# Learning Online Smooth Predictors for Realtime Camera Planning using Recurrent Decision Trees

Jianhui Chen * Hoang M. Le † Peter Carr ‡ Yisong Yue † James J. Little *
*University of British Columbia † California Institute of Technology ‡ Disney Research
{jhchen14, little}@cs.ubc.ca {hmle, yyue}@caltech.edu carr@disneyresearch.com

## Abstract

*We study the problem of online prediction for realtime camera planning, where the goal is to predict smooth trajectories that correctly track and frame objects of interest (e.g., players in a basketball game). The conventional approach for training predictors does not directly consider temporal consistency, and often produces undesirable jitter. Although post-hoc smoothing (e.g., via a Kalman filter) can mitigate this issue to some degree, it is not ideal due to overly stringent modeling assumptions (e.g., Gaussian noise). We propose a recurrent decision tree framework that can directly incorporate temporal consistency into a data-driven predictor, as well as a learning algorithm that can efficiently learn such temporally smooth models. Our approach does not require any post-processing, making online smooth predictions much easier to generate when the noise model is unknown. We apply our approach to sports broadcasting: given noisy player detections, we learn where the camera should look based on human demonstrations. Our experiments exhibit significant improvements over conventional baselines and showcase the practicality of our approach.*

## 1. Introduction

We investigate the problem of determining where a camera should look when broadcasting a team sporting event, such as basketball or soccer (see Fig. 1). Realtime camera planning shares many similarities with online object tracking: in both cases, the algorithm must constantly revise an estimated target position as new evidence is acquired. Noise and other ambiguities can cause non-ideal jittery trajectories, which in camera planning lead to unaesthetic results (see Fig. 2). In contrast to object tracking, smoothness is of paramount importance in camera control: fluid movements that maintain adequate framing are preferable to erratic motions that constantly pursue perfect composition.

Non-parametric or model-free estimation methods, such as random forests [10], are very popular because they can learn (almost) arbitrary predictors directly from training

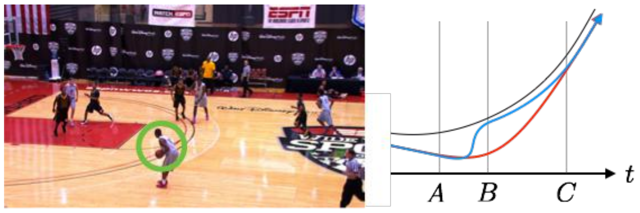


Figure 1. **Camera Planning**. *The goal is to predict the pan angle for a broadcast camera based on noisy player detections. Consider two planning algorithms (the blue and red curves) which both make the same mistake at time $A$ but recover to a good framing by $C$ (the ideal camera trajectory is shown in black). The blue solution quickly corrects by time $B$ using a jerky motion, whereas the red curve conducts a gradual correction. Although the red curve has a larger discrepancy with the ideal motion curve, its velocity characteristics are most similar to the ideal motion path.*

data. When applied to smooth trajectory prediction, the estimator is often learned within a time-independent paradigm, with temporal regularization integrated afterwards as a post-processing stage (e.g., via a Kalman filter) [28, 30]. One major limitation of this two-stage approach for camera planning is that the smoothing is done in a context-independent way, which can lead to uninformed tradeoffs between accuracy and smoothness (see Fig. 1).

In this paper, we propose a recurrent decision tree framework that can make predictions *conditioned on its own previous predictions*, which allows it to learn temporal patterns within the data (in addition to any direct feature-based relationships). However, this recursive formulation (similar to reinforcement learning [31]) makes the learning problem much more challenging compared to the time-independent approach. We develop a learning algorithm based on the "search and learn" (SEARN) approach [12] to efficiently converge to a stable recurrent model.

We applied our approach to autonomous camera control in sports, where the goal is to generate smooth camera motion that imitates a human expert. We provide both quantitative and qualitative evidence showing our approach significantly outperforms several strong baselines.

## 2. Related work

**Sequential Supervised Learning** Sequential supervised learning is broadly applied in many domains, including natural language processing tasks such as part-of-speech tagging [9] and computational biology tasks such as protein alignment [32]. Unlike standard supervised learning, sequential supervised learning operates on sequences: each training example $(\mathbf{x}_i, \mathbf{y}_i)$ is a sequence of features $\mathbf{x}_i = \langle x_{i,1}, x_{i,2}, \ldots, x_{i,T_i} \rangle$ and a corresponding sequence of labels $\mathbf{y}_i = \langle y_{i,1}, y_{i,2}, \ldots, y_{i,T_i} \rangle$. The learned predictor outputs a sequence of labels for an input sequence of features.

Our setting is distinguished from conventional sequential learning because we must learn an online predictor. In other words, conventional sequential learning typically assumes access to all of $\mathbf{x}$ before predicting $\mathbf{y}$ [1, 9, 13, 23, 24, 32]. Our setting is further distinguished from most previous sequential prediction work by aiming to learn model-free or non-parametric predictors. For instance, the bulk of previous work utilize linear predictors and thus require a well-specified feature representation [1, 9, 24, 32].

One approach for utilizing arbitrary predictors is via a sliding window [13, 14, 23], in which any supervised learning method can be used to learn the sliding window predictor. However, if the predictor is defined recurrently (i.e., it depends on its previous predictions), then it is not obvious what values to use for previous predictions when generating supervised training examples. One way to address this issue is via "stacked" sequential learning [8, 28, 30], which is essentially the two-stage approach of first training a non-recurrent predictor, and then employing recurrent smoothing (e.g., a hidden Markov model or Kalman filter).

Our approach instead directly trains a predictor to make good predictions given previous predictions, which leads to a "chicken and egg" problem of how to define a training set that depends on the predictions of the model to be trained. We employ a learning reduction approach, based on SEARN [12], that iteratively constructs a sequence of supervised training sets such that the resulting sequence of predictors efficiently converges to a stable recurrent predictor. Other learning reduction approaches for sequential learning include DAgger [29], which can be more efficient in the number of iterations needed to converge, but with each iteration being more computationally expensive.

**Camera Planning** Algorithms for determining where a camera should look have been investigated for a variety of scenarios from scripted cooking shows and college lectures to team sports [5, 22]. It is widely accepted that a smoothly moving camera is critical for generating aesthetic video [16]. One approach is post-processing to regulate the sequential predictions, which leads to a trade-off between smoothness and responsiveness [6]. Alternatively, offline batch processes [19] can be used if there is not an online requirement. Significant filtering constrains the motion of the camera, making it unable to track fast moving objects [26]. Other approaches can offer more refined control but first require users to select the main features of interest [17].

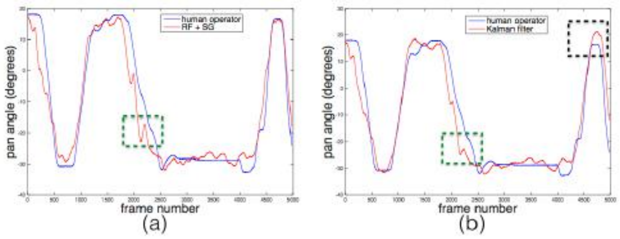


Figure 2. **Jitter and Overshooting Artifact**. *(a) A plan generated by [6] includes a sudden direction change (green dashed box) and jitter. (b) Post-processing independent predictions with a Kalman filter reduces jitter but causes over shooting (black dashed box).*

**Camera Motion Models** Camera motion is well-studied in the context of stabilization. Liu *et al*. [26] used a low-pass filter to create constant angular velocity motions for video stabilization. The rotation component was filtered by linearizing the rotation space using derivatives with respect to time to obtain angular velocities [25]. The method was extended to polynomial eigen-trajectories for subspace video stabilization [27]. In contrast, we are interested in the problem of camera planning. In other words, rather than trying to reconstruct a stable camera trajectory from existing trajectories, our goal is to plan a brand new trajectory.

## 3. Problem Setup

Let $d_{\mathbf{x}}$ denote a distribution of input sequences: $\mathbf{x} = \langle x_1, \ldots, x_T \rangle \sim d_{\mathbf{x}}$. In camera planning, $\mathbf{x}$ can be a sequence of (noisy) detected player locations from stationary cameras. For clarity, we assume each sequence has the same length $T$, but in general $T$ can vary and/or be unknown.

Let $\Pi$ denote a class of policies that our learner is considering. Given a stream of inputs $\mathbf{x} = \langle x_1, \ldots, x_T \rangle$, each $\pi \in \Pi$ generates a stream of outputs $\mathbf{y} = \langle y_1, \ldots, y_T \rangle$. In camera planning, $\mathbf{y}$ can be a sequence of pan angles of the broadcast camera. Section 4 describes our recurrent decision tree framework that instantiates $\Pi$ in our experiments.

