As is common for incremental learning methods, the error does not monotonically decrease. In practice one would use a validation set to select the best stopping point for training.

As an additional assessment of tracking performance, we also counted the number of players that were never lost by the tracker during testing. Interestingly, our untrained filters lost every player for at least a very small portion of the video, while Yu and Wu's and Khan *et al.*'s methods were successful on 174 and 119 players (out of a total of 440), respectively. In comparison, the best performing filters trained with the MSE update, the residual update, and the perceptron update were successful on 280, 264, and 223 players, respectively. Examining these players further, we found the best performing trained filters achieved an average per-frame MSE of 74.0 (MSE update), 76.3 (residual update), and 82.4 (perceptron update), while Yu and Wu's and Khan *et al.*'s methods respectively achieved average per-frame MSEs of 110.0 and 100.1. These results show that our trained filters can fully track a significantly larger number of players more accurately. In our model 6 pixels corresponds to one yard on the field, indicating that MSE and residual updates achieve an error of approximately 1.5 yards for these players, which, in our experience, is more than enough to infer many types of player behavior.

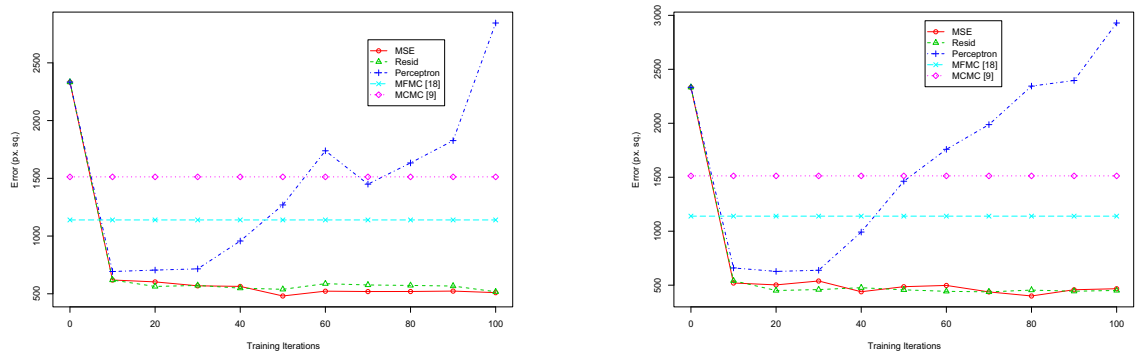
Average run times for pseudo-independent log-linear filters, Yu and Wu's method and Khan *et al.*'s method are summarized in Table 1. We can see that, in addition to improved filter performance, pseudo-independent log-linear filters also offer faster tracking times.

	PILLFs	Yu and Wu	Khan *et al.*
Avg. run time	18.30 m	30.55 m	84.50 m

Table 1: *Average run time in minutes for pseudo-independent log-linear filters, Yu and Wu's method [18], and Khan* et al.*'s method [9]. These are computed over the entire football dataset on a standard desktop computer.*

