Learning to Transform Time Series with a Few Examples

by

Ali Rahimi

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Electrical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Feb 2005

Author .....
Department of Electrical Engineering and Computer Science
4 Nov 2005

Certified by...
.....
Trevor J. Darrell
Associate Professor
Thesis Supervisor

Accepted by...
.....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Learning to Transform Time Series with a Few Examples

by

Ali Rahimi

Submitted to the Department of Electrical Engineering and Computer Science
on 4 Nov 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Electrical Engineering

Abstract

I describe a semi-supervised regression algorithm that learns to transform one time series into another time series given examples of the transformation. I apply this algorithm to tracking, where one transforms a time series of observations from sensors to a time series describing the pose of a target. Instead of defining and implementing such transformations for each tracking task separately, I suggest learning a memoryless transformations 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. I relate this algorithm and its unsupervised extension to nonlinear system identification and manifold learning techniques. I demonstrate it 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, and tracking a target in a completely uncalibrated network of sensors. 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.

Thesis Supervisor: Trevor J. Darrell
Title: Associate Professor

Acknowledgments

This thesis is the result of a collaboration with Ben Recht. It is the culmination of many brainstorming sessions and a few papers we coauthored. I thank him for the most fruitfuil collaboration I have ever had.

James Patten made the *Sensetable* available to us and helped us record data from it. Sam Roweis, Matt Beal, and Dan Klein provided stimulating conversations, and gave me several helpful pointers. This document benefited from many helpful comments and edits from my committee members Dr. TD, Tommi Jaakkola, and Stefano Soatto, who suggested additional experiments, found errors in the mathematical presentation, and generally suggested ways to make the document more convincing.

I am indefinitely indebted to my advisors, Trevor Darrell (Dr. TD) and Sandy Pentland, for their guidance and for bankrolling many years of graduate school. Dr. Ahmad Waleh was my second advisor and mentor ever, and set me on the path to good engineering. His admonishment "be systematic" still resound in my head. Most importantly, I thank my father, my very first and most influential mentor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Many problems in machine perception, computer graphics, and controls can be framed as mapping one time series to another time series. For example, in tracking, one transforms observations from sensors to the pose of a target; by transforming a time series representing the motions of an animator to vectorized graphics, one can generate computer animation; and fundamentally, a controller maps a time series of measurements from a plant to a time series of control signals. Typically, all of these time series transformations are expressed programmatically, and by devising a different algorithm for each application. If instead, we could teach a machine to perform these transformations by supplying examples of the transformation, there could be significant savings in the time required to develop such tools, and it would become easier to customize them at a later time. As a step in this direction, I derive an algorithm that learns how to transform time series from examples. If focus on applying the abstraction of transforming time series to learning trackers from examples. I explore an unsupervised version of this algorithm that can perform these transformations given no examples at all, relying only on aggregate *a priori* knowledge about the structure of the output time series.

The time series transformation algorithm takes as input a time series to transform, and some examples of how to perform the requested transformation. It then returns a function generalizes these input-output examples, and as a side-effect, also transforms the input time series. The time series are multivariate functions of a discrete time index, and the examples are expressed as pairs of input samples and their corresponding output samples. The algorithm returns a memoryless and time-invariant mapping from samples of the input time series to samples of an output time series. In learning this mapping, the algorithm takes advantage of both the input-output examples and the input time series. The algorithm fits in the framework of semi-supervised learning because it takes advantage of both labeled data (the input-output examples) and unlabeled data (the input time series) to learn this memoryless mapping. I also discuss a fully unsupervised version of the algorithm that replaces the input-output examples with a description of the aggregate behavior of the output time series. This unsupervised algorithm often recovers the low-dimensional process underlying the input time series. It can be used a as a knowledge discovery tool. It can learn how to track a moving object in a field of networked sensors when almost nothing is known

about the sensor a priori (I will only assume that the transduction from the position of the target to measurement signals is smooth. The algorithm is supplied with no other information about the measurement model, and will recover the observation function automatically).

I apply the semi-supervised learning algorithm to learning visual trackers, since a video sequence is a very high-dimensional time series of pixel values. I focus on applications where a user is given a video sequence and asked to annotate all of its frames with a low-dimensional representation of the scene. This desired low-dimensional representation is specific to each tracking application, and can vary from task to task. For example, it could refer to the image-plane position of a target, or its 3D position, to the contour of a moving shape, or the articulated pose of a human body. A tool allows user to annotate a few frames of a video sequence, and invokes the semi-supervised algorithm to automatically annotate the other frames in the video sequence. For example, from a video of a moving person and the annotations of a few frames of the video with the position of the person's limbs, the algorithm can learn a nonlinear function that maps images of the person to the pose of the person's limbs. This function can be applied to the rest of the video sequence to recover poses for the unlabeled frames. The specified example outputs can be any vector representation, as long as the values of the representation change smoothly over time. The very same algorithm can be used to learn to track deformable contours like the contour of lips (represented with a spline curve), or to transform videos of an animator to the control points of an animated character.

These trackers work well despite their simplistic representation of video as a time series of pixel values. There need not be any explicit reasoning about occlusion, or edges, or many other issues traditionally associated with visual tracking. The time series representation, along with the supplied examples and the various smoothness assumptions that are explicitly made in the algorithm seem to convey enough information to build trackers in many cases. Of course, many caveats apply for this algorithm to work well in computer vision 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, and motions of the camera. But occlusions due to stationary objects, or even self occlusions are allowed. This is because the algorithm represents mappings from images to poses with radial basis functions, with a kernel centered on each image of the video sequence. In this thesis, I use Gaussian kernels, which ultimately result in sum-of-squared comparisons between image pairs. Thus, the algorithm is also invariant to orthonormal operators applied to images, such as permutations of the pixels, as these operations do not effect the result of the comparison. If distractors do exist in the scene, they may be removed by preprocessing the 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 the output example. In practice, this smoothness requirement is not a problem, even when dealing with 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, and will perform only as well as fully-supervised nonlinear regression algorithms. A more detailed description of the intuition and requirements behind the performance of this algorithm are given in Chapter 3.

Videos are perhaps the most challenging time series to process, exhibiting high-dimensional observations, complex nonlinear relationships between the input and output representations, observation noise, and distractors and occlusions. The semi-supervised algorithm isn't limited to computer vision applications. We have used it to learn to track a radio frequency ID tag by mapping the voltages it induces in electromagnetic coils to the tag's position. Chapter 4 documents some of our computer vision and non-computer vision experiments.

1.1 The Value of Examples

In Chapter 4, I show that a generic algorithm can solve some complex computer vision tasks with a raw pixel representation, a handful of examples, and some smoothness assumptions. In the second half of the thesis, I present an unsupervised algorithm that learns how to track without requiring any input-output examples. This shows that in certain cases, a few smoothness assumptions about the raw pixel values capture all the necessary structure of the scene to perform tracking.

The unsupervised algorithm is based on the premise that the appearance of a dynamic scene is governed by a low-dimensional dynamical process, and that tracking often seeks to recover that process. The algorithm assumes that the dynamics of this process are known *a priori*, and searches for a function that transforms the input time series so its dynamics adhere to those of the underlying process. The algorithm is an extension of the semi-supervised algorithm, with a few terms modified, and like its semi-supervised counterpart, it relies only on fast matrix operations.

Both the semi-supervised and unsupervised function learning algorithms are closely related to the problems of nonlinear dimensionality reduction using manifold learning, and to nonlinear system identification. Manifold learning algorithms map high-dimensional observations to low-dimensional coordinates on a manifold embedded in a high-dimensional space. These low-dimensional coordinates are chosen so as to preserve various geometric properties observed in the high-dimensional observations. Instead of enforcing a geometric property, the algorithms in this thesis learn a smooth mapping between two spaces that force the output coordinates to obey given dynamics. As far as I am aware, these are the only manifold learning algorithms that that explicitly model dynamics in the low-dimensional space.

Nonlinear system identification seeks to recover the parameters of a generatic model for observed data. 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 a nonlinear observation functions. Learning an observation function is difficult, and requires optimiziation procedures that are prone to local minima, and signficant storage requirements. As in conditional random fields [44], the algorithms in this thesis learn a mapping in the reverse

directions, from observations to states. This alleviates both the storage and the local minimum problem, making it possible to characterize the pseudo inverse of the observation function for very high dimensional observations, such as video frames.

1.2 Basics of the Approach

We wish to learn a memoryless mapping that transform a sample of the input time series at any time point to a sample of an output time series. One way to learn such a transformation from examples is to apply standard nonlinear regression techniques to find a function that fits specified input-output examples. Such functions can typically represent any smooth mapping, and take the form of multilayer perceptrons, radial basis functions, or any other sufficiently general family of functions. But for the applications I tackle, nonlinear regression techniques require too many examples to be of practical use. To accommodate the dearth of available examples, the semi-supervised regression algorithm in this thesis utilizes easy-to-obtain side information such as unlabeled examples and a prior distribution on the output. A prior on the output time series allows our algorithm to automatically take advantage of any available unlabeled data. Although the final learned mapping does not have memory, finding such a mapping is a batch process that takes advantage of the entire data set.

In tracking applications, the output time series represents the motion of physical mass, so we expect that this time series will exhibit physical dynamics. This *a priori* knowledge can be approximately captured using a linear-Gaussian autoregressive model, a dynamics model commonly used in tracking applications. Like nonlinear regression methods, this semi-supervised algorithm searches for a smooth function that fits the example input-output pairs. But it also simultaneously estimates missing output labels. The missing output labels are made to agree with the dynamics model, as well as the output of the function. As such, the function is required to produce a time series that behaves according to the given dynamics when applied to the input time series. The search is expressed as a joint optimization over missing labels and a set of functions in a Reproducing Kernel Hilbert Space (RKHS) [84].

The unsupervised learning algorithm derived in this thesis operates without the benefit of input-output examples. It searches for a mapping that results in an output sequence that evolves according to the given dynamics and whose moments match those of the prior on the dynamics. It reduces to an eigenvalue problem reminiscent of various manifold learning algorithms.

The cost functional for the unsupervised and semi-supervised algorithms are computationally easy to optimize, so that the algorithms can be executed at interactive rates on modern computers. The cost functions are crafted so as to not require expensive inference algorithms such as MCMC methods, or slow-converging optimization procedures such as EM, or even Newton iterations. Because all the penalty terms in our cost function are quadratic, after applying the Representer Theorem [84], all of the optimizations reduce to least-squares or eigenvalue problems.

It is important to note that in this thesis, I assume that these dynamics models are fully known *a priori*. I do not attempt to estimate their parameters from data.

A lot is usually known about the motion of objects before applying our algorithm, so it is convenient to specify these dynamics by hand. This is why in much of the work on tracking, these dynamics are also specified manually when building filtering algorithms. Further, it is not essential that the dynamics be known exactly, because the algorithm is empirically not very sensitive to the parameter settings of the dynamics model.

1.3 Contributions

This thesis draws from the areas of semi-supervised and unsupervised learning, computer vision, and system identification. Its contributions also lie across these areas. It contributes the following developments:

- A tool for quickly annotating video sequences, and for rapidly prototyping vision and signal processing applications.
- Enhancing manifold learning algorithms by explicitly representing temporal dynamics in the low-dimensional space.
- Showing that Kernel PCA learns a function in a Reproducing Kernel Hilbert space that projects high-dimensional points to uncorrelated low-dimensional coordinates.
- Kernelizing on observations instead of latent states to efficiently learn an approximation of the pseudo-inverse of the observation function in nonlinear system identification, when the dynamics are known.

Chapter 2

Background

This chapter introduces some of the prior work upon which this work is built. The relationship to existing algorithms will be clarified in subsequent chapters.

2.1 Notation

In this thesis, scalars are denoted by Greek letters, as in λ, or in upper case, as in N. When it is not confusing to do so, scalar indices that appear in subscripts and superscripts are written in lower case letters, as in x_t. Vectors are denoted by lower case letters, such as c, and are column vectors unless transposed with the $'$ mark, as in c'. Matrices are bold and upper case, as in $\mathbf{A}$. The vector $\mathbf{1}$ is a column vector consisting of all ones, and $\mathbf{I}$ is the identity matrix. Sets of indices are written in a calligraphic font, as in $\mathcal{L} = \{1 \cdots T\}$, which describes the set of integers from 1 to T, inclusive. A set consisting of vectors x_t with $t \in \mathcal{L}$ is written as $\mathbf{X} = \{x_t\}_{\mathcal{L}}$, and also acts as a matrix consisting of the x_t's stacked horizontally. When such a set is indexed by a set of increasing integers, it describes a time series.

2.2 Time Series Model and State Estimation

The time series abstraction provides a convenient notation for taking the temporal correlation of data sets into account. There are many auto-regressive models for time series that model this temporal correlation. I focus on state-space models because they provide both a physically plausible, and a compact generative model for many time series. Common operations with time series models include estimating the parameters of the model from time series data, attenuating observation noise, predicting future samples, or inferring a latent state underlying each sample of the time series. The book by Shumway and Stoffer [78] offers a thorough review of time series analysis methods.

Methods for estimating the parameters of a linear autoregressive model driven by Gaussian noise find a model whose second order output statistics agree with the observed second order statistics of the observed time series data [49]. This results in the Yule-Walker equations, a set of linear equations in the parameters of the model,

which can be solved efficiently. For linear state-space models, subspace methods [91, 49] have proved to be the most effective way of fitting these second order statistics, as they are consistent and do not suffer from local minima.

Matching second order moments is not adequate for identifying nonlinear systems since the nonlinearities in the system make the distribution over the outputs non-Gaussian. When a training data set consisting of states, their corresponding observations, and subsequent states is available, a straightforward solution is to learn a pair of nonlinear functions representing the state transition function and the observation function, usually represented with radial basis functions or neural networks [59, 22, 50]. When the states are not available in the training data set, a common approach is to learn a 1-step-ahead predictor that maps a short history of past observations to a predicted output. Such a function is again represented using radial basis functions or neural networks [66, 17]. Such techniques are useful for building controllers, but do not learn state-space models, so state estimation, dimensionality reduction, and smoothing are not possible. The only unsupervised methods for learning the parameters of nonlinear state-space models use the EM algorithm to estimate the model parameters while marginalizing over the latent states [26, 90]. Unfortunately, these methods are computationally intensive, subject to local minima, and do not scale well to very high-dimensional observations. See Section 5.5.2 for more detail.

In Chapter 5, we devise an approximate method for estimating the inverse of the observation function of a state-space model with nonlinear observations. Estimating the inverse of the observation function instead of the observation function itself reduces the estimation problem to a quadratic optimization problem that circumnavigates the problems introduced by the EM algorithm.

2.3 Function Fitting

Function fitting is the process of learning an input-output mapping given L training example pairs $\{x_i, y_i\}_{i=1..L}$ of example inputs $x_i \in \mathcal{R}^M$ and their corresponding labels $y_i \in \mathcal{R}^N$. Regression is a fully-supervised learning problem because both inputs $\{x_i\}$ and outputs $\{y_i\}$ of a function are given. The mapping itself is a function f, and can take any form, such as linear, polynomial, Radial Basis Function (RBF), neural networks, or the nearest neighbors rule. The function f may have a fixed number of parameters θ, in which case we refer to the function learning problem as being parameteric, or the number of parameters of f may grow with L, in which case the problem is called non-parameteric. We use the notation $f_\theta(x)$ to denote label predicted by the mapping given an input x.

To find the best mapping, one defines a loss $V(y, z)$ between output labels, and one minimizes a cost of the form

$$\min_\theta \sum_{i=1}^{L} V(f_\theta(x_i), y_i) \tag{2.1}$$

Additionally, one may place a reguarlizer on f or on its parameters to provide shrinkage towards a prior, or to improve stability [13].

$$\min_{\theta} \sum_{i=1}^{L} V(f_\theta(x_i), y_i) + P(\theta). \tag{2.2}$$

When $V(y, z) = \|y - z\|^2$ is the quadratic loss, and f_θ, we obtain the standard linear linear squares problem, with θ denoting the slops and intercept of a hyperplane. When the reguarlizer $P(\theta)$ is also quadratic, we get ridge regression. When θ are the coefficients of a polynomial in the components of x, we get polynomial regression. All of these fitting problems can be solved with straightforward least-squares.

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

$$f_\theta(x) = \sum_{j=1}^{C} \theta_j k(x, c_j). \tag{2.3}$$

Here, θ consists of vectors $\theta_i \in \mathcal{R}^N$, and k maps $\mathcal{R}^M \times \mathcal{R}^M \rightarrow \mathcal{R}^N$. Estimating θ with a quadratic loss still reduces to a least-squares problems, since the output of f is linear in the parameters of f.

The nearest neighbor rule provides a simple way to learn mappings without much training. A function f with the nearest neighbors form returns the label corresponding to the training point with the closes x_i to the queried x:

$$f_\theta(x) = y_{\arg\min_i k(x_i, x)}, \tag{2.4}$$

for some distance measure k between points in $\mathcal{R}^M$. The k-nearest neighbors form or the ϵ-neighbors form are similar in that they return the average of the labels within a neighborhood of the query point:

$$f_\theta(x) = \frac{1}{|\mathcal{N}(x)|} \sum_{j \in \mathcal{N}} y_i, \tag{2.5}$$

where $\mathcal{N}$ are the indices of the training input examples x_i which are close to x in some sense, usually either the k nearest neighbors of x in $\{x_i\}_{1 \cdots L}$, or the elements of $\{x_i\}_{1 \cdots L}$ whose distance to x is less than some ϵ. Note that the parameters θ of this model constist of the entire collection of training examples. Also, since the neighbor hood around x changes in discrete steps, f is a piecewise constant function. We will use the nearest neighbors form to derive alternatives to our algortihms, which are based on the RBF representation for the mapping.

2.3.1 Reproducing Kernel Hilbert Spaces

Rather than chosing a class of functions first, we can search for a mapping directly in an infinite Hilbert Space of functions. This approach requires us only to define an

inner product in a Reproducing Kernel Hilbert Space (RKHS) by specifying a kernel function. The choice of an inner product specifies a norm in the Hilbert space, which can be used as a regularizer. The optimal mapping according to this technique takes a specific finite RBF form, allowing the function fitting to be carried out efficiently. Like the nearest neighbors form, this approach can generalize any mapping if given enough data. But unlike nearest neighbors function, even when the sample size is small, it learns a smooth differential mapping, more closely capturing the physical mapping that underlies the training data.

Every positive definite kernel $k : \mathcal{R}^M \times \mathcal{R}^M \rightarrow \mathcal{R}$ defines a Hilbert space on bounded functions whose domains is a compact subset of $\mathcal{R}^N$ and whose range is $\mathcal{R}$ [84]. This function space is called a Reproducing Kernel Hilbert Space because the inner product in this space 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.

According to Mercer's theorem [84], every positive definite kernel k has a countable representation on a compact domain: $k(x_1,x_2) = \sum_{i=1}^{\infty} \lambda_i \phi_i(x_1)\phi_i(x_2)$, with $\phi_i : \mathcal{R}^M \rightarrow \mathcal{R}$. Combining this with the reproducing property reveals that the set of ϕ are a countable basis for the RKHS:

$$f(x) = \langle f(\cdot), k(x,\cdot)\rangle \tag{2.6}$$

$$= \langle f(\cdot), \sum_{i=1}^{\infty} \lambda_i \phi_i(\cdot)\phi_i(x)\rangle = \sum_{i=1}^{\infty} \phi_i(x)\lambda_i \langle f(\cdot), \phi_i(\cdot)\rangle \tag{2.7}$$

$$= \sum_{i=1}^{\infty} \phi_i(x) c_i, \tag{2.8}$$

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.

A similar argument shows that ϕ_i form an orthogonal basis under this inner product:

$$\phi_j(x) = \langle \phi_j(\cdot), k(x,\cdot)\rangle = \sum_{i=1}^{\infty} \phi_i(x)\lambda_i \langle \phi_j(\cdot), \phi_i(\cdot)\rangle. \tag{2.9}$$

Since the ϕ's are linearly independent, $\langle \phi_i, \phi_j\rangle = \delta_{ij}/\lambda_i$.