Operationally, each $\pi$ is a streaming predictor that takes a *state* $s_t = \{x_t, \ldots x_{t-\tau}, y_{t-1}, \ldots, y_{t-\tau}\}$ composed of the recent inputs and predictions, and generates a new prediction $y_t = \pi(s_t)$. Note that the next state $s_{t+1}$ depends only on the new input $x_{t+1}$, the current state $s_t$, and the current prediction $y_t$. Hence, for any input sequence $\mathbf{x}$, $\pi$ equivalently generates a state sequence $\mathbf{s} = \langle s_1, \ldots, s_T \rangle$. We also abuse notation to say that $\mathbf{y} = \pi(\mathbf{x})$ denotes the sequence of predictions $\mathbf{y}$ generated by streaming $\mathbf{x}$ into $\pi$ (with the construction of the state sequence $\mathbf{s}$ being implicit).

For any policy $\pi \in \Pi$, let $d_t^\pi$ denote the distribution of states at time $t$ if $\pi$ is executed for the first $t-1$ time steps ($d_t^\pi$ is defined exactly by $d_{\mathbf{x}}$ and $\pi$). Furthermore, let $d_\pi = \frac{1}{T}\sum_{t=1}^T d_t^\pi$ be the average distribution of states if we follow $\pi$ for all $T$ steps. The goal of the learner is to find a policy $\hat{\pi} \in \Pi$ that minimizes the imitation loss under its own induced distribution of states:

$$\hat{\pi} = \underset{\pi \in \Pi}{\operatorname{argmin}} \, \mathbb{E}_{s \sim d_\pi} \left[ \ell(s, \pi, \pi^*) \right]. \tag{1}$$

Since our goal is to learn a policy $\pi$ that both imitates the (human) expert $\pi^*$ and is also smooth, we decompose our loss function into precision and smoothness components:

$$\ell(s, \pi, \pi^*) = \ell_*(s, \pi, \pi^*) + \omega \ell_R(s, \pi), \tag{2}$$

where $\ell_*$ measures how well $\pi(s)$ agrees with $\pi^*(s)$, and $\ell_R$ measures how smooth the prediction of $\pi(s)$ is relative to the current state $s$, with $\omega \geqslant 0$ controlling the trade-off between the two. The precision error $\ell_*$ is typically the squared deviation: $\ell_*(s, \pi, \pi^*) = \|\pi^*(s) - \pi(s)\|^2$.

A standard way to instantiate the smoothness error $\ell_R$ is via the squared deviation of the velocity: $\ell_R(s, \pi) = \|v(s) - v(\pi(s))\|^2$, where $v(s)$ denotes the velocity in state $s$, which can be computed from the information encoded in $s$. One can thus interpret the smoothness error as encouraging the curvature of the predicted trajectories to be low.

We assume the agnostic setting, where the minimizer of (1) does not necessarily achieve 0 loss (i.e., we cannot perfectly imitate the human expert). In practice, we approximate the expectation in (1) with a finite sample (e.g., a training set of basketball gameplay sequences), and also solve (1) via alternating minimization: collect training data according to current $\hat{\pi}$, and then train a new $\hat{\pi}$ using that training data (see Section 5 for more details).

**Discussion and Interpretation.** By utilizing a recurrent policy class $\Pi$, such as our recurrent decision tree framework (see Section 4), the learner is able to reason about inherent temporal relationships. For instance, suppose the learner incorrectly predicted a previous value (see Fig. 1). Then the learner could make a drastic correction (blue line) to minimize the discrepancy as quickly as possible, which might result in unaesthetic high-frequency camera motion. Instead, a more gradual correction (red line) may better trade off between instantaneous error and smooth motion.

Estimating the best $\hat{\pi}$ is challenging due to its dependence on its own previous predictions. Most notably, this recurrent relationship makes it nontrivial to extract a set of independent training examples to use with conventional supervised learning. This issue is formally highlighted in the learning objective (1), where the distribution that the loss is evaluated over is induced by the policy under consideration. In other words, the distribution $d_\pi$ that the loss is evaluated on in (1) depends on the $\pi$ being considered, which leads to complicated learning problem since conventional supervised learning assumes that distribution is fixed.

One straightforward approach is to approximate $d_\pi$ in (1) using the distribution of states, $d_{\pi^*}$, that the human expert $\pi^*$ induces on $d_{\mathbf{x}}$, and then select the $\hat{\pi} \in \Pi$ to minimize:

$$\hat{\pi} = \underset{\pi \in \Pi}{\operatorname{argmin}} \, \mathbb{E}_{s \sim d_{\pi^*}} \left[ \ell(s, \pi, \pi^*) \right]. \tag{3}$$

Note that (3) is easy to evaluate since the expectation is over $d_{\pi^*}$ and does not depend on the policy $\pi$ under consideration.[1] However, as soon as the behavior of $\hat{\pi}$ deviates from $\pi^*$, then the distribution of states experienced by $\hat{\pi}$ will differ from $\pi^*$, and thus optimizing (3) will not be aligned with optimizing (1).[2] We show in Section 5 how to address this distribution mismatch problem via an alternating minimization approach that efficiently converges to a stable solution.

## 4. Recurrent Decision Tree Framework

Decision trees are amongst the best performing learning approaches, and are popular whenever a non-parametric predictor is desirable [10, 14, 23]. We propose a recurrent extension, where the prediction at the leaf node is not necessarily constant, but rather is a (smooth) function of both static leaf node prediction and previous predictions from the tree. For simplicity, we present our framework using a single decision tree, although our approach extends trivially to ensemble methods such as random forests [3, 10].

A decision tree specifies a partitioning of the input space (i.e, the space of all possible states $s$). Let $D = \{(s_m, y_m^*)\}_{m=1}^M$ denote a training set of state/target pairs. Conventional regression tree learning aims to learn a partitioning such that each leaf node, denoted by `node`, makes a constant prediction to minimize the squared loss:

$$\bar{y}_{\texttt{node}} = \underset{y}{\operatorname{argmin}} \sum_{(s, y^*) \in D_{\texttt{node}}} (y^* - y)^2, \tag{4}$$

where $D_{\texttt{node}}$ denotes the training data from $D$ that has partitioned into the leaf `node`. The solution to (4) is:

$$\bar{y}_{\texttt{node}} = \frac{1}{size(D_{\texttt{node}})} \sum_{(s, y^*) \in D_{\texttt{node}}} y^* \tag{5}$$

One typically trains via greedy top-down induction [10], which is an iterative process that repeatly chooses a leaf node to be split based on a binary query of the input state $s$.

The above formulation (4) can be viewed as the simplest version of our recurrent decision tree framework. In particular, the decision tree is allowed to branch on the input state

[1] In practice, optimizing (3) reduces to a standard supervised learning scenario. One simply collects the decisions made by the human expert $\pi^*$ to use as a static training set, and then choose the $\hat{\pi} \in \Pi$ that agrees most with $\pi^*$ on the states induced by $\pi^*$.

[2] One simplified interpretation is that training on (3) does not teach the predictor how to recover from its own mistakes.

$s$, which, as discussed in Section 3, includes the previous predictions $\{y_{-1}, \ldots, y_{-\tau}\}$. Thus, decision trees that minimize (4) form a valid recurrent policy class, and can be used to instantiate $\Pi$. In the following, we will describe a more general class of recurrent decision trees that enforce more explicit smoothness requirements.

Note that recurrent neural networks (RNNs) [2] impose a slightly different notion of "recurrent" than our approach. Whereas our approach is only recurrently defined with respect to the previous predictions of our approach, RNNs are recurrently defined with respect to the previous hidden unit activations and (or) previous predictions.

### 4.1. Jointly Capturing Dynamics and Control

Let $f_\pi(y_{-1}, \ldots, y_{-\tau})$ denote an autoregressor of the temporal dynamics of $\pi$ over the distribution of input sequences $d_{\mathbf{x}}$, while *ignoring* the exogenous inputs $x$. In other words, at time step $t$, $f_\pi$ predicts the behavior $y_t \equiv \pi(s_t)$ given only $y_{t-1}, \ldots, y_{t-\tau}$. Typically, $f_\pi$ is selected from a class of autoregressors $F$ (e.g., smooth autoregressors). For our experiments, we use regularized linear autoregressors as $F$, although one could also use more complex functions (e.g., based on Kalman filters). See the supplemental material for more details on linear autoregressors.

We now present a policy class $\Pi$ of recurrent decision trees $\pi$ that make smoothed predictions by regularizing to be close to its autoregressor $f_\pi$. For any tree/policy $\pi$, each leaf node is associated with a "control" or "target" signal $\bar{y}_{\texttt{node}}$ such that the prediction $\hat{y}$ given input state $s$ is:

$$\hat{y} \equiv \pi(s) \equiv \underset{y}{\operatorname{argmin}} \ (y - \bar{y}_{node(s)})^2 + \lambda(y - f_\pi(s))^2, \quad (6)$$

where $\texttt{node}(s)$ denotes the leaf node of the decision tree that $s$ branches to, and $\lambda \geqslant 0$ trades off between $\hat{y}$ matching the target signal $\bar{y}_{\texttt{node}(s)}$ versus the smooth autoregressor $f_\pi(s)$. The closed-form solution to (6) is:

$$\hat{y}(s) = \frac{\bar{y}_{\texttt{node}(s)} + \lambda f_\pi(s)}{1 + \lambda}. \quad (7)$$

Compared to just predicting $\hat{y} \equiv \bar{y}_{\texttt{node}(s)}$, the prediction in (6) varies much more smoothly with $s$, since $\hat{y}$ is now an interpolation between the target signal $\bar{y}_{\texttt{node}(s)}$ and smooth extrapolation $f_\pi(s)$ from previous predictions.

