Learning to Transform Time Series with a Few Examples

Ali Rahimi, Ben Recht, Trevor Darrell

Abstract

We describe a semi-supervised regression algorithm that learns to transform one time series into another time series given examples of the transformation. This algorithm is applied to tracking, where a time series of observations from sensors is transformed to a time series describing the pose of a target. Instead of defining and implementing such transformations for each tracking task separately, our algorithm learns a memoryless transformation of time series from a few example input-output mappings. The algorithm searches for a smooth function that fits the training examples and, when applied to the input time series, produces a time series that evolves according to assumed dynamics. The learning procedure is fast and lends itself to a closed-form solution. It is closely related to nonlinear system identification and manifold learning techniques. We demonstrate our algorithm on the tasks of tracking RFID tags from signal strength measurements, recovering the pose of rigid objects, deformable bodies, and articulated bodies from video sequences. For these tasks, this algorithm requires significantly fewer examples compared to fully-supervised regression algorithms or semi-supervised learning algorithms that do not take the dynamics of the output time series into account.

Keywords: *Semi-supervised learning, example-based tracking, manifold learning, nonlinear system identification*.

Learning to Transform Time Series with a Few Examples

I. Introduction

Many fundamental problems in machine perception, computer graphics, and controls involve the transformation of one time series into another. For example, in tracking, one transforms a time series of observations from sensors to the pose of a target; one can generate computer animation by transforming a time series representing the motions of an actor to vectorized graphics; and fundamentally, a controller maps a time series of measurements from a plant to a time series of control signals. Typically, such time series transformations are specified programmatically with application-specific algorithms. We present an alternative: algorithms that learn how to transform time series from examples. This article demonstrates how nonlinear regression techniques, when augmented with a prior on the dynamics of their output, can transform a variety of time series with very few output examples.

Given enough input-output examples, nonlinear regression techniques can learn and represent any smooth mapping using any sufficiently general family of functions such as multilayer perceptrons or radial basis functions. But for many of the time series transformation applications addressed here, traditional nonlinear regression techniques require too many input-output examples to be of practical use. To accommodate the dearth of available examples, our algorithms utilize easy-to-obtain side information in the form of a prior distribution on the output time series. Utilizing this prior to regularize the output allows our algorithms to take advantage of "unlabeled," examples, or examples for which no output example is provided.

In tracking applications, the output time series represents the motion of physical objects, so we expect that this time series will exhibit physical dynamics. We assume that a linear-Gaussian autoregressive model that roughly captures the dynamics of the output time series is *a priori* available. It is convenient to specify these dynamics by hand, as is done in much of the tracking literature. Like nonlinear regression methods, our algorithms search for a smooth function that fits the given input-output examples. In addition, this function is also made to map inputs to outputs that exhibit temporal behavior consistent with the given dynamical model. The search

for this function is expressed as a joint optimization over the labels of the unlabeled examples and a mapping in a Reproducing Kernel Hilbert Space. We show empirically that the algorithms are not very sensitive to their parameter settings, including those of the dynamics model, so fine tuning this model is often not necessary.

We demonstrate our algorithms with an interactive tracking system for annotating video sequences: the user specifies the desired output for a few key frames in the video sequence, and the system recovers labels for the remaining frames of the video sequence. The output examples are real-valued vectors that may represent any temporally smoothly varying attribute of the video frames. For example, to track the limbs of a person in a 1-minute-long video, only 13 frames of the video needed to be manually labeled with the position of the person's limbs. The estimated mapping between video frames and the person's pose is represented using radial kernels centered on the frames of the video sequence. Because the system can take advantage of unlabeled data, it can track limbs much more accurately than simple interpolation or traditional nonlinear regression given the same number of examples. The very same algorithm can be used to learn to track deformable contours like the contour of lips (represented with a spline curve). The system is interactive, so the user may specify additional examples to improve the performance of the system where needed. Our method works on directly on the provided feature space, which in the case of images may be raw pixels, extracted silhouttes, or tracked contours. No explicit representation or reasoning about occlusions or 3-D is required in our approach. Our algorithms can also be applied in non-visual settings. We can learn to transform the voltages induced in a set of antennae by a Radio Frequency ID (RFID) tag to the position of the tag with only four labeled examples.

Our main contribution is to demonstrate empirically that for a large variety of tracking problems, sophisticated generative models and nonlinear filters that are prone to local minima are not needed. Instead, a few examples coupled with very generic assumptions on the dynamics of the latent space, and simple quadratic optimization are sufficient. We demonstrate this by regularizing the output of a regression algorithm with a dynamical model. This results in a nonlinear system identification algorithm that, rather than estimating an observation function that maps latent states to observations, estimates a function that maps observations to latent states. This affords us significant computational advantages over existing nonlinear system identification algorithms when the observation function is invertible and the dynamics are known, linear and

Gaussian. When the dimensionality of each output is lower than that of each observation, this estimated function performs dimensionality reduction. Thus, our contribution is also a semi-supervised manifold learning algorithm that takes advantage of the dynamics of the low-dimensional representation.

II. Related Work

This work is closely related to the problem of nonlinear dimensionality reduction using manifold learning. Manifold learning techniques [1]–[6] find low-dimensional representations that preserve geometric attributes high-dimensional observed points. To define these attributes, these algorithms identify local neighborhoods along the manifold of observations by thresholding pairwise distances in the ambient space. When the manifold is sparsely sampled, neighboring points along the manifold are difficult to identify, and the algorithms can fail to recover any meaningful geometric attributes [7]. Our approach utilizes the time-ordering of data points to obviate the need to construct neighborhoods. While we do rely on the distance between pairs of points, these distances are used only to enforce the smoothness of the manifold.

Jenkins and Mataric [8] suggest artificially reducing the distance between temporally adjacent points to provide an additional hint to Isomap about the local neighborhoods of image windows. We take advantage of dynamics in the low-dimensional space to allow our algorithm to better estimate the distance between pairs of temporally adjacent points along the manifold. This requires only fine enough sampling over time to retain the temporal coherence between video frames, which is much less onerous than the sampling rate required to correctly estimate neighborhood relationships in traditional manifold learning algorithms. Various semi-supervised extensions to manifold learning algorithms have been proposed [9], [10], but these algorithms still do not take advantage of the temporal coherence between adjacent samples of the input time series.

Our technique is semi-supervised in that it takes advantage of both labeled data (the input-output examples) and unlabeled data (the portions of the input time series without output examples). The semi-supervised regression approaches of [11] and [12] take into account the manifold structure of the data, but they also rely on estimates of the neighborhood structure, and do not take advantage of the time ordering of the data set. These semi-supervised regression methods are similar to our method in that they also impose a random field on the low-dimensional representation. The work presented here augments these techniques by introducing the temporal

dependency between output samples in the random field. It can be viewed as a special case of estimating the parameters of a continuous-valued conditional random field [13] or a manifold learning algorithm based on function estimation [14]. The algorithms in this article are based on our earlier work [15]. Here, we provide a simpler formulation of the problem along with a variant, a more thorough comparison against existing algorithms, and some new tracking applications.

Nonlinear system identification seeks to recover the parameters of a generative model for observed data (see [16]–[19] and references within). Typically, the model is a continuous-valued hidden Markov chain, where the state transitions are governed by an unknown nonlinear state transition function, and states are mapped to observations by an unknown nonlinear observation function. In the context of visual tracking, states are physical configurations of objects, and observations are frames of the video. Common representations for the observation function (such as RBF [16] or MLP [17]) require a great amount of storage, and find the maximum *a posteriori* estimate of the observation function requires optimization procedures that are prone to local optima. Discrete dynamical models, such as HMMs, have also been proposed [20]. Dynamic Textures [21] sidesteps these issues by performing linear system identification instead, which limits it to linear appearance manifolds.

Like conditional random fields [13], the algorithms in this article learn a mapping in the reverse direction, from observations to states, though the states here are Gaussian random variables. Adopting an RBF representation for the mapping results in an optimization problem that is quadratic in the latent states and the parameters of the function to estimate. This makes the problem computationally tractable, not subject to local minima, and significantly reduces the storage requirement in the case of very high-dimensional observations.

III. Background: Function Fitting

We wish to learn a memoryless and time-invariant function that transforms each sample x_t of an input time series $\mathbf{X} = \{x_t\}_{t=1}^T$ to a sample y_t of the output time series $\mathbf{Y} = \{y_t\}_{t=1}^T$. The samples of $\mathbf{X}$ and $\mathbf{Y}$ are M and N-dimensional respectively. We will let sets of vectors such as $\mathbf{X} = \{x_t\}$ also denote matrices that stack the vectors in $\mathbf{X}$ horizontally. In a visual tracking application, each x_t might represent the pixels of an image, with $M \approx 10^6$, y_t might be the joint angles of the limbs of a person in the scene, with $N \approx 20$, and we would seek a transformation from images to joint angles. Function fitting is the process of fitting a function $f : \mathcal{R}^M \rightarrow \mathcal{R}^N$

given a set of input-output examples $\{x_i, y_i\}$, with $0 \leq i \leq T$. Scenarios where not all the y's are given are discussed later.

To find the best mapping f one defines a loss function over the space of functions using a loss $V(y, z)$ between output labels. Additionally, one may place a regularizer on f or on its parameters to improve stability [22], resulting in an optimization of the form:

$$\min_f \sum_{i=1}^{T} V(f(x_i), y_i) + P(f). \tag{1}$$

The mapping f can take any form that is algorithmically convenient, such as linear, polynomial, radial basis function (RBF), neural networks, or the nearest neighbors rule. In this article, we focus on the RBF representation, though other representations, such as nearest neighbors, also result in simple optimizations.

The Radial Basis Functions (RBF) form consists of a weighted sum of radial basis functions centered at pre-specified centers $\{c_j\}_{1 \cdots J}$.

$$f_\theta(x) = \sum_{j=1}^{J} \theta_j k(x, c_j). \tag{2}$$

Here, the parameters θ of the function consist of vectors $\theta_j \in \mathcal{R}^N$, and $k : \mathcal{R}^M \times \mathcal{R}^M \rightarrow \mathcal{R}$ is a function of the Euclidean distance between its arguments. When V in Equation (1) is the quadratic loss, estimating θ reduces to a least-squares problem, since the output of f is linear in its parameters θ.

A. Reproducing Kernel Hilbert Spaces

The theory of Reproducing Kernel Hilbert Spaces (RKHS) provides a guide for a stabilizer P and a set of basis functions for function fitting. Every positive definite kernel $k : \mathcal{R}^M \times \mathcal{R}^M \rightarrow \mathcal{R}$ defines an inner product on bounded functions whose domains is a compact subset of $\mathcal{R}^N$ and whose range is $\mathcal{R}$ [23]. This inner product is defined so that it satisfies the so-called *reproducing property* $\langle k(x, \cdot), f(\cdot) \rangle = f(x)$. That is, in the RKHS, taking the inner product of a function with $k(x, \cdot)$ evaluates that function at x. The norm $\| \cdot \|$ in this Hilbert space is defined in terms of this inner product in the usual way.

This norm favors smooth functions, and will serve as the stabilizer P for function fitting. According to Mercer's theorem [23], k has a countable representation on a compact domain: $k(x_1, x_2) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x_1) \phi_i(x_2)$, where the functions $\phi_i : \mathcal{R}^M \rightarrow \mathcal{R}$ are linearly independent.

Combining this with the reproducing property reveals that the set of ϕ are a countable basis for the RKHS:

$$f(x) = \langle f(\cdot), k(x,\cdot)\rangle = \langle f(\cdot), \sum_{i=1}^{\infty} \lambda_i \phi_i(\cdot)\phi_i(x)\rangle = \sum_{i=1}^{\infty} \phi_i(x)\lambda_i \langle f(\cdot), \phi_i(\cdot)\rangle = \sum_{i=1}^{\infty} \phi_i(x) c_i, \quad (3)$$

where $c_i = \lambda_i \langle f(\cdot), \phi_i(\cdot)\rangle$ are the coefficients of f in the basis set defined by the ϕ_i.