An analog to Parsevals' theorem shows that the norm in the RKHS can be expressed in terms of these coefficients:

$$\|f\|_k^2 = \langle f, f\rangle = \langle \sum_{i=1}^{\infty} \phi_i c_i, \sum_{i=1}^{\infty} \phi_i c_i\rangle = \sum_{ij} c_i c_j \langle \phi_i, \phi_j\rangle \tag{2.10}$$

$$= \sum_i c_i^2/\lambda_i. \tag{2.11}$$

A kernel that satisfies the stationarity condition $k(x,x') = k(x-x')$ has sinusoidal bases ϕ [93, 84]. Since the norm $\|f\|_k^2$ penalizes the coefficients the projection of f on sinusoids, the norm effeictively penalizes the amplitude of different frequency bands

according to λ_i. A common choice for the kernel k are Gaussian kernels $k(x', x) = \exp(-\|x - x'\|^2/\sigma_k^2)$, whose Mercer expansion has has decaying λ_i. Thus $\|f\|_k$ under this kernel penalizes the high frequency content in f more than the low-frequency content, favoring smoother f's [93, 84].

2.3.2 Nonlinear Regression with Tikhonov Regularization on an RKHS

Tikhonov regularization learns a function $f : \mathcal{R}^M \to \mathcal{R}^N$ that can determine good y's for as-yet-unseen inputs x [84, 92]. To learn a multivariate function $f = [f^1(x) \dots f^N(x)]$, Tikhonov regularization can be applied to each component of f independently. Denoting the dth component of each y_i by y_i^d, the Tikhonov problem for each component g^d is:

$$\min_{f^d} \sum_{i=1}^{L} V(f^d(x_i), y_i^d) + \lambda_k \|f^d\|_k^2. \tag{2.12}$$

The minimization is over the RKHS defined by k. The loss V penalizes deviations between $g^d(x_i)$ and its corresponding label y_i, so the first term favors functions that fit the training examples. The RKHS norm, $\|\cdot\|_k$, serves as a stabilizer, akin to the norm penalty in ridge regression, and favors smoothness in g^d.

Although the optimization (2.12) is a search over a function space, it can be shown that regardless of the form of V, the minimizer of (2.12) can be represented as a weighted sum of kernels placed at each x_i: [73]

$$f^d(x) = \sum_{i=1}^{L} c_i^d k(x, x_i). \tag{2.13}$$

To prove that the optimum of (2.12) has this form, we show that any solution containing a component that is orthogonal to this form must have a greater cost according to (2.12), and therefore cannot be optimal. Specifically, suppose the optimal solution has the form $g = f + h$, with f and h in the RKHS, with f having the form (2.13), and h non zero and not representable with this form so that for all c_i, $\langle \sum_{i=1} c_i k(\cdot, x_i), h \rangle = 0$. By the reproducing property, we have $\sum_i c_i \langle h(\cdot), k(\cdot, x_i) \rangle = 0$ for all c. Thus $\langle h(\cdot), k(\cdot, x_i) \rangle = 0 = h(x_i)$. Therefore, $g(x_i) = f(x_i)$. But $\|g\|_k^2 = \|f\|_k^2 + 2 \cdot 0 + \|h\|_k^2$, so $\|g\|_k^2$ is strictly greater than $\|f\|_k^2$, even though the data term is equal. Therefore, g cannot be optimal.

When $V(y, z)$ is quadratic, Equation (2.12) reduces to familiar forms. In spatial statistics, the resulting problem is known as Krigging [32], it finds the MAP estimate of data points under a Gaussian process prior with Gaussian observation noise [75], and it amounts to finding the best RBF coefficients under a quadratic prior. The optimal solution given by Equation (2.13) can be written in vector form as $K_x' c^d$, where the ith component of the column vector K_x is $k(x, x_i)$, and c^d is a column vector of coefficients. 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 (2.13) is $\|g^d\|_k^2 = c^{d'} K c^d$. Substituting these into (2.12) yields a finite dimensional quadratic problem:

$$\min_{c^d} \|Kc^d - y^d\|^2 + \lambda_k c^{d'} K c^d. \tag{2.14}$$

This optimization problem can be solved using least squares. We have shown that Tikhonov regularization on an RKHS guides the choice of RBF centers as and the regularizer when fitting RBFs.

The cost function (2.12) generalizes a variety of other learning algorithms. When the output labels are binary, $f : \mathcal{R}^M \rightarrow \mathcal{R}$, and $V(z, y)$ is set to the hinge loss $V(y, z) = \max(0, 1 - yz)$, the cost function (2.12) becomes the Support Vector Machine's loss function with slack, and searches for an f with maximum margin to the training data [84]. The optimization (2.12) is equivalent to

$$\min_{f \in \mathcal{F}, \zeta} \|f\|_k^2 + C \sum_i \zeta_i \tag{2.15}$$

$$\text{s.t. } y_i f(x_i) \geq 1 - \zeta_i \tag{2.16}$$

$$\zeta_i \geq 0, \tag{2.17}$$

since the concavity of $C \sum_i \zeta_i$ ensures that each optmal ζ_i lies on a vertex of the constraint set [12], so that ζ_i is either 0 or $1 - y_i(x_i)$, so that the optimal ζ_i satisfies $\zeta_i = \max(0, 1 - y_i f(x_i))$. This optimization searches for a signed distance function f in an RKHS $\mathcal{F}$ so that the signed distance between each point x_i and the implicit surface $\{x | f(x) = 0\}$ is made to agree in sign with y_i, and made to be greater than the margin 1. The slack variable ζ_i allows each point to venture within this margin, but this slack is penalized in the cost function by $C\zeta_i$. Substituting the RBF form for f gives the quadratic cost functional for the SVM [92].

The optimization of Equation (2.12) also has a Bayesian interpretation [84]. The RKHS norm serves as a the negative log prior, $-\log p(g|x)$, over the space of functions, and the data term serves as a negative likelihood, $-\log p(\{y_i\}|g, x)$ over the space of functions. Thus the optimum g^d is the maximum a posteriori g^d given the training data.

There are useful guarantees on the performance of Tikhonov regularization. Bousquet and Elisseff [13] showed that in order to find functions that generalize to as-yet-unseen x's, a learning algorithm must be stable under perturbations of the training data and must return a function that fits the given data set well. The norm penalty in Tikhonov regularization provides the stability, while the data term provides fidelity to the training data. Tikhonov regularization has also been used as an approximation to Structural Risk Minimization by Vapnik [92].

2.4 Manifold Learning

Manifold learning algorithms reduce the dimensionality of a high-dimensional data sets. The high-dimensional data set $\{x_i\}$ is assumed to lie on a manifold. Low-

dimensional coordinates $\{y_i\}$ corresponding to each high-dimensional point are found so that the low-dimensional coordinates preserve some desired neighborhood attribute of the manifold [86, 70, 8, 19, 94, 14]. Manifold learning is an unsupervised problem because only inputs $\{x_i\}$ are given, a function that maps these to unspecified low-dimensional outputs $\{y_i\}$ is to be estimated.

Isomap [86] finds low-dimensional coordinates that preserve the geodesic distance between high-dimensional points. It assumes that the high-dimensional data are generated by lifting low-dimensional points that lie in a convex set through an isometric lifting. Donoho and Grimes [19] pointed out that due to foreshortening effects, imaging processes are are more accurately represented by local isometry, and the location of multiple objects in a scene cannot be represented by a convex low-dimensional set. They presented Hessian LLE to handle these conditions. LLE [70] finds a conformal mapping that preserves the affine relationship between high-dimensional points in local neighborhoods. Like LLE, Laplacian Eigenmaps [8] and Semidefinite Embedding [94] attempt to preserve some notion of local geometry observed in the high dimensional data set (local isometry in the case of Semidefinite Embedding and proximity weighted by a local distance metric in the case of Laplacian Eigenmaps). If the high-dimensional data points do not densely sample the manifold, the local neighborhood structure of the manifold becomes difficult to estimate, and these algorithms recover low-dimensional points that do not exhibit the desired neighborhood attribute [6].

In the manifold learning literature, only the algorithm of Jenkins and Mataric [36] takes advantage of the temporal coherence between adjacent samples of the input time series. Their algorithm extends Isomap by grouping temporally adjacent samples and favoring temporally adjacent groups to have similar low-dimensional coordinates. While it does not model dynamics, this algorithm does take advantage of the time ordering of points. In Chapter 5, we introduce a manifold learning algorithm that incorporates knowledge about the temporal dynamics of the low-dimensional points, allowing it to implicitly estimate a velocity for each data point in the low-dimensional space. This estimate implicitly provides a distance measurement between temporally adjacent points in the low-dimensional space, obviating the need for the brittle neighborhood relationships estimated from high-dimensional data points, and favors low-dimensional coordinates to agree with the latent process that generated the high-dimensional data set. The proposed algorithm is closest to Principal Manifolds [80], a function estimation framework for learning a function that lifts low-dimensional coordinates to the observed high-dimensional points.

Some manifold learning algorithms have been extended to allow the low-dimensional coordinates for certain points to be specified by hand [31, 63]. This allows the low-dimensional coordinate system to be fixed, and improves the estimated coordinates. Similarly, the algorithm of Chapter 3 allows labeled points to be supplied, making it a semi-supervised version of the algorithm of Chapter 5.

2.5 Manifold Structure for Semi-supervised Learning

The semi-supervised regression approaches of [96] and [7] take advantage of the manifold structure of the data to estimate missing outputs when only inputs $\{x_i\}$ and some of the outputs $\{y_i\}$ are given.

Input points with missing outputs aid in estimating the manifold structure of the input space. Knowledge of this structure can in turn be used to improves the estimate of the missing outputs if we assume that these must preserve the manifold structure in some way. However, these algorithms can suffer from the same brittleness as unsupervised manifold learning algorithms, because they estimate the local manifold structure from the neighborhood structure of the inputs $\{x_i\}$. These semi-supervised methods are similar to the algorithms derived in Chapter 3 thesis in that they impose a random field on the output labels. In Chapter 3, we augment these techniques by introducing the temporal dependency between output samples in the random field.

The semi-supervised learning algorithm of Belkin and Niyogi [7, 9] attempts to preserve the neighborhood structure of the input points by ensuring that the pairwise distance between neighboring output points is similar to the distance between corresponding input points. Let w_{ij} denote the distance between the input point x_i and a point x_j in the neighborhood of x_i. If x_j is not in the neighborhood of x_i, $w_{ij} = 0$. For each x_i, let y_i be the corresponding scalar output to be estimated, and let the set $\{z_i\}_{\mathcal{L}}$ denote the set of given outputs. The Belkin and Niyogi algorithm finds scalar output labels by solving

$$\min_{y} \sum_{ij} (y_i - y_j)^2 w_{ij} + \lambda \sum_{i \in \mathcal{L}} (y_i - z_i)^2 \tag{2.18}$$

$$\text{s.t.} \quad \sum_{i} y_i = 0, \tag{2.19}$$

The first term in the cost function ensures that the distance between two outputs is weighted according to the distance of their corresponding inputs. The second term favors a match between estimated outputs and given outputs. The constraint makes it possible to prove that the algorithm is stable.

Taking advantage of prior information about missing outputs is crucial in semi-supervised learning. Consider, for example, an attempt at converting the fully-supervised Tikhonov regularization algorithm of Section 2.3.2 into a semi-supervised learning algorithm without any prior over the missing outputs. Searching for f and the outputs y gives the following problem:

$$\min_{f,y} \sum_{i=1}^{L} (f(x_i) - y_i)^2 + \sum_{i \in \mathcal{L}} (f(x_i) - z_i)^2 + \lambda_k \|f\|_k^2. \tag{2.20}$$

Since there are no constraints on y, we may set it to $y_i = f(x_i)$ to minimize $(f(x_i) - y_i)^2$, effectively eliminating this term from the optimization. Hence the optimization

reduces to the original, fully-supervised cost functional

$$\min_f \sum_{i \in \mathcal{L}} (f(x_i) - z_i)^2 + \lambda_k \|f\|_k^2. \tag{2.21}$$

The need for a prior over missing data is discussed further in Chapter 3.

2.6 Linear Gaussian Markov Chains

Our semi-supervised learning algorithm takes advantage of a prior on outputs to improve its estimates of the missing outputs and of the regressor. Since we are mainly interested in learning to track objects, we have chosen a prior that is suitable for modeling the physical dynamics of moving objects. In this section, I define this prior and introduce some useful operations on this prior.

Suppose the state s_t of an object at time t evolves with linear Gaussian Markovian dynamics:

$$s_t = \mathbf{A} s_{t-1} + \omega_t. \tag{2.22}$$

The Gaussian random variable ω_t has zero mean and covariance Λ_ω. Premultiplying the state vector s_t by $\mathbf{A}$ deterministically evolves it to the state at time $t+1$. The Gaussian random variable ω_w models nondeterministic effects not captured by $\mathbf{A}$. The random variables ω_t are iid over time, have zero mean, and have covariance Λ_ω. We will let the initial state s_0 be a zero-mean Gaussian with a very large covariance $\sigma_0^2 \mathbf{I}$, so that its influence on the chain is small.

When describing the motion of an object, we will consider Markov chains where Λ_ω is diagonal, and where $\mathbf{A}$ has the special form

$$\mathbf{A} = \begin{bmatrix} 1 & \alpha_v & 0 \\ 0 & 1 & \alpha_a \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.23}$$

In this case, the components of s_t have intuitive physical analogs: the first component corresponds to a position, the second to velocity, and the third to acceleration. The dot product of each s_t with the vector $h = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}'$ extracts the position component from each s_t.

The sequence of states from time $t = 1$ to $t = T$, denoted by $\mathbf{S} = \{s_t^d\}_{t=1..T}$, is a zero-mean Gaussian random variable, since each of its components is a sum of zero-mean Gaussian random variables. Thus a distribution over it can be written as

$$p(\mathbf{S}) = \mathcal{N}(\mathrm{vec}\,(\mathbf{S})\,|0, \Omega) \propto \exp\left(-\frac{1}{2}\mathrm{vec}\,(\mathbf{S})'\,\Omega \mathrm{vec}\,(\mathbf{S})\right), \tag{2.24}$$

where the $\mathrm{vec}\,(\cdot)$ operator stacks the elements of $\mathbf{S}$ vertically into a column vector. The covariance matrix Ω is generally dense, but its inverse is block-tri-diagonal. This

can be seen by regrouping the terms in the prior:

$$p(\mathbf{S}) = p(s_0)\prod_{t=1}^{T} p(s_t|s_{t-1}) \tag{2.25}$$

$$\propto \exp\left(-\frac{1}{2}\|s_0\|^2/\sigma_0^2\right)\prod_{t=1}^{T}\exp\left(-\frac{1}{2}\|s_t - \mathbf{A}s_{t-1}\|^2_{\Lambda_\omega}\right) \tag{2.26}$$

$$= \exp\left[-\frac{1}{2}\left(\|s_0\|^2/\sigma_0^2 + \sum_{t=1}^{T}\|s_t - \mathbf{A}s_{t-1}\|^2_{\Lambda_\omega}\right)\right]. \tag{2.27}$$

Since the exponent is a sum of quadratics involving adjacent entries of $\mathbf{S}$, it can be written as the vectorized quadratic (2.24) after some manipulation.

To provide a prior over the position of an object, we will need a distribution over the position components $s'_p = h'\mathbf{S}$ of $\mathbf{S}$. This distribution can be obtained by marginalizing over all other components of $\mathbf{S}$. Since $\mathbf{S}$ is a zero-mean Gaussian random variable, the position components $\mathbf{S}$ will also form a zero-mean Gaussian random variable. To find the inverse covariance of this Gaussian, write $h'\mathbf{S}$ in terms of vec $(\mathbf{S})$ using the identity vec $(\mathbf{ABC}) = (\mathbf{C}' \otimes \mathbf{A})$ vec $(\mathbf{B})$, where $\otimes$ is the Kronecker product [55]. We get $s_p = (\mathbf{I} \otimes h')$ vec $(\mathbf{S}) = \mathbf{H}$vec $(\mathbf{S})$. Thus the covariance of s_p is $\mathbf{H}\Omega^{-1}\mathbf{H}'$, and its inverse covariance is $(\mathbf{H}\Omega^{-1}\mathbf{H}')^{-1}$. Note that although the inverse covariance of vec $(\mathbf{S})$ is block tri-diagonal, the inverse covariance of the position components is dense, which means the position components are conditionally fully dependent on each other when the other components of the state are marginalized out.

2.7 Easy to Solve Quadratic Problems

The optimization problems in this thesis are crafted to be solvable by fast linear algebraic methods. We will encounter equality constrained quadratic optimizations of the form

$$\min_x \frac{1}{2}x'\mathbf{A}x \tag{2.28}$$

$$\text{s.t. } \mathbf{B}x = c. \tag{2.29}$$

This optimization can be performed in closed form, without invoking the heavy machinery of quadratic programming. In general, both $\mathbf{A}$ and $\mathbf{B}$ are rank deficient. The feasible set is the affine subspace $\mathbf{B}^\perp u + x_0$, where x_0 is any feasible point, and $\mathbf{B}^\perp$ is a basis set that spans the nullspace of $\mathbf{B}$. Then the optimization becomes a simple 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 $x^* = \mathbf{B}^\perp u^* + x_0$.

Computing $\mathbf{B}^\perp$ explicitly may be both computationally and storageally expensive. At various points in the thesis, we have situations where $\mathbf{A}$ 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 (2.28) 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 this problem because the problem is closed, convex and smooth [12]. 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 (2.28) is

$$x^* = \mathbf{A}^\dagger\mathbf{B}'\left(\mathbf{B}\mathbf{A}^\dagger\mathbf{B}'\right)^\dagger c. \tag{2.30}$$

This provides an alternative way to solve (2.28) without explicitly finding the null space of the constraint set.

The following matrix sphere optimization problem reduces to a Rayleigh quotient:

$$\min_{\mathbf{X}} \quad \mathrm{tr}\mathbf{X}\mathbf{H}\mathbf{X}' \tag{2.31}$$

$$\text{s.t.} \quad \mathbf{X}\mathbf{X}' = \mathbf{I}, \tag{2.32}$$

where $\mathbf{H}$ is positive semi-definite. First, observe that there is not a unique optimum. If $\mathbf{X}^*$ is optimal, due to the cyclic property of the trace [55], so is $\mathbf{Q}\mathbf{X}^*$, where $\mathbf{Q}$ is square and orthonormal. One optimal solution is obtained by setting the rows of $\mathbf{X}$ to the eigenvectors of $\mathbf{H}$ with the smallest corresponding eigenvalues [27].

We will also encounter cylinder programming problems of the following form:

$$\min_{\mathbf{X}} \quad \mathrm{tr}\mathbf{X}\mathbf{H}\mathbf{X}' \tag{2.33}$$

$$\text{s.t.} \quad \mathbf{X}\mathbf{X}' = \mathbf{I} \tag{2.34}$$

$$\mathbf{X}\mathbf{1} = 0. \tag{2.35}$$

These also reduce to eigenvalue problems. Let $\mathbf{X}$ be $\mathcal{R}^{n\times m}$. Define a $\mathcal{R}^{n-1\times m}$ matrix $\mathbf{Z}$ with rows that span the nullspace of $\mathbf{1}$. So the rows of $\mathbf{Z}$ span the space of solutions to the mean constraint (2.35), and $\mathbf{X}$ must take the form $\mathbf{U}\mathbf{Z}$ for some matrix $\mathbf{U}$. Then we may rewrite the optimization problem as

$$\min_{\mathbf{U}} \quad \mathrm{tr}\mathbf{U}\mathbf{Z}\mathbf{H}\mathbf{Z}'\mathbf{U} \tag{2.36}$$

$$\text{s.t.} \quad \mathbf{U}\mathbf{Z}\mathbf{Z}'\mathbf{U} = \mathbf{I}. \tag{2.37}$$

This problem has the form (2.28), and its optimal $\mathbf{U}$ can be found by the method described at the beginning of this section, and $\mathbf{X}$ can be set to $\mathbf{U}\mathbf{Z}$.