Training requires estimating both the autoregressor $f_\pi$ and the decision tree of target signals $\bar{y}_{\texttt{node}}$. In practice, given training data $D$, we perform greedy training by first estimating $f_\pi$ on $D$ (which ignores the exogenous inputs $x$),[3] and then estimating the decision tree of target signals to "course correct" $f_\pi$. Given a fixed $f_\pi$ and decision tree structure, one can set $\bar{y}_{\texttt{node}}$ as:

$$\bar{y}_{\texttt{node}} = \underset{y}{\operatorname{argmin}} \sum_{(s,y^*) \in D_{\texttt{node}}} (y^* - \hat{y}(s|y))^2, \quad (8)$$

for $\hat{y}(s|y)$ defined as in (7) with $y \equiv \bar{y}_{\texttt{node}(s)}$. Similar to (5), we can write the closed-form solution of (8) as:

$$\bar{y}_{\texttt{node}} = \frac{1}{size(D_{\texttt{node}})} \sum_{(s,y^*) \in D_{\texttt{node}}} \left((1 + \lambda)y^* - \lambda f_\pi(s)\right). \quad (9)$$

Note that when $\lambda = 0$, then the autoregressor has no influence, and (9) reduces to (5). Note also that (8) is a simplified setting that only looks at imitation loss $\ell_*$, but not smoothness loss $\ell_R$. We refer to the supplemental material for more details regarding our full training procedure.

One can interpret our recurrent decision tree framework as holistically integrating model-based temporal dynamics $f$ with model-free control $\bar{y}_{\texttt{node}}$. The target signal $\bar{y}_{\texttt{node}}$ can thus focus on generating "course corrections" of the smooth temporal dynamics imposed by $f$. In practice, the target signal $\bar{y}_{\texttt{node}}$ can also be set using an ensemble such as random forests [3, 10, 18], or regression trees that predict piecewise-linear variables in the leaf nodes [15]. In this way, our framework is completely complementary with previous work on learning smoother regression tree predictors.

**Extensions.** One could also define the predictions $y$ as velocity rather than absolute coordinates, which coupled with the current state $s$, will encode the absolute coordinates. Such an approach would be desirable if one wants to do perform autoregressor regularization in the velocity domain.

Another extension is to replace the L2 autoregression penalty in (8) with an L1 penalty. With an L1 penalty, small target signals would not deviate the prediction from the autoregressor, thus making the temporal curvature potentially piecewise linear (which may be desirable). However, the non-smooth nature of L1 regularization would make the prediction more sensitive to the choice of $\lambda$.

## 5. Iterative Training Procedure

In general, it can be difficult to exactly solve (1) due to the circular dependency between the distribution of states and the policy under consideration. One meta-learning approach is to alternate between estimating $d_\pi$ over a fixed $\pi$, and optimizing $\pi$ over a fixed $d_\pi$. At a high level, this can be described as:

1. Start with some initial policy $\hat{\pi}_0$
2. At the $i$-th iteration, use the previously estimated policies $\hat{\pi}_0, \ldots, \hat{\pi}_{i-1}$ to construct a new distribution $d_i$
3. Estimate the best policy $\hat{\pi}_i$ via:

$$\hat{\pi}_i = \underset{\pi \in \Pi}{\operatorname{argmin}} \, \mathbb{E}_{s \sim d_i} \left[\ell_i(s, \pi, \pi^*)\right]. \quad (10)$$

[3] Intuitively, if we train $f_\pi$ on $D$, then the state distribution of $D$ should be similar to the state distribution that will be induced by the resulting trained $\pi$. We address this point further in Section 5.

Note that the loss function $\ell_i$ need not be the original loss function, but simply needs to converge to it.

4. Repeat from Step 2 with $i \leftarrow i + 1$ until some termination condition is met.

One typically initializes using the human expert $\hat{\pi}_0 = \pi^*$, which we have demonstrations for in the training set (i.e., $\pi^*$ is memorized on the training set). The estimation problem in (10) can be solved via standard supervised learning (see the supplemental material for how we solve (10) for our recurrent decision tree framework). It remains to decide how to construct a stable and converging sequence of distributions $d_i$. For instance, it is known that simply choosing $d_i = d_{\hat{\pi}_{i-1}}$ does not guarantee stable behavior [12, 29].

We build upon the SEARN approach [12] to iteratively construct a stable sequence of distributions $d_i$ for training each $\hat{\pi}_i$. Algorithm 1 describes our approach, which is a meta-learning approach that can be applied to other policy classes beyond our recurrent decision tree framework (i.e., by using a different implementation of `Learn`). Note also that we approximate each state distribution $d_i$ using a finite empirical sample $\tilde{\mathbf{S}}_i$ — i.e., $\tilde{\mathbf{S}}_i$ is constructed from the given input sequences $\mathbf{X}$ and the predictions iteration $i$, $\tilde{\mathbf{Y}}_i$.

Algorithm 1 is an instance of the alternating procedure described above. Given a state distribution $d_i$, the training problem is a straightforward supervised learning problem (Line 7). Note that the iteration-specific loss $\ell_i$ is simply the original loss $\ell$ using a (potentially) modified imitation target $\mathbf{Y}_i^*$ that converges to the original $\mathbf{Y}$ (Line 6).

The key innovation of SEARN [12] is that the new distribution $d_i$ should be induced by an exponentially decaying interpolation between every $\hat{\pi}_0, \ldots, \hat{\pi}_{i-1}$. In practice, we found it more convenient to interpolate the trajectories $\hat{\mathbf{Y}}_0, \ldots, \hat{\mathbf{Y}}_{i-1}$ (Line 10), which leads to a set of trajectories $\tilde{\mathbf{Y}}_i$ that can be combined with the input $\mathbf{X}$ to form an empirical sample of states $\tilde{\mathbf{S}}_i$ to approximate $d_i$ (Line 5). Because of this interpolation, the sequence of distributions $d_i$ will converge in a stable way (see [12] for a rigorous analysis of the convergence behavior). This stable behavior is especially important considering our greedy training procedure for our recurrent decision tree framework: if the resulting $\hat{\pi}_i$ has a very different behavior from the training distribution $d_i$, then $f_{\hat{\pi}_i}$ will not be a good autoregressor of $\hat{\pi}_i$.

### 5.1. Implementation Details

**Choosing $\beta$.** A straightforward choice of interpolation parameter is to set $\beta$ to a small constant (close to 0) to ensure that the new distribution $d_i$ stays close to the previous distribution $d_{i-1}$, which is the original approach of SEARN [12]. One drawback of this conservative approach is slower convergence rate, especially when the learner needs to quickly move away from a bad policy $\hat{\pi}_i$.

We can also adaptively select $\beta$ based on relative empirical loss of learned policies (Line 9 of Algorithm 1). Let

**Algorithm 1** Iterative Training Procedure

```
Input: input sequences $\mathbf{X}$
Input: expert demonstrations $\mathbf{Y}$ for $\mathbf{X}$
Input: autoregression function class $F$
Input: history time horizon $\tau$ for generating states //see Section 3
Input: $\beta(\ldots)$                          //distribution drift step size
 1: Initialize $\tilde{\mathbf{S}}_0 \leftarrow$ state sequences defined by $\mathbf{Y}$ and $\mathbf{X}$
 2: $\pi_0 \leftarrow \texttt{Learn}(\tilde{\mathbf{S}}_0, \mathbf{Y})$.
 3: Initialize $\tilde{\mathbf{Y}}_1 \leftarrow \{\pi_0(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$     //initialize exploration
 4: for $i = 1, \ldots, N$ do
 5:     $\tilde{\mathbf{S}}_i \leftarrow$ state sequences defined by $\tilde{\mathbf{Y}}_i$ and $\mathbf{X}$
 6:     $\mathbf{Y}_i^* \leftarrow$ expert feedback on each $\tilde{\mathbf{s}} \in \tilde{\mathbf{S}}_i$   //see Section 5.1
 7:     $\hat{\pi}_i \leftarrow \texttt{Learn}(\tilde{\mathbf{S}}_i, \mathbf{Y}_i^*)$            //minimizing (10)
 8:     $\hat{\mathbf{Y}}_i \leftarrow \{\hat{\pi}_i(\mathbf{x}) | \mathbf{x} \in \mathbf{X}\}$                //roll-out $\hat{\pi}_i$
 9:     $\hat{\beta}_i \leftarrow \beta(\texttt{error}(\hat{\mathbf{Y}}_i), \texttt{error}(\tilde{\mathbf{Y}}_i))$    //see Section 5.1
10:     $\tilde{\mathbf{Y}}_{i+1} \leftarrow \hat{\beta}_i \hat{\mathbf{Y}}_i + (1 - \hat{\beta}_i) \tilde{\mathbf{Y}}_i$  //distribution interpolation
11: end for
12: return $\hat{\pi} \in \{\hat{\pi}_1, \ldots, \hat{\pi}_N\}$ with best validation performance
```