The functions ϕ are orthogonal under this inner product: by the reproducing property, we have $\phi_j(x) = \langle \phi_j(\cdot), k(x,\cdot)\rangle = \sum_{i=1}^{\infty} \phi_i(x)\lambda_i \langle \phi_j(\cdot), \phi_i(\cdot)\rangle$. The ϕ's form a basis, so ϕ_j cannot be written as a linear combination of other ϕ's. This implies $\langle \phi_i, \phi_j\rangle = \delta_{ij}/\lambda_i$, or that the ϕ's are orthonormal.

The norm of a function in the RKHS can therefore be expressed in terms of its coefficients:

$$\|f\|_k^2 = \langle f, f\rangle = \langle \sum_{i=1}^{\infty} \phi_i c_i, \sum_{i=1}^{\infty} \phi_i c_i\rangle = \sum_{ij} c_i c_j \langle \phi_i, \phi_j\rangle = \sum_i c_i^2/\lambda_i. \quad (4)$$

An RBF kernel $k(x,x') = k(\|x - x'\|)$ has sinusoidal bases ϕ [23], [24], so the norm $\|f\|_k^2$ penalizes the coefficients the projection of f on sinusoids. When k is a Gaussian kernel $k(x',x) = \exp(-\|x-x'\|^2/\sigma_k^2)$, λ_i are positive and decaying with increasing i. Thus $\|f\|_k$ under this kernel penalizes the high frequency content in f more than the low-frequency content, favoring smoother f's [23], [24].

B. Nonlinear Regression with Tikhonov Regularization on an RKHS

Since the RKHS norm for the Gaussian kernel favors smooth functions, we may use it as a stabilizer for function fitting. We fit a multivariate function $f = [f^1(x) \dots f^N(x)]$ to data by applying Tikhonov regularization to each scalar-valuded component of f independently. Denoting the dth component of each y_i by y_i^d, the Tikhonov problem for each f^d becomes:

$$\min_{f^d} \sum_{i=1}^{T} V(f^d(x_i), y_i^d) + \lambda_k \|f^d\|_k^2. \quad (5)$$

The minimization is over the RKHS defined by the kernel k, and λ_k is a scalar that controls the trade-off between smoothness and agreement with the training data.

Although the optimization (5) is a search over a function space, the *Representer Theorem* states that its minimizer can be represented as a weighted sum of kernels placed at each x_i [25]:

$$f^d(x) = \sum_{i=1}^{T} c_i^d k(x, x_i). \quad (6)$$

To see that the optimum of (5) must have this form, we show that any solution containing a component that is orthogonal to the space of functions spanned by this form must have a greater cost according to (5), and therefore cannot be optimal. Specifically, suppose the optimal solution has the form $f = g + h$, with g having the form (6), and h non-zero and orthogonal to all functions of the form (6), i.e., for all c, $\langle \sum_{i=1} c_i k(\cdot, x_i), h \rangle = 0$. By the reproducing property, we have $\sum_i c_i h(x_i) = 0$ for all c, so $h(x_i) = 0$. Therefore, $f(x_i) = g(x_i)$. But by the orthogonality of f and g, $\|f\|_k^2 = \|g\|_k^2 + \|h\|_k^2$, so $\|f\|_k^2$ is strictly greater than $\|g\|_k^2$, even though their corresponding data terms are equal. Therefore, f cannot be optimal.

When $V(x, y)$ has the quadratic form $(x - y)^2$, we can use the representer theorem to reduce (5) into a finite-dimensional least-squares problem in the coefficients of f. The optimal solution given by Equation (6) can be written in vector form as $K_x' c^d$, where the ith component of the column vector K_x is $k(x, x_i)$, c^d is a column vector of coefficients, and $'$ is the transpose operator. The column vector consisting of f^d evaluated at every x_i can be written as Kc^d, where $K_{ij} = k(x_i, x_j)$. Using the reproducing property of the inner product, it can be seen that the RKHS norm of a functions of the form (6) is $\|f^d\|_k^2 = c^{d'} K c^d$. Substituting these into (5) yields a finite-dimensional least-squares problem in the coefficients of f:

$$\min_{c^d} \|Kc^d - y^d\|^2 + \lambda_k c^{d'} K c^d. \tag{7}$$

One c is found, f can be evaluated at arbitrary points by evaluating the form (6).

IV. Semi-supervised Nonlinear Regression with Dynamics

It is appealing to use fully supervised regression to learn a mapping from the samples of the input time series to those of the output time series. But for many of the applications we consider here, obtaining adequate performance with these techniques has required supplying so many input-output examples that even straightforward temporal interpolation between the examples yields adequate performance. This is not surprising, since *a priori* most nonlinear regression algorithms take into account very little of the structure of the problem at hand. In addition, they ignore unlabeled examples.

Taking advantage of even seemingly trivial additional information about the structure of the problem can not only improve regression with supervised points, but also renders unlabeled points informative, which in turn provides a significant boost in the quality of the regressor. For

example, explicitly enforcing the constraint that missing labels must be binary and the regressor smooth, as in a transductive SVM [26]–[29], results in performance gains over an SVM that only makes the latter assumption [30].

To take advantage of missing labels, we augment the cost functional for Tikhonov regularized function regression with a penalty function, $\mathcal{S}$, over missing labels. The penalty function $\mathcal{S}$ favors label sequences that exhibit plausible temporal dynamics. Under this setting, semi-supervised learning becomes a joint optimization over a function f and an output sequence $\mathbf{Y}$. Let $\mathbf{Z} = \{z_i\}$, $i \in \mathcal{L}$ be the set of labeled outputs, where the index set $\mathcal{L}$ of input-output examples may index a subset of samples of the time series, or may index additional out-of-sample examples.

The following optimization problem searches for an assignment to missing labels that is consistent with $\mathcal{S}$, and a smooth function f that fits the labeled data:

$$\min_{f,\mathbf{Y}} \sum_{i=1}^{T} V(f(x_i), y_i) + \lambda_l \sum_{i \in \mathcal{L}} V(f(x_i), z_i) + \lambda_s \mathcal{S}(\mathbf{Y}) + \lambda_k \sum_{d=1}^{N} \|f^d\|_k^2 \tag{8}$$

This cost functional adds two terms to the cost functional of Equation (1). As before, the second term ensures that f fits the given input-output examples, and the term weighted by λ_k favors smoothness of f. The first term ensures that f also fits the estimated missing labels, and the term weighted by λ_s favors sequences $\mathbf{Y}$ that adhere to the prior knowledge about $\mathbf{Y}$. The scalar λ_l allows points with known labels to have more influence than unlabeled points.

We consider applications where the output time series represents the motion of physical objects. In many cases, a linear-Gaussian random walk process is a reasonable model for the time evolution of the dynamical state of the object, so the negative log likelihood of this process provides the penalty function $\mathcal{S}$. For simplicity, we assume that each coordinate of the object's pose evolves independently of the other coordinates, and that the state sequence $\mathbf{S} = \{s_t\}_{t=1 \cdots T}$ for each coordinate evolves according to a chain of the form:

$$s_t = \mathbf{A} s_{t-1} + \omega_t \tag{9}$$

$$\mathbf{A} = \begin{bmatrix} 1 & \alpha_v & 0 \\ 0 & 1 & \alpha_a \\ 0 & 0 & 1 \end{bmatrix}, \tag{10}$$

where the Gaussian random variable ω_t has zero mean and diagonal covariance Λ_ω. These parameters, along with the scalars α_v and α_a specify the desired dynamics of the output time

series. When describing the motion of an object, each component of s_t has an intuitive physical analog: the first component corresponds to a position, the second to velocity, and the third to acceleration.

We would like to define $\mathcal{S}$ so that it favors output time series $\mathbf{Y}$ that adhere to the position component of the process defined by Equation (9). Equation (9) defines a zero-mean Gaussian distribution, $p(\mathbf{S})$, with

$$\log p(\mathbf{S}) = k - \frac{1}{2}\sum_{t=1}^{T} \|s_t - \mathbf{A}s_{t-1}\|^2_{\Lambda_\omega} + p(s_0) = k - \frac{1}{2}s'\mathbf{T}s, \tag{11}$$

where k is a normalization constant, $p(s_0)$ is a Gaussian prior over the first time step, and $\mathbf{T}$ can be written as a block tri-diagonal matrix. Since $p(\mathbf{S})$ is a zero-mean Gaussian pdf, marginalizing over the components of $\mathbf{S}$ not corresponding to position yields another zero-mean Gaussian distribution $p_1(s^1)$, where $s^1 = [s_1^1 \cdots s_T^1]'$ denotes the column vector consisting of the first component of each s_t. The Schur complement of $\mathbf{T}$ over s^1 gives the inverse covariance Ω_1 of $p_1(s^1)$ [31], and we get $\log p_1(s^1) = k_1 - \frac{1}{2}s^{1\prime}\Omega_1 s^1$. To favor time series $\mathbf{Y}$ that adhere to these dynamics, we penalize each component $y^d = [y_1^d \cdots y_T^d]$ of $\mathbf{Y}$, using $\log p_1(y^d)$. This amounts to penalizing each component of $\mathbf{Y}$ with $\left(y^d\right)' \Omega_1 y^d$.

We can substitute this quadratic form into the semi-supervised learning framework of Equation (8). Letting V be the quadratic penalty function, and defining $z^d = \{z_i^d\}_{\mathcal{L}}$ as a column vector consisting of the dth component of the labeled outputs, we get:

$$\min_{f,\mathbf{Y}} \sum_{d=1}^{N}\sum_{i=1}^{T}(f^d(x_i) - y_i^d)^2 + \lambda_l \sum_{i\in\mathcal{L}}(f^d(x_i) - z_i^d)^2 + \lambda_k\|f^d\|_k^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d. \tag{12}$$

Because our choice of $\mathcal{S}$ decouples over each dimension of $\mathbf{Y}$, the terms in the summation over d are decoupled, so each term over d can be optimized separately:

$$\min_{f^d,y^d} \sum_{i=1}^{T}(f^d(x_i) - y_i^d)^2 + \lambda_l \sum_{i\in\mathcal{L}}(f^d(x_i) - z_i^d)^2 + \lambda_k\|f^d\|_k^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d. \tag{13}$$

This is an optimization over f, nested within an optimization over y^d, so the Representer Theorem still applies. Letting the set $\mathcal{I} = \mathcal{L} \cup \{1 \cdots T\}$ denote the index set of labeled and unlabeled data, the optimum f^d has the RBF form:

$$f^d(x) = \sum_{i\in\mathcal{I}} c_i^d k(x, x_i), \tag{14}$$

Note that in contrast to fully-supervised nonlinear regression, where kernels are only centered on labeled points, the kernels here are centered on the labeled as well as the unlabeled points. This allows the function f to have larger support in the input data space without making it overly smooth.

Substituting the RBF form (14) for f in Equation (13) turns it into a quadratic problem in the coefficients of the RBF form and the missing labels:

$$\min_{c^d, y^d} \|\mathbf{K}_T c^d - y^d\|^2 + \lambda_l \|\mathbf{K}_{\mathcal{L}} c^d - z^d\|^2 + \lambda_k c^{d\prime} \mathbf{K} c^d + \lambda_s \left(y^d\right)' \Omega_1 y^d, \tag{15}$$

where $\mathbf{K}$ is the kernel matrix corresponding to labeled and unlabeled data, $\mathbf{K}_T$ is the matrix consisting of the rows of $\mathbf{K}$ that correspond to the unlabeled examples, and $\mathbf{K}_{\mathcal{L}}$ is the kernel matrix consisting of the rows of $\mathbf{K}$ that correspond to the labeled examples.