A sparse matrix $\mathbf{Z}$ can be found to make computing $\mathbf{Z}\mathbf{H}\mathbf{Z}'$, $\mathbf{Z}\mathbf{Z}'$, and $\mathbf{U}\mathbf{Z}$ fast. Simply set $\mathbf{Z}$ to a bi-diagonal matrix with 1 on the diagonal, and -1 on the upper diagonal. The product $\mathbf{Z}\mathbf{1}$ is 0 because $\mathbf{Z}$ simply subtracts adjacent entries of 1.

Chapter 3

Semi-supervised Nonlinear Regression with Dynamics

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 $\mathbf{Y} = \{y_t\}_{t=1}^T$. Each sample of the input time series is an M-dimensional column vector, and each sample of the output time series is an N-dimensional column. To illustrate the setup, in a visual tracking application, each x_t represents the pixels of an image, with $M \approx 10^6$, y_t could be the joint angles of the limbs of a person in the scene, with $N \approx 20$, and we seek a transformation from images to joint angles. To learn the transformation, we are given a set of input-output examples $\{x_i, z_i\}_{\mathcal{L}}$ of how to map an input sample $x_i \in \mathcal{R}^M$ to an output $z_i \in \mathcal{R}^N$. The index set $\mathcal{L}$ can refer to samples within $\mathbf{X}$ or outside of $\mathbf{X}$. The learning problem is to find a function $g : \mathcal{R}^M \rightarrow \mathcal{R}^N$ that generalizes these examples, and can be used to map each sample of the input time series $\mathbf{X}$ to the elements of $\mathbf{Y}$.

It is appealing to use a fully-supervised nonlinear regression algorithm to learn the mapping g (see Section 2.3). But for many of the applications I consider in Chapter 4, obtaining adequate performance with nonlinear regression has required supplying so many input-output examples that straightforward temporal interpolation between the examples yields adequate performance as well. 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, in learning g, they ignore the unlabeled portions of $\mathbf{X}$.

Taking advantage of even seemingly trivial additional information about the structure of the problem can significantly improve the quality of the regressor. For example, explicitly enforcing the constraint unlabeled data points must be binary, as in a transductive SVM [92, 18, 37, 11], results in performance gain over RLSC [69], which does not impose *a priori* constraints on missing labels. Such additional information in conjunction with the requirement that g be smooth can not only improve regression from supervised points, but also renders unlabeled points informative, which in turn provides a significant boost in the quality of the regressor. See Figure 3-1 for an illustration.

In the applications I consider, output time series represent the motion of physical

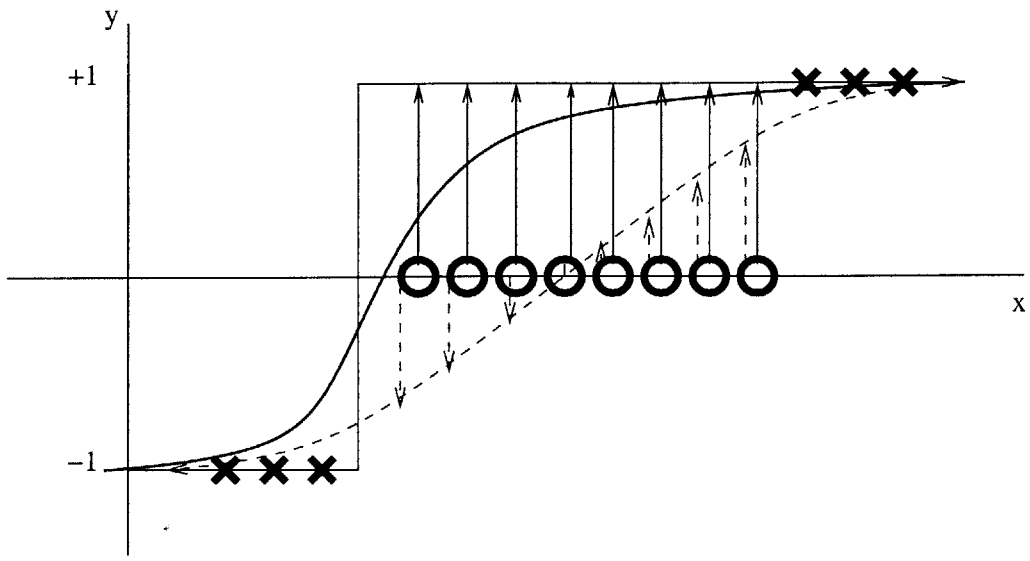


Figure 3-1: Imposing constraints on missing labels renders unsupervised points informative. Crosses represent labeled points with known x and y-values. Circles represent unlabeled points with only known x-values. The black step function represents the true mapping used to generate y-values from x-values. When the regressor is allowed to assign arbitrary y-values to the unsupervised points, supervised points will completely guide the fit and unsupervised points will be assigned whatever y-values make the function the smoothest (dashed blue line). But when y-values are required to be binary, the function may no longer assign arbitrary values to the unlabeled points. These constrained y-values in turn tug the function towards -1 or +1. The resulting function (thick solid blue line) identifies the decision boundary more accurately than the alternative of not constraining the missing labels.

objects. Since variations over time in the output time series represent the displacement of mass over time, and since physical systems have finite energy, these displacements must be smooth over time. This *a priori* knowledge about the output time series is the domain knowledge that allows us to take advantage of unlabeled data. We encode it in the form of a penalty function that favors output time series that exhibit such temporal regularity.

The semi-supervised regression algorithms I present in this chapter couple this penalty function with a functional for nonlinear regression. The interaction between these two functions not only regularizes the quality of the regressor, but also allows us to take advantage of unlabeled examples. This way, a dearth of input-output examples can be compensated for by a plethora of unsupervised points.

Even though the learning algorithm processes the entire training data set (input-output examples as well as unlabeled inputs) in batch, the resulting regressor it learns is memoryless, mapping a single input vector x to a single output vector y without internal state. Once the regressor g is learned, it can be used to map individual input samples to individual output samples memorylessly, or we may insert g in a conditional random field that provides it access to the rest of the sequence, allowing us to transform time series with memory.

3.1 Semi-Supervised Function Learning

We obtain a cost functional for semi-supervised learning by augmenting the cost functional for Tikhonov regularized least squares regressions with a penalty term that favors regular outputs. Recall from Section 2.3 that a nonlinear regressor can be found by minimizing the following cost functional over a space of functions:

$$\min_g \sum_{i \in \mathcal{L}} V(g(x_i), z_i) + \lambda \sum_{d=1}^{N} \|g^d\|_k^2. \tag{3.1}$$

This minimization searches for a regressor $g : \mathcal{R}^M \to \mathcal{R}^N$ that, according to the first term, fits the given example pairs $\{x_i, z_i\}_{\mathcal{L}}$, and according to the second term, is smooth.

To account for missing labels, we augment this cost functional with a penalizer $\mathcal{S} : \mathcal{R}^M \times \ldots \times \mathcal{R}^M \to \mathcal{R}$ over missing labels. The function $\mathcal{S}$ maps a sequence $\mathbf{Y}$ of output vectors to a scalar penalty, favoring a label sequence that adheres to our *a priori* knowledge about the output sequence. It may, for example, favor binary output sequences over others, or as suggested below, sequences that exhibit plausible temporal dynamics. Under this setting, semi-supervised learning becomes a joint optimization over a function g and an output sequence $\mathbf{Y}$. Let $\mathcal{I} = \mathcal{L} \cup \{1 \cdots T\}$ denote the index set of labeled as well as unlabeled data.

The following optimization problem searches for an assignment to missing labels

that is consistent with $\mathcal{S}$, and a smooth function g that fits the labeled data:

$$\min_{g,\mathbf{Y}} \sum_{i=1}^{T} V(g(x_i), y_i) + \lambda_l \sum_{i \in \mathcal{L}} V(g(x_i), z_i) + \lambda_s \mathcal{S}(\mathbf{Y}) + \lambda_k \sum_{d=1}^{N} \|g^d\|_k^2 \tag{3.2}$$

This cost function adds two terms to the one in Equation (3.1). As before, the second term ensures that g fits the given input-output examples, and the term weighted by λ_k favors smoothness of g. The first term ensures that g also fits the estimated missing labels, and the term weighted by λ_s favors regular sequences $\mathbf{Y}$. The scalar λ_l allows points with known labels to have more influence than unlabeled points. The appendix provides a probabilistic interpretation for this cost functional.

Later, we also consider variants of this optimization that constrain some of these terms to be zero instead of penalizing their deviation from zero as is done in this cost function:

$$\min_{g,\mathbf{Y}} \sum_{i=1}^{T} V(g(x_i), y_i) + \lambda_l \sum_{i \in \mathcal{L}} V(g(x_i), z_i) + \lambda_s \mathcal{S}(\mathbf{Y}) + \lambda_k \sum_{d=1}^{N} \|g^d\|_k^2 \tag{3.3}$$

$$\text{s.t. } C_y(\mathbf{Y}) = 0 \tag{3.4}$$

$$C_g(g) = 0. \tag{3.5}$$

Because this is an optimization over g, nested within an optimization over $\mathbf{Y}$, the Representer Theorem still applies, so the optimum g^d still has the representer form:

$$g^d(x) = \sum_{i \in \mathcal{I}} c_i^d k(x, x_i). \tag{3.6}$$

Note that the kernels are centered on the labeled as well as the unlabeled points. This is in contrast to fully-supervised nonlinear regression, where kernels are only centered on labeled points. Placing kernels on unlabeled points allows the function g to have larger support in the input data space without making it overly smooth. Unlike the trivial augmentation of nonlinear regression with Tikhonov regression presented in Section 2.5, the coefficients of this expansion are tied together by $\mathcal{S}$, and so are not trivial.

3.2 Algorithm: Semi-supervised Learning of Time Series Transformation

In our applications, the output time series $\mathbf{Y}$ is known to be smooth over time because it obeys physical dynamics. In many cases, a reasonable model for these dynamics is a linear-Gaussian random walk process, and the negative log likelihood of such a process provides the desired penalty function. The side information provided by this penalizer, along with the training data, provides the necessary information to apply the semi-supervised learning framework of the previous section.

The evolution of many physical systems can be approximated with a linear Gaussian Markov process (see Section 2.6 for an introduction). Such models are defined by a stochastic recurrence equation that produces a sequence of state vectors $\mathbf{S} = \{s_t\}_{t=1\cdots T}$. In certain applications, the dynamics of the output time series are well-known *a priori*, and the full model described in Section 2.6 may be used. But, throughout this thesis, to reduce the number of parameters required to specify the dynamics, we assume that the components of the output time series evolve independently according to a chain of the form

$$s_t = \mathbf{A}s_{t-1} + \omega_t \tag{3.7}$$

$$\mathbf{A} = \begin{bmatrix} 1 & \alpha_v & 0 \\ 0 & 1 & \alpha_a \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.8}$$

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 expect *a priori* that each component of the output time series $\mathbf{Y}$ adheres to such a dynamical model.

We would like to define $\mathcal{S}$ so that it favors output time series $\mathbf{Y}$ that could plausibly have been generated by extracting the position component of the process defined by Equation (3.7). Equation (3.7) defines a zero-mean Gaussian distribution, $p(\mathbf{S})$, over a state sequence $\mathbf{S}$. Clearly, $p(\mathbf{S})$ assigns a higher probability to sequences $\mathbf{S}$ that adhere to this model, and a low probability to those that don't. Letting $s^1 = [s_1^1 \cdots s_T^1]'$ denote the column vector consisting of the first component of each s_t, the joint distribution $p(s^1)$ similarly defines a penalty that favors sequences of position that adhere to these dynamics. Letting $y^d = [y_1^d \cdots y_T^d]$ denote a column vector consisting of the dth component of the elements of the series $\mathbf{Y}$, the negative log of $p(s^1)$ can be used to regularize each y^d. Since $p(\mathbf{S})$ is Gaussian with mean zero, $p(s^1)$ is also a zero-mean Gaussian. Denoting its inverse covariance by Ω_1, the penalty on each component of $\mathbf{Y}$ becomes the quadratic form $\left(y^d\right)' \Omega_1 y^d$. This quadratic form favors sequences y^d that evolve according to the sequence of positions produced by the dynamics model (3.7).

Defining $z^d = [y_1^d \cdots y_T^d]$ as a column vector consisting of the dth component of the labeled outputs, we can substitute this quadratic form into the semi-supervised learning framework summarized by Equation (3.2). Letting V be the quadratic penalty function, we get:

$$\min_{g,\mathbf{Y}} \sum_{d=1}^{N} \sum_{i=1}^{T} (g^d(x_i) - y_i^d)^2 + \lambda_l \sum_{i \in \mathcal{L}} (g^d(x_i) - z_i^d)^2 + \lambda_k \|g^d\|_k^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d. \tag{3.9}$$

The index set $\mathcal{L}$ may overlap with the time index set $1 \cdots T$, and may refer to labeled examples that do not appear in the input time series.

Because $\mathcal{S}$ 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:

$$\min_{g^d, y^d} \sum_{i=1}^{T} (g^d(x_i) - y_i^d)^2 + \lambda_l \sum_{i \in \mathcal{L}} (g^d(x_i) - z_i^d)^2 + \lambda_k \|g^d\|_k^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d. \tag{3.10}$$

Substituting the RBF form (3.6) for the optimal g, this optimization becomes a quadratic problem in terms of the coefficients of the representer form and the missing labels y^d:

$$\min_{c^d, y^d} \|\mathbf{K}_T c^d - y^d\|^2 + \lambda_l \|\mathbf{K}_{\mathcal{L}} c^d - z^d\|^2 + \lambda_k c^{d\prime} \mathbf{K} c^d + \lambda_s \left(y^d\right)' \Omega_1 y^d, \tag{3.11}$$

where, if we denote by $\mathbf{K}$ the kernel matrix corresponding to labeled and unlabeled data, the matrix $\mathbf{K}_T$ is the matrix consisting of the rows of $\mathbf{K}$ that correspond to the missing labels, 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 (3.11) as a quadratic form plus a linear term[1]:

$$\min_{c^d, y^d} \begin{bmatrix} c^d \\ y^d \end{bmatrix}' \begin{bmatrix} \mathbf{K}_T' \mathbf{K}_T + \lambda_k \mathbf{K} + \lambda_l \mathbf{K}_{\mathcal{L}}' \mathbf{K}_{\mathcal{L}} & -\mathbf{K}_T' \\ -\mathbf{K}_T & \mathbf{I} + \lambda_s \Omega_1 \end{bmatrix} \begin{bmatrix} c^d \\ y^d \end{bmatrix} + \begin{bmatrix} -2\lambda_l \mathbf{K}_{\mathcal{L}}' z^d \\ \mathbf{0} \end{bmatrix}' \begin{bmatrix} c^d \\ y^d \end{bmatrix} \tag{3.12}$$

For convenience of notation, we denote by $\mathbf{A}$ the matrix that appears in the quadratic form, and partition it according to $\mathbf{A} = \begin{bmatrix} \mathbf{A}_c & \mathbf{A}_{cy} \\ \mathbf{A}_{cy}' & \mathbf{A}_y \end{bmatrix}$. Taking derivatives of the cost and setting to zero yields $\begin{bmatrix} \mathbf{A}_{cc} & \mathbf{A}_{cy} \\ \mathbf{A}_{cy}' & \mathbf{A}_{yy} \end{bmatrix} \begin{bmatrix} c^d \\ y^d \end{bmatrix} = \begin{bmatrix} \lambda_l \mathbf{K}_{\mathcal{L}}' z^d \\ \mathbf{0} \end{bmatrix}$.

Because the right hand side of this inversion problem has a zero block, we can reduce the complexity of solving this equation by solving for c only. Using the matrix inversion lemma yields a solution for the optimal c^d:

$$c^{d*} = \lambda_l \left(\mathbf{A}_{cc} - \mathbf{A}_{cy} \mathbf{A}_{yy}^{-1} \mathbf{A}_{cy}'\right)^{-1} \mathbf{K}_{\mathcal{L}}' z^d. \tag{3.13}$$

Once the coefficients of g are recovered, labels can be estimated by evaluating g at various x's by plugging c^{d*} into the Representer Form (3.6). Since g is only a function of x, it does not take the history of x's seen so far into account. For many of the experiments in the next chapter, this memoryless mapping provides good results. But g can be enhanced with memory by smoothing or filtering its output over time. For example, once g is learned, to transform a new time series $\mathbf{X}^{new}$, we can transform it with the following smoothing cost functional:

$$\min_{\mathbf{Y}} \sum_{d=1}^{N} \sum_{i=1}^{T} (g^d(x_i^{new}) - y_i^d)^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d. \tag{3.14}$$

This optimization can be efficiently implemented using Rauch-Tung-Striebel smooth-

[1]The presence of the regularizer Ω_1 ensures that disregarding the potentially helpful least-squares form of Equation (3.11) does not significantly harm the conditioning of the problem.

ing [41]. A Kalman filter can also be used to implement the analogous filtering operation. Smoothing and filtering are most helpful when the observation noise is large, or when a sequence over observations are necessary to infer the labels. However, in the applications we have considered, transforming the samples individually with g provides qualitatively similar results to smoothing or filtering. This is because the observation noise is typically small, and because we usually seek a one-to-one mapping between states and observations, so that states can be inferred from a single observation.

The parameters of the algorithm 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 merely scales Λ_ω, it is subsumed by the parameters of Λ_ω. We let Λ_ω be diagonal, which leaves us with a total of seven parameters, plus the parameters of k (which is usually just a scalar). The algorithm usually works with default parameter settings, and its parameters need to be changed only when its results are incorrect. In that case usually only one or two parameters need to be modified in practice. Sections 3.5 and 4.5 provide some intuition and guideline to help tune these. Also, the variations presented in the next two sections require fewer parameters.

3.3 Algorithm Variation: Noise-free Examples

The learning functional in the previous section does not require g to fit the 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 g fit them exactly. This has the advantage of eliminating the free parameter λ_l, which weights the influence of the labeled points.

We can enforce an exact fit 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 (3.2):

$$\min_{g,\mathbf{Y}} \sum_{i=1}^{T} V(g(x_i), y_i) + \lambda_s \mathcal{S}(\mathbf{Y}) + \lambda_k \sum_{d=1}^{N} \|g^d\|_k^2 \tag{3.15}$$

$$\text{s.t. } g(x_i) = z_i, \qquad \forall i \in \mathcal{L} \tag{3.16}$$

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

$$\min_{c^d, y^d} \begin{bmatrix} c^d \\ y^d \end{bmatrix}' \begin{bmatrix} \mathbf{K}_T'\mathbf{K}_T + \lambda_k \mathbf{K} + & -\mathbf{K}_T' \\ -\mathbf{K}_T & \mathbf{I} + \lambda_s \Omega_1 \end{bmatrix} \begin{bmatrix} c^d \\ y^d \end{bmatrix} \tag{3.17}$$

$$\text{s.t. } \mathbf{K}_{\mathcal{L}} c^d = z^d. \tag{3.18}$$

For convenience of notation, label the blocks of the matrix that appears in the quadratic form as $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{cc} & \mathbf{A}_{cy} \\ \mathbf{A}_{cy}' & \mathbf{A}_{yy} \end{bmatrix}$. Solving for the optimal y^d and plugging back in yields a quadratic form in terms of the Schur complement of $\mathbf{A}$, which we denote

by $\mathbf{H} = \mathbf{A}_{cc} - \mathbf{A}_{cy}\mathbf{A}_{yy}^{-1}\mathbf{A}'_{cy}$:

$$\min_{c^d} \quad (c^d)'\mathbf{H}c^d \tag{3.19}$$

$$\text{s.t. } \mathbf{K}_{\mathcal{L}}c^d = z^d. \tag{3.20}$$

I showed in Section 2.7 how to find the minimum of such a linearly constrained quadratic cost function in closed form. Applying that result, we obtain the optimal coefficients:

$$c^{d*} = \mathbf{H}^{-1}\mathbf{K}'_{\mathcal{L}}\left(\mathbf{K}'_{\mathcal{L}}\mathbf{H}^{-1}\mathbf{K}_{\mathcal{L}}\right)^{-1}z^d. \tag{3.21}$$

Compared the optimum (3.13) that accounted for noise in the labels, this solution is computationally somewhat more costly, but requires fewer parameters to evaluate.