$\texttt{error}(\hat{\mathbf{Y}}_i)$ and $\texttt{error}(\tilde{\mathbf{Y}}_i)$ denote the loss (2) of rolled-out trajectories $\hat{\mathbf{Y}}_i$, $\tilde{\mathbf{Y}}_i$, respectively, with respect to ground truth $\mathbf{Y}$. We can then set $\beta$ as:

$$\hat{\beta}_i = \frac{\texttt{error}(\tilde{\mathbf{Y}}_i)}{\texttt{error}(\hat{\mathbf{Y}}_i) + \texttt{error}(\tilde{\mathbf{Y}}_i)}. \tag{11}$$

This adaptive selection of $\beta$ encourages the learner to disregard bad policies when interpolating, thus allowing fast convergence to a good policy.

**Choosing $\mathbf{Y}^*$.** Intuitively, whenever the current policy $\hat{\pi}_i$ makes a mistake, the expert feedback $y^*$ should recommend smooth corrections (e.g., follow the red line in Figure 1). In many cases, it suffices to simply use the original expert trajectories $\mathbf{Y}_i^* \leftarrow \mathbf{Y}$. However, when $\tilde{\mathbf{Y}}_i$ differs substantially from $\mathbf{Y}$, especially during early iterations, it can be beneficial to use "smoothed" expert feedback $\mathbf{Y}_i^*$ in place of $\mathbf{Y}$.

Specifically, each step along $\mathbf{Y}_i^*$ is computed to be a gradual 1-step correction of $\tilde{\mathbf{Y}}_i$ towards $\mathbf{Y}$, which will gradually converge to $\mathbf{Y}$ in later iterations. It is not strictly necessary, as our recurrent decision tree framework can offset non-smooth corrections by enforcing a larger regularization parameter $\lambda$ (trading higher smoothness for lower imitation accuracy, cf. 8). Generally however, given the same smoothness weight $\lambda$, smoothed expert feedback leads to more stable learning. One simple way to provide smoothed $\mathbf{Y}_i^*$ is to reduce the distance ratio $(y^* - y)/(\tilde{y} - y)$ by a small decaying rate: $y^* = y + e^{-\eta}(\tilde{y} - y)$.

## 6. Experiments

We evaluate our approach automated broadcasting for basketball and soccer (see Fig. 3). A high-school basketball match was recorded using two cameras: one near the ceiling

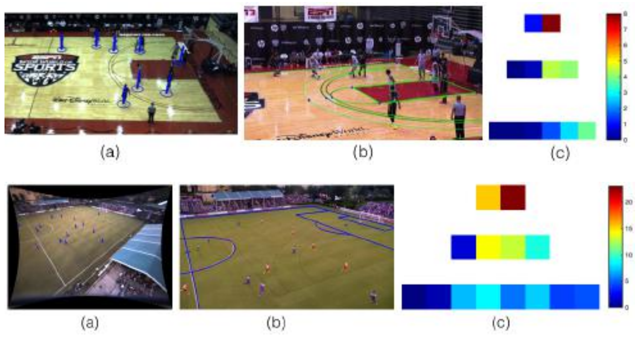


Figure 3. **Features and Labels**. *(a) player detections, (b) pan/tilt/zoom parameters, and (c) spherical quantization scheme for generating features.*

for player detection, and one at ground level for broadcasting (operated by a human expert). The videos were synchronized at 60fps. 'Timeouts' were manually removed, resulting in 32 minutes of 'in-play' data divided into roughly 50 segments (each about 40 seconds long), with two held out for validation and testing.

A semi-professional soccer match was recorded using three cameras: two near the flood lights for player detection, and a robotic PTZ located at mid-field remotely operated by a human expert. The videos were synchronized at 60 fps. About 91 minutes was used for training, and two 2 minute sequences were held out for validation and testing.

**Features** The ground locations of players were determined from 3D geometric primitives which best justified the background subtraction results [4]. Each ground position was projected to a spherical coordinate system centered and aligned with the broadcast camera. Because the number of detections varies due to clutter and occlusions, a fixed length feature vector was constructed using spatial frequency counts. The surface of the sphere was quantized at three resolutions ($1 \times 2, 1 \times 4$, and $1 \times 8$) resulting in a 14 dimensional feature vector $\mathbf{x}_t$ [6].

**Labels** Pan/tilt/zoom parameters are estimated for each frame of the broadcast video by matching detected SIFT key points to a collection of manually calibrated reference frames in a similar fashion to [20]. The homography between the current frame and the best match in the database of reference images is estimated, from which the camera's pan-tilt-zoom settings are extracted. Because the tilt and zoom of the broadcast camera do not vary significantly over the dataset, our experiments only focus on building an estimator for online prediction of pan angles.

### 6.1. Baselines

**Savitzky-Golay.** [6] learns a predictor using a random forest trained using only current player locations. A Savitzky-Golay (SG) filter smooths the predictions, but induces a delay. Our implementation of this method augments the current player locations with previous player locations. This modification makes the instantaneous predictions more reliable, as the predictor has more temporal information.

**Kalman Filter.** We replace the Savitzky-Golay filter with a Kalman filter employing a constant velocity process model. Parameters were determined through validation (see supplemental material).

**Dual Kalman Filter.** A dual Kalman filter [21] simultaneously estimates the unknown state of the system, as well as its process matrix. Similar to our formulation, we assume the system adheres to an autoregressive model. This method then applies two Kalman filters in parallel: one to estimate the coefficients of the autoregressive function, and a second to estimate the trajectory of the camera, based on the current estimate of the autoregressive model. Again, parameters were tuned through validation.

**Conditional Regression Forests.** Conditional regression forests (CRFs) [11] split the training data into multiple subsets. We tested various splitting methods based on camera position and velocity, such as dividing the data into $4, 8$ and $16$ subsets of pan angle. We also tried both disjoint sets and joint sets with different overlap ratios. We report the best result from 8 subsets with 50% overlap. The output of the CRF is further smoothed by a SG filter.

**Filter Forests.** Filter forests (FF) [15] is an efficient discriminative approach for predicting continuous variables given a signal. FF can learn the optimal filtering kernels to smooth temporal signals. Our implementation includes some adaptations, such as limited candidate window sizes, to improve the performance on our datasets.

### 6.2. Benchmark Experiments

Fig. 4 shows the benchmark performance evaluated for both basketball and soccer. We evaluate using joint loss (2) with $\omega = 500$. The precision and smoothness losses are plotted separately to illustrate their relative contributions to the joint loss. For both settings, we see that our approach achieves the best performance, with the performance gap being especially pronounced in basketball.

We note that the soccer setting is significantly more challenging than basketball, and no method performs particularly well for soccer. One possible explanation is that soccer camera planning using only player detections is unreliable due to players not following the ball (unlike in basketball). A visual inspection of the generated videos also suggests that lack of ball tracking in the input signal $\mathbf{x}$ is a significant limitation in the soccer setting.

We also observe that our approach achieves very low smoothness loss, despite not utilizing a post-processing smoothing step (see Table 1 for a summary description of

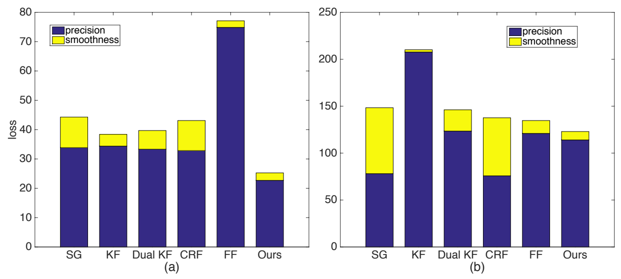


Figure 4. **Prediction loss**. (a) Basketball; (b) Soccer. Loss measured by (2). Our method achieves the lowest overall loss.

| Method | Noise Model | Post-Smooth | Delay |
|---|---|---|---|
| SG & CRF | high-freq | required | Yes |
| KF & Dual KF | Gaussian | required | No |
| FF | self-learned | not-required | No |
| Ours | self-learned | not-required | No |

Table 1. **Qualitative comparison**. Both our method and FF [15] learn noise model from the data, do not require post-processing and achieve real-time response. However, our model requires less data and is efficient for training and testing.

the different approaches). For instance, in the basketball setting, our approach actually achieves the smoothest performance. In fact, for the basketball setting, our approach dominates all baselines in both imitation loss and smoothness loss. The KF and FF baselines tend to achieve competitive smoothness loss, but can suffer substantial imitation loss. For soccer, the SG and CRF baselines achieve lower imitation loss, suggesting that they might be better able to track the action. However, they suffer from substantial jitter.