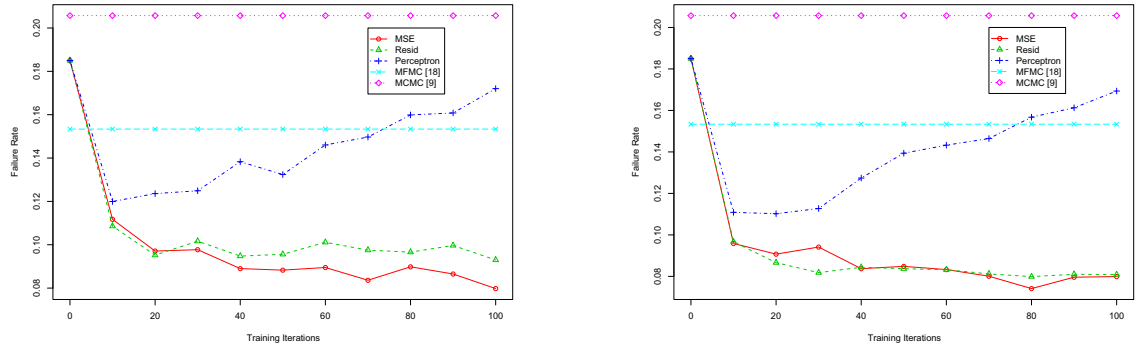
7. Summary and Future Work

We have described a framework for multi-object tracking based on pseudo-independent log-linear particle filters. Our main contribution is an error-driven discriminative training algorithm for this model. We gave results in the domain of American football that show the effectiveness of the method in comparison to recently proposed multi-object trackers. To our knowledge, our learned trackers are state-of-the-art in the football domain. In future work, we plan to extend our approach to learn the parameters of more complex MRF-style models to allow for more significant joint inference about object interactions. In addition, we plan to study the theoretical convergence of our error-driven algorithms.



(a) Testing folds (b) Training folds

Figure 2: *Mean squared per-frame pixel error at every 10 training iterations, averaged over all players in (a) the testing and (b) the training folds for each of the three weight update equations (5), (6), and (8). Included for reference are MSE values for Yu and Wu's method [18] and Khan* et al.*'s method [9].*



(a) Testing folds (b) Training folds

Figure 3: *Failure rate at every 10 training iterations, averaged over all players in (a) the testing folds and (b) the training folds. Failure rate is computed as the proportion of a player's frames in which the player was lost by the tracker.*

8. Acknowledgments

This work is supported by NSF grant IIS-0546867 and DARPA contract FA8750-05-2-0283. We would like the thank the coaching staff of the Oregon State Football team for use of their video and William Brendel and Jervis Pinto for assistance in constructing the labeled data set.

References

[1] P. Abbeel, A. Coates, M. Montemerlo, A. Y. Ng, and S. Thrun. Discriminative training of Kalman filters. In *Proc. Robitics: Science and Systems*, 2005.

[2] M. Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. In *EMNLP*, 2002.

[3] M. Collins and B. Roark. Incremental parsing with the perceptron algorithm. In *Proc. Annual Meeting for the Assoc. for Computational Linguistics*, 2004.

[4] B. Cowan, I. Kučerová, and M. Collins. A discriminative model for tree-to-tree translation. In *EMNLP*, 2006.

[5] H. Daume III and D. Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *ICML*, 2005.

[6] A. Doucet, N. de Freitas, and N. Gordon, editors. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[7] R. Hess and A. Fern. Improved video registration using non-distinctive local image features. In *CVPR*, 2007.

[8] R. Hess, A. Fern, and E. Mortensen. Mixture-of-parts pictorial structures for objects with variable part sets. In *ICCV*, 2007.

[9] Z. Khan, T. Balch, and F. Dellaert. MCMC-based particle filtering for tracking a variable number of interacting targets. *PAMI*, 27(11):1805–1819, 2005.

[10] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[11] B. Limketkai, D. Fox, and L. Liao. CRF-filters: Discriminative particle filters for sequential state estimation. In *ICRA*, 2007.

[12] A. Y. Ng and M. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *NIPS*, 2002.

[13] K. Okuma, A. Taleghani, N. de Freitas, J. J. Little, and D. G. Lowe. A boosted particle filter: multitarget detection and tracking. In *ECCV*, 2004.

[14] P. Pérez, C. Hue, J. Vermaak, and M. Ganget. Color-based probabilistic tracking. In *ECCV*, 2002.

[15] B. Taskar, C. Guestrin, and D. Koller. Max-margin Markov networks. In *NIPS*, 2004.

[16] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *ICML*, 2004.

[17] Y. Xu and A. Fern. On learning linear ranking functions for beam search. In *ICML*, 2007.

[18] T. Yu and Y. Wu. Collaborative tracking of multiple targets. In *CVPR*, 2004.

[19] T. Yu and Y. Wu. Decentralized multiple target tracking using netted collaborative autonomous trackers. In *CVPR*, 2005.

[20] L. Zettlemoyer and M. Collins. Online learning of relaxed CCG grammars for parsing to logical form. In *EMNLP/CoNLL*, 2007.