3.4 Algorithm Variation: Nearest Neighbors Functions

The RBF representation for g is particularly attractive because it reduces our optimizations a straightforward least squares problems. Other representations, such as MLP, where the output of the function is not linear with respect to the parameters will require a more costly nonlinear optimization procedures. However, there is another class of functions g that yields a simple optimization procedure: nearest neighbor interpolators. In addition we can define optimizations over this class of functions with fewer free parameters.

Suppose we are given a set of input-output pairs $\{x_i, y_i\}$, with scalar outputs y_i. To evaluate a nearest neighbors function g at a given x, we first identify the index set of the nearby neighbors $\mathcal{N}(x, \{x_i\})$ of x, and compute their average labeling:

$$g(x) = \frac{\sum_{i\in\mathcal{N}(x,\{x_i\})} y_i}{\sum_{i\in\mathcal{N}(x,\{x_i\})} 1}, \tag{3.22}$$

These neighbors could be x's k-nearest neighbors in $\{x_i\}$ according to some distance metric, or all x_i that fall within an *epsilon*-ball around x (called the ϵ-ball neighbors of x). Since the $\{x_i\}$ are observed points in the input space, the only parameters of these functions are the labels $\{y_i\}$. In addition, a function that performs nearest neighbors interpolation depends linearly on these labels.

We can search for functions in this family, instead of an RKHS. This function family is inherently smooth because the averaging that occurs in a neighborhood smooths $g(x)$ as a function of x, with the size of the neighborhood acting as an implicit smoothness parameter. Hence, we will not need to explicitly penalize the smoothness of g. To plug a function of this form into (3.2), we first express it in vector form as

$$g(x) = \frac{g'_x}{g'_x e}y = w'_x y, \tag{3.23}$$

where the vector g_x is x's neighborhood indicator in $\{x_i\}$, with its ith element set to

1 if $i \in \mathcal{N}(x, \{x_i\})$, and to zero otherwise. The column vector e consists of all ones. For convenience, we have defined the column vector $w_x = g_x / g_x' e$.

Substituting into (3.9) yields:

$$\min_{y^d} \sum_{d=1}^{N} \sum_{i=1}^{T} \left(w'_{x_i} \begin{bmatrix} y^d \\ z^d \end{bmatrix} - y_i^d\right)^2 + \lambda_l \sum_{i \in \mathcal{L}} \left(w'_{x_i} \begin{bmatrix} y^d \\ z^d \end{bmatrix} - z_i^d\right)^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d. \tag{3.24}$$

Note that we have dropped the RKHS norm penalty and eliminated the optimization over g, since the only parameters of g are $\mathbf{Y}$, over which we are already optimizing.

Rewriting (3.24) in matrix form yields

$$\min_{y^d} \left\| \mathbf{W_X} \begin{bmatrix} y^d \\ z^d \end{bmatrix} - y^d \right\|^2 + \lambda_l \left\| \mathbf{W}_{\mathcal{L}} \begin{bmatrix} y^d \\ z^d \end{bmatrix} - z^d \right\|^2 + \lambda_s \left(y^d\right)' \Omega_{\mathbf{Y}} y^d, \tag{3.25}$$

where the matrices $\mathbf{W_X}$ and $\mathbf{W}_{\mathcal{L}}$ are derived from the adjacency matrix $\mathbf{G}$ of the nearest neighbors graph of $\{x_i\}$. If we normalize each row of $\mathbf{G}$ by the sum of its entries, $\mathbf{W_X}$ contains the rows corresponding to the unlabeled points, and $\mathbf{W}_{\mathcal{L}}$ contains the rows corresponding to the labeled points.

Equation (3.25) is a quadratic cost function in the missing labels, and so can be solved using least squares again. The term $\|\mathbf{W_X} y^d - y^d\| = y^{d'}(\mathbf{I} - \mathbf{W_X})'(\mathbf{I} - \mathbf{W_X}) y^d$ is reminiscent of the term that appears in the LLE algorithm. Section 5.2.2 relates the unsupervised version of this cost functional to the LLE algorithm.

This optimization has seven parameters: λ_l, α_v, α_a, the three variance parameters for the driving noise of the dynamical model, and a scalar parameter that governs the neighborhood size, either ϵ in case of ϵ-neighborhoods or k in case of k-nearest neighbors.

3.5 Intuitive Interpretation

Gaining an informal understanding of these algorithms will be helpful in understanding when they will work, and how to tune their parameters. The optimization (3.2) fits g with Tikhonov regularization to given as well as estimated labels, and simultaneously interpolates the missing labels using $\mathcal{S}$. The interaction between these two operations imputes missing labels where there were no given examples, and improves the imputation with function fitting. This interaction can be better understood by regrouping the terms of (3.2):

Function fitting: The data penalty terms fit g to to given and estimated labels. To see this, rewrite (3.2) as:

$$\min_{\mathbf{Y}} \lambda_s \mathcal{S}(\mathbf{Y}) + \left[\min_{g} \sum_{i=1}^{T} V(g(x_i), y_i) + \lambda_l \sum_{i \in \mathcal{L}} V(g(x_i), z_i) + \lambda_k \sum_{d=1}^{N} \|g^d\|_k^2 \right]. \tag{3.26}$$

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 $g(x_i)$ as observations and $\mathcal{S}$ as a prior. To see this, rewrite (3.2) as:

$$\min_g \lambda_k \sum_{d=1}^{N} \|g^d\|_k^2 + \lambda_l \sum_{i \in \mathcal{L}} V(g(x_i), z_i) + \left[\sum_{i=1}^{T} V(g(x_i), y_i) + \lambda_s \mathcal{S}(\mathbf{Y}) \right]. \quad (3.27)$$

This nested smoothing operation corrects the output of g at unlabeled points. This in turn guiding 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 hallucinated by interpolating them from temporally adjacent known outputs. This effect is starkly illustrated with the *Sensetable* data set in the next chapter.

Due to our choice of dynamics model and smoothness penalty in the cost (3.9), removing labeled points causes the optima of the functional to collapse to zero. This observation will become significant in Chapter 5 when we devise an unsupervised extension of this algorithm. Our choice of the RKHS norm favors functions with a small magnitude, because $\|\alpha g\|_k = \alpha \|g\|_k$. So αg is considered to be smoother than g if the magnitude of α is small. Similarly, the regularizer $\mathcal{S}$ on $\mathbf{Y}$ is quadratic in $\mathbf{Y}$, and favors a label sequence $\alpha\mathbf{Y}$ over $\mathbf{Y}$ if the magnitude of α is small. But because g is required to fit the given input-output examples, it is not allowed to collapse to zero. In turn, this does no allow $\mathbf{Y}$ to collapse to zero.

Chapter 4

Learning to Track from Examples with Semi-supervised Learning

This chapter exhibits various applications of the semi-supervised technique I described in Chapter 3. It demonstrates that nonlinear regression techniques, when augmented with a prior on the temporal dynamics of their output, can solve difficult tracking problems with surprisingly few labeled data points. For example, we can learn to track objects using signal strength measurements from an RFID tag reader with only four labeled data points. We can learn to track the arms of person from a video sequence with only twelve examples. A fully-supervised nonlinear regression algorithm would require significantly more examples to learn these operations.

In the applications I consider here, it is reasonable to require that the output time series fit the specified examples exactly, so I use the semi-supervised learning algorithm of Section 3.3. In these experiments, the RKHS is defined by a Gaussian kernel $k(x_1, x_2) = \exp\left(-\frac{1}{2}\|x_1 - x_2\|^2/\sigma^2\right)$. The bandwidth parameter σ is a free parameter of the algorithm. Section 4.5 provides some guidance in tuning the algorithm's free parameters.

4.1 Synthetic Manifold Learning Problems

I 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.

The manifold is constructed by lifting a random walk on a 2D euclidean patch to $\mathcal{R}^3$ via a diffeomorphism (a diffeomorphism is a map that is differentiable and has a differentiable inverse). A few examples of how to map points in $\mathcal{R}^3$ to their coordinates in $\mathcal{R}^2$ are supplied. See Figure 4-1(left-middle,left-top). The task is to recover the projection function $g : \mathcal{R}^3 \rightarrow \mathcal{R}^2$ to invert this lifting.

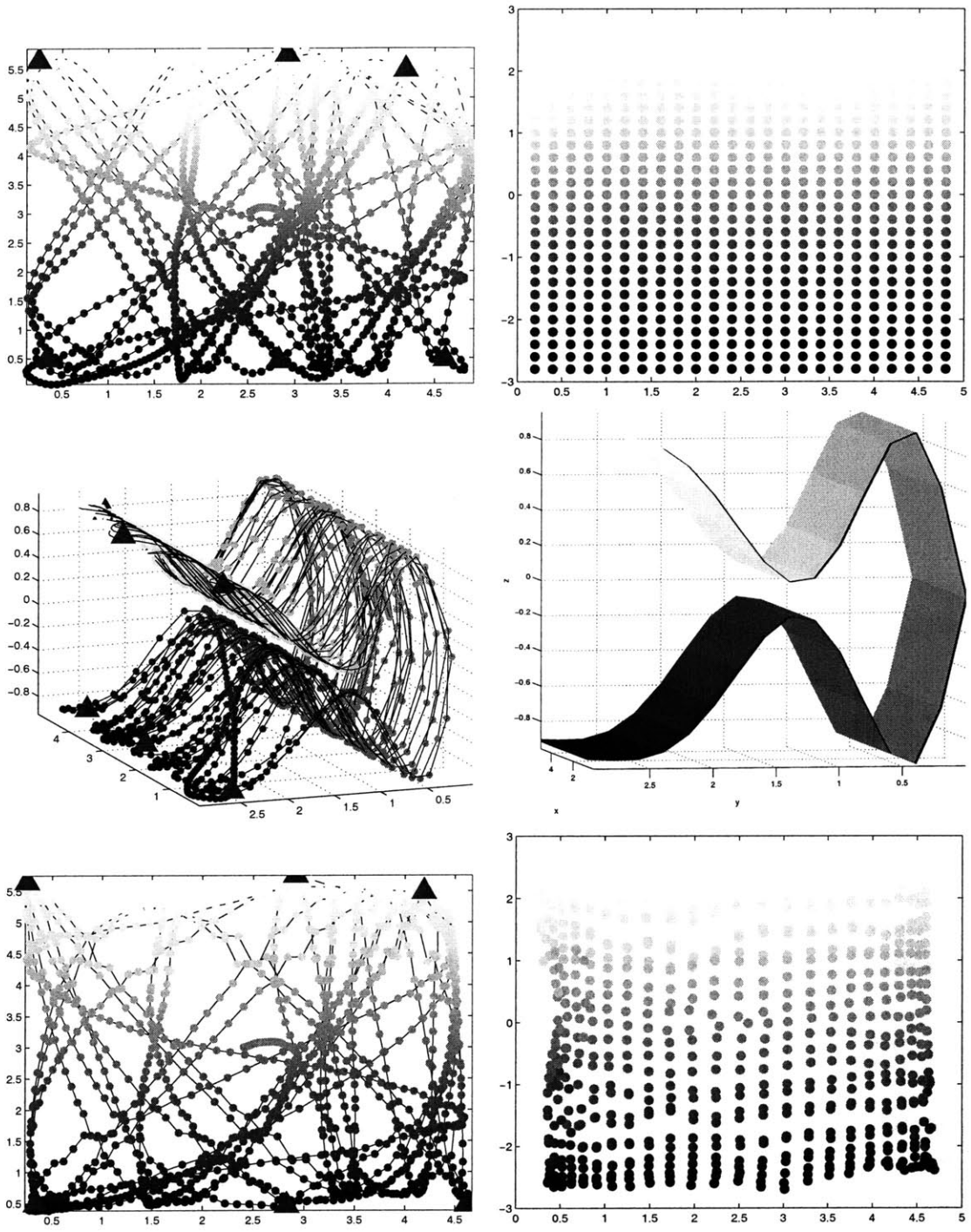


Figure 4-1: (left-top) The true 2D parameter trajectory. Semi-supervised points are marked with big blue triangles. The trajectory has 1500 points. In all these plots, the color of each trajectory point is based on its y-value, with higher intensities corresponding to higher y-values. (left-middle) Embedding of a 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 correctly 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 g. The mapping from 3D to 2D is learned accurately.

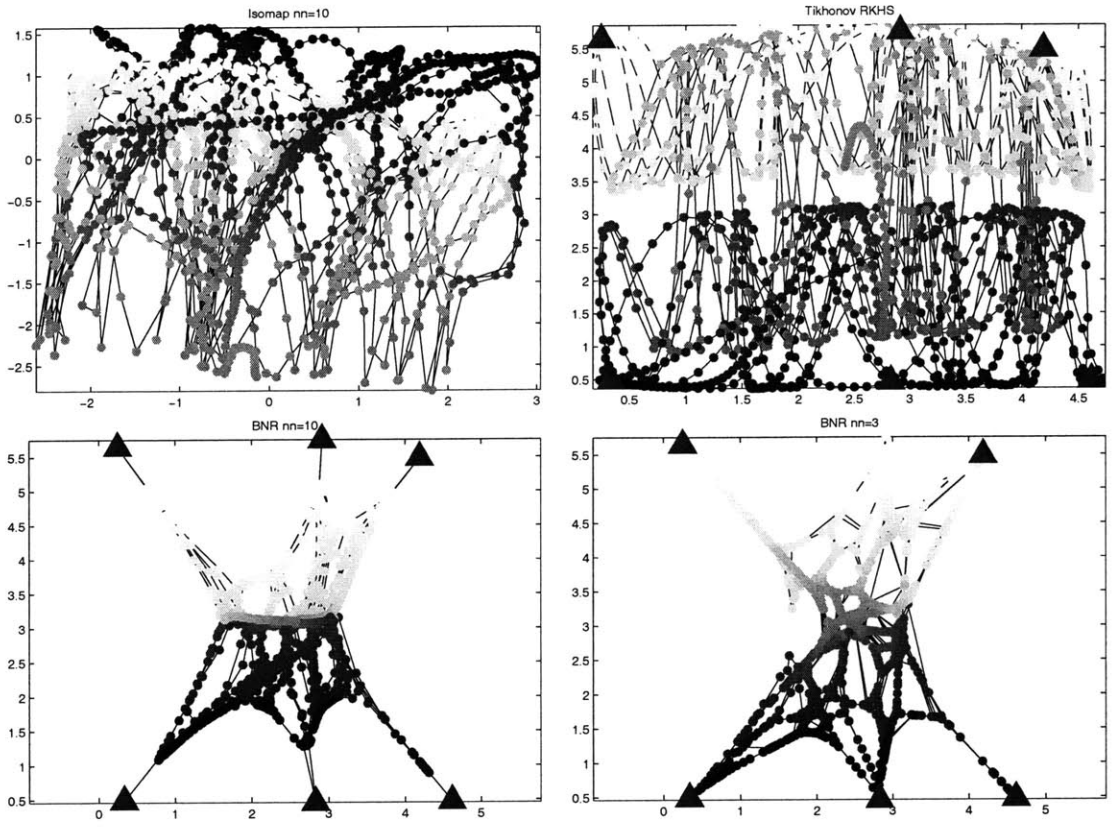


Figure 4-2: (top-left) Isomap's recovered 2D coordinates for the dataset of Figure 4-1(top-middle). Errors in estimating the neighborhood relations at the neck of the manifold cause the projection to fold over itself in the center. The neighborhood size was 10, but smaller neighborhoods produce similar results. (top-right) Without taking advantage of unlabeled points, the the coordinates of unlabeled points cannot be recovered correctly, since only points at the edges of the shape are labeled. (bottom-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. (bottom-right) BNR with neighborhood size of 3 prevents most of the folding, but not all of it. Further, the points are still shrunk to the center, so the low-dimensional values are not recovered accurately.

Recall that as a first step, manifold learning algorithms such as LLE [70], Isomap [86], and Laplacian Eigenmaps [8], and some semi-supervised learning algorithms [7], construct a nearest-neighbor graph based on the proximity of the data points to each other in $\mathcal{R}^3$. They seek to maintain this proximity structure in the low-dimensional representation they recover. In this data set, the neighborhood structure is difficult to estimate from high-dimensional data, because the manifold almost intersects itself, and it is sparsely sampled. As a result, 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. As a result, existing manifold learning algorithms will assign similar coordinates to points that are in fact very far from each other on the manifold. We applied LLE, Laplacian Eigenmaps, and Isomap to the data set of Figure 4-1(left-middle). Isomap produced the result shown in Figure 4-2(top-left). Due to the difficulty in estimating the neighborhood structure near the neck, Isomap creates folds in the projection. Neither LLE nor Laplacian Eigenmaps produced sensible results, projecting the data to a straight line, even at higher sampling rates (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 coordinate trivial. To show this, we also compare against Belkin and Nyogi's graph Laplacian-based semi-supervised regression algorithm [7], which I refer to as BNR (see Section 2.5 for a description). Six points on the boundary of the manifold were labeled with their ground truth low-dimensional coordinates. Figures 4-2(bottom-left,bottom-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. Finally, Figure 4-2(top-right) shows the result of Tikhonov regularization on an RKHS with quadratic loss (the solution of (3.1) applied to the high-dimensional points). This algorithm use only labeled points, and ignores unlabeled data. Because all the labeled points have the same y coordinates, Tikhonov reguarlization cannot generalize the mapping to the rest of the 3D shape.

Taking into account the temporal coherence between data points alleviates these problems. First, folding problems are alleviated because our algorithm does not need to explicitly estimate the neighborhood structure of points based on their proximity in $\mathcal{R}^3$. Instead, it takes advantage of the time ordering of data points. Figure 4-1(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 g on as-yet unseen points. Figure 4-1(right-top and right-middle) shows a lifting of a 2D grid spanning $[0, 5] \times [-3, 3]$ by the same mapping used to generate the training data. Each of these points in $\mathcal{R}^3$ is passed through g to obtain the 2D representation shown in Figure 4-1(right-bottom). These projections fall close to the true 2D location of these samples, implying that g has correctly generalized an inverse for the true lifting.

In comparing the algorithm with Isomap, LLE and Laplacian Eigenmaps, I relied on source code available from the respective authors' web sites. To compute eigenvalues and eigenvectors, I tried both MATLAB's EIGS routine and JDQR [25], a drop-in replacement for EIGS. I used my own implementation of BNR, but relied on the code supplied by the authors to compute the Laplacian.

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

- 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. This renders our algorithm less sensitive to errors in estimates of neighborhoods [6].

- The assumed dynamics model does not need to be very accurate. The the true low-dimensional 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 (3.7) 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 that fact that the examples only showed how to map points whose y coordinate is 2.5 to their low-dimensional coordinate, our semi-supervised learning algorithm learned the low-dimensional coordinate of points with any y-coordinate.