## 6.3. Visual Inspection

Figs. 5 and 6 show a visualization of the predictions. From this visual inspection, we can verify qualitatively that our method achieves the best balance of precision and smoothness. Our predicted trajectory remains close to the human operator's trajectory and has less jitter than the other methods. Even with post-smoothing, SG and CRF exhibit significant jitter. KF struggles between jitter and over shooting when the noise is not Gaussian. Surprisingly, dual KF performance is worse than KF, which is again presumably because the noise is not Gaussian, and errors in the process estimation propagate to the state estimation (see supplemental material). FF is very competitive in the basketball dataset, but its performance suffers from large jitter in the more challenging soccer dataset.

Table 1 summarizes qualitative properties of all the methods. SG and CRF assume the noise only has high-frequency components. As a result, they struggle with low-frequency noise. KF and dual KF rely on a Gaussian noise assumption, which is not reasonable on our datasets.

Both our method and FF [15] learn control and dynamics aspects from the data. Neither requires post-processing and both are able to generate a zero delay response. In camera planning, our method has three advantages over FF. First, our method has fewer parameters so that it requires less data to train. Second, our model can be trained much faster than FF because FF has to solve a large linear equation in each split. Third, the experimental results show that our method is more stable in both datasets.

Fig. 7 provides a visual comparison of our framework using a random forest of 100 trees and varying influence of the autoregressor, $\lambda$. When $\lambda = 0$, the autoregressor has no influence, and so the smoothness of the prediction depends solely on the smoothness of decision tree ensemble and the size of the history window, $\tau$. Since decision trees are non-

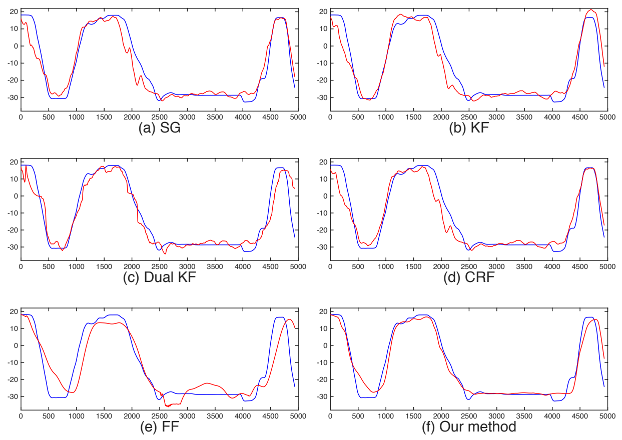


Figure 5. **Comparison on basketball data.** (a) Method of [6], (b) Kalman filter, (c) Dual Kalman filter , (d) Conditional regression forests [11], (e) Filter forests [15], (f) Our method. The blue line is from human operators and the red line is from predictions. Note our method does not need any post-processing.

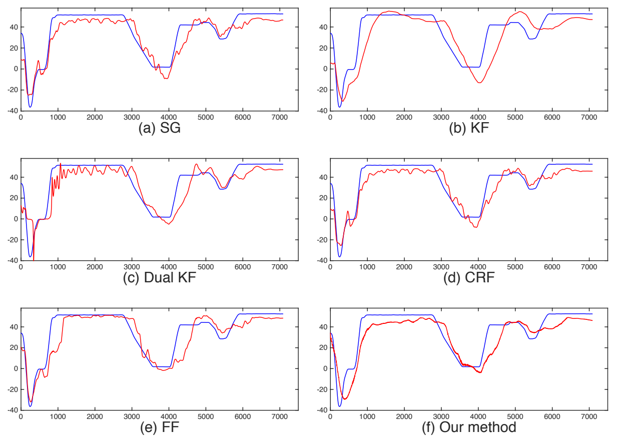


Figure 6. **Comparison on soccer data.**. (a) Method of [6], (b) Kalman filter, (c) Dual Kalman filter , (d) Conditional regression forests [11], (e) Filter forests [15], (f) Our method. The blue line is from human operators and the red line is from predictions. Note our method does not need any post-processing.

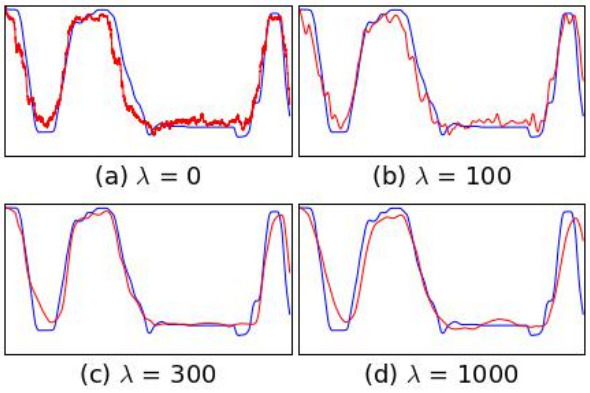


Figure 7. **Comparing varying** $\lambda$ for basketball data. $\lambda = 300$ is a good trade-off between smoothness and accuracy. Very small $\lambda$ allows more accurate but noisier predictions, and very large $\lambda$ leads to smoother but less accurate predictions.

parametric, one could in principle learn a smooth predictor given sufficient training examples and enough trees, but the data and computational costs would be immense. As $\lambda$ increases, the autoregressor causes the predictions to be increasingly smooth. Recall that the learner (Algorithm 1) always seeks to find the predictor within the policy class with the lowest loss, and so it can adaptively trade off between smoothness and accuracy depending on the input signal $\mathbf{x}$ (rather than rely on post-processing). As $\lambda$ becomes very large, the policy class becomes restricted to extremely smooth predictors. In practice, $\lambda$ can be set via a user preference study or validation performance.

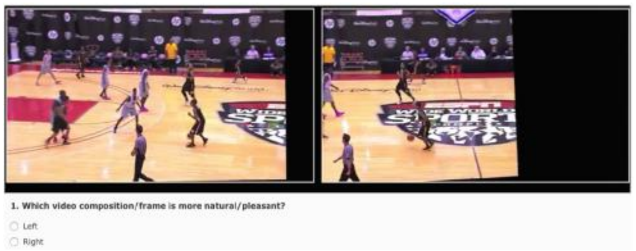
## 6.4. User Preference Study

We also conducted a user preference study to complement our benchmark experiments. Fig. 8 shows our user study interface. Videos were generated by warping the video captured by the human operator. We evaluated our approach against the five baseline algorithms for both basketball and soccer. In each trial, participants were instructed to choose the video that was more appealing (our method was randomly assigned the left or right view).

Table 2 shows the results. For basketball, our method is significantly preferred over the other methods based on a two-tailed binomial test ($p < 0.001$). For soccer, none of the methods performed particularly well (see Section 6.2), making it challenging for users to judge which method generated better videos. Note that there is still a sizable preference gap compared to the human expert.

## 7. Discussion

Although our approach achieved good results for basketball, the results for soccer were much poorer. It is likely that we require more expressive inputs in order to learn good policy, such as tracking the ball in addition to the players.


1. Which video composition/frame is more natural/pleasant?

- Left
- Right

Figure 8. **User study screenshot.** Users were asked which video was more pleasant to watch, and to consider both composition and smoothness in their assessment.

| Comparison | Basketball win / loss | Soccer win / loss |
|---|---|---|
| vs SG | 22 / 3 | 14 / 11 |
| vs KF | 23 / 2 | 12 / 13 |
| vs dual KF | 25 / 0 | 24 / 1 |
| vs CRF | 24 / 1 | 12 / 13 |
| vs FF | 23 / 2 | 14 / 11 |
| vs human | 1 / 24 | 4 / 21 |

Table 2. **User study results**. For basketball, our method is significantly preferred over all baselines. For soccer, all methods performed poorly, and users did not have a strong preference. There is still a sizable preference gap compared to expert human.

The human demonstrations in the soccer dataset were almost entirely piece-wise linear. In other words, the human expert is almost always directing the camera in a straight line with very sharp course corrections. As such, it may be that we require L1 regularization to an autoregressor that prefers zero acceleration in order to better capture the temporal dynamics of camera planning in soccer.

We chose decision trees due to their non-parametric or model-free properties as well as ease of training. Using other complex predictors, such as deep neural nets or Gaussian processes, could potentially also work well.

Finally, our approach only addresses the planning problem, and cannot be directly applied to physical camera control without a control model. It would be interesting to jointly learn to both planning and physical control [7].