A simple way to perform this minimization is to rewrite the cost function (15) as a quadratic form plus a linear term:

$$\min_{c^d, y^d} \begin{bmatrix} c^d \\ y^d \end{bmatrix}' \begin{bmatrix} \mathbf{K}_T' \mathbf{K}_T + \lambda_k \mathbf{K} + \lambda_l \mathbf{K}_{\mathcal{L}}' \mathbf{K}_{\mathcal{L}} & -\mathbf{K}_T' \\ -\mathbf{K}_T & \mathbf{I} + \lambda_s \Omega_1 \end{bmatrix} \begin{bmatrix} c^d \\ y^d \end{bmatrix} + \begin{bmatrix} -2\lambda_l \mathbf{K}_{\mathcal{L}}' z^d \\ \mathbf{0} \end{bmatrix}' \begin{bmatrix} c^d \\ y^d \end{bmatrix} \tag{16}$$

The optima can be found by matrix inversion, but because the right hand side of this inversion problem has a zero block, we can reduce the complexity of the inversion by solving for c only. Denote by $\mathbf{P}$ the matrix that appears in the quadratic term, and partition it according to $\mathbf{P} = \left[\begin{smallmatrix} \mathbf{P}_c & \mathbf{P}_{cy} \\ \mathbf{P}_{cy}' & \mathbf{P}_y \end{smallmatrix}\right]$. Taking derivatives of the cost and setting to zero yields $\left[\begin{smallmatrix} \mathbf{P}_{cc} & \mathbf{P}_{cy} \\ \mathbf{P}_{cy}' & \mathbf{P}_{yy} \end{smallmatrix}\right] \left[\begin{smallmatrix} c^d \\ y^d \end{smallmatrix}\right] = \left[\begin{smallmatrix} \lambda_l \mathbf{K}_{\mathcal{L}}' z^d \\ \mathbf{0} \end{smallmatrix}\right]$. Using the matrix inversion lemma yields a solution for the optimal c^d:

$$c^{d*} = \lambda_l \left(\mathbf{P}_{cc} - \mathbf{P}_{cy} \mathbf{P}_{yy}^{-1} \mathbf{P}_{cy}'\right)^{-1} \mathbf{K}_{\mathcal{L}}' z^d. \tag{17}$$

Once these coefficients are recovered, labels can be estimated by evaluating f at various x's by evaluating the RBF form (14).

This algorithm is not very sensitive to the settings of the its parameters, and usually works with default parameter settings. These parameters are λ_k, λ_l, λ_s, α_v, α_a, Λ_ω, and the parameters required to define the kernel $k(\cdot, \cdot)$. Since λ_s scales Λ_ω, it is subsumed by Λ_ω, which is diagonal, leaving us with a total of seven parameters, plus the parameters of k (which is usually just a scalar). In [32], we present a variant based on a nearest-neighbors representation of f that requires fewer parameters. The next section presents a variation based on the RBF form that eliminates one parameter. Sections VI and VIII provide some intuition to guide any manual parameter tuning that may be necessary.

V. Algorithm Variation: Noise-free Examples

The learning functional in the previous section does not require f to fit the given input-output examples exactly, allowing some noise to be present in the given output labels. But if the given labels are accurate, we may require that f fit them exactly. This has the advantage of eliminating the free parameter λ_l, which weights the influence of the labeled points.

An exact fit to the given examples can be enforced by making λ_l very large, but this makes the cost functional poorly conditioned. A better solution is to turn the second term into a set of constraints, resulting in the following alternative to (8):

$$\min_{f,\mathbf{Y}} \sum_{i=1}^{T} V(f(x_i), y_i) + \lambda_s \mathcal{S}(\mathbf{Y}) + \lambda_k \sum_{d=1}^{N} \|f^d\|_k^2 \tag{18}$$

$$\text{s.t. } f(x_i) = z_i, \qquad \forall i \in \mathcal{L} \tag{19}$$

When V is quadratic, this reduces to minimizing a quadratic form subject to linear constraints

$$\min_{c^d, y^d} \begin{bmatrix} c^d \\ y^d \end{bmatrix}' \begin{bmatrix} \mathbf{K}_T' \mathbf{K}_T + \lambda_k \mathbf{K} & -\mathbf{K}_T' \\ -\mathbf{K}_T & \mathbf{I} + \lambda_s \Omega_1 \end{bmatrix} \begin{bmatrix} c^d \\ y^d \end{bmatrix} \tag{20}$$

$$\text{s.t. } \mathbf{K}_{\mathcal{L}} c^d = z^d. \tag{21}$$

To solve for c^d, partition again the matrix that appears in the quadratic term as $\mathbf{P} = \begin{bmatrix} \mathbf{P}_{cc} & \mathbf{P}_{cy} \\ \mathbf{P}_{cy}' & \mathbf{P}_{yy} \end{bmatrix}$. Solving for the optimal y^d and plugging back in yields a quadratic form in terms of the Schur complement of $\mathbf{P}$, which we denote by $\mathbf{H} = \mathbf{P}_{cc} - \mathbf{P}_{cy}\mathbf{P}_{yy}^{-1}\mathbf{P}_{cy}'$:

$$\min_{c^d} \quad \left(c^d\right)' \mathbf{H} c^d \tag{22}$$

$$\text{s.t. } \mathbf{K}_{\mathcal{L}} c^d = z^d. \tag{23}$$

The optimal coefficients, derived in the appendix, are given by:

$$c^{d*} = \mathbf{H}^{-1}\mathbf{K}_{\mathcal{L}}'\left(\mathbf{K}_{\mathcal{L}}'\mathbf{H}^{-1}\mathbf{K}_{\mathcal{L}}\right)^{-1} z^d. \tag{24}$$

VI. Intuitive Interpretation

An informal understanding of these algorithms will be helpful in understanding when they will work and how to tune their parameters. The optimization (8) simultaneously interpolates the missing labels using $\mathcal{S}$ and fits a function f to missing and given labels with Tikhonov regularization. This can be better understood by regrouping the terms of (8):

- Function fitting: The data penalty terms fit f to given and estimated labels. To see this, rewrite (8) as:

$$\min_{\mathbf{Y}} \lambda_s \mathcal{S}(\mathbf{Y}) + \left[\min_{f} \sum_{i=1}^{T} V(f(x_i), y_i) + \lambda_l \sum_{i \in \mathcal{L}} V(f(x_i), z_i) + \lambda_k \sum_{d=1}^{N} \|f^d\|_k^2 \right]. \quad (25)$$

  The inner optimization is Tikhonov regularization and assigns a different weight to known labels and imputed labels.

- Interpolation: The optimization over $\mathbf{Y}$ implements a smoother over label trajectories that uses $f(x_i)$ as observations and $\mathcal{S}$ as a prior. To see this, rewrite (8) as:

$$\min_{f} \lambda_k \sum_{d=1}^{N} \|f^d\|_k^2 + \lambda_l \sum_{i \in \mathcal{L}} V(f(x_i), z_i) + \left[\min_{\mathbf{Y}} \sum_{i=1}^{T} V(f(x_i), y_i) + \lambda_s \mathcal{S}(\mathbf{Y}) \right]. \quad (26)$$

  This nested smoothing operation corrects the output of f at unlabeled points. This in turn guides the function fitting step.

The coupling between these two operations allows the algorithm to learn the correct mapping in regions where labeled data is scarce. In those regions, the labels can be inferred by interpolating them from temporally adjacent known outputs. This effect is starkly illustrated with the *Sensetable* data set in the next section.

VII. Learning to Track from Examples with Semi-supervised Learning

This section exhibits the effectiveness of regularizing the output nonlinear regression when solving tracking problems. The relationship to manifold learning algorithms is clarified with some synthetic examples. We then report our experiments in learning a tracker for RFID tags, and our sequence annotation tool for labeling video sequences. Throughout this section, we show that a fully-supervised nonlinear regression algorithm would require significantly more examples to learn these operations.

The applications demonstrated here rely on the semi-supervised learning algorithm of Section V, which requires the output time series to fit the input-output examples exactly. We use a Gaussian kernel $k(x_1, x_2) = \exp\left(-\frac{1}{2}\|x_1 - x_2\|^2/\sigma^2\right)$, whose bandwidth parameter σ is a free parameter of the algorithm. Section VIII provides some guidance in tuning the algorithm's free parameters.

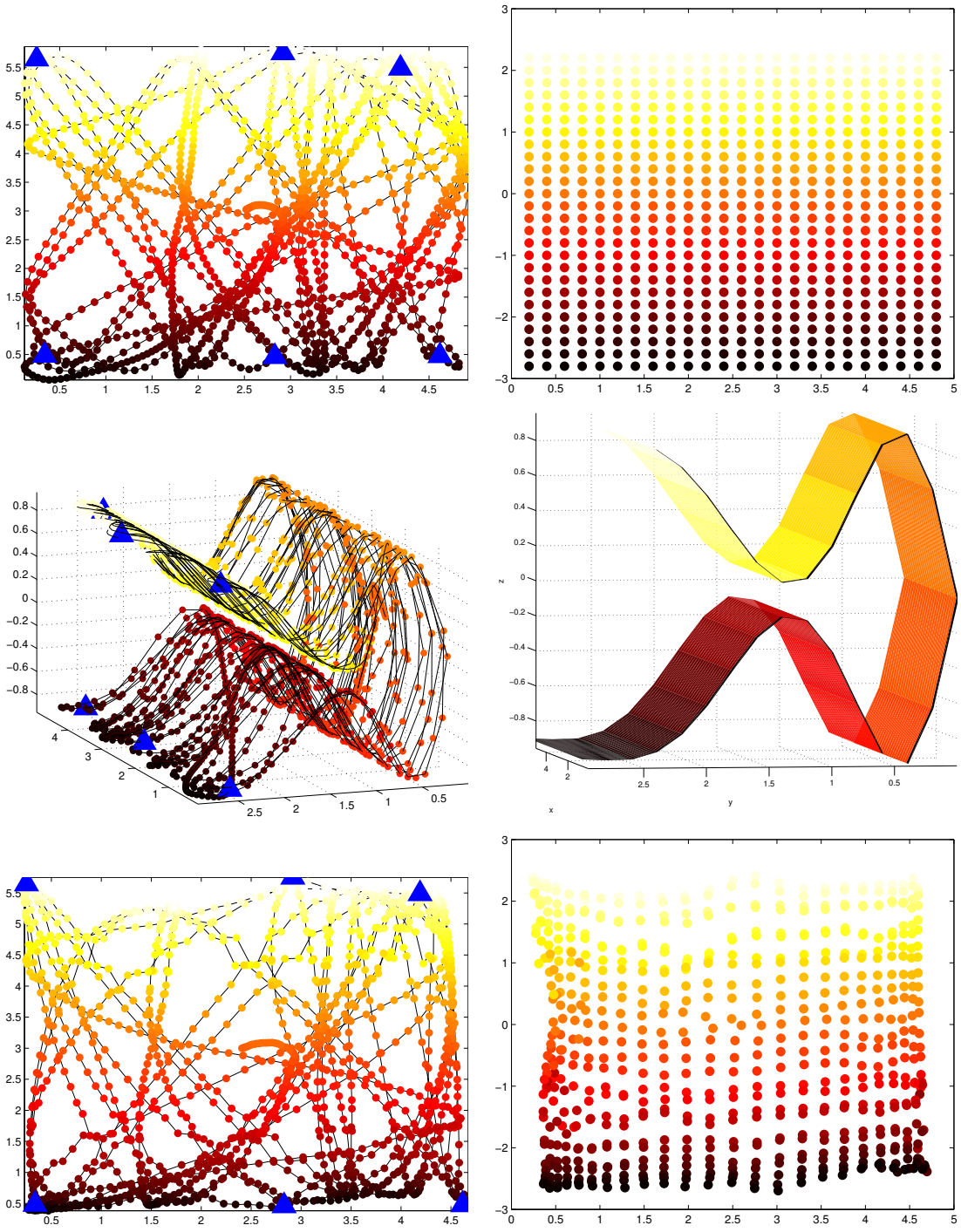

Fig. 1. (left-top) The true 2D parameter trajectory. The six labeled points are marked with big blue triangles. The trajectory has 1500 samples. In all these plots, the color of each trajectory point is based on its y-value, with higher intensities corresponding to greater y-values. (left-middle) Embedding of the path via the lifting $F(x, y) = (x, |y|, \sin(\pi y)(y^2 + 1)^{-2} + 0.3y)$. (left-bottom) Recovered low-dimensional representation using our algorithm. The original data in (top-left) is recovered. (right-top) Even sampling of the rectangle $[0, 5] \times [-3, 3]$. (right-middle) Lifting of this rectangle via F. (right-bottom) Projection of (right-middle) via the learned function f. The recovered locations are close to their 2D locations, showing that the inverse of F has been learned correctly.