4.2 Learning to Track: Tracking with the *Sensetable*

The *Sensetable* is a hardware platform for tracking the position of radio frequency identification (RFID) tags. 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 4-3. We wish to learn to map these 10 values to the 2D position of the RFID tag. Such a mapping can be recovered by hand through an arduous analysis process that involves building a physical model of the inner-workings of the *Sensetable*, and resorting to trial and error to refine the resulting mappings [61] [1]. Rather than reverse-engineering

[1]The *Sensetable* has a story worth telling. In 2000, James Patten of the Media Lab extracted the core components of the *Sensetable* from a toy developed by a company called Zowie (incidentally, Zowie was a spin-off company of Interval Research Corporation, my employer before I entered graduate school). James and Jason Alonzo, his then-UROP, reverse-engineered the inner-workings of the Zowie hardware. After a few weeks, they figured out how to control the board via RS232 to extract the 10 signal strength measurements from it. Once they had these numbers, they spend a few months trying to convert them to positions. They overlayed the *Sensetable* on a Wacom tablet, and attached the RFID tag to the Wacom pen to obtain ground truth tag locations. James's initial attempt at learning a mapping from positions to the coordinates reported by the the Wacom board by neural nets failed. I briefly had a hand in one of these reverse-engineering sessions, pouring

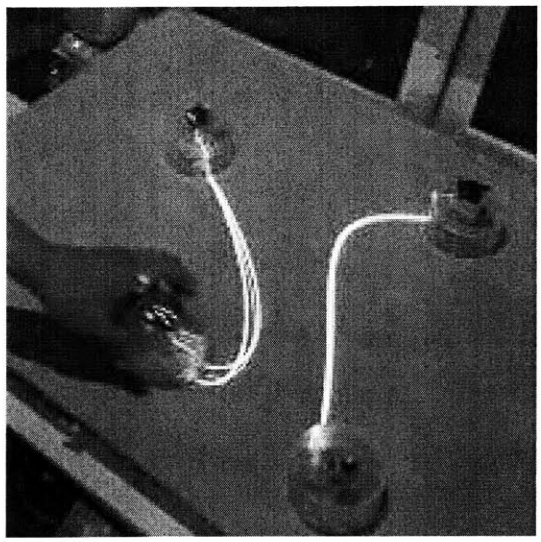


Figure 4-3: A top view of the *Sensetable*, an interactive environment that consists of an RFID tag tracker and a projector for providing user feedback. To track tags, it measures the signal strength between each tag and the antennae embedded in the table. These measurements must then be mapped to the tag's position.

this device by hand, we show that it is possible to recover these mappings semi-automatically, with only 4 labeled examples and some unlabeled data points. This is a challenging task because 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. Of course, this procedure is quite general, and can be applied to a variety of other hardware.

To collect labeled examples, we placed the tag on each of the four corners of the *Sensetable*and recorded the *Sensetable*'s output. 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. The four labeled points, along with the few minutes of recorded data were passed to the semi-supervised learning algorithm to recover the mapping. Figure 4-4(left) shows the ground truth trajectory of the RFID tag, as recovered by the manually reverse-engineered *Sensetable*mappings. The four triangles in the corners of the figure depict

with James over packet sniffs of the board using MATLAB. The only upshot of that sessions was that I had managed to indoctrinate James into using MATLAB as a data analysis tool. James eventually recovered the mapping by trial and error (most likely without ever using MATLAB). A part of James' Master's thesis at the Media Lab involved an interactive setup where this hardware tracked RFID tags as they were moved about on a table, and a projector that overlayed on the table a display that was aware of the location of these tags. He dubbed this setup the *Sensetable*, and developed various applications for it, including a program for visualizing supply chains. In 2003, James and our now-mutual collaborator, Ben Recht, integrated the *Sensetable* into the *Audiopad* (`http://www.jamespatten.com/audiopad/`), an interactive disc jockey system, where the DJ can visualize and browse music clips with two hands by manipulating RFID tags. James and Ben have exhibited the *Audiopad* at various high profile venues around the world, where it has been received with critical acclaim. It was Ben's idea to re-calibrate the *Sensetable* using semi-supervised learning.

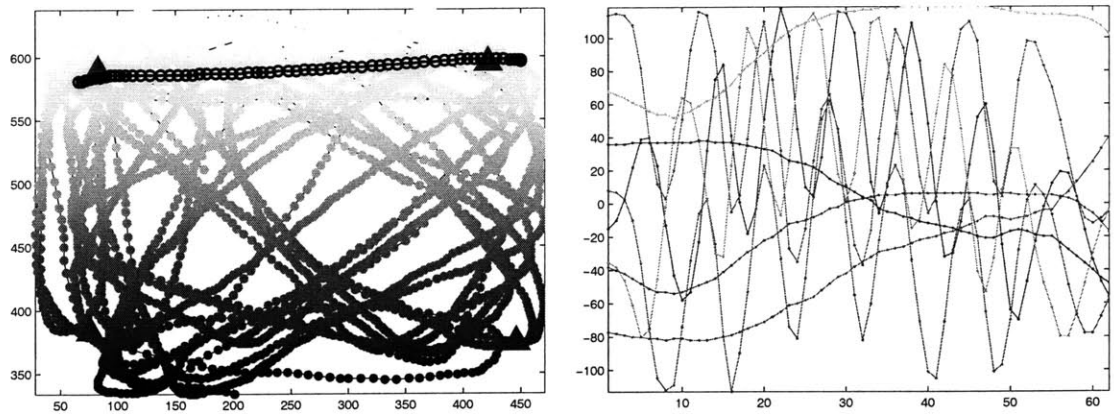


Figure 4-4: (left) The ground truth trajectory of the tag. The tag was moved around smoothly on the surface of the *Sensetable* for about 400 seconds, producing about 3600 samples after downsampling. Triangles indicate the four locations where the true location of the tag was provided to the algorithm. The color of each point is based on its y-value, with higher intensities corresponding to higher y-values. (right) Samples from the output of the *Sensetable* over a six second period, taken over trajectory marked by large circles in the left panel. After downsampling, there are 10 measurements, updating at about 10 Hz.

the location of the labeled examples. The rest of the 2D trajectory was not made available to the algorithm. Figure 4-4(right) shows an example of the output from the *Sensetable*. Contrary to what one might hope, each trace of the output does not have a straightforward one-to-one relationship to a component of the 2D position. Rather, this relationship is smooth but sinusoidal. For example, when the tag is moved in a straight line from left to right, it generates sinusoidal traces similar to those shown in Figure 4-4(right).

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 4-5. Its RMS distance to the ground truth trajectory is about 1.3 cm, though we do not know how accurate the ground truth itself is. Figure 4-5(right) shows the regions that are most prone to errors. The errors are greatest outside the bounding box of the labeled points, but points in the center of the board are recovered very accurately, despite the lack of labeled points there. This phenomenon is discussed in Section 3.5.

Once the mapping from measurements to positions is learned, we can use it to track tags. Individual samples of 10 measurements can be passed to g to recover the corresponding tag position, but because the *Sensetable*'s output is noisy, the results must be filtered. Figure 4-6 shows the output of a few test paths after smoothing (see Equation (3.14) in Section 3.2). The recovered trajectories match the patterns traced by the tag.

The mapping cannot be learned from the four labeled examples alone using Tikhonov regularization, demonstrating that access to unlabeled data and prior knowledge about dynamics is very helpful in real-world applications. See Figure 4-7(left).

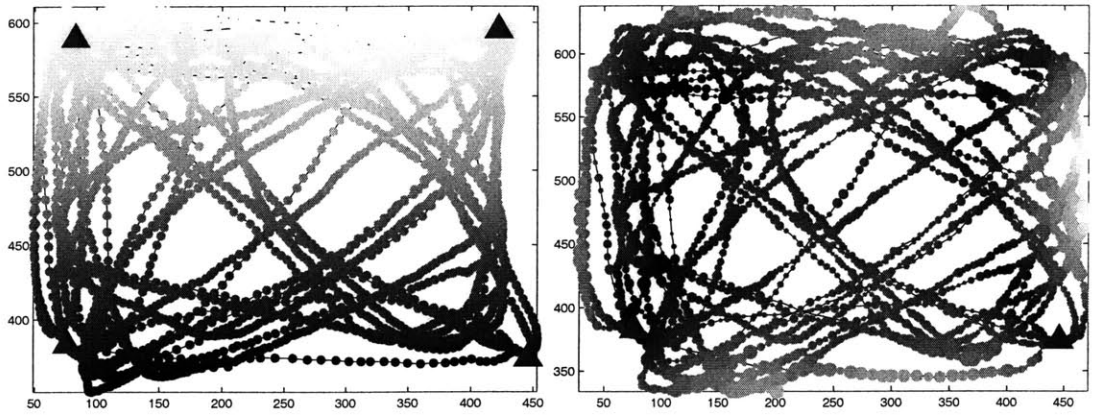


Figure 4-5: (left) The recovered missing labels match the original trajectory depicted in Figure 4-4. (right) Errors in recovering the ground truth trajectory. The ground locations are plotted, with the intensity and size of each circle proportional to the Euclidean distance between a point's true position and its recovered position. The largest errors are outside the bounding box of the labeled data, and points in the center are recovered accurately, despite the lack of labeled points there.

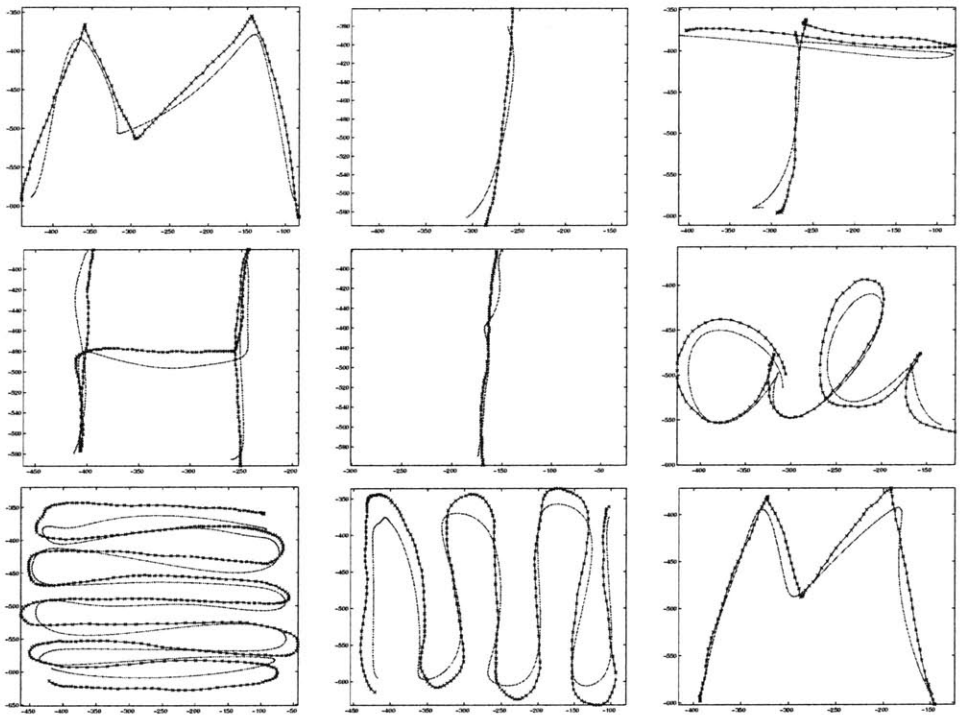


Figure 4-6: Once g is learned, we can use it to track tags. Each panel shows a ground truth trajectory (blue crosses) and the estimated trajectory (red dots). The recovered trajectories match the intended shapes.

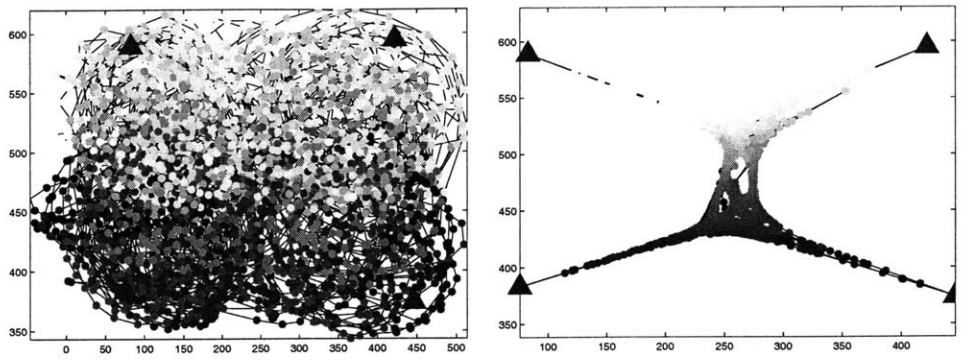


Figure 4-7: (left) Tikhonov regularization with labeled examples only. The trajectory is not recovered. (right) BNR with a neighborhood size of three. There is folding at the bottom of the plot, where black points appear under the red points, and severe shrinking towards the mean.

Figure 4-7(right) shows the trajectory recovered by BNR with its most favorable parameter setting for this data set. As with the synthetic data set, there is severe shrinkage toward the mean of the labeled points, and some folding at the bottom.

The features of the *Sensetable* are present in many other tracking applications, like outdoor localization using GSM signal measurements [57], and indoor localization with 802.11 signal measurements [5, 45]. Current approaches to wireless localization rely on a large corpus of signal strength measurements for a large variety node locations. A context-sensitive table lookup of signal strength measurements then recovers the position of the node. An alternative that requires a less arduous data collection step relies on analytical models that relate position to signal strengths [85], but these models are violated in practice, resulting in poor localization. General Semi-supervised learning algorithms may provide a hybrid solution, where a model is learned from dense measurements of the signal strength throughout the environment, but where only a few of these measurements would need to be labeled with the position of the node. Because radio strength measurements display some of the properties of the measurements reported by the *Sensetable*, I believe it will be possible to apply semi-supervision to the problem of wireless localization.

4.3 Learning to Track: Visual Tracking

With semi-supervised learning, we can learn visual trackers with very few examples. I focus on applications where a user is given a video sequence and asked to annotate all of its frames with a low-dimensional representation of the scene. I demonstrate the algorithm with an interactive tool that allows the user to provide examples by graphically labeling a few key frames of a video sequence. This labeling takes the form of a collection of vectorized drawing primitives, such as splines and polylines. The output representation consists of the control points of these drawing primitives. The semi-supervised learning algorithm recovers the control points for the unlabeled portions 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. This system is reminiscent of rotoscoping tools [3, 2], which allow the user to interactively adjust the output of contour trackers by annotating key frames. The difference is that our algorithm is not limited to tracking contours. Since the 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 need to correspond to contours. For this reason, it is robust to occlusions, though changing backgrounds that are not removed with preprocessing are still detrimental. The tool is demonstrated on three kinds of videos: 1) a video of a rotating synthetic object, where the pose of the rigid object is specified for a few key frames, 2) a lip tracking video where the user specifies the shape of the lips of a subject, and 3) two articulated body tracking experiments where the user specifies positions of the subject's limbs.

The idea of learning to track from examples is not new. Various researchers have applied nonlinear regression to image patches, though they rely on fully-supervised regression algorithms, typically based on the nearest neighbors. For example, Efros et al. used thousands of labeled images of soccer players to recover the articulated pose players in new images [21], El Gammal [23] recovered small rigid deformations using RBFs, and Shaknarovich et al. [76] used a fast nearest-neighbors algorithm to recover the pose of hands. Agarwal and Triggs [1] used a fully-supervised nonlinear regression algorithm similar to the one described in Section 2.3.2 to learn a mapping from features of an image to the pose of the body. When the labeled images are frames of a video sequence, not all of them need to be labeled if we take advatnage of their time ordering. Throughout this chapter, I compare our semi-supervised learning algorithm, which does take advatnage of this ordering, against fully-supervised nonlinear regression to demonstrate this point.

When the input time series is a video sequence, each of its samples x_t represents an image or an image patch. The most straightforward representation of an image or an image patch is the concatenation of its pixel values into a column vector. Thus, a 640×480 gray scale image would be represented as a vector in $\mathcal{R}^{307200}$. All of the results in this thesis use this simple representation, without applying any preprocessing to the images. The functions learned by our algorithm are thus represented using radial basis kernels centered each frame of the video sequence. Other example-based tracking systems preprocess the images to facilitate finding similar images by selecting a relevant patch in the image [21, 23], and optionally extracting silhouettes [76, 28]. This allows them to rely on far fewer examples. In our case, the reduction in the number of required labeled images comes from taking into account the dynamics of the output representation. Though we do not do it here, we could further reduce the number of required examples and make the learned mapping applicable to inputs that are significantly different from the training images by preprocessing the input images.

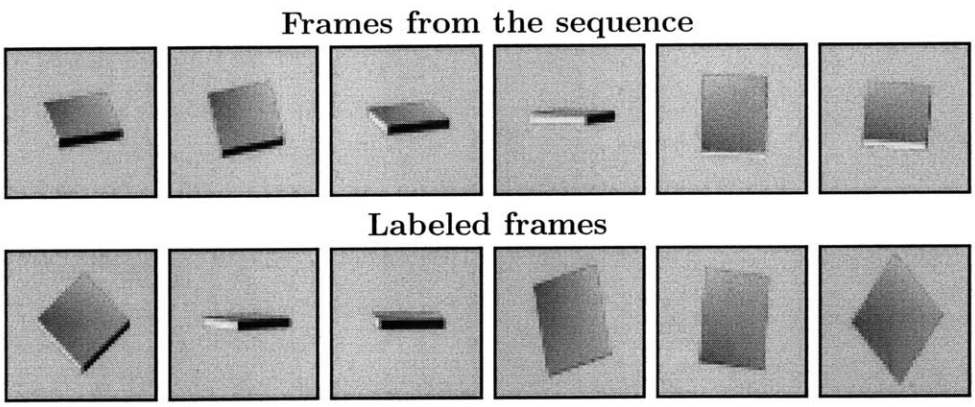


Figure 4-8: (top) A few frames of a synthetically-generated 1500 frame sequence of a rotating cube. (bottom) The six frames labeled with the true rotation of the cube. The rotation for each frame in the sequence was recovered with an average deviation of 4° from ground truth.

4.3.1 Synthetic Images

We quantitatively evaluated the performance of the algorithm on images by running it on a synthetic image sequence where the ground truth was readily available. Figure 4-8 shows frames in a synthetically generated sequence of 50×50 pixel images of a rigidly rotating object. Six images were chosen for supervision by providing the true elevation and azimuth of the object to the algorithm.

The recovered azimuth and elevation of the object in the unlabeled images deviate from the ground truth by an average of 3.5°. We evaluated BNR on the same data set, with the same labeled frames, and obtained average errors of 17° in elevation and 7° in azimuth for the most favorable settings of BNR. To test the learned function g, we generated a video sequence that swept through the range of azimuths and elevations in 4° increments. These images were passed through g to estimate their azimuth and elevation. The mean squared error of the resulting rotations was about 4° in each axis.

4.3.2 Interactive Tracking

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 4-9(top) shows a few frames of this sequence. The contour of lips are represented with cubic splines with four control 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 4-9(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,

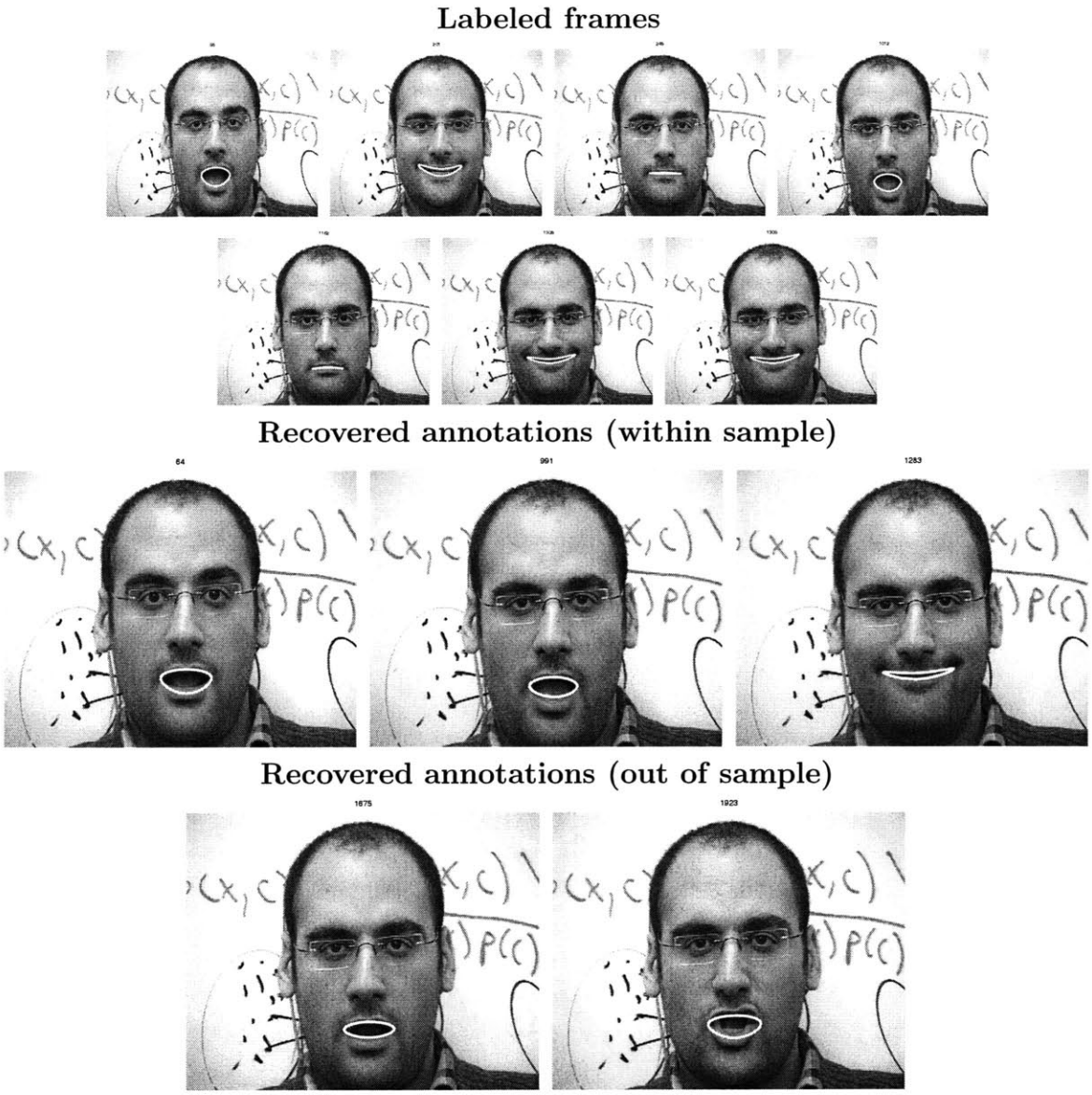


Figure 4-9: (top) The contour of the lips was annotated in 7 frames of a 2000 frame video. The contour is represented using cubic splines, controlled by four control points. The desired output time series is the position of the control points over time. These labeled points and first 1500 frames were 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 at all. 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.

where fully-supervised nonlinear regression does not perform well with the given examples.