## 8. Summary

We have introduced the problem of smooth online imitation learning, where the goal is to learn a predictor to smoothly imitate a human expert given a stream of input signals. We have also presented a recurrent non-parametric model class for generating smooth predictions, which jointly integrates a model-free control signal with a model-based autoregressor. In order to guarantee stability of training, we extended previous work on iterative learning of dynamical models to our setting. We applied our approach to camera control in sports, where we demonstrated significant improvements over several strong baselines.

## References

[1] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden markov support vector machines. In *International Conference on Machine Learning (ICML)*, 2003. 2

[2] S. Bengio, O. Vinyals, N. Jaitly, and N. Shazeer. Scheduled sampling for sequence prediction with recurrent neural networks. In *Neural Information Processing Systems (NIPS)*, 2015. 4

[3] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 3, 4

[4] P. Carr, Y. Sheikh, and I. Matthews. Monocular object detection using 3D geometric primitives. In *ECCV*, 2012. 6

[5] J. Chen and P. Carr. Autonomous camera systems: A survey. In *Workshops at the AAAI Conference on Artificial Intelligence*, 2014. 2

[6] J. Chen and P. Carr. Mimicking human camera operators. In *Applications of Computer Vision (WACV), 2015 IEEE Winter Conference on*, pages 215–222. IEEE, 2015. 2, 6, 7

[7] A. Coates, P. Abbeel, and A. Y. Ng. Apprenticeship learning for helicopter control. *Communications of the ACM (CACM)*, 52(7):97–105, 2009. 8

[8] W. Cohen and V. Carvalho. Stacked sequential learning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005. 2

[9] M. Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2002. 2

[10] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2–3):81–227, 2012. 1, 3, 4

[11] M. Dantone, J. Gall, G. Fanelli, and L. Van Gool. Real-time facial feature detection using conditional regression forests. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 6, 7

[12] H. Daumé III, J. Langford, and D. Marcu. Search-based structured prediction. *Machine learning*, 75(3):297–325, 2009. 1, 2, 5

[13] T. G. Dietterich. Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer, 2002. 2

[14] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *IEEE International Conference on Computer Vision (ICCV)*, 2013. 2, 3

[15] S. R. Fanello, C. Keskin, P. Kohli, S. Izadi, J. Shotton, A. Criminisi, U. Pattacini, and T. Paek. Filter forests for learning data-dependent convolutional kernels. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 4, 6, 7

[16] V. R. Gaddam, R. Eg, R. Langseth, C. Griwodz, and P. Halvorsen. The cameraman operating my virtual camera is artificial: Can the machine be as good as a human ? *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 11(4):56, 2015. 2

[17] V. R. Gaddam, R. Langseth, S. Ljødal, P. Gurdjos, V. Charvillat, C. Griwodz, and P. Halvorsen. Interactive zoom and panning from live panoramic video. In *Network and Operating System Support on Digital Audio and Video Workshop*, 2014. 2

[18] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006. 4

[19] M. Grundmann, V. Kwatra, and I. Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 2

[20] A. Gupta, J. J. Little, and R. J. Woodham. Using line and ellipse features for rectification of broadcast hockey video. In *Canadian Conference on Computer and Robot Vision (CRV)*, 2011. 6

[21] S. Haykin. *Kalman filtering and neural networks*, volume 47. John Wiley & Sons, 2004. 6

[22] K. Kim, D. Lee, and I. Essa. Detecting regions of interest in dynamic scenes with camera motions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 2

[23] T. Kim, Y. Yue, S. Taylor, and I. Matthews. A decision tree framework for spatiotemporal sequence prediction. In *ACM Conference on Knowledge Discovery and Data Mining (KDD)*, 2015. 2, 3

[24] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning (ICML)*, 2001. 2

[25] J. Lee and S. Y. Shin. General construction of time-domain filters for orientation data. *Visualization and Computer Graphics, IEEE Transactions on*, 8(2):119–128, 2002. 2

[26] F. Liu, M. Gleicher, H. Jin, and A. Agarwala. Content-preserving warps for 3d video stabilization. *ACM Transactions on Graphics (TOG)*, 28(3):44, 2009. 2

[27] F. Liu, M. Gleicher, J. Wang, H. Jin, and A. Agarwala. Subspace video stabilization. *ACM Transactions on Graphics (TOG)*, 30(1):4, 2011. 2

[28] G. Rogez, J. Rihan, S. Ramalingam, C. Orrite, and P. H. Torr. Randomized trees for human pose detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. 1, 2

[29] S. Ross, G. Gordon, and J. A. Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *Conference on Artificial Intelligence and Statistics (AISTATS)*, 2011. 2, 5

[30] J. Schimert and A. Wineland. Coupling a dynamic linear model with random forest regression to estimate engine wear. In *Annual Conference of the Prognostics and Health Management Society*, 2010. 1, 2

[31] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press Cambridge, 1998. 1

[32] C.-N. J. Yu, T. Joachims, R. Elber, and J. Pillardy. Support vector training of protein alignment models. *Journal of Computational Biology*, 15(7):867–880, 2008. 2

# Supplemental Material for Learning Online Smooth Predictors for Realtime Camera Planning using Recurrent Decision Trees

Jianhui Chen [*] Hoang M. Le [†] Peter Carr [‡] Yisong Yue [†] James J. Little [*]
[*]University of British Columbia [†] California Institute of Technology [‡] Disney Research
{jhchen14, little}@cs.ubc.ca {hmle, yyue}@caltech.edu carr@disneyresearch.com

## 1. Linear Autoregressor Function Class

The autoregresor $f_\pi(y_{-1}, \ldots, y_{-\tau})$ is typically selected from a class of autoregressors $F$. In our experiments, we use regularized linear autoregressors as $F$.

Consider a generic learning policy $\hat{\pi}$ with rolled-out trajectory $\hat{\mathbf{Y}} = \{\hat{y}_t\}_{t=1}^T$ corresponding to the input sequence $\mathbf{X} = \{x_t\}_{t=1}^T$. We form the state sequence $\mathbf{S} = \{s_t\}_{t=1}^T = \{[x_t, \ldots, x_{t-\tau}, \hat{y}_{t-1}, \ldots, \hat{y}_{t-\tau}]\}_{t=1}^T$. We approximate the smoothness of the curve $\hat{\mathbf{Y}}$ by a linear autoregressor

$$f_\pi \equiv f_\pi(s_t) \equiv \sum_{i=1}^{\tau} c_i \hat{y}_{t-i}$$

for a set of constants $\{c_i\}_{i=1}^\tau$ such that $\hat{y}_t \approx f_\pi\,(s_t)$. Given expert feedback $\mathbf{Y}^* = \{y_t^*\}$, the joint loss function becomes

$$\begin{aligned}\ell(y, y_t^*) &= \ell_d(y, y_t^*) + \lambda \ell_R(y, s_t)\\ &= (y - y_t^*)^2 + \lambda (y - \sum_{i=1}^{\tau} c_i \hat{y}_{t-i})^2\end{aligned}$$

Here $\lambda$ trade-offs smoothness versus absolute imitation accuracy. The autoregressor $f_\pi$ acts as a smooth linear regularizer, the parameters of which can be updated at each iteration based on expert feedback $Y^*$ according to

$$\begin{aligned}f_\pi &= \operatorname*{argmin}_{f \in F} \|Y^* - f(Y^*)\|_{\ell_2}\\ &= \operatorname*{argmin}_{c_1, \ldots, c_\tau} (\sum_{t=1}^{T} (y_t^* - \sum_{i=1}^{\tau} c_i y_{t-i}^*)^2), \qquad (1)\end{aligned}$$

In practice we use a regularized version of equation (1) to learn a new set of coefficients $\{c_i\}_{i=1}^\tau$. The `Learn` procedure (Line 7 of algorithm 1) uses this updated $f_\pi$ to train a new policy that optimizes the trades off between $\hat{y}_t \approx y_t^*$ (expert feedback) versus smoothness as dictated by $\hat{y}_t \approx \sum_{i=1}^{\tau} c_i \hat{y}_{t-i}$.