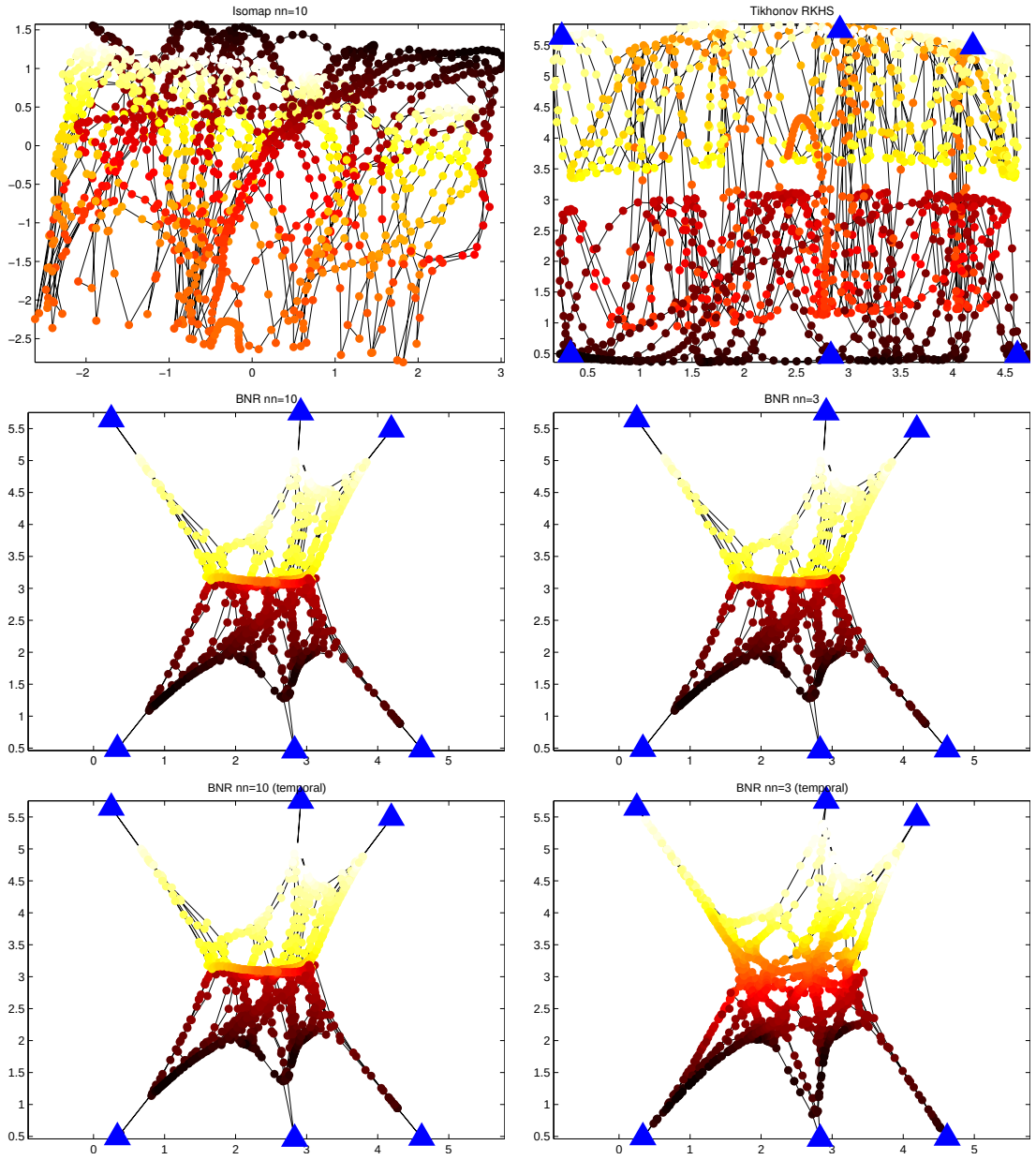

Fig. 2. (top-left) Isomap's recovered 2D coordinates for the dataset of Figure 1(left-middle). Errors in estimating the neighborhood relations at the neck of the manifold cause the projection to fold over itself in the center. The neighborhood size was 10, but smaller neighborhoods produce similar results. (top-right) Fully supervised algorithms, which do not take advantage of unlabeled points, cannot correctly recover the coordinates of unlabeled points because only points at the edges of the shape are labeled. (middle-left) Projection with BNR, a semi-supervised regression algorithm, with neighborhood size of 10. Although the structure is recovered more accurately, all the points behind the neck are folded into one thin strip. (middle-right) BNR with neighborhood size of 3 prevents only some of the folding. Points are still shrunk to the center, so the low-dimensional values are not recovered accurately. (bottom row) BNR as before, with temporally adjacent points included in neighborhoods. There is no significant improvement over building neighborhoods using nearest neighbors only.

A. Synthetic Manifold Learning Problems

We first demonstrate the effectiveness of the semi-supervised learning algorithm on a synthetic dimensionality reduction problem where the task is to recover low-dimensional coordinates on a smooth 2D manifold embedded in $\mathcal{R}^3$. The data set considered here proves challenging for existing manifold learning techniques, which estimate the neighborhood structure of the manifold based on the proximity of high-dimensional points. Taking advantage of temporal dynamics, and a few points labeled with their low-dimensional coordinates, makes the problem tractable using our algorithm. To assess the importance of labeled points, we also compare against a semi-supervised learning algorithm that does not take temporal dynamics into account.

The high-dimensional dataset is constructed by lifting a random walk on a 2D euclidean patch to $\mathcal{R}^3$ via a smooth and invertible mapping. See Figure 1(left-middle,left-top). The task is to recover the projection function $f : \mathcal{R}^3 \rightarrow \mathcal{R}^2$ to invert this lifting.

In this data set, the nearest neighbors in $\mathcal{R}^3$ of points near the region of near-self-intersection (the "neck") will straddle the neck, and the recovered neighborhood structure will not reflect the proximity of points on the manifold. This causes existing manifold learning algorithms such as LLE [2], Isomap [1], and Laplacian Eigenmaps [3] to assign similar coordinates to points that are in fact very far from each other on the manifold. Isomap creates folds in the projection. See Figure 2(top-left). Neither LLE nor Laplacian Eigenmaps produced sensible results, projecting the data to a straight line, even with denser sampling of the manifold (up to 7000 samples) and with a variety of neighborhood sizes (from 3 neighbors to 30 neighbors).

These manifold learning algorithms ignore labeled points, but the presence of labeled points does not make the recovery of low-dimensional coordinates trivial. To show this, we also compare against Belkin and Nyogi's graph Laplacian-based semi-supervised regression algorithm [12], referred to here as BNR. Six points on the boundary of the manifold were labeled with their ground truth low-dimensional coordinates. Figures 2(middle-left,middle-right) show the results of BNR on this data set when it operates on large neighborhoods. There is a fold in the resulting low-dimensional coordinates because BNR assigns the same value to all points behind the neck. Also, the recovered coordinates are shrunk towards the center, because the Laplacian regularizer favors coordinates with smaller magnitudes. For smaller settings of the neighborhood size, the

folding disappears, but the shrinking remains. [1]

It is also not sufficient to merely take temporal adjacency into account when building the neighborhood structure of the high-dimensional points. Figure 2(bottom row) shows the result of BNR when the neighborhood of each point includes temporally adjacent points. Including these neighbors does not improve the result.

Finally, Figure 2(top-right) shows the result of Tikhonov regularization on an RKHS with quadratic loss (the solution of (5) applied to the high-dimensional points). This algorithm uses only labeled points, and ignores unlabeled data. Because all the labeled points have the same y coordinates, Tikhonov regularization cannot generalize the mapping to the rest of the 3D shape.

Taking into account the temporal coherence between data points and the dynamics of the low-dimensional coordinates alleviates these problems. Folding problems are alleviated because our algorithm takes advantage of the time ordering of data points, and the explicit dynamical model alleviates the shrinking towards zero by implicitly modeling velocity in the low-dimensional trajectory. Figure 1(left-bottom) shows the low-dimensional coordinates recovered by our algorithm. These values are close to the true low-dimensional coordinates.

We can also assess the quality of the learned function f on as-yet unseen points. Figure 1(right-top and right-middle) shows a 2D grid spanning $[0, 5] \times [-3, 3]$ lifted by the same mapping used to generate the training data. Each of these points in $\mathcal{R}^3$ is passed through the recovered mapping f to obtain the 2D representation shown in Figure 1(right-bottom). These projections fall close to the true 2D location of these samples, implying that f has correctly generalized an inverse for the true lifting.

This synthetic experiment illustrates three features that recur in subsequent experiments:

- While the kernel matrix $\mathbf{K}$ takes into account the similarity of the high-dimensional data points, explicitly taking into account the dynamics of the low-dimensional process obviates the need to build the brittle neighborhood graph that is common in manifold learning and semi-supervised learning algorithms.
- The assumed dynamics model does not need to be very accurate. The true low-dimensional

[1] In comparing the algorithm with Isomap, LLE and Laplacian Eigenmaps, we relied on source code available from the respective authors' web sites. To compute eigenvalues and eigenvectors, we tried both MATLAB's EIGS routine and JDQR [33], a drop-in replacement for EIGS. We used our own implementation of BNR, but relied on the code supplied by the authors to compute the Laplacian.

random walk used to generate the data set bounced off the boundaries of the rectangle $[0, 5] \times [-3, 3]$, an effect not modeled by a linear-Gaussian Markov chain. Nevertheless, the assumed dynamics of Equation (9) are sufficient for recovering the true location of unlabeled points.

- The labeled examples do not need to capture all the modes of variation of the data. Despite the fact that the examples only showed how to transform points whose y coordinate is 2.5, our semi-supervised learning algorithm learned the low-dimensional coordinate of points with any y-coordinate.

B. Learning to Track: Tracking with the Sensetable

The *Sensetable* is a hardware platform for tracking the position of radio frequency identification (RFID) tags [35]. It consists of 10 antennae woven into a flat surface that is 30 cm on a side. As an RFID tag moves along the flat surface, analog-to-digital conversion circuitry reports the strength of the RF signal from the RFID tag as measured by each antenna, producing a time series of 10 numbers. See Figure 3(left). The *Sensetable* has been integrated into various hardware platforms, such as a system for visualizing supply chains, and the Audiopad [34], an interactive disc jockey system.

We wish to learn to map the 10 signal strength measurements from the antennae to the 2D position of the RFID tag. Previously, a mapping was recovered by hand through an arduous reverse-engineering process that involved building a physical model of the inner-workings of the *Sensetable*, and resorting to trial and error to refine the resulting mappings [35]. Rather than reverse-engineering this device by hand, we show that it is possible to recover this mapping with only 4 labeled examples and some unlabeled data points, even though the relationship between the tag's position an the observed measurements is highly oscillatory. Once it is learned, we can use the mapping to track RFID tags. The procedure we follow is quite general, and can be applied to a variety of other hardware.