Figure 4-10(top) shows some of the labeled images of a 2300 frame sequence of a subject moving his arms. Thirteen frames were manually labeled with line segments denoting the upper and lower arms. The middle portion of Figure 4-10 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 there is no ambiguity between poses. 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 4-10. In contrast to the semi-supervised case, the resulting recovered positions are often wrong.

A researcher who was investigating didactics in expository discourse wanted to recover the position of the hand of subjects in a video database [2]. He provided us with a sequence that posed an infeasible challenge for the color blob-based hand tracker at his disposal. Figure 4-11(top) shows 12 labeled images in this 2000 frame sequence. These labeled frames were sufficient for recovering the outline of the subject's right arm throughout the sequence. See the rest of Figure 4-11. 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 12 examples explicitly annotate these cases.

To numerically assess the quality of our algorithm, we corrected its output by hand and use the amount of correction required as a measure of quality. Table 4.1 shows the magnitude of that correction. The recovered label for every fifth frame in the sequence was adjusted to match the subjective contour of the arm. These corrections were not supplied to the algorithm and serve only to numerically assess the quality of the output from the 12 labeled images. The table shows the average distance (in pixels) between the recovered and corrected location for each corner of the outline. To evaluate magnitude of these distances, the largest distance each corner traveled from its average position is also shown. Our algorithm outperforms temporal linear interpolation between the 13 labeled frames, and fully-supervised regression using Tikhonov regularization with the 13 labeled frames using its best parameter setting. The output of each algorithm was corrected separately to avoid unintentionally favoring any one algorithm.

In these experiments, we sought to recover the dominant factors affecting the appearance of the scene. The algorithm can naturally handle occlusions in these case. There are several ways of handling more pernicious distractors such as background effects and other motions not accounted for by the annotations:

- Preprocess the images. To handle distracting background effects, one could explicitly remove the background as has been done in previous example-based

[2] I thank Jacob Eisenstein for suggesting this experiment and providing me with this data set.

Figure 4-10: (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.

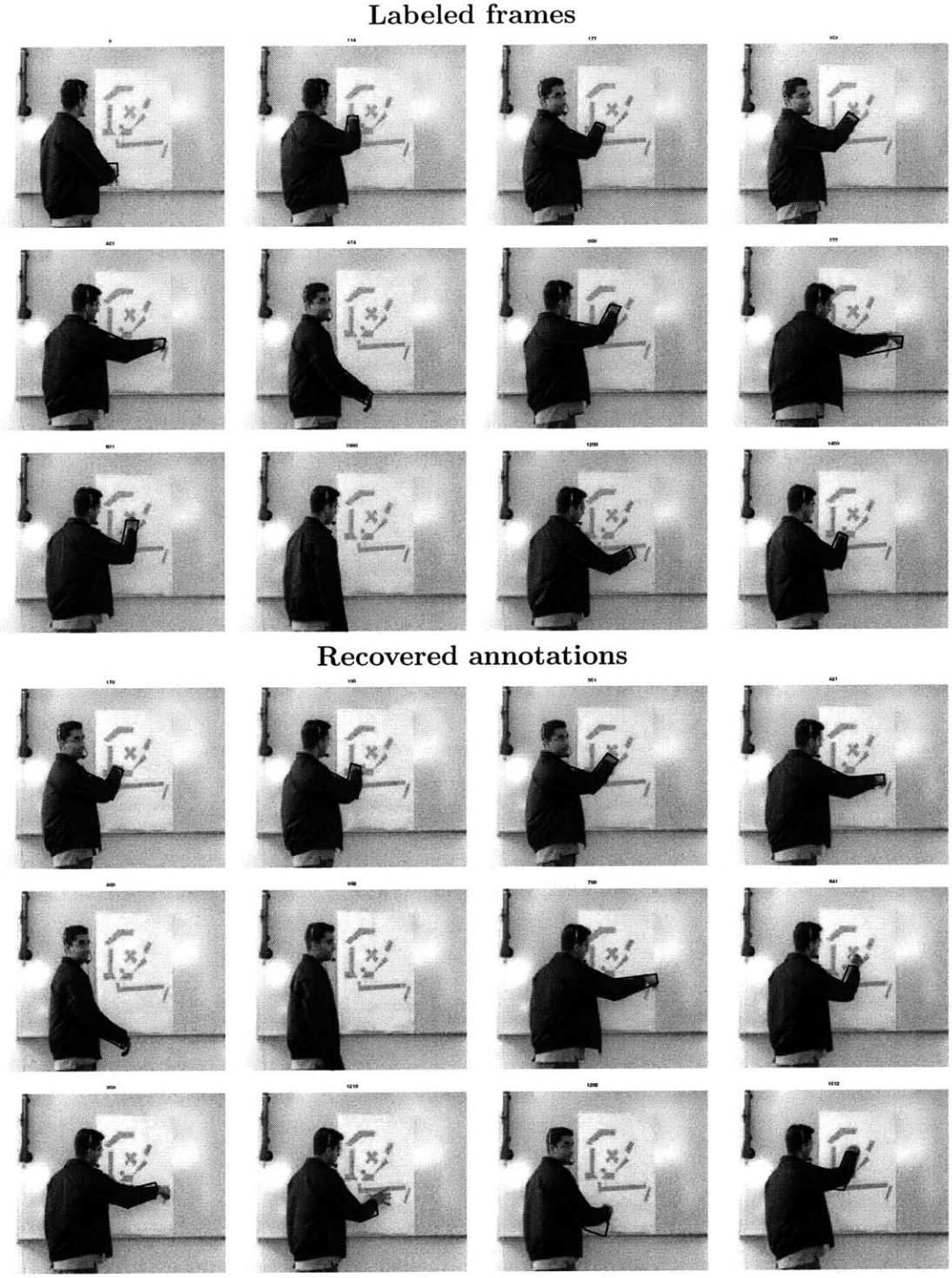


Figure 4-11: (top) 12 of the 13 annotated frames for the arm tracking experiment. The labeling is a closed polygin with six corners. The corners are placed at the shoulder, elbow and hand. Each of these body parts is associated with two corners. To handle the subject turning his head, we annotate a few frames with the subject's head turned towards 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.

	Semi-supervised	Tikhonov regression	Temporal interpolation	Travel
Shoulders	(no correction)	(no correction)	(no correction)	10,10
Elbows	.4, .6	4, 5	8,8	30,34
Hands	3.5, 4.8	10, 11	14,14	56, 59

Table 4.1: Comparison between output and hand-labeled ground truth for the data set of Figure 4-11. The recovered labeling for every fifth frame in the sequence was corrected by hand to assess the quality of the output of our algorithm. The first column gives the average distance (in pixels) between the position of each corner as recovered by our algorithm and its hand-labeled location. Since there are two corners for each body part (shoulder, elbow, and hand), the error for the two corners associated with each body part is reported separately. For comparison, the last column gives the maximum divation between each corner's position and its average position in the sequence. The hand moves the farthest, and its position is recovered with the least accuracy. The second and third columns report the errors for temporally interpolating between the 13 labels, and applying fully-supervised Tikhonov regularization on the 13 labels. These perform worse than our algorithm.

trackers. To be robust to further effects, such as lighting variations, one could operate on silhouettes instead of pixel intensities.

- Annotate frames exhibiting anomalies. In the lip tracking experiments, we annotated two frames where the mouth was open because the pose of the head is different between these two frames (similarly for and the neutral mouth pose in this sequence). In the second arm tracking example, the subject turned his head on several occasions. To account for this head motion, we explicitly annotated the pose of the subject's arm with the subject's head at various orientations. The same strategy can be applied to many other kinds of distractors.

- Automatically select relevant features. As part of learning the function, we could learn the set of pixels that are relevant to the tracking task. This cannot be achieved economically by adapting the weights of g, but can be represented economically by tuning the covariance matrix of the kernel k. We leave the problem of selecting a subset of relevant feature for future work.

4.4 Video Synthesis

The recovered mapping g, which transforms images to a low-dimensional representation, can be inverted to give a convenient way to control video. With this inverse in hand, we could reshape the control points of a spline to manipulate the shape of a mouth, or drag the control points in Figure 4-10 to manipulate articulated bodies. Because g is defined on all of $\mathcal{R}^M$, and not just on an N-dimensional manifold in $\mathcal{R}^M$, it is not strictly one-to-one so we must define its pseudo-inverse. There are two ways

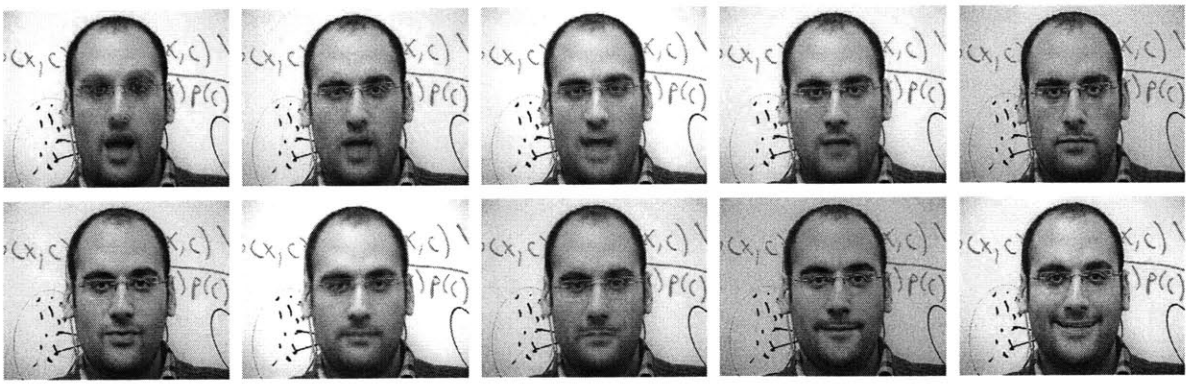


Figure 4-12: Synthesized frames using radial basis functions. The two rows show the output of the pseudo-inverse of g as the mouth is closed by pulling the control points together vertically (top) and as the mouth is widened by pulling the control points apart horizontally (bottom). Because the pseudo-inverse performs interpolation between the frames in the training set, there is some blurring in the output.

to do this, each resulting in a different algorithm for computing the pseudo-inverse of g.

We can define a pseudo-inverse for g as a minimizer of

$$\min_f \int \|x - f(g(x))\|^2 d\mu(x), \tag{4.1}$$

where there minimization is over a space of function $f : \mathcal{R}^N \rightarrow \mathcal{R}^M$, and μ is a measure on the space of images. This cost function favors functions f so that $f(g(x))$ is as close to x as possible. As shown in Section 2.3.2, the cost functional can be approximated by an empirical risk $\sum_i \|x_i - f(g(x_i))\|^2$, and the minimization can be performed with a suitable stabilizer on f to yield the minimization

$$\min_f \sum_i \|x_i - f(g(x_i))\|^2 + \lambda \|f\|_k^2. \tag{4.2}$$

Once g is learned, we have a collection of pairs $\{x_i, g(x_i)\}_{\mathcal{I}}$. Equation (4.2) suggests simply using fully-supervised nonlinear regression on these pairs to learn a mapping f from $g(x_i)$ to x_i.

Figure 4-12 shows how to use this technique to map the control points of lip contours to images of a person whose lips exhibit that shape. The learned pseudo-inverse takes as input a contour, represented as four control points (the same as the output of g in the lip tracking experiment) and returns the pixels of an image. This function is represented using RBFs, so the output interpolates between the examples in the training set (labeled as well as unlabeled). Hence, there is some blurring in some of these images.

Alternatively, we can define the pseudo-inverse of g as

$$f(y) = \arg\min_x \|g(x) - y\|. \tag{4.3}$$

Figure 4-13: Synthesized video using nearest neighbors. (top) The left hand moves straight up while keeping the right hand fixed. (middle) The same motion, but with the hands switched. (bottom) Both arms moving in opposite directions at the same time.

For a given y, this pseudo-inverse returns the input to g that produces y. Whereas the the inverse defined by Equation (4.1) searched for an f so that $f(g(x)) \approx x$, this inverse insures that $g(f(x)) \approx y$. When the training set is large, this optimization admits an efficient approximation. Instead of searching over all images, we may search over the set of training images:

$$f(y) = \arg\min_{i \in \mathcal{I}} \|g(x_i) - y\|. \tag{4.4}$$

This approximation searches for the image whose assigned label is as close as possible to the given label y. This reduces the task of computer an inverse to that of identifying the nearest neighbor of y in the set of given and estimated labels in the training set.

Figure 4-13 shows the result of inverting the mapping learned in Figure 4-10. It shows the result of manipulating the coordinate of the hand, and searching for an image in the training sequence whose estimated hand coordinate is closest to the specified hand coordinate.

These two approaches have complementary properties. The first approach represents the pseudo-inverse with RBFs, and thus interpolates between the training images to generate the desired image. This interpolation can cause the output image to be blurred, but in exchange, the resulting video sequence will be smooth. The second approach avoids blurring by returning a frame from the training set. Although it avoids blurring, the resulting video sequence will be jittery if the training set does not sample the space of images densely.

4.5 Choosing Examples and Tuning Parameters

The parameters and the choice of labeled points supplied to the semi-supervised learning algorithm developed in the previous chapter can be tuned interactively. 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 samples that lie on the boundary 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 4-10, we labeled extreme arm positions. As is shown in Figure 4-5(right), the labels of points in the interior 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 direct path through the interior between two labeled points , and the label of the interior points can be recovered with temporal interpolation. However, if such direct paths through the interior between the boundary examples do not exist, interior points may need to be labeled as well. For example, in Figure 4-11, 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 interior of the other labeled frames. The problem with these frames is that 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 wasn't very likely according to the dynamics model, we had to explicitly label some points along it. The unsupervised learning algorithm presented in the following chapter can be used to identify such boundary points in a first pass, if these points cannot be easily identified by inspection.

Typically, only the parameters of the kernel need to be tuned. The algorithm is insensitive to settings of the other parameters up to several orders of magnitude. To tune the parameters, we initially set the kernel bandwidth parameter so that the kernel matrix $\mathbf{K}$ captures both similarities and discrepancies between pairs of points. 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: Like BNR, the algorithm may have shrinkage to 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. This creates the necessary overshoot at the boundary to correct the shrinkage.

The output is static, except for abrupt jumps at example points: This happens when the kernel bandwidth parameter is too small, forcing $\mathbf{K}$ to become diagonal. Increasing this bandwidth fixes the problem.

Jittery outputs: If the recovered labels are not smooth over time, one can either force g 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. If there are enough labeled examples, we can search for the parameter settings that minimize the leave-one-out cross validation error on these these example points. 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 algorithm. 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 semi-supervised learning algorithm once for each left-out point. Though it is possible to speed it up by devising a recursive version of the learning algorithm that can quickly update its solution when new labeled examples are added or removed, the experiments above show that applying the rules of thumb above is sufficient for obtaining adequate performance since the output of the algorithm is not strongly influenced by the settings of the parameters.

Such fine-tuning is not usually necessary because the output of the algorithm does not vary much with the parameter settings. Figure 4-14 shows the performance of the algorithm on the expository discourse sequence of Figure 4-11 as the kernel width of the kernel 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 4-11 and Table 4.1. The figure of merit is the average distance to the corrected sequence used in the evaluation of Table 4.1. Wider kernels product better results, until numerical problems are encountered. The algorithm produces the same result over a large range of settings for this parameter. When the kernel bandwidth becomes very small, $\mathbf{K}$ becomes diagonal, and the algorithm simply temporally smooths between the labeled data. The algorithm also reports the same result for a large range of settings for λ_k, which governs the smoothness of g. The algorithm is also resilient to a wide range of settings for the dynamics 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 4-11 and Table 4.1. All the trajectories recovered with these parameter settings had similar performance, with the average location error ranging from 3.528 to 3.53 pixels for one corner associated with the hand, 4.77 to 4.78 pixels for the other corner associated with the hand. The algorithm is similarly resilient to different settings of the driving noise. We varied the diagonal elements of Λ_ω in a similar fashion, restricting all three

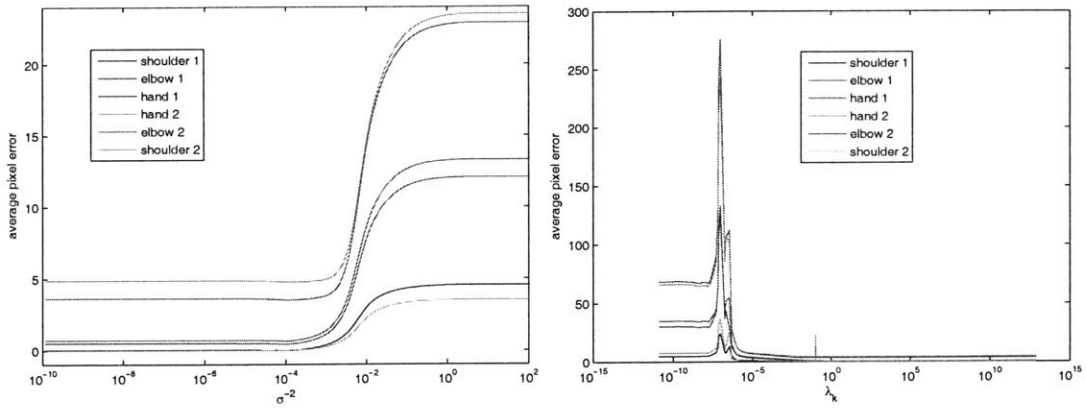


Figure 4-14: (left) Average error in the position of each recovered corner in the data set of Figure 4-11 as the kernel width parameter is varied over several orders of magnitude. The parameter controls $k(x, x') = \exp\left(-\frac{1}{2\sigma^2}\|x - x'\|^2\right)$. (right) Performance as the weight λ_k, which favors the smoothness of g, is varied. The algorithm has the same performance over a wide range of settings for these two parameters.

parameters to lie between 10^{-7} and 10^{6}. All the trajectories recovered with these parameter settings had similar performance, with the average location error ranging from 3.528 to 3.53 pixels for one corner associated with the hand, 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 4-11 and Table 4.1. The parameters of the algorithm were fixed. Figure 4-15 shows the accuracy with which the algorithm recovered one of the corners corresponding to the hand 200 of these subsets. 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.

We have used the guidelines of this section in selecting the parameters and labeled points for all of the experiments in this chapter.

