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 λ trade-offs smoothness versus absolute imitation accuracy. The autoregressor f_π 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_π to train a new policy that optimizes the trades off between $\hat{y}_t \approx y_t^*$ (expert feedback) versus smoothness as dictated by $\hat{y}_t \approx \sum_{i=1}^{\tau} c_i \hat{y}_{t-i}$.

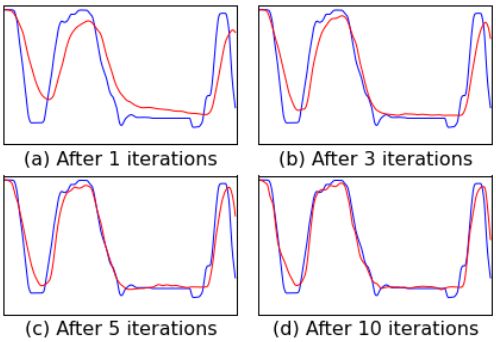

Figure 1: *Performance of learned policy for basketball data after different number of iterations*

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_π 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_π, 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_π 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_π. 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_π.

Fig. 1 illustrates the effect of applying Algorithm 1

as outlined above using the adaptive interpolation parameter β to the basketball data. Throughout iterations, the linear autoregressor f_π 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:

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

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

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

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 π over the distribution of input sequences $d_{\mathbf{x}}$, while *ignoring* the inputs x. At time step t, f_π predicts the behavior $y_t = \pi(s_t)$ given only $y_{t-1}, \ldots, y_{t-\tau}$.

Our policy class Π of recurrent decision trees π makes smoothed predictions by regularizing the predictions to be close to its autoregressor f_π. The new loss function incorporates both the squared distance loss ℓ_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 λ is a hyper-parameter that controls how much we care about smoothness versus absolute distance loss.

Making prediction: For any any tree/policy π, 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:

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

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

Setting terminal node value: Given a fixed f_π 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:

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

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:

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

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

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

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

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 θ_t and pan velocity $\dot{\theta}_t$. The internal state of the camera Φ_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 Φ_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 Φ_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).

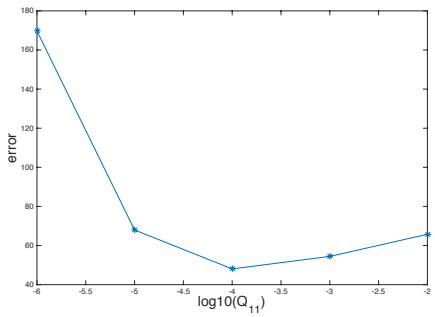

Figure 2: *Cross validation on constant velocity with no external control. The minimum error is achieved when* $Q_{11} = 1.0e^{-4}$.

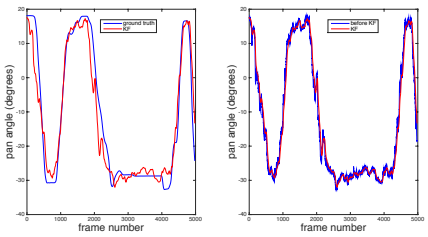

Figure 3: *Kalman filter testing result. The spatiotemporal loss is 30.95.*

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 Φ_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

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

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

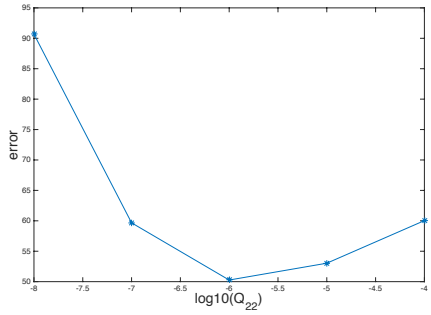

Figure 4: *Cross validation on constant velocity. The minimum error is achieved when* $Q_{22} = 1.0e^{-6}$.

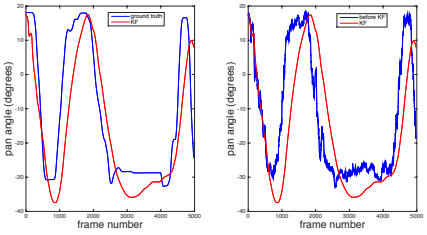

Figure 5: *Kalman filter testing result. The spatiotemporal loss is 38.12.*

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.

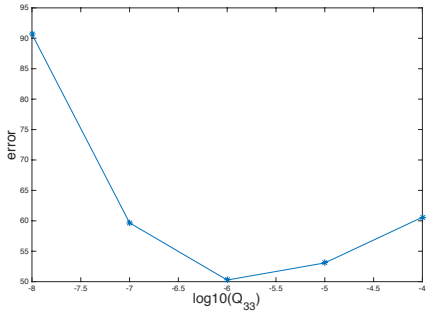

Figure 6: *Cross validation on constant velocity with external acceleration. The minimum error is achieved when* $Q_{33} = 1.0e^{-6}$.

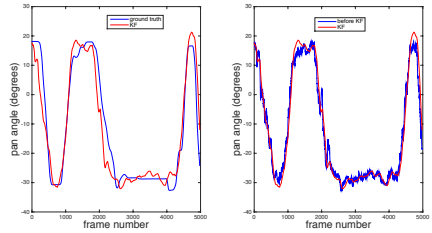

Figure 7: *Kalman filter (constant velocity with external acceleration) testing result. The spatiotemporal loss is 37.43.*

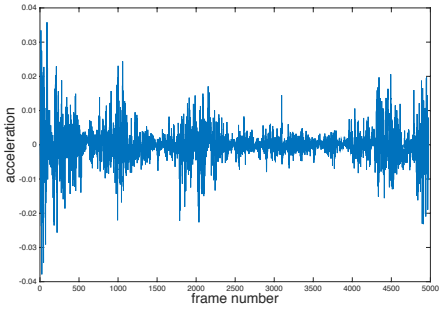

Figure 8: *Estimated accelerations.*

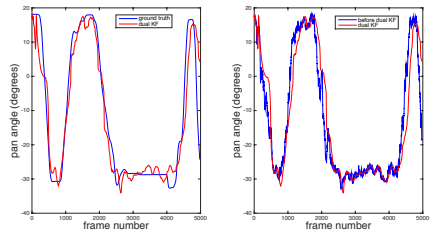

Figure 9: *Dual Kalman filter result. The spatiotemporal loss is 39.43.*

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