To collect the four labeled examples, the tag was placed on each of the four corners of the *Sensetable*, the *Sensetable*'s output was recorded. We collected unlabeled data by sweeping the tag on the *Sensetable*'s surface for about 400 seconds, and down-sampled the result by a factor of 3 to obtain about 3600 unlabeled data points. Figure 3(middle) shows the ground truth trajectory of the RFID tag, as recovered by the manually reverse-engineered *Sensetable* mappings. The

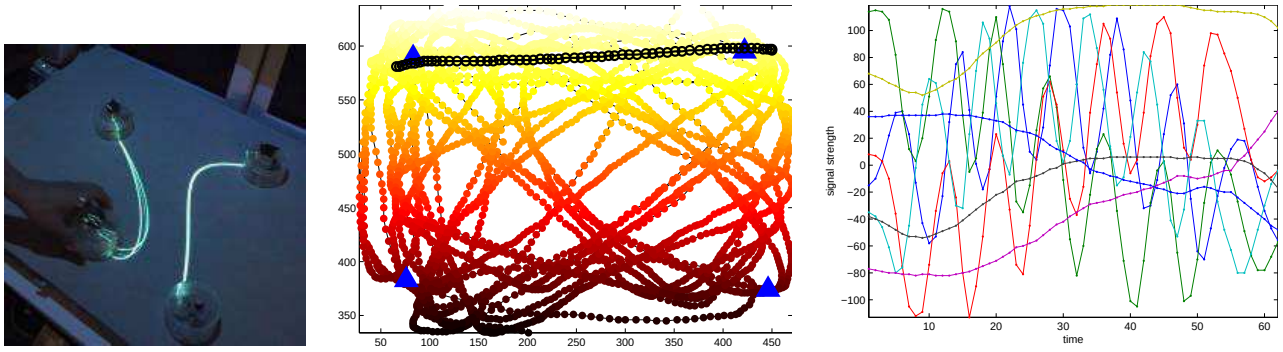

Fig. 3. (left) A top view of the *Sensetable*, an interactive environment for tracking RFID tags. Users manipulate RFID tagged pucks, and a projector overlays visual feedback on the surface of the table. Coils under the table measure the strength of the signals induced by RFID tags. Our algorithm recovers a mapping from these signal strengths to the position of the tags. (middle) The ground truth trajectory of the tag. The tag was moved around smoothly on the surface of the *Sensetable* for about 400 seconds, producing about 3600 signal strength measurement samples after downsampling. Triangles indicate the four locations where the true location of the tag was provided to the algorithm. The color of each point is based on its y-value, with higher intensities corresponding to higher y-values. (right) Samples from the antennae of the *Sensetable* over a six second period, taken over the trajectory marked by large circles in the left panel.

four triangles in the corners of the figure depict the location of the labeled examples. The rest of the 2D trajectory was not made available to the algorithm. Contrary to what one might hope, the output from each antenna of the *Sensetable* does not have a straightforward one-to-one relationship with a component of the 2D position. For example, when the tag is moved in a straight line from left to right, it generates oscillatory traces similar to those shown in Figure 3(right).

The four labeled points, along with the few minutes of recorded data were passed to the semi-supervised learning algorithm to recover the mapping. The algorithm took 90 seconds to process this data set on a 3.2 Ghz Xeon machine. The trajectory is recovered accurately despite the complicated relationship between the 10 outputs and the tag position (see Figure 4). The RMS distance to the ground truth trajectory is about 1.3 cm, though the ground truth itself is based on the reverse engineered tracker and may be inaccurate. Figure 4(right) shows the regions that are most prone to errors. The errors are greatest outside the bounding box of the labeled points, but points near the center of the board are recovered very accurately, despite the lack of labeled points there. This phenomenon is discussed in Section VI.

The recovered mapping from measurements to positions can be used to track tags. Individual

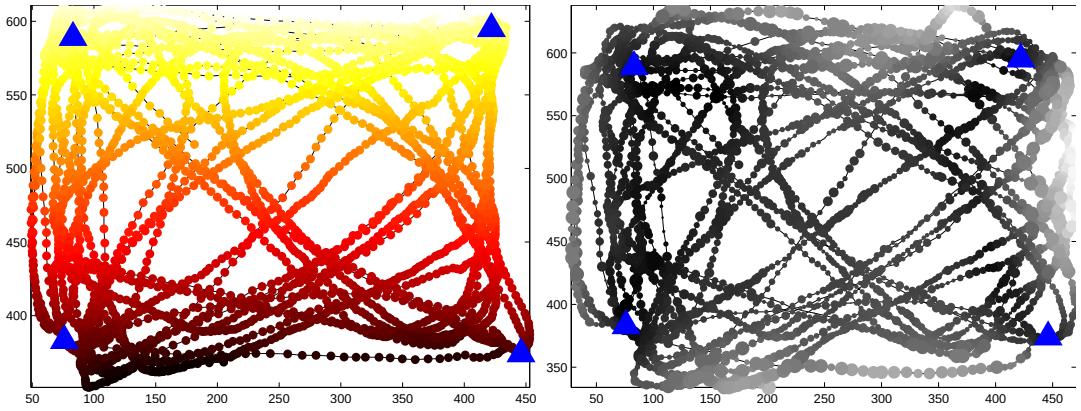

Fig. 4. (left) The recovered tag positions match the original trajectory depicted in Figure 3. (right) Errors in recovering the ground truth trajectory. Circles depict ground truth locations, with the intensity and size of each circle proportional to the Euclidean distance between a point's true position and its recovered position. The largest errors are outside the bounding box of the labeled points. Points in the center are recovered accurately, despite the lack of labeled points there.

samples of 10 measurements can be passed to the recovered mapping f to recover the corresponding tag position, but because the *Sensetable*'s output is noisy, the results must be filtered. Figure 5 shows the output of a few test paths after smoothing using the assumed dynamics. The recovered trajectories match the patterns traced by the tag.

Unlabeled data, and explicitly taking advantage of dynamics is critical in this dataset because the relationship between signal strengths and tag positions is complex, and few input-output examples are available. Thus, fully-supervised learning with Tikhonov regularization on an RKHS fails to recover the mapping. See Figure 6(left). Figure 6(middle) shows the trajectory recovered by BNR with its most favorable parameter setting for this data set. Figure 6(right) shows the trajectory recovered by BNR when temporally adjacent neighbors are counted as part of the adjacency graph when computing the Laplacian. As with the synthetic data set, there is severe shrinkage toward the mean of the labeled points, and some folding at the bottom, and taking temporal adjacency into account does not significantly improve the results.

C. Learning to Track: Visual Tracking

In this section, we demonstrate an interactive application where our algorithm helps a user quickly annotate every frame of a video sequence with a low-dimensional representation of a the scene given a few manually-labeled key frames. These labels are specified with an interactive graphical tool as a collection of vectorized drawing primitives, such as splines and polylines.

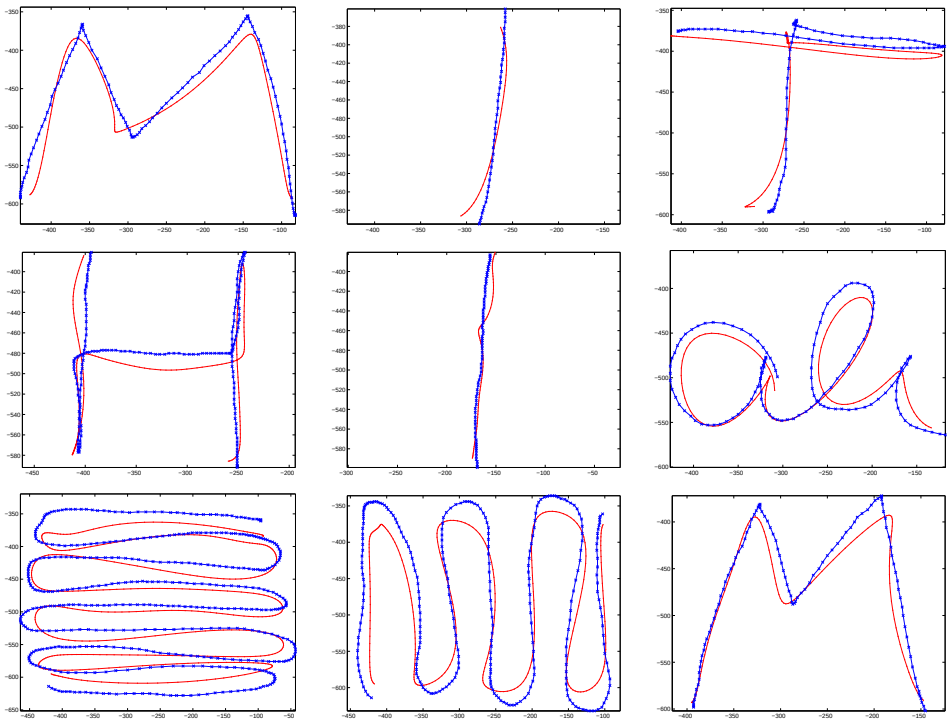

Fig. 5. Once f is learned, it can be used it to track tags. Each panel shows a ground truth trajectory (blue crosses) and the estimated trajectory (red dots). The recovered trajectories match the intended shapes.

The output representation consists of the control points of these drawing primitives. Given the video sequence and the labeled examples, our algorithm recovers the control points for the unlabeled frames of the video sequence. If the user is not satisfied with the rendering of these control points, he can modify the labeling and rerun the algorithm at interactive rates. The tool is demonstrated on a lip tracking video where the user specifies the shape of the lips of a subject, and two articulated body tracking experiments [36], [37], where the user specifies positions of the subject's limbs. The annotation tool is available online [38].

There is a rich body of work in learning visual trackers from examples using fully-supervised regression algorithms. For example, relying on the nearest neighbors representation, Efros et al. [39] used thousands of labeled images of soccer players to recover the articulated pose of players in new images, and Shaknarovich et al. [40] used a database of synthetically rendered hands to recover the pose of hands. Relying on the RBF representation, El Gammal [41] recovered the deformation of image patches, and Agarwal and Triggs [42] learned a mapping from features of an image to the pose of human bodies. In our case, the training sequence are sequential frames

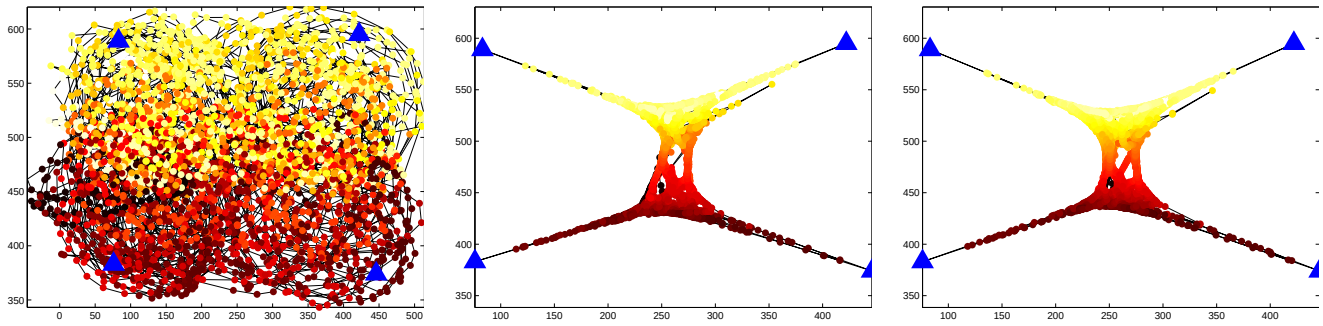

Fig. 6. (left) Tikhonov regularization with labeled examples only. The trajectory of the RFID tag is not recovered. Both BNR with 3-nearest neighbors (middle) and BNR with 3-nearest neighbors including temporal neighbors (right) exhibit folding at the bottom of the plot, where black points appear under the lighter red points. There is also severe shrinking towards the mean.

of a video, and taking advantage of the temporal coherence between these frames allows us to take advantage of unlabeled examples, and greatly reduces the need for labeled outputs.

Because it is interactive, our system is reminiscent of rotoscoping tools [43], [44], which allow the user to interactively adjust the output of contour trackers by annotating key frames. Since our algorithm does not rely explicitly on edges or the spatial coherence of pixels in images, it can learn arbitrary smooth mappings from images to vectorized representations that do not correspond to contours. For this reason, it is robust to occlusions.

While other example-based tracking systems preprocess the images by selecting a relevant patch [39], [41], or extracting silhouettes [40], [45], we represent images as by concatenating their pixel values into a column vector. Thus, we represent a 640×480 gray scale image as a vector in $\mathcal{R}^{307200}$. The visual tracking results in this section use this simple representation, without applying any preprocessing to the images, and represent the mappings using Gaussian radial basis functions.

Because it performs no preprocessing on the input images, several caveats apply for this algorithm to work well in visual settings. First, the motion being tracked must be the dominant source of differences between the images: there may be no large distractors that are not governed by the underlying representation. Such distractors include other moving objects in the scene, significant lighting variations over time, and motions of the camera. But occlusions due to stationary objects, or even self occlusions are allowed because the Gaussian RBF representation relies only on pixel differences between pairs of images. Thus, the algorithm is also invariant to orthonormal operators applied to images, such as permutations of the pixels, as these operations

do not affect the response of the Gaussian kernel. If distractors do exist in the scene, they may be removed by preprocessing the frames, and adopting a more invariant representation. This could further reduce the number of required examples and make the learned mapping applicable to inputs that are significantly different from the training images. The tracker can also be made to ignore these distractors by providing additional labeled data. Second, the algorithm assumes that the mapping from images to labels is smooth: a small change in the input image should result in a small change in its label. This smoothness requirement is satisfied in practice, even in the presence of occlusions. Third, the algorithm assumes that the output time series evolves smoothly over time. If this is not the case, the algorithm cannot take advantage of unlabeled data.