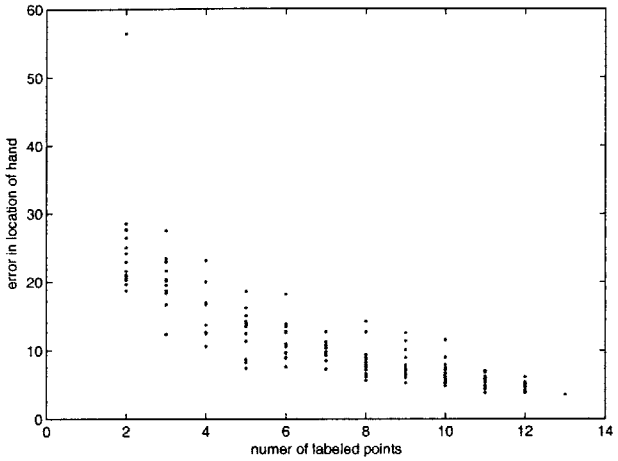


Figure 4-15: Average error in the position of one of the corners corresponding to the hand, as a function of the number of labeled examples used. Labeled examples were chosen randomly from a fixed set of 13 labeled examples. Reducing the number of labels reduces accuracy. Also, the choice of labels has a strong influence on the performance, as demonstrated by the vertical spread of each column.

Chapter 5

Uncovering Intrinsic Dynamical Processes without Labeled Examples

In the previous chapter, the combination of labeled data points and a prior that enforced temporal coherency on the output allowed us to learn a transformation from an input time series to a desired smooth time series. When this desired time series represents a dynamical process that underlies the input time series, labeled points are not necessary. In these cases, labeled points serve mainly to fix a coordinate system.

In this chapter, I devise an unsupervised algorithm based on the learning framework of Chapter 3. This unsupervised learning algorithm is a fast way to uncover a low-dimensional dynamical process underlying a high-dimensional time series, and the inverse of the observation function from a hidden markov chain. Like the algorithm of Chapter 3, this algorithm favors outputs that exhibit temporal dynamics, but in contrast to the algorithm of Chapter 3, it requires no labeled input. Instead, it replaces labeled points with aggregate constraints on the output time series, and searches for a mapping that transforms observations to low-dimensional vectors that evolve according to the specified temporal dynamics. These low-dimensional coordinates are often recovered up to a rotation and scaling of the coordinates of the true underlying dynamical process. When applied to the data sets in this chapter, other learning algorithms that not take advantage of temporal coherence either completely fail to recover the intrinsic process, or recover it with severe distortion.

The unsupervised learning algorithm in this chapter illuminates the relationship between its semi-supervised counterpart described in Chapter 3 and various unsupervised learning algorithms. I show that by removing terms in the cost function we use for unsupervised learning, we obtain a cost functional whose optimum coincides to the output of Kernel PCA on the unlabeled data set. By adopting a functional representation based on nearest neighbors, and again removing some terms from our cost functional, we obtain the embedding step of LLE. Thus, the semi-supervised learning algorithms of the previous chapters are a supervised version of KPCA and LLE's embedding step that take temporal coherence into account. Our algorithm's cost functional is also very similar to a cost functional for nonlinear system identifi-

cation. We show that our unsupervised learning algorithm approximately finds the inverse of the observation function in a hidden Markov chain when the dynamics of the chain are known and the observation function is invertible.

In addition to demonstrating the importance of dynamics in unsupervised learning, our algorithm has applications in many domains. Existing nonlinear identification techniques [26, 90] are computationally expensive and susceptible to local minima, and do not scale well with the dimensionality of the observations. By contrast, our unsupervised learning algorithm is fast, does not get stuck in local minima, and because observations are used only through pairwise kernel evaluations, does not grow in complexity with the dimension of the observed data, except to compute pairwise similarities. This makes it particularly well suited for applications involving large, high-dimensional data sets, such as video sequences or measurements from large-scale sensor networks. Given a video sequence of a rotating object, we can recover the parameterization of its pose up to a scale and rotation, with no training data. This points to way to learning how to track without examples. Using our algorithm as a practical nonlinear system identification method, we show how to track a target moving in a large field of sensors when it is not known how the sensors map the target's position to the measurements they provide. The algorithm uses only the raw output of the sensors to recover the position of the target. To demonstrate this idea on a real-world example, we apply it to the *Sensetable* data set of the previous chapter, and recover, up to a scale, the mapping from its measurements to the position of an RFID tag.

5.1 Algorithm: Unsupervised Recovery of Intrinsic Dynamical Processes

As explained in Section 3.5, simply removing observations from the quadratic cost function (3.9) does not result in an unsupervised learning algorithm, because the optima of the resulting functional collapse to zero. To apply the learning framework of Equation (3.3) to unsupervised situations, we replace terms involving labeled data with constraints that capture *a priori* knowledge about the aggregate statistics of the intrinsic process, and prevent it from collapsing to zero, to obtain the following optimization:

$$\min_{g,\mathbf{Y}} \sum_{i=1}^{T} \|g(x_i) - y_i\|^2 + \lambda_s \mathcal{S}(\mathbf{Y}) + \lambda_k \sum_{d=1}^{N} \|g^d\|_k^2 \tag{5.1}$$

$$\text{s.t. } \frac{1}{T}\sum_{t=1}^{T} y_t y_t' = \mathbf{I} \tag{5.2}$$

$$\frac{1}{T}\sum_{t=1}^{T} y_t = 0 \tag{5.3}$$

The first term ensures that g fits the estimated missing labels. The term weighted by λ_s favors regular sequences $\mathbf{Y}$, and the term weighted by λ_k favors smoothness of g. The first constraint is common in manifold learning algorithms like LLE, Laplacian Eigenmaps, and even Isomap (since it uses MDS). It ensures that each dimension of $\mathbf{Y}$ has unit variance and is orthogonal to all other directions. This captures a belief that the coordinates of the intrinsic process consist of independently evolving Markov chains with known dynamics. The second constraint sets the origin of the coordinate system to the mean of the output samples. It prevents the possibility of $\mathbf{Y}$ collapsing into a thin shape far from the origin, as we have observed in the output of LLE (see Chapter 4 and Section 5.5).

As in Section 3.2, applying the Representer theorem, adopting the penalty function $\mathcal{S}$ for linear Gaussian dynamics, and vectorizing, the optimization becomes:

$$\min_{\mathbf{C},\mathbf{Y}} \sum_{d=1}^{N} \|\mathbf{K}c^d - y^d\|^2 + \lambda_k c^{d\prime}\mathbf{K}c^d + \lambda_s \left(y^d\right)' \Omega_1 y^d \tag{5.4}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I} \tag{5.5}$$

$$\mathbf{Y}\mathbf{1} = 0, \tag{5.6}$$

where the kernel matrix $\mathbf{K}$ has $k(x_s, x_t)$ in its stth entry, and $\mathbf{C}$ are the coefficients of the RBF representation for $g(y) = \sum_{t=1}^{T} c_t k(x, x_t)$. The column vectors c^d and y^d collect the dth component of the vectors c_t and y_t respectively.

Rewriting Equation (5.4) in matrix form, we get

$$\min_{\mathbf{C},\mathbf{Y}} \sum_{d=1}^{N} \begin{bmatrix} c^d \\ y^d \end{bmatrix}' \begin{bmatrix} \mathbf{K}^2 + \lambda_k \mathbf{K} & -\mathbf{K} \\ -\mathbf{K} & \mathbf{I} + \lambda_s \Omega_1 \end{bmatrix} \begin{bmatrix} c^d \\ y^d \end{bmatrix} \tag{5.7}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I} \tag{5.8}$$

$$\mathbf{Y}\mathbf{1} = 0. \tag{5.9}$$

This cost is in the form of Equation (2.31), whose solution reduces to an eigenvalue problem. We can reduce the size of the problem before applying the result of Section 2.7 by eliminating $\mathbf{C}$. The optimal coefficients can be found by least squares as a function of $\mathbf{Y}$:

$$\mathbf{C}^* = \mathbf{Y}(\mathbf{K} + \lambda_k \mathbf{I})^{-1} \tag{5.10}$$

Plugging this value back into the cost function yields a minimization only over $\mathbf{Y}$:

$$\min_{\mathbf{Y}} \sum_{d=1}^{N} \left(y^d\right)' \left(\mathbf{I} + \lambda_s \Omega_1 - \mathbf{K}(K + \lambda_k \mathbf{I})^{-1}\right) y^d \tag{5.11}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I} \tag{5.12}$$

$$\mathbf{Y}\mathbf{1} = 0. \tag{5.13}$$

The optimal $\mathbf{Y}$ can be found by applying the result of Section 2.7:

$$\mathbf{Z}\left(\mathbf{I} + \lambda_s \Omega_1 - \mathbf{K}(K + \lambda_k \mathbf{I})^{-1}\right)\mathbf{Z} = \mathbf{U}\mathbf{D}\mathbf{U}' \tag{5.14}$$

$$\mathbf{Y}^* = \mathbf{U}_{1\cdots d}\mathbf{Z} \tag{5.15}$$

$$\mathbf{C}^* = \mathbf{Y}^*(\mathbf{K} + \lambda_k \mathbf{I})^{-1} \tag{5.16}$$

The $\mathbf{U}\mathbf{D}\mathbf{U}'$ is the eigen decomposition of the matrix on the left of Equation (5.14), where as explained in Section 2.7 The columns of $\mathbf{Z}$ span the null space of $\mathbf{1}$, so they span the space of solutions of the mean constraint $\mathbf{Y}\mathbf{1} = 0$. The eigenvalue problem (5.14) is of size $(dT - 1) \times (dT - 1)$, and can be solved quickly since we only require the top d eigenvectors by a variety of eigenvalue solvers such as the power, Lancosz, or Nÿstrom methods [27].

5.2 Relationship to Manifold Learning

The algorithm of the previous section is closely related to manifold learning and dimensionality reduction methods such as LLE [70] and Kernel PCA (KPCA) [74]. A variant of KPCA [74] is obtained by dropping some of the terms in the cost functional (5.1). By further adopting a nearest-neighbors form for the mapping instead of the RBF form, we obtain an algorithm that is similar to LLE. The modifications of the algorithm of Chapter 3 to an unsupervised algorithm, and the modification of the unsupervised algorithm to manifold learning sheds light on the effect of adding labeled points and dynamics to unsupervised manifold learning algorithms. As we will see in Section 5.5, incorporating knowledge about dynamics is often sufficient for obtaining coordinates on the manifold that capture the physical process that underlies the high-dimensional data.

5.2.1 Relationship to Kernel PCA

The unsupervised algorithm of the previous section is a variant of Kernel PCA. By removing the dynamics prior $\mathcal{S}$ and the zero-mean constraint on the output, we obtain a an algorithm that, like KPCA [74], recovers the eigenvectors of the kernel matrix. This derivation also is a new insight on KPCA, showing that it finds a non-linear function that projects high-dimensional points to uncorrelated low-dimensional representations.

To see this, set λ_s to zero in the learning cost functional (5.1) to remove the influence of the dynamics prior, and remove the zero-mean constraint to obtain the following optimization:

$$\min_{g,\mathbf{Y}} \sum_{t=1}^{T} \|g(x_i) - y_i\|^2 + \lambda \sum_{d=1}^{N} \|g^d\|_k^2 \tag{5.17}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}^\top = \mathbf{I}. \tag{5.18}$$

Applying the representer theorem, writing in matrix form and solving for $\mathbf{C}$, as in the previous section, results in this optimization problem:

$$\min_{\mathbf{Y}} \sum_{d=1}^{N} \left(y^d\right)' \mathbf{K}(K + \lambda_k \mathbf{I})^{-1} y^d \tag{5.19}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I}. \tag{5.20}$$

Factoring $\mathbf{K}$ as $\mathbf{K} = \mathbf{U}\mathbf{S}\mathbf{U}'$, with $\mathbf{U}'\mathbf{U} = \mathbf{I}$, and $\mathbf{S}$ diagonal, this becomes a Rayleigh quotient:

$$\min_{\mathbf{Y}} \sum_{d=1}^{N} \left(y^d\right)' \mathbf{U} \left(\mathbf{S}(\mathbf{S} + \lambda_k \mathbf{I})^{-1}\right) \mathbf{U}' y^d \tag{5.21}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I}. \tag{5.22}$$

The optimum of this problem is achieved by assigning the N largest eigenvectors of $\mathbf{K}$ to $\mathbf{Y}$, so that $\mathbf{Y}^* = \mathbf{U}'_{1\ldots N}$. Accordingly, the optimal coefficients are given by Equation (5.10) to be $\mathbf{C}^* = (\mathbf{S}_{1\ldots N} + \lambda \mathbf{I})^{-1}\mathbf{U}'_{1\ldots N}$.

The original KPCA algorithm as derived in [74], on the other hand, learns a function whose coefficients are given by $\mathbf{C}_{kpca} = \mathbf{U}'_{1\ldots N}$. This discrepancy between $\mathbf{C}_{kpca}$ and $\mathbf{C}^*$ only influences the scale of the output coordinate system, so that $g^*(x) = (\mathbf{S}_{1\ldots N} + \lambda \mathbf{I})^{-1} g_{kpca}(x)$. KPCA is usually described as performing linear PCA in a high-dimensional feature space in which the inner product $< \cdot, \cdot >$ is evaluated using the kernel $k(\cdot, \cdot)$. This interpretation does not provide any insight about its recovered low-dimensional coordinates, because principal components in feature space need not correspond to sensible directions of variation in low-dimensional space. The Gaussian kernel maps points to the surface of a high-dimensional sphere [1], so the subsequent application of linear PCA on points that lie on this curved surface is not well-motivated. Bengio et al. [10] showed that KPCA searches for a sequence of functions whose empirical outer-product best reconstruct the kernel matrix $\mathbf{K}$.

An analysis similar to the above shows that a simpler derivation of KPCA can

[1] This is easy to see because every point in feature space has norm $< x, x >= k(x, x) = 1$. The inner product between every pair of point is positive because $k(x, x') \geq 0$, so that the angle between every point on the surface of the sphere is acute. Hence, all points in the feature space lie on the same orthant of the hypersphere.

be obtained by solving the optimization $\min_g \sum_{d=1}^{N} \|g^d\|_k^2$ subject to the constraint $\sum_{t=1}^{T} g(x_t)g(x_t)' = \mathbf{I}$. The resulting low-dimensional coordinates $y_t = g(x_t)$ are exactly the coordinates recovered by KPCA. Whereas in standard PCA, g is forced to be linear, KPCA allows for nonlinear projections.

We have shown that our algorithm extends KPCA to time series data by placing additional priors on the hidden sequence $\mathbf{Y}$. Based on this interpretation, we also provided a simple derivation of KPCA as learning a nonlinear function that decorrelates high-dimensional observations.

5.2.2 LLE

Adopting the nearest-neighbors function representation of Section 3.4 and ignoring temporal coherency reduces our unsupervised learning algorithm to an algorithm that is very similar to the embedding step of the Locally Linear Embedding (LLE) algorithm [70].

To find a low-dimensional representation for a high-dimensional data set, LLE first captures the local neighborhood relationship between high-dimensional points. It does this by tuning the entries of an adjacency matrix $\mathbf{W}$ that reconstructs each observed high-dimensional point using its neighbors. Since reconstruction from neighbors can be expressed as a linear operation, the reconstruction matrix can can be found by minimizing reconstruction error $\|\mathbf{XW} - \mathbf{X}\|_F$ over $\mathbf{W}$ using least squares. The low-dimensional coordinates $\mathbf{Y}$ are assumed to preserve this neighborhood structure. Once $\mathbf{W}$ is learned, LLE embeds the high-dimensional data by searching for low-dimensional coordinates that are best reconstructed using these weights. To find these, LLE solves

$$\min_{\mathbf{Y}} \sum_{d=1}^{N} \|\mathbf{W}y^d - y^d\|^2 \tag{5.23}$$

$$\text{s.t. } \mathbf{YY}' = \mathbf{I} \tag{5.24}$$

The constraint ensures the the solution does not trivially collapse to $\mathbf{Y} = 0$. Our algortihm performs this embedding step if we adopt a nearest neighbors form for the nonlinear mapping.

As in Section 3.4, let g take the nearest neighbors form. Each component of g can be written as $g^d(x) = w_x' y^d$, with w_x a weighted indicator vector for the neighbors of x. Applying g^d to each high-dimensional point and stacking the results vertically, results in a linear relationship between g^d and the output sequence:

$$\begin{bmatrix} g^d(x_1) \\ \cdots \\ g^d(x_T) \end{bmatrix} = \begin{bmatrix} w_{x_1}' \\ \cdots \\ w_{x_T}' \end{bmatrix} y^d = \mathbf{W}y^d. \tag{5.25}$$

Substituting this form into (5.1), with $\mathcal{S}$ enforcing temporal coherency, and ignoring the smoothness penalty on g (since the neighborhood size already serves as a

smoothness penalty), we get

$$\min_{\mathbf{Y}} \sum_{d=1}^{N} \sum_{i=1}^{T} \|\mathbf{W} y^d - y^d\|^2 + \lambda_s \left(y^d\right)' \Omega_1 y^d \tag{5.26}$$

$$\text{s.t. } \frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I} \tag{5.27}$$

$$\mathbf{Y}\mathbf{1} = 0. \tag{5.28}$$

Compared to the LLE embedding step, this cost function adds a zero-mean constraint $\mathbf{Y}$ to fix the coordinate axis, and a penalty to favor temporal smoothness. We have chosen $\mathbf{W}$ to be the re-weighted adjacency matrix of the neighborhood graph in the input space, but it could be trained, as it is in LLE, for optimal linear reconstruction in input space. We have shown that our unsupervised algorithm, and therefore the semi-supervised algorithm of Chapter 3, are versions of LLE's embedding step that are enhanced with a prior on dynamics and use RBFs instead of nearest neighbor functions to represent the mapping between high-dimensional inputs and low-dimensional outputs.

5.3 Relationship to System Identification

The unsupervised learning algorithm of this chapter approximately estimates the parameters of a generative model for the observations, even though we derived it as searching for a projection from observations to low-dimensional trajectories. Estimating the parameters of a generative model allows us to perform a variety of useful operations such as predicting, smoothing or filtering the observed sequence. Our algorithm finds the inverse of the observation function in a hidden Markov chain whose latent states evolve according to known linear-Gaussian dynamics. The observations correspond to the samples of the the input time series $\mathbf{X}$, and the states are the samples of the output time series $\mathbf{Y}$. A nonlinear observation function f, whose inverse the algorithm estimates, maps latent states to observations. Exact parameter estimation in such a generative models is computationally expensive. Current approximate techniques [26, 90] are susceptible to local minima, and do not scale to very high-dimensional data sets such as video sequences and measurements from sensor networks. Rather than applying approximate inference methods to the exact estimation problem, by searching for the inverse of the observation function instead of the function itself, our approach modifies the generative model slightly and estimates the parameters in this approximate model exactly. When applied to very high-dimensional data sets, this allows our algorithm to overcome the significant computational and storage burdens of current nonlinear system identification techniques.

Nonlinear system identification is the process of estimating the parameters of a nonlinear dynamical system given only observations. Consider the following genera-

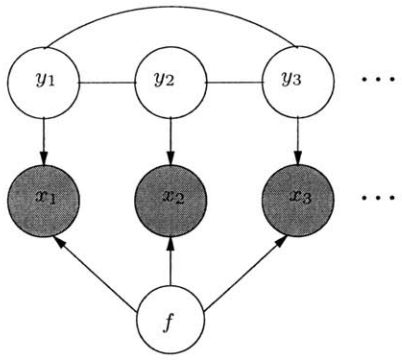


Figure 5-1: A generative model for time series. Each state y_t is an underlying representations of the observed samples x_t. The observations are obtained by applying the observation function f to y_t and corrupting the result with noise.

tive model for the observed time series $\mathbf{X}$:

$$\mathbf{Y} \sim p(\mathbf{Y}) \tag{5.29}$$

$$x_t = f(y_t) + \nu_t, \tag{5.30}$$

where $f : \mathcal{R}^N \rightarrow \mathcal{R}^M$ is a nonlinear observation function, and the ν_t are iid observation noise. We will assume that the dynamics of the latent process $\mathbf{Y}$ is known, and is captured by a zero-mean Gaussian distribution $p(\mathbf{Y}) \propto \exp\left(-\frac{1}{2}\sum_{d=1}^{N}\left(y^d\right)'\Omega_1 y^d\right)$ [2]. These equations define a generative model $p(\mathbf{X}, \mathbf{Y}|f)$ over states and observations, with Equation (5.30) defining the observation model $p(x_t|y_t, f)$ and Equation (5.29) defining a prior over states. The RKHS norm of f may also inform a prior distribution $p(f)$ on f. Figure 5-1 depicts the independence relationships in this generative model. Given a generative model, one can perform smoothing by evaluating $p(\mathbf{Y}|\mathbf{X})$, evaluate evidence by computing $p(\mathbf{X})$, generate fantasies from $p(\mathbf{X})$, or predict output frames with $p(X|\mathbf{X})$.