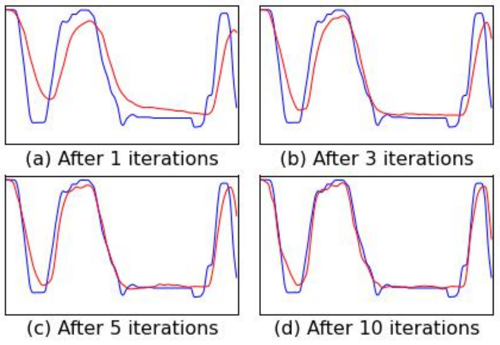


Figure 1: *Performance of learned policy for basketball data after different number of iterations*

### 1.1. Training with Linear Autoregressor

Our application of Algorithm 1 to realtime camera planning proceeds as follows: At each iteration, we form a state sequence $\tilde{\mathbf{S}}$ based on the exploration trajectory $\tilde{\mathbf{Y}}$ and the tracking input data $\mathbf{X}$. We collect expert feedback $\mathbf{Y}^*$ based on each $\tilde{s} \in \tilde{\mathbf{S}}$. At this point, a new linear autoregressor $f_\pi$ is learned based on $\mathbf{Y}^*$, as described in the previous section. We then train a new model $\hat{\pi}$ based on $\tilde{\mathbf{S}}$, $\mathbf{Y}^*$, and this updated autoregressor $f_\pi$, using our recurrent decision tree framework (Line 7 of Algorithm 1). Note that typically this creates a "chicken-and-egg" problem. As the newly learned policy $\hat{\pi}$ is greedily trained with respect to $\mathbf{Y}^*$, the rolled-out trajectory of $\hat{\pi}$ may have a state distribution that is different from what the previously learned $f_\pi$ would predict. Our approach offers two remedies to this circular problem. First, by allowing expert feedback to vary smoothly relative to the current exploration trajectory $\tilde{\mathbf{Y}}$, the new policy $\hat{\pi}$ should induce a new autoregresor that is similar to previously learned $f_\pi$. Second, by interpolating distributions (Line 10 of Algorithm 1) and having $\mathbf{Y}^*$ eventually converge to the original human trajectory $\mathbf{Y}$, we will have a stable and converging state distribution, leading to a stable and converging $f_\pi$.

Fig. 1 illustrates the effect of applying Algorithm 1 as outlined above using the adaptive interpolation parameter $\beta$ to the basketball data. Throughout iterations, the linear autoregressor $f_\pi$ enforces smoothness of the rolled-out trajectory, while the recurrent decision tree framework learns an increasingly accurate imitation policy. We generally achieve a satisfactory policy after 5-10 iterations in our basketball and soccer data sets. In the following section, we describe the mechanics of our recurrent decision tree training.

## 2. Recurrent Decision Tree Training

Empirically, decision tree-based ensembles are among the best performing supervised machine learning method [3, 4]. Due to the piece-wise constant nature of decision tree-based prediction, the results are typically non-smooth. We propose a recurrent extension, where the prediction at each leaf node is not necessarily constant, but rather is a smooth function of both static leaf node prediction and its previous predictions.

Given state $s$ as input, a decision tree specifies a partitioning of the input state space. Let $D = \{(s_m, y^*_m)\}_{m=1}^M$ denote a training set of state/target pairs. Conventional regression tree learning aims to learn a partitioning such that each leaf node, $\texttt{node}$, makes a constant prediction via minimizing the squared loss function:

$$\begin{aligned}\bar{y}_{\texttt{node}} &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} \ell_d(y, y^*) \\ &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} (y^* - y)^2, \qquad (2)\end{aligned}$$

where $D_{\texttt{node}}$ denotes the training data from $D$ that has partitioned into the leaf $\texttt{node}$. For squared loss, we have:

$$\bar{y}_{\texttt{node}} = \texttt{mean}\left\{y^* \,|(s, y^*) \in D_{\texttt{node}}\right\}. \qquad (3)$$

In the recurrent extension, we allow the decision tree to branch on the input state $s$, which includes the previous predictions $y_{-1}, \ldots, y_{-\tau}$. To enforce more explicit smoothness requirements, let $f_\pi(y_{-1}, \ldots, y_{-\tau})$ denote an autoregressor that captures the temporal dynamics of $\pi$ over the distribution of input sequences $d_{\mathbf{x}}$, while *ignoring* the inputs $x$. At time step $t$, $f_\pi$ predicts the behavior $y_t = \pi(s_t)$ given only $y_{t-1}, \ldots, y_{t-\tau}$.

Our policy class $\Pi$ of recurrent decision trees $\pi$ makes smoothed predictions by regularizing the predictions to be close to its autoregressor $f_\pi$. The new loss function incorporates both the squared distance loss $\ell_d$, as well as a smooth regularization loss such that:

$$\begin{aligned}\mathcal{L}_D(y) &= \sum_{(s,y^*)\in D} \ell_d(y, y^*) + \lambda \ell_R(y, s) \\ &= \sum_{(s,y^*)\in D} (y - y^*)^2 + \lambda (y - f_\pi(s))^2\end{aligned}$$

where $\lambda$ is a hyper-parameter that controls how much we care about smoothness versus absolute distance loss.

**Making prediction:** For any any tree/policy $\pi$, each leaf node is associated with the terminal leaf node value $\bar{y}_{\texttt{node}}$ such that prediction $\hat{y}$ given input state $s$ is:

$$\begin{aligned}\hat{y}(s) \equiv \pi(s) &= \underset{y}{\operatorname{argmin}}\ (y - \bar{y}_{\texttt{node}(s)})^2 + \lambda (y - f_\pi(s))^2 \\ &= \frac{\bar{y}_{\texttt{node}(s)} + \lambda f_\pi(s)}{1+\lambda}. \qquad (4)\end{aligned}$$

where $\texttt{node}(s)$ denotes the leaf node of the decision tree that $s$ branches to.

**Setting terminal node value:** Given a fixed $f_\pi$ and decision tree structure, navigating through consecutive binary queries eventually yields a terminal leaf node with associated training data $D_{\texttt{node}} \subset D$.

One option is to set the terminal node value $\bar{y}_{\texttt{node}}$ to satisfy:

$$\begin{aligned}\bar{y}_{\texttt{node}} &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} \ell_d(\hat{y}(s|y), y^*) \\ &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} (\hat{y}(s|y) - y^*)^2 \qquad (5) \\ &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} (\frac{y + \lambda f_\pi(s)}{1+\lambda} - y^*)^2 \qquad (6)\end{aligned}$$

for $\hat{y}(s|y)$ defined as in (4) with $y \equiv \bar{y}_{\texttt{node}(s)}$. Similar to (3), we can write the closed-form solution of (5) as:

$$\bar{y}_{\texttt{node}} = \texttt{mean}\left\{(1+\lambda)y^* - \lambda f_\pi(s) \,|(s, y^*) \in D_{\texttt{node}}\right\}. \qquad (7)$$

When $\lambda = 0$, (7) reduces to (3).

Note that (5) only looks at imitation loss $\ell_d$, but not smoothness loss $\ell_R$. Alternatively in the case of joint imitation and smoothness loss, the terminal leaf node is set to minimize the joint loss function:

$$\begin{aligned}\bar{y}_{\texttt{node}} &= \underset{y}{\operatorname{argmin}}\, \mathcal{L}_{D_{\texttt{node}}}(\hat{y}(s|y)) \\ &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} \ell_d(\hat{y}(s|y), y^*) + \lambda \ell_R(\hat{y}(s|y), s) \\ &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} (\hat{y}(s|y) - y^*)^2 + \lambda(\hat{y}(s|y) - f_\pi(s))^2 \\ &= \underset{y}{\operatorname{argmin}} \sum_{(s,y^*)\in D_{\texttt{node}}} (\frac{y + \lambda f_\pi(s)}{1+\lambda} - y^*)^2 \\ &+ \lambda(\frac{y + \lambda f_\pi(s)}{1+\lambda} - f_\pi(s))^2 \qquad (8) \\ &= \texttt{mean}\left\{y^* \,|(s, y^*) \in D_{\texttt{node}}\right\}, \qquad (9)\end{aligned}$$

**Node splitting mechanism:** For a node representing a subset $D_{\texttt{node}}$ of the training data, the node impurity is defined as:

$$\begin{aligned} I_{\texttt{node}} &= \mathcal{L}_{D_{\texttt{node}}}(\bar{y}_{\texttt{node}}) \\ &= \sum_{(s,y^*)\in D_{\texttt{node}}} \ell_d(\bar{y}_{\texttt{node}}, y^*) + \lambda \ell_R(\bar{y}_{\texttt{node}}, s) \\ &= \sum_{(s,y^*)\in D_{\texttt{node}}} (\bar{y}_{\texttt{node}} - y^*)^2 + \lambda(\bar{y}_{\texttt{node}} - f_\pi(s))^2 \end{aligned}$$

where $\bar{y}_{\texttt{node}}$ is set according to equation (7) or (9) over $(s, y^*)$'s in $D_{\texttt{node}}$. At each possible splitting point where $D_{\texttt{node}}$ is partitioned into $D_{\texttt{left}}$ and $D_{\texttt{right}}$, the impurity of the left and right child of the node is defined similarly. As with normal decision trees, the best splitting point is chosen as one that maximizes the impurity reduction: $I_{\texttt{node}} - \frac{|D_{\texttt{left}}|}{|D_{\texttt{node}}|} I_{\texttt{left}} - \frac{|D_{\texttt{right}}|}{|D_{\texttt{node}}|} I_{\texttt{right}}$

**Parameters:** The window size of history time is $\tau = 40$ of previous time frames. The number of iterations is between $5 - 10$ for the basketball and soccer data sets.

## 3. Baselines

We build two baseline methods based on Kalman filter with unknown noise covariances [6].