To demonstrate the algorithm's ability to track deformable shapes, the interactive tool was used to annotate the contour of a subject's lip in a 2000 frame video sequence. Figure 7(top) shows a few frames of this sequence. The contour of lips are represented with cubic splines controlled by four points. Two of the control points were placed on the corners of the mouth, and the other two were placed above and below the lips. Only seven labeled frames were necessary to obtain good lip tracking performance for the rest of the sequence. See Figure 7(bottom). The tracker is robust to all the artifacts manifest in this video sequence, including blinking, facial expressions, small movements of the head, and the appearance and disappearance of teeth. Applying fully-supervised Tikhonov regularization on these seven examples yields identical results. But this is not the case with the following video sequences, where fully-supervised nonlinear regression does not perform well with the given examples.

Figure 8(top) shows the labeled images of a 2300 frame sequence of a subject moving his arms. These twelve frames were manually labeled with line segments denoting the upper and lower arms. The middle portion of Figure 8 shows that the limb positions were recovered accurately for the unlabeled portions of the sequence. The bottom portion of the figure shows that limb positions are recovered accurately for novel frames that were not in the training sequence. Because the raw pixel representation is used, the mapping between observations and pose is nearly one-to-one, so poses can be recovered from individual frames. For the same reason, the mapping is robust to self-occlusions when the subject's arms cross themselves. Fully-supervised nonlinear regression produced the limb locations shown in black in Figure 8. In contrast to the semi-supervised case, the resulting recovered positions are often wrong.

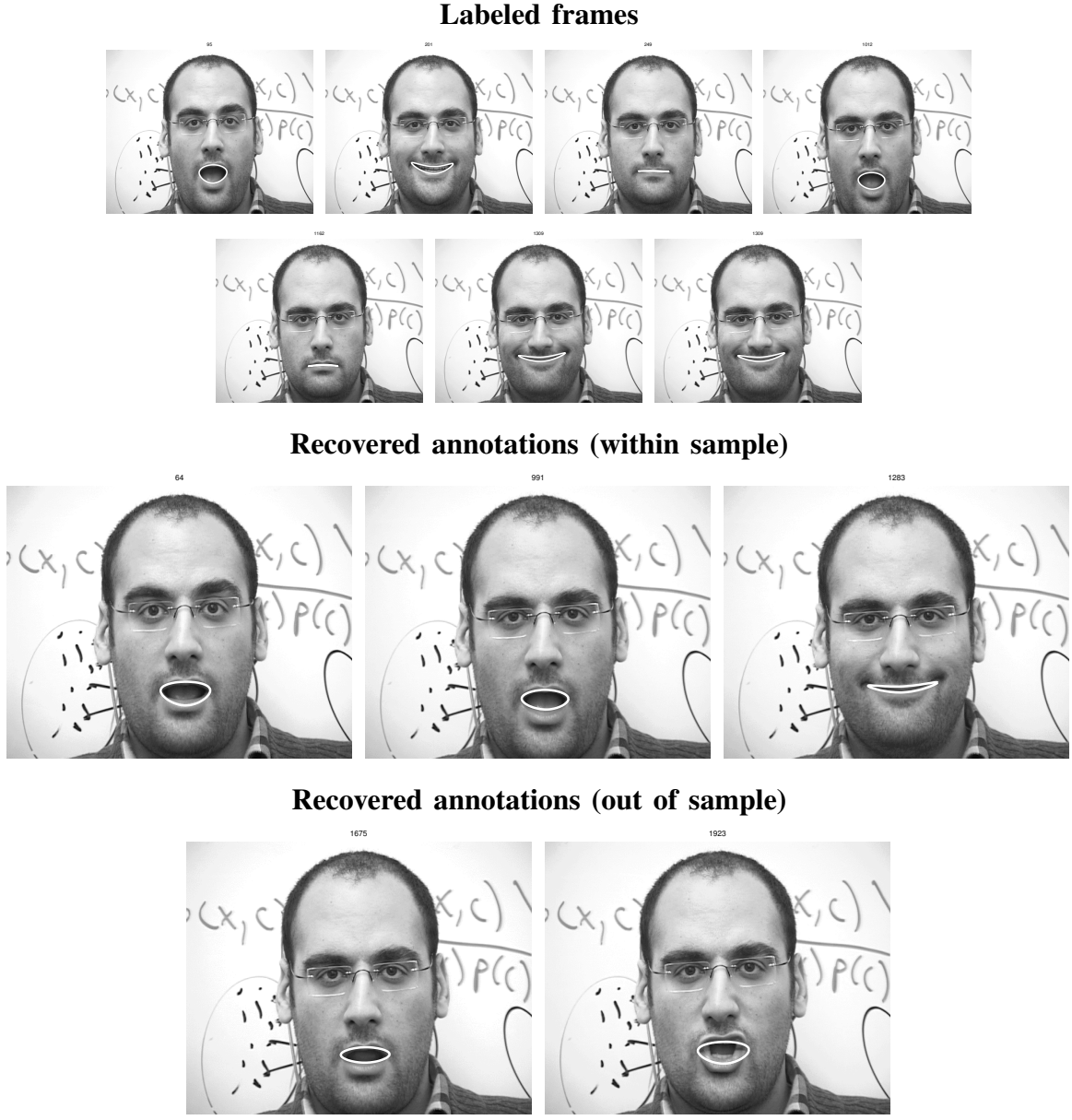

Fig. 7. (top) The contour of the lips was annotated in 7 frames of a 2000 frame video. The contour is represented using cubic splines, controlled by four control points. The desired output time series is the position of the control points over time. These labeled points and first 1500 frames were used to train our algorithm. (bottom) The recovered mouth contours for various frames. The first three images show the labeling recovered for to unlabeled frames in the training set, and the next two show the labeling for frames that did not appear in the training set. The tracker is robust to natural changes in lighting (ie, the flicker of fluorescent lights), blinking, facial expressions, small movements of the head, and the appearance and disappearance of teeth.

Fig. 8. (top) Twelve frames were annotated with the joint positions of the subject in a 1500 frame video sequence. (middle) The recovered positions of the hands and elbows for the unlabeled frames are plotted in white. The output of fully-supervised nonlinear regression using only the 12 labeled frames and no unlabeled frames is plotted in black. Using unlabeled data improves tracking significantly. (bottom) Recovered joint positions for frames that were not in the training set. The resulting mapping generalizes to as-yet unseen images.

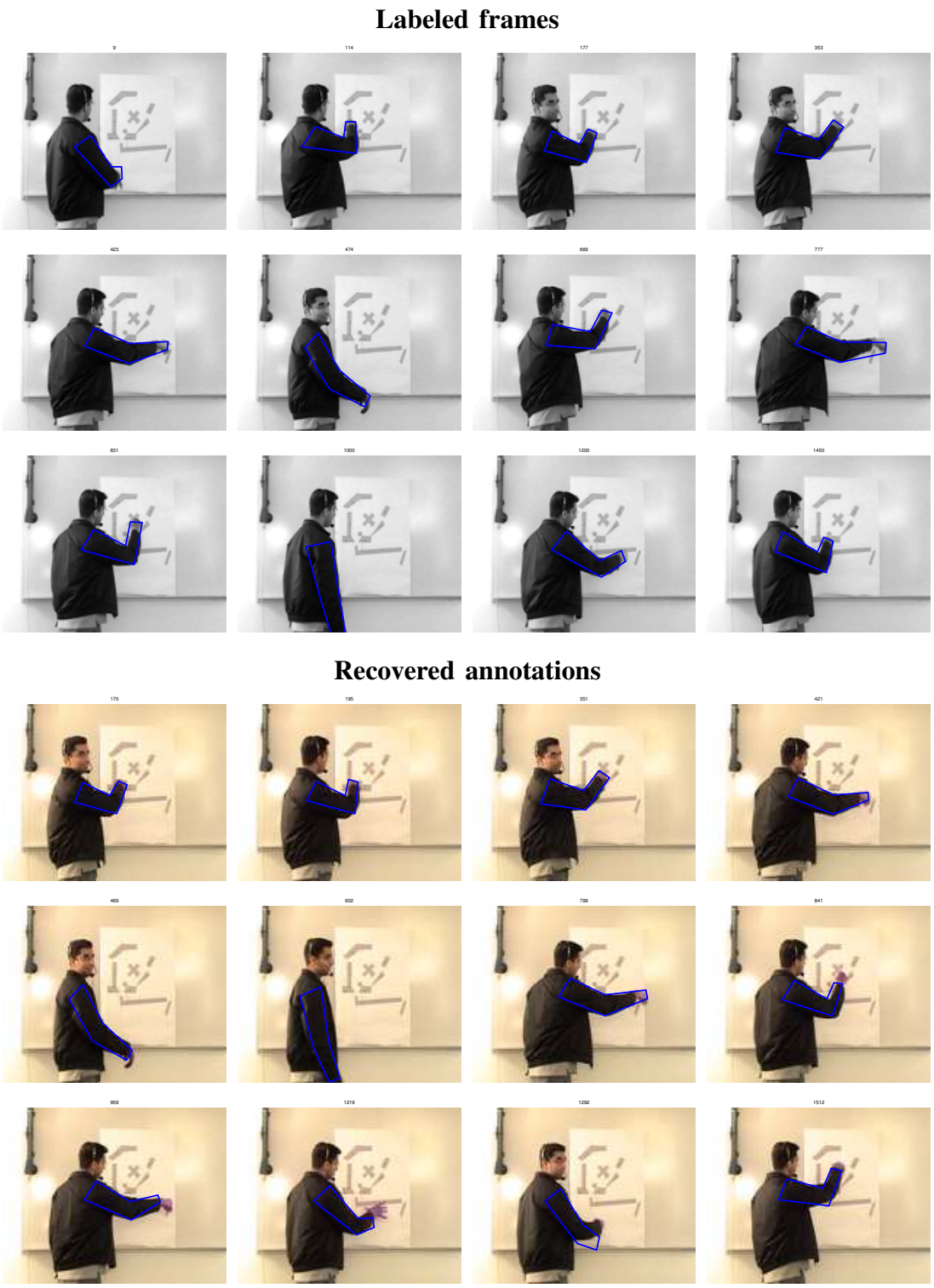

Fig. 9. (top) 12 of the 13 annotated frames for the arm tracking experiment. The labeling is a closed polygon with six corners. The corners are placed at the shoulder, elbow and hand. Two corners are associated with each of these body parts. To handle the subject turning his head, we annotated a few frames with the subject's head turned towards the camera and away from the camera. (bottom) A few recovered annotations. Tracking is robust to head rotations and small motions of the torso because we explicitly annotated the arm position in frames exhibiting these distractors.

Figure 9(top) shows some of the labeled images in another 2000 frame sequence. Thirteen labeled frames were sufficient for recovering the outline of the subject's right arm throughout the sequence. See the rest of Figure 9. The resulting tracker misses the fine motions of the hand (for example, when the subject moves his hand in a small circle, the annotation remains stationary), but captures the gross motion of the arm. Tracking is robust to the motion of the subject's torso and the subject turning his head, because some of the 13 examples explicitly annotate these cases.

To numerically assess the quality of these results, we corrected its output at every fifth frame by hand and used the magnitude of this correction as a measure of quality. These corrections were not supplied to the algorithm and serve only to numerically assess the quality of the output from the 13 labeled images. Table I shows the magnitude of these corrections averaged over the whole sequence. Our algorithm outperforms temporal linear interpolation between the 13 labeled frames, and fully-supervised regression using Tikhonov regularization with the 13 labeled frames. The output of each algorithm was corrected separately to avoid unintentionally favoring any one algorithm.

VIII. Choosing Examples and Tuning Parameters

This section provides guidelines for selecting which data points to label, and how to set the parameters. We find that the algorithm returns similar results for a wide setting of parameters, but that the choice of labeled examples has a strong effect on the accuracy of the result.

Typically, it is only necessary to label input samples whose labels lie on the corners of the output space. For example, in the *Sensetable* experiment, we labeled the corners of the table, and in the arm tracking example of Figure 8, we labeled extreme arm positions. As is shown in Figure 4(right), labels in the hull of the labeled examples can usually be recovered very accurately, so it is not necessary to label them. This is because there is often a path between two labeled points that either follows the given dynamics and passes through the interior, so the label of the interior points can be recovered with temporal interpolation, or because these points lie in regions where f returns accurate results due to proximity to other labels that are well-estimated. An unsupervised version of our learning algorithm [46] can be used to identify boundary points in a first pass, if these points cannot be easily identified by inspection. However, if good paths through the interior are rare, interior points may need to be labeled as well. For

TABLE I

COMPARISON BETWEEN OUTPUT AND HAND-LABELED GROUND TRUTH FOR THE DATA SET OF FIGURE 9. ALL DISTANCES ARE IN PIXELS. THE FIRST COLUMN GIVES THE AVERAGE DISTANCE BETWEEN THE POSITION OF EACH CORNER AS RECOVERED BY OUR ALGORITHM AND ITS HAND-CORRECT LOCATION. THE ERROR FOR THE TWO CORNERS ASSOCIATED WITH EACH BODY PART (SHOULDER, ELBOW, AND HAND) IS REPORTED SEPARATELY. TO GAUGE THE MAGNITUDE OF THE ERRORS, THE LAST COLUMN GIVES THE MAXIMUM DEVIATION BETWEEN EACH CORNER'S POSITION AND ITS AVERAGE POSITION IN THE SEQUENCE. THE SECOND AND THIRD COLUMNS REPORT THE ERRORS FOR TEMPORALLY INTERPOLATING BETWEEN THE 13 LABELS, AND APPLYING FULLY-SUPERVISED REGRESSION ON THE 13 LABELS. TIKHONOV REGULARIZATION WAS RUN WITH THE BEST SETTINGS OF ITS PARAMETERS (KERNEL VARIANCE AND THE WEIGHTING OF THE STABILIZER). THESE PARAMETERS WERE FOUND BY GRIDDING THE PARAMETER SPACE. OUR ALGORITHM OUTPERFORMS BOTH OF THESE APPROACHES.

	Semi-supervised	Tikhonov regression	Temporal interpolation	Travel
Shoulders	0	0	0	10,10
Elbows	.4, .6	4, 5	8,8	30,34
Hands	3.5, 4.8	10, 11	14,14	56, 59

example, in Figure 9, frames where the subject pointed to the 'X' in the middle of the board had to be labeled, even though the appropriate parameters of the shape of the arm for these frames lie in the convex hull of the other labeled frames. In this video sequence, to reach the 'X' from a boundary point, the subject's arm followed a circuitous path through previously unexplored interior regions of the board. Because this path was neither very likely according to the dynamics model nor near other points whose labels were recovered, we had to explicitly label some points along it.

The algorithm is insensitive to settings of the other parameters up to several orders of magnitude, and typically, only the parameters of the kernel need to be tuned. When using a Gaussian kernel, if the bandwidth parameter is too small, $\mathbf{K}$ becomes diagonal and all points are considered to be dissimilar. If the bandwidth parameter is too large, $\mathbf{K}$ has 1 in each entry and all points are considered to be identical. We initially set the kernel bandwidth parameter so that the minimum entry in $\mathbf{K}$ is approximately 0.1. Other parameters, including the scalar weights and the

parameters of dynamics are initially set to 1. After labeling a few boundary examples, we run the algorithm with this default set of parameters and adjust them or add new examples depending on the way in which the output falls short of the desired result. Some of the symptoms and possible adjustments are:

- Boundary points are not correctly mapped: The output may exhibit slight shrinkage toward the center. One way to fix this issue is to provide more labels on the boundary. Another solution is to increase α_v and α_a to under-damp the dynamics.
- All outputs take on the same value, except for abrupt jumps at example points: This happens when the kernel bandwidth parameter is too small, causing the algorithm to treat all points as being dissimilar. Increasing the bandwidth fixes this problem.
- Jittery outputs: If the recovered labels are not smooth over time, one can either force f to become smoother as a function of x, or force the label sequence to become smoother over time. The first fix is achieved by increasing λ_l, and the second by decreasing the variance of the driving noise in the dynamics.

It takes two or three iterations of applying these rules before one converges on a good set of parameters and labeled examples.

One can also search for the parameters of the model automatically by searching for the parameter settings that minimize the leave-one-out cross validation error on the labeled outputs. To compute this error, we withhold one of the labeled examples from the learning algorithm, and estimate its label based on all the other examples. The error in this estimate, averaged over several left-out examples provides a measure of performance for a given setting of the parameters. We use the simplex method to find the parameter settings that minimize this cross validation error. This procedure can take many hours to converge, because evaluating the leave-one-out error requires running the algorithm once for each left-out point. Though it is possible to speed it up with a recursive version of the learning algorithm that can quickly update its solution when labeled examples are added or removed, our experiments show that fine parameter tuning is not necessary.

Figure 10 shows the performance of the algorithm on the sequence of Figure 9 as the kernel bandwidth and regularization weight are varied by several orders of magnitude. The other parameters were fixed to values we used to generate the results of Figure 9 and Table I. The figure of merit is the average distance to the corrected sequence used in the evaluation of Table

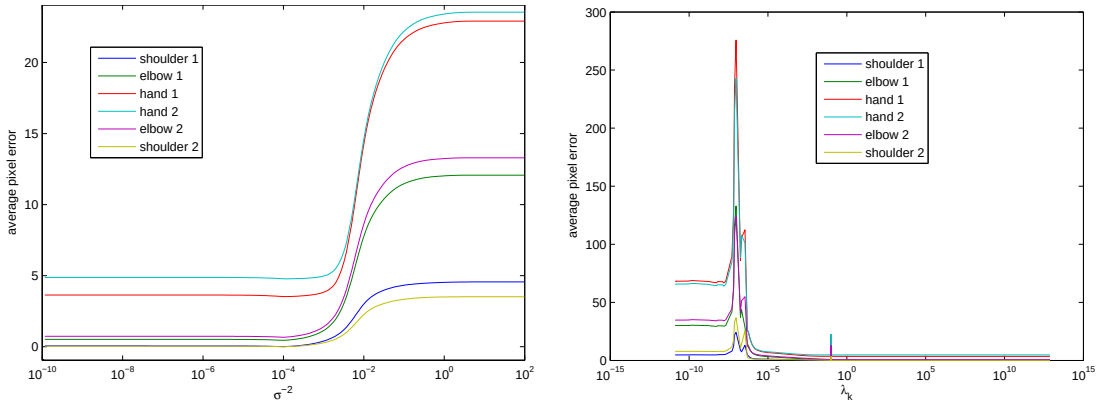

Fig. 10. (left) Average error in the position of each recovered corner in the data set of Figure 9 as the kernel bandwidth parameter is varied over several orders of magnitude. The parameter is σ^2 in the kernel $k(x,x') = \left(2\pi\sigma^2\right)^{-\frac{M}{2}} \exp\left(-\frac{1}{2\sigma^2}\|x-x'\|^2\right)$. (right) Performance as the weight λ_k, which favors the smoothness of f, is varied. The algorithm has the same performance over a wide range of settings for both parameters.

I. The algorithm produces the same result over a large range of settings for this parameter, until numerical problems emerge for wider kernels. The same result are also reported for a wide range of settings for λ_k, which governs the smoothness of f.

The algorithm is also resilient to a wide range of settings for the dynamical model. We drew 400 random joint samples of α_v and α_a, restricting both parameters in the ranges 10^{-4} to 10^3, and fixing all other parameters to the settings used to generate the results of figure 9 and Table I. Similar errors were reported for all the trajectories recovered with these parameter settings, with the average location error ranging from 3.528 to 3.53 pixels for one corner associated with the hand, and 4.77 to 4.78 pixels for the other corner. The algorithm is similarly resilient to settings of the driving noise of the dynamical model. We sampled the diagonal elements of Λ_ω in a similar fashion, restricting all three parameters to lie between 10^{-7} and 10^6. All recovered trajectories again exhibit similar errors, with the average location error ranging from 3.528 to 3.53 pixels for one corner, and 4.781 to 4.783 pixels for the other corner associated with the hand.

On the other hand, the number and choice of labels has a strong influence on the quality of the recovered trajectory. We ran our algorithm on randomly selected subsets of the 13 labeled points used to generate the results of Figure 9 and Table I. The parameters of the algorithm were fixed. Figure 11 shows the accuracy with which the algorithm recovered one of the corners

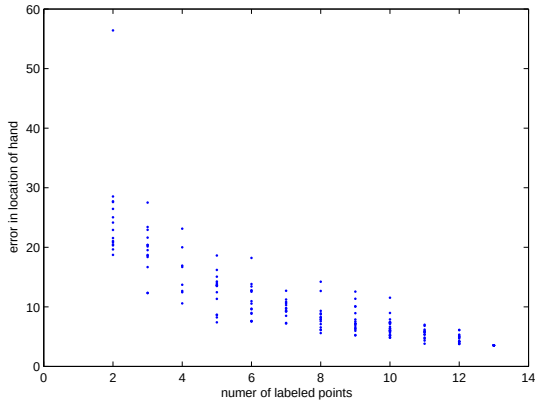

Fig. 11. Average error in the position of one of the corners corresponding to the hand, as a function of the number of labeled examples used. Labeled examples were chosen randomly from a fixed set of 13 labeled examples. Reducing the number of labels reduces accuracy. Also, the choice of labels has a strong influence on the performance, as demonstrated by the vertical spread of each column.

corresponding to the hand when training on 200 randomly selected subsets of the 13 labeled points. The accuracy of the algorithm drops with the number of labeled examples. Also note that given a fixed number of labeled examples, the choice of the examples to label can affect the accuracy by as much as 16 pixels when only 3 examples are labeled.

IX. Conclusion

We have presented algorithms that learn to transform time series. Our algorithms search smooth memoryless functions that fit input-output training examples and, when applied to the input time series, produce a time series that evolves according to assumed dynamics. In this way, they can take advantage of unlabeled input examples. The learning procedures are fast, and lend themselves to closed-form solutions.

This work augments previous work on example-based tracking by supplying a prior on the dynamics of the output. This prior can leverage unlabeled data, thereby significantly reducing the number of examples required to learn trackers. We were able to recover the pose of articulated bodies and the contour of deformable shapes from videos using very few hand-labeled frames. We were also able to learn the complicated mapping between the signal strength measurements induced by an RFID tag in a set of antennae to the position of the RFID tag. Fully-supervised regression algorithms or semi-supervised learning algorithms that do not take the dynamics of

the output time series into account were unable to perform many of these tasks.

The success of our technique on these data sets demonstrates some characteristics that many other tracking problems may exhibit:

1) When pose is the dominant factor governing the measurements, an abundance of unlabeled data can obviate the need for hand-crafted image representations.
2) Simple dynamical models capture enough of the temporal coherence to learn an appearance model.
3) The combination of labeled and unlabeled examples can obviate the need for sophisticated appearance models.

It is important to point out that in our data sets, pose was the dominant factor in the appearance of the scene, so changes in the background were negligible. Further, it was possible to construct a function to map appearance to state without the benefit of context. An extension to address the latter issue is discussed in the next section.

In [46], we develop unsupervised learning algorithms that take temporal coherency into account, and rely on no labeled data. The success of these algorithms further underlines the importance of capturing temporal coherency in learning algorithms.

X. Future Work

Several interesting directions remain to be explored. We would like to apply the semi-supervised learning algorithm to more application areas, and to extend this work by exploring various kernels that might provide invariance to more distractors, more sophisticated dynamical models, and an automatic way of selecting data points to label.

In this work, the pose of the target was the only factor governing the appearance of the target. This allowed us to use a simple Gaussian kernel to compare observations. to provide more invariance to distractors, we could either summarize images by a list of interest points and their descriptors, as in [42], or compute the similarity matrix with different kernels, as in [47] and references within. We would also like to explore an automatic way to select features by tuning the covariance matrix of the Gaussian kernel.