### 3.1. Kalman Filter

The noisy, non-smooth target pan positions $\hat{y}'_t$ are generated by a random decision forest (equivalent to the time invariant predictions of Equation 3 in the main paper). A Kalman filter is used to estimate a smooth variant $\hat{y}_t$ from the noisy time invariant predictions $\hat{y}'_t$ (Kalman smoothing of noisy predictions according to Equation 5 in the main paper).

We represent the unknown, smoothly varying state $\Phi_t = [\theta_t, \dot{\theta}_t]$ of the camera as a combination of instantaneous pan angle $\theta_t$ and pan velocity $\dot{\theta}_t$. The internal state of the camera $\Phi_t$ evolves over time based on a state transition matrix $\texttt{F}$. The internal state can also be influenced by an external signal $u_t$ and corresponding control matrix $\texttt{B}$. The discrepancy $\mathbf{w}_t$ is modeled as random noise.

$$\Phi_{t+1} = \texttt{F}\Phi_t + \texttt{B}u_t + \mathbf{w}_t. \tag{10}$$

Each time invariant prediction $\hat{y}'_t$ is an observation of the unknown state $\Phi_t$. Using the measurement matrix $\texttt{H}$, we can generate the expected observation $\texttt{H}\Phi_t$. The discrepancy $v_t$ between the actual observation and the expected observation is modeled as random noise.

$$\hat{y}'_t = \texttt{H}\Phi_t + v_t. \tag{11}$$

The filter estimates $\Phi_t$, which is the basis for the outputted smooth approximation $\hat{y}_t = \theta_t$ of the input noisy signal $\hat{y}'_t$ (see Equation 5 of the paper).

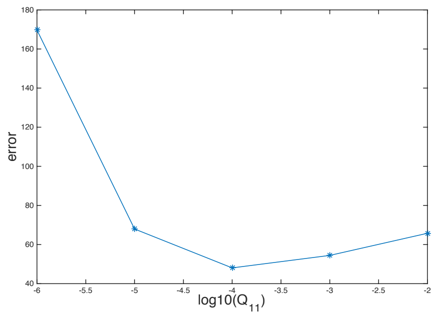


Figure 2: *Cross validation on constant velocity with no external control. The minimum error is achieved when* $Q_{11} = 1.0e^{-4}$.

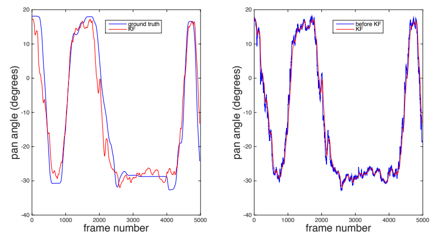


Figure 3: *Kalman filter testing result. The spatiotemporal loss is 30.95.*

In practice, both process (10) and measurement (11) are noisy. The sources of noise are assumed to be independent normal distributions with zero mean and covariance matrices $Q$ and $R$, respectively.

By setting different dynamic model and observation model, the smoothly varying state $\Phi_t$ can be recovered using the standard Kalman filtering method [2]. We explore the smoothing ability of Kalman filter by setting different $F$ and $H$. The measurement covariance matrix $R$ is set as the the standard deviation of the raw predictions $y'_t$ relative to the ground truth $y_t$ (on the training data). The process covariance matrix $Q$ is set by cross validation using the simplification method from [1]. The simplification method only puts a noise term in the lower rightmost element in $Q$ to approximate continuous white noise model. The cross validation error is measured by the joint loss function:

$$\frac{1}{T}\sum_t (y_t - \theta_t)^2 + 500 \times \dot{\theta}^2. \tag{12}$$

#### 3.1.1 Constant Position

In this simple model, we only model the pan angle in the dynamic and observation model, thus

$$\begin{aligned} F &= \begin{bmatrix}1\end{bmatrix} \\ H &= \begin{bmatrix}1\end{bmatrix} \end{aligned}$$

Fig. 2 shows the cross validation error. Fig. 3 shows the testing result.

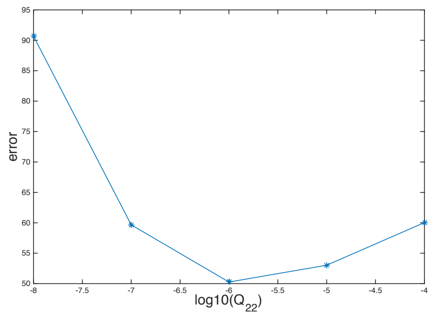


Figure 4: *Cross validation on constant velocity. The minimum error is achieved when* $Q_{22} = 1.0e^{-6}$.

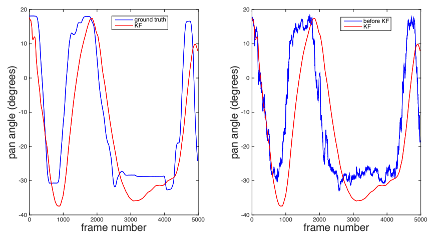


Figure 5: *Kalman filter testing result. The spatiotemporal loss is 38.12.*

### 3.1.2 Constant Velocity

In this model,

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}$$

$$H = \begin{bmatrix} 1 & 0 \end{bmatrix}$$

Fig. 4 shows the cross validation error. Fig. 5 shows the testing result.

### 3.1.3 Constant Velocity with External Acceleration

In this model,

$$F = \begin{bmatrix} 1 & 1 & 0.5 \\ 0 & 1 & 1 \\ 0 & 0 & 0 \end{bmatrix}$$
$$H = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

The camera is undergoing external acceleration, leading to change in velocity as well as the position. Assume the external control is instantaneous accelerations, and there is no inherent pattern (e.g. smoothness in the control signal). As a result, there is no correlation between $\ddot{\phi}_t$ and $\ddot{\phi}_{t+1}$, which is the reason that the last row of `F` consists of all zeros.

Fig. 6 shows the cross validation error. Fig. 7 shows the testing result.

## 3.2. Dual Kalman filter

In the dual Kalman filter, both the states of the dynamic system and its parameters are estimated simultaneously, given only noisy observation [5]. Fig. 9 shows the result of the dual Kalman filter.

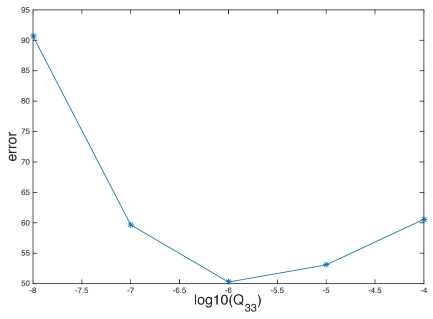


Figure 6: *Cross validation on constant velocity with external acceleration. The minimum error is achieved when* $Q_{33} = 1.0e^{-6}$.

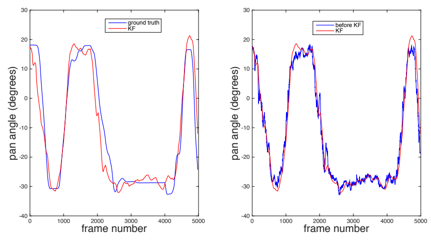


Figure 7: *Kalman filter (constant velocity with external acceleration) testing result. The spatiotemporal loss is 37.43.*

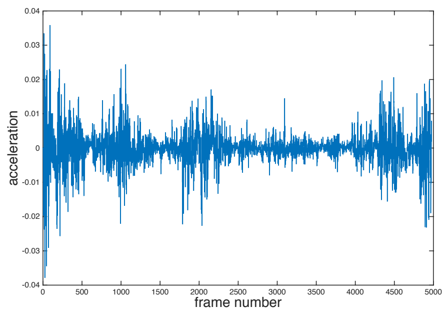


Figure 8: *Estimated accelerations.*

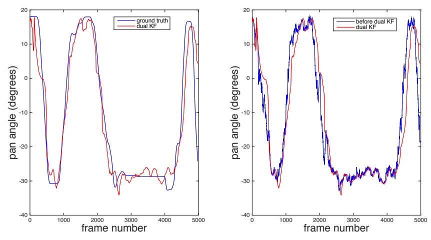


Figure 9: *Dual Kalman filter result. The spatiotemporal loss is 39.43.*

## References

[1] Kalman and Bayesian filters in python. http://gitxiv.com/posts/4wYYffue4WfnhKZoB/book-kalman-and-bayesian-filters-in-python. Accessed: 2015-10-24. 3

[2] G. Bishop and G. Welch. An introduction to the Kalman filter. Technical report, 2001. 3

[3] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *International Conference on Machine Learning (ICML)*, 2006. 2

[4] A. Criminisi, J. Shotton, and E. Konukoglu. Decision forests: A unified framework for classification, regression, density estimation, manifold learning and semi-supervised learning. *Foundations and Trends in Computer Graphics and Vision*, 7(2–3):81–227, 2012. 2

[5] S. Haykin. *Kalman filtering and neural networks*, volume 47. John Wiley & Sons, 2004. 4

[6] M. Nilsson. Kalman filtering with unknown noise covariances. 2006. 3