We have also assumed that *a priori* the components of the output evolve independently of each other. In some settings, such as when tracking articulated objects in 3D with a weak-perspective camera, *a priori* correlation between the outputs becomes an important cue, because

the observation function is not invertible and pose can only be recovered up to a subspace from a single frame. To address this issue, the data matching terms in the cost function can be set to $V(f(x_i), \mathbf{H}y_i)$, where $\mathbf{H}$ spans this subspace. A correlated prior on $\mathbf{Y}$ would then allow the algorithm to recover the best path within the subspace. We have not tried our algorithm on a data set where correlated outputs and one-to-many mappings are critical, but it would be interesting to examine the benefits of these additions, and the use of other priors.

We provided some guidelines for choosing the inputs to label, but it would be interesting to let the system guide the user's choice of inputs to label via active learning [48]–[50],

In the future, we hope to explore many other application areas. For example, we could learn to transform images to cartoon sequences, add special effects to image sequences, extract audio from muted video sequences, and drive video with audio signals.

REFERENCES

[1] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, no. 5500, pp. 2319–2323, 2000.

[2] S. Roweis and L. Saul, "Nonlinear dimensionality reduction by locally linear embedding," *Science*, vol. 290, no. 5500, pp. 2323–2326, 2000.

[3] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Computation*, vol. 15, no. 6, pp. 1373–1396, 2003.

[4] D. Donoho and C. Grimes, "Hessian eigenmaps: new locally linear embedding techniques for highdimensional data," TR2003-08, Dept. of Statistics, Stanford University, Tech. Rep., 2003.

[5] K. Weinberger and L. Saul, "Unsupervised learning of image manifolds by semidefinite programming," in *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[6] M. Brand, "Charting a manifold," in *Neural Information Processing Systems (NIPS)*, 2002.

[7] M. Balasubramanian, E. L. Schwartz, J. B. Tenenbaum, V. de Silva, and J. C. Langford, "The isomap algorithm and topological stability," *Science*, vol. 295, no. 5552, 2002.

[8] O. Jenkins and M. Mataric, "A spatio-temporal extension to isomap nonlinear dimension reduction," in *International Conference on Machine Learning (ICML)*, 2004.

[9] J. Ham, D. Lee, and L. Saul, "Learning high dimensional correspondences from low dimensional manifolds," in *ICML*, 2003.

[10] R. Pless and I. Simon, "Using thousands of images of an object," in *CVPRIP*, 2002.

[11] X. Zhu, Z. Ghahramani, and J. Lafferty, "Semi-supervised learning using gaussian fields and harmonic functions," in *ICML*, 2003.

[12] M. Belkin, I. Matveeva, and P. Niyogi, "Regularization and semi-supervised learning on large graphs," in *COLT*, 2004.

[13] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *International Conf. on Machine Learning*, 2001, pp. 282–289.

[14] A. Smola, S. Mika, B. Schoelkopf, and R. C. Williamson, "Regularized principal manifolds," *Journal of Machine Learning*, vol. 1, pp. 179–209, 2001.

[15] A. Rahimi, B. Recht, and T. Darrell, "Learning appearance manifolds from video," in *Computer Vision and Pattern Recognition (CVPR)*, 2005.

[16] Z. Ghahramani and S. Roweis, "Learning nonlinear dynamical systems using an EM algorithm," in *Neural Information Processing Systems (NIPS)*, 1998, pp. 431–437.

[17] H. Valpola and J. Karhunen, "An unsupervised ensemble learning method for nonlinear dynamic state-space models," *Neural Computation*, vol. 14, no. 11, pp. 2647–2692, 2002.

[18] L. Ljung, *System identification: theory for the user*. Prentice-Hall, 1987.

[19] A. Juditsky, H. Hjalmarsson, A. Benveniste, B. Delyon, L. Ljung, J. Sjöberg, and Q. Zhang, "Nonlinear black-box models in system identification: Mathematical foundations," *Automatica*, vol. 31, no. 12, pp. 1725–1750, 1995.

[20] K.-C. Lee and D. Kriegman, "Online learning of probabilistic appearance manifolds for video-based recognition and tracking," in *Computer Vision and Pattern Recognition (CVPR)*, 2005.

[21] G. Doretto, A. Chiuso, and Y. W. S. Soatto, "Dynamic textures," *International Journal of Computer Vision (IJCV)*, vol. 51, no. 2, pp. 91–109, 2003.

[22] O. Bousquet and A. Elisseeff, "Stability and generalization," *Journal of Machine Learning Research*, 2002.

[23] M. P. T. Evgeniou and T. Poggio, "Regularization networks and support vector machines," *Advances in Computational Mathematics*, 2000.

[24] G. Wahba, "Spline models for observational data," *SIAM*, vol. 59, 1990.

[25] B. Schölkopf, R. Herbrich, A. Smola, and R. Williamson, "A generalized representer theorem," NeuroCOLT, Tech. Rep. 81, 2000.

[26] V. Vapnik, *Statistical learning theory*. Wiley, 1998.

[27] T. De Bie and N. Cristianini, "Convex methods for transduction," in *Neural Information Processing Systems (NIPS)*, 2003.

[28] T. Joachims, "Transductive inference for text classification using support vector machines," in *International Conference on Machine Learning*, 1999, pp. 200–209.

[29] K. Bennett and A. Demiriz, "Semi-supervised support vector machines," in *Advances in Neural Information Processing Systems (NIPS)*, 1998, pp. 368–374.

[30] R. Rifkin, G. Yeo, and T. Poggio, "Regularized least squares classification," *Advances in Learning Theory: Methods, Model and Applications, NATO Science Series III: Computer and Systems Sciences*, vol. 190, 2003.

[31] T. Kailath, A. H. Sayed, and B. Hassibi, *Linear Estimation*. Prentice Hall, 2000.

[32] A. Rahimi, "Learning to transform time series with a few examples," Ph.D. dissertation, Massachusetts Institute of Technology, Computer Science and AI Lab, Cambridge, Massachusetts, USA, 2005.

[33] D. Fokkema, G. Sleijpen, and H. van der Vorst, "Jacobi-davidson style qr and qz algorithms for the reduction of matrix pencils," *SIAM J. Sc. Comput.*, vol. 20, no. 1, pp. 94–125, 1998.

[34] J. Patten, B. Recht, and H. Ishii, "Audiopad: A tag-based interface for musical performance," in *Conference on New Interfaces for Musical Expression (NIME)*, 2002.

[35] J. Patten, H. Ishii, J. Hines, and G. Pangaro, "Sensetable: A wireless object tracking platform for tangible user interfaces," in *CHI*, 2001.

[36] H. Sidenbladh, M. J. Black, and D. Fleet, "Stochastic tracking of 3d human figures using 2d image motion," in *European Conference on Computer Vision (ECCV)*, 2000, pp. 702–718.

[37] C. Bregler and J. Malik, "Tracking people with twists and exponential maps," in *Computer Vision and Pattern Recognition (CVPR)*, 1998.

[38] A. Rahimi, "Spline drawing tool for MATLAB," MIT CSAIL, http://people.csail.mit.edu/rahimi/splines/html, Tech. Rep., Aug. 2005.

[39] A. A. Efros, A. C. Berg, G. Mori, and J. Malik, "Recognizing action at a distance," in *International Conference on Computer Vision (ICCV)*, 2003.

[40] G. Shakhnarovich, P. Viola, and T. Darrell, "Fast pose estimation with parameter sensitive hashing," in *International Conference on Computer Vision (ICCV)*, 2003.

[41] A. M. Elgammal, "Learning to track: Conceptual manifold map for closed-form tracking," in *Computer Vision and Pattern Recognition (CVPR)*, 2005, pp. 724–730.

[42] A. Agarwal and B. Triggs, "3D human pose from silhouettes by relevance vector regression," in *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[43] A. Agarwala1, A. Hertzmann, and S. M. S. D. H. Salesin1, "Keyframe-based tracking for rotoscoping and animation," in *SIGGRAPH*, 2004.

[44] A. Agarwala, "Snaketoonz: A semi-automatic approach to creating cel animation from video," in *International Symposium on Non-Photorealistic Animation and Rendering*, 2002. [Online]. Available: http://www.agarwala.org/Pages/snaketoonz.html

[45] K. Grauman and T. Darrell, "Fast contour matching using approximate earth mover's distance," in *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[46] A. Rahimi and B. Recht, "Estimating observation functions in dynamical systems using unsupervised regression," in *Advances in Neural Information Processing Systems (NIPS)*, 2006.

[47] K. Grauman and T. Darrell, "The pyramid match kernel: Discriminative classification with sets of image features," in *International Conference on Computer Vision (ICCV)*, 2005.

[48] N. Roy and A. McCallum, "Toward optimal active learning through sampling estimation of error reduction," in *IEEE International Conference on Machine Learning (ICML)*, 2001, pp. 441–448.

[49] G. Schohn and D. Cohn, "Less is more: Active learning with support vector machines," in *IEEE International Conference on Machine Learning (ICML)*, 2000, pp. 839–846.

[50] R. Yan, J. Yang, and A. Hauptmann, "Automatically labeling video data using multi-class active learning," in *International Conference on Computer Vision (ICCV)*, 2003, pp. 516–523.

[51] D. P. Bertsekas, A. Nedic, and A. E. Ozdaglar, *Convex Analysis and Optimization*. Athena Scientific, 2001.

APPENDIX

Equation (22) has the form:

$$\min_x \frac{1}{2} x' \mathbf{A} x \tag{27}$$

$$\text{s.t. } \mathbf{B}x = c. \tag{28}$$

This optimization can be performed in closed form, without invoking the heavy machinery of quadratic programming. In general, both $\mathbf{A}$ and $\mathbf{B}$ may be rank-deficient. The feasible set is the affine subspace $\mathbf{B}^{\perp}u + x_0$, where $\mathbf{B}^{\perp}$ is a basis set that spans the nullspace of $\mathbf{B}$, and x_0 is any feasible point. Then the optimization becomes an unconstrained quadratic minimization over u: $\min_u (\mathbf{Z}u + x_0)' \mathbf{A} (\mathbf{Z}u + x_0)$, which is solved by least squares. The optimal x can then be obtained via $x^* = \mathbf{B}^{\perp}u^* + x_0$.

The matrix $\mathbf{B}^{\perp}$ may be computationally expensive to compute or store. In this article, $\mathbf{A}$ is symmetric positive semidefinite, and $span(\mathbf{B}') \subseteq span(\mathbf{A})$, or equivalently, $null(\mathbf{B}) \supseteq null(\mathbf{A})$. In these cases, another solution is available.

The dual of (27) is $\max_l \min_x \frac{1}{2} x' \mathbf{A} x + l'(c - \mathbf{B}x)$, where l is the vector of dual variables. There is no duality gap in convex quadratic programs [51, chap. 2]. For a fixed value of l, an optimal x must satisfy $\mathbf{A}x = \mathbf{B}'l$. Therefore, given l, we know that $x^* \in \{\mathbf{A}^{\dagger}\mathbf{B}'l + \delta | \delta \in null(\mathbf{A})\}$, where $\mathbf{A}^{\dagger}$ is the pseudo-inverse of $\mathbf{A}$. Plugging back into the Lagrangian yields the problem $\max_l \min_{\delta \in null(\mathbf{A})} \frac{1}{2} l' \mathbf{B} \mathbf{A}^{\dagger} \mathbf{B}' l + l'c - l' \mathbf{B} \delta$. By assumption, δ is also in the nullspace of $\mathbf{B}$, so the last term is zero, which results in the minimization $\min_l l' \mathbf{B} \mathbf{A}^{\dagger} B' l - l'c$. Thus $l^* = \left(\mathbf{B}\mathbf{A}^{\dagger}\mathbf{B}'\right)^{\dagger} c$ is a feasible dual variable at the optimum. Using $\mathbf{A}x = \mathbf{B}'l$, we get that the smallest norm optimal solution to (27) is

$$x^* = \mathbf{A}^{\dagger}\mathbf{B}' \left(\mathbf{B}\mathbf{A}^{\dagger}\mathbf{B}'\right)^{\dagger} c. \tag{29}$$

This provides an alternative way to solve (27) without explicitly finding the null space of the constraint set.