In this thesis, I use the term "nonlinear system identification" to mean the problem of inferring f and $\mathbf{Y}$ only, whereas typically, the term refers to estimating all the parameters of the model, including those of the observation noise and the parameters of the dynamics. In this setting, all the parameters of the model (5.29)-(5.30) are known except for the observation function f. Finding the maximum *a posteriori* states and the observation function requires a joint maximization over f and $\mathbf{Y}$ on the distribution $p(\mathbf{Y}, \mathbf{X}, f)$. Assuming zero-mean spherical Gaussian observation noise, taking logs, and removing constant terms and factors, this amounts to the following optimization:

$$\min_{f,\mathbf{Y}} \sum_{t=1}^{T} \|f(y_t) - x_t\|^2 + \lambda_k \sum_{d=1}^{M} \|f^d\|_k^2 + \lambda_s \sum_{d=1}^{N} \left(y^d\right)'\Omega_1 y^d. \tag{5.31}$$

[2] In the dynamical system literature, latent variables are denoted by $\mathbf{X}$ and observations by $\mathbf{Y}$. In order to remain consistent with the rest of the thesis, I have denoted the quantities we wish to estimate by $\mathbf{Y}$, and the input of the algorithm by $\mathbf{X}$.

The first term favors a fit between observations and latent variables, the second term favors a smooth observation function, and the third term favors state sequences that evolve according to the given dynamics. A similar cost function is obtained if we treat $\mathbf{Y}$ as a nuisance variable by solving $\max_f \int_{\mathbf{Y}} p(\mathbf{Y}, \mathbf{X}, f)$. By Expectation Maximization [54], we get the optimization $\max_{f,q} \int_{\mathbf{Y}} q(\mathbf{Y}) \log p(\mathbf{Y}, \mathbf{X}, f) + \mathcal{H}(q)$, where q is a probability distribution over $\mathbf{Y}$. As shown in Section 5.5.2, the difficulties in the joint optimization remain, and are amplified by the integration.

Existing techniques for identifying hidden Markov models with nonlinear observation functions [26, 90, 67] are based on the coordinate ascent optimization strategy. Our experiments show that they require many iterations to converge even when initialized near the true answer, and they tend to get stuck in local minima despite the simplifying approximations they make. For example, [26] uses EM to find the maximum likelihood parameters of the model while marginalizing out the state sequence $\mathbf{Y}$, and [90] uses Variational Bayes to find the posterior over $\mathbf{Y}$ and the parameters of the model. Both of these algorithms are approximate learning algorithms: the Variational Bayes formulation of [90] assumes a convenient functional form for the posteriors, and the E-step of both algorithms relies on approximate state estimation. Despite these approximations, these algorithms are subject to local minima because f is a nonlinear function of y with no other guarantees beyond smoothness.

When the observations are high-dimensional, as is the case with video frames, these methods run into another difficulty. Traditional representations of nonlinear functions, such as multilayer perceptrons (MLP) [90] and radial basis functions (RBF) [26], require many parameters to represent a mapping to a high-dimensional space, resulting in an optimization over a large number of variables. For example, in analyzing a sequence of observed images, if we adopt the RBF form $f(y) = \sum_{t=1}^{T} c_t k(y, y_t)$, each c_t must have as many elements as there are pixels in each image. If T is the length of a video sequence, estimating the parameters of the function would require performing an optimization over as many coefficients as there are pixels in the video sequence. This storage requirement may not be acceptable for large sequences, and operating on such a representation may be too computationally intensive. The MLP representation has a similar requirement. The most common system identification approach is to directly model an autoregressive k-step-ahead predictor that predicts future measurements given a history of measurements [38]. These predictor models also requires a large number of parameters to represent high-dimensional outputs. Further, because there are no latent variables in these models, they cannot support many of the tasks we mentioned earlier. Hence, as far as we are aware, only linear system identification techniques have been used to learn state-space models of video [20, 83].

The unsupervised learning algorithm presented earlier in this Chapter circumnavigates the problems that arise when searching for f by assuming that f is smooth and invertible, and by approximately searching for its inverse. It approximates the functional of Equation (5.31) with a quadratic cost function, so that when adopting an RBF representation for g, this search reduces to an eigenvalue problem in which high-dimensional observations are summarized by entries of a kernel matrix. Hence finding g is not subject to local minima, and the dimensionality of the data set does

not factor into the computational or storage complexity of the algorithm, except to compute the kernel matrix. The inverse of this projection can later be implicitly plugged back into the generative model without explicitly computing and storing f.

One strength of the cost function for nonlinear system identification problem posed in Equation (5.31) is that it does not require the observation function to be one-to-one. When f is not invertible, a given x may correspond to many ys, and observing a particular y may make multiple values of x feasible. In our model the mapping g transforms noisy observations to states, and so cannot entertain the possibility of multiple xs that are very different from each other. Our model can only maintain the possibility of nearby xs generating the same observation. The invertibility requirement means that every observation must be uniquely mappable to its underlying representation. This is the case in many applications of interest. For example, in visual tracking, as long as an object remains visible, its pose corresponds uniquely to its appearance, so its instantaneous appearance can be mapped to its pose. This invertibility requirement means that the manifold of noiseless observations may not cross itself. This assumption is implicit in some manifold learning algorithms. For example, Isomap requires that the manifold must be an isometric embedding of a convex compact ball. The isometry requirement is that $\forall_{x,x'} \|y - y'\|_2 = d_G(f(y), f(x'))$, where $f(y)$ is the lifting of a coordinate y, and d_G is the geodesic distance along the manifold in the embedding space. This formula implies that $f(y) = f(y') \iff y = y'$, which is exactly the one-to-one requirement on f. If f is not invertible, the interpretation in this section is no longer applicable, though the algorithm still produces results, and the manifold learning interpretations of the previous section are still valid.

The objective function for nonlinear system identification, Equation (5.31), is similar to the objective function for our unsupervised learning algorithm, Equation (5.1), with two differences. First, our learning cost functional has constraints. These constraints can be added to Equation (5.31) to fix a coordinate system. Second, the data terms involve functions that map in opposite directions. Despite these discrepancies, under certain conditions, the optimum function according to Equation (5.1) closely approximates the inverse of the optimum function according to Equation (5.31).

If we force f to be a locally area-preserving one-to-one function with a small curvature compared to the observation noise, Equation (5.31) can be recast as an optimization that looks similar to (5.1). If f is isometric, it is locally area-preserving, so this requirement is less restrictive than Isomap's requirement. Since f is a diffeomorphism, it has a unique inverse, g, so that $f(g(x)) = x$. The following approximation for the data fitting term holds:

$$\|f(y_i) - x_i\|^2 = \|f(y_i + g(x_i) - g(x_i)) - x_i\|^2 \tag{5.32}$$

$$\approx \left\|f(g(x_i)) - x_i + \left.\frac{\partial f}{\partial y}\right|_{y=g(x_i)} (y_i - g(x_i))\right\|^2 \tag{5.33}$$

$$= \|y_i - g(x_i)\|^2_{\left(\frac{\partial g}{\partial x}\frac{\partial g}{\partial x}'\right)^{-1}} \tag{5.34}$$

$$= \|y_i - g(x_i)\|^2. \tag{5.35}$$

In the second line, the first order Taylor expansion of f about $g(x_i)$ becomes more accurate as the curvature of f becomes smaller compared to the excursion $g(x_i) - y_i$, whose magnitude is determined by the amount of observation noise. The final step utilizes the local area-preserving property of f.

This approximation can be substituted into (5.31) to yield an optimization that is identical to (5.1), after imposing a suitable smoothness penalty on g, and adding the constraints of (5.1) to fix a coordinate system

5.3.1 Substituting into the Generative Model

Once the function g and the latent states $\mathbf{Y}^*$ have been estimated, the inverse $\hat{f}$ of g can be evaluated. This in turn allows us to perform a variety of inference tasks on the generative model (5.29)-(5.30). Because the output of $\hat{f}(y)$ is a high-dimensional quantity, representing $\hat{f}$ in an explicit non-parametric form such as MLP or RBF would be unwieldy, as argued before. Instead, we can approximately evaluate $\hat{f}$ at a particular y without requiring additional storage. We do this by searching for the k nearest neighbors of y in $\mathbf{Y}^*$, and interpolating between their corresponding x's to obtain $\hat{x} \approx \hat{f}(y)$. Filtering, smoothing, and calculating the evidence under the generative model can then be performed by standard techniques that do not require the derivatives of $\hat{g}$, such as the Unscented Kalman Filter [39].

5.4 Relationship to other Methods

In addition to our use of dynamics, a notable difference between our algorithm and the general manifold learning framework laid out in [80], the nonlinear ICA algorithm of [89], or [46] is that instead of learning a mapping from states to observations, we learn mappings from observations to states, which reduces the storage and computational requirements when processing high-dimensional data.

Independent Component Analysis algorithms (ICA) [65] find mappings from observations to low-dimensional representations that exhibit various information theoretic properties. Instead, our algorithm requires that each low-dimensional coordinate exhibit temporal coherency, and be uncorrelated from the other coordinates.

As far as I am aware, in the manifold learning literature, only Jenkins and Mataric [36] explicitly take temporal coherency into account. They have extended Isomap [86] by explicitly assigning similar low-dimensional coordinates to temporally adjacent samples.

5.5 Experiments

The small storage requirement and the fast and local-minimum-free computations of the unsupervised learning algorithm make it well-suited for analysing high-dimensional data sets. These experiments show that the recovered latent states are close to the true states of the process that underlies the high-dimensional observations, and that when the observation function is invertible, the mapping recovered by this algorithm

is close to the inverse of the true observation function. All of our experiments use a spherical Gaussian kernel.

5.5.1 Recovering the Inverse Observation Function in Low-dimensional Datasets

I first show that the recovered mapping is empirically close to the inverse of the true measurement function in low-dimensional settings. Because in these experiments the observation function maps scalars to scalars, visualizing the results will be easier than subsequent experiments that deal with higher-dimensional data. Section 5.5.2 shows that our algorithm compares favorably to the nonlinear system identification procedure of [26] on these data sets.

Figure 5-2(left) depicts a segment of a 1500 sample 1D signal $\mathbf{X}$ generated by observing a latent signal $\mathbf{Y}$ through an invertible nonlinearity. The latent process was produced using a linear-Gaussian autoregressive model $s_{t+1} = \mathbf{A}s_t + \omega_t$ with $\mathbf{A} = \begin{bmatrix} .9 & .2 & 0 \\ 0 & .5 & .1 \\ 0 & 0 & .1 \end{bmatrix}$, and ω_t zero-mean with covariance $\Lambda_\omega = diag(\begin{bmatrix} 2 \times 10^{-5} & 2 \times 10^{-5} & 20 \end{bmatrix})$. The samples y_t of $\mathbf{Y}$ consisted of the first component (the position component) of each s_t. $\mathbf{Y}$ is shown in Figure 5-2(middle). To generate the observations $\mathbf{X}$ shown in the left panel, each y_t was passed through the observation function $f(y) = \tan^{-1}(10y)$, whose inverse is shown in Figure 5-2(right).

The observed sequence shown in Figure 5-2(left) was processed by the algorithm of Section 5.1 with parameters $\lambda_s = 1$, $\lambda_k = 1$, kernel variance $\sigma^2 = 1$, and with $\mathbf{A}$ and Λ_ω set to their ground truth values. To recover the correct scale, the constraint $\frac{1}{T}\mathbf{Y}\mathbf{Y}' = \mathbf{I}$ was changed to $\frac{1}{T}\mathbf{Y}\mathbf{Y}' = \sigma_{ss}^2\mathbf{I}$, where σ_{ss}^2 is the steady-state variance of the position component of the Markov chain. This ensures that the second moments of the recovered $\mathbf{Y}$ match the second moments of the true latent variable sequence.

Figure 5-2(middle) compares the recovered latent process to the true latent process. Note that its scale is recovered correctly because a correct generative model of the latent process was available. Usually, we only know this model very approximately, and the scale of the recovered signal will not be recovered accurately. The recovered coefficients define the function g depicted in Figure 5-2(right) along with the inverse of the true observation function. Figure 5-3 shows the recovered functions g and the recovered states for a variety of other nonlinear observation functions. These recovered functions coincide with the inverse of the true function.

Figure 5-4 shows the embedding of a 1500 step 2D random walk into $\mathcal{R}^3$ by the function $f(x, y) = (x, y\cos(2y), y\sin(2y))$. The 2D walk was limited to a rectangular areas and reflected off the boundaries of the rectangle. In addition to this behavior, the dynamics model used to generate the walk was different from the one specified in our algorithm, to demonstrate resilience to errors in the dynamics model. Using the recovered g, the figure plots $g(x)$ where x are generated by lifting a 2D grid using f. The result is a flat surface that preserves the global characteristics of the true low-dimensional coordinates up to a scale and some shrinking in the lower left corner. Therefore the recovered g is close the inverse of the original mapping, up to a scale change. Varying the dynamics up to an order of magnitude on the parameters of the

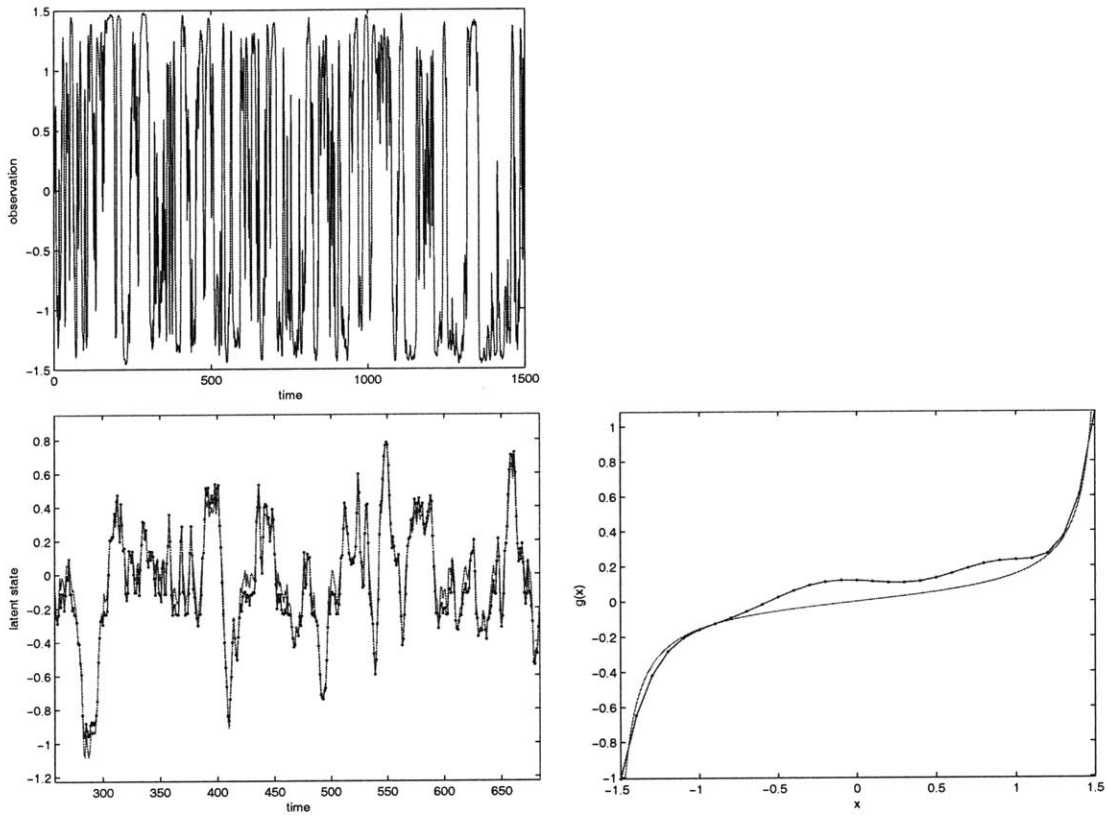


Figure 5-2: (top) Observed 1D signal. (bottom-left) Latent process underlying the observations in the left panel (solid line), and recovered latent process (dotted line). (bottom-right) The inverse of true observation function $f(y) = \tan^{-1}(10y)$ (solid line) and its recovered inverse (dotted line) The latent states and the inverse of the observation function are recovered accurately.

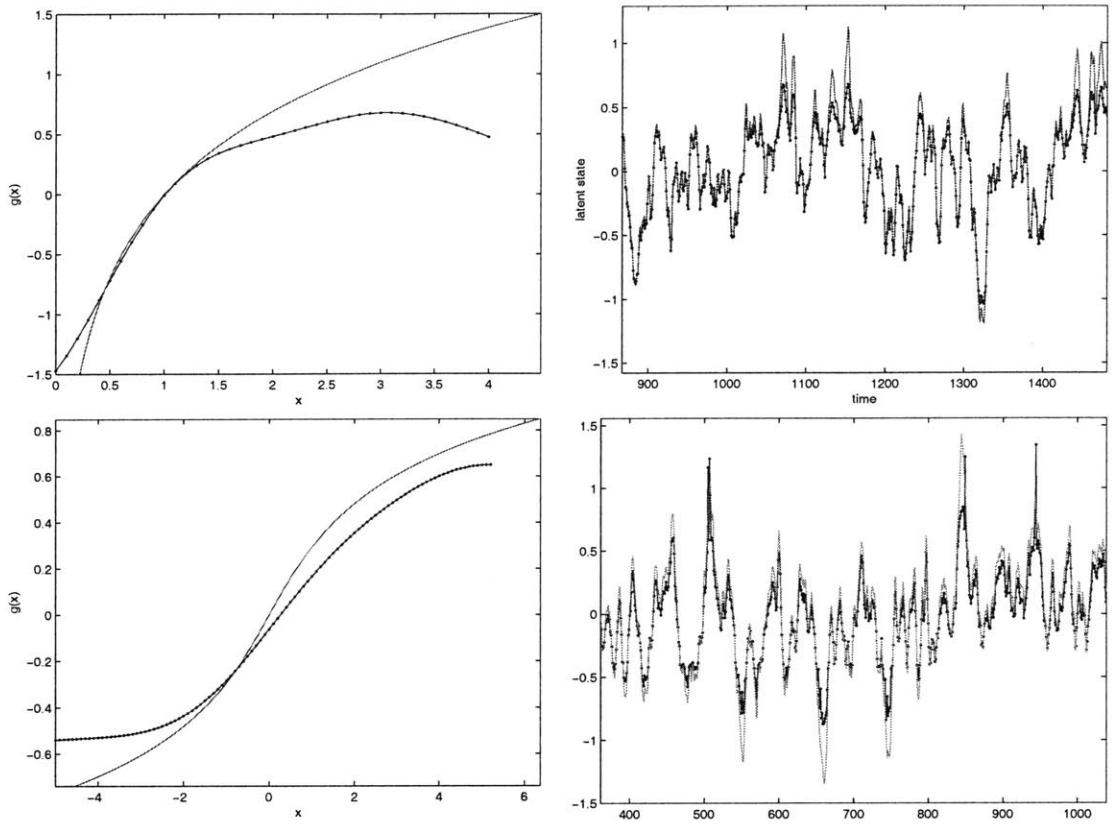


Figure 5-3: Experiments with two more observation functions. (top-left) The inverse of the true observation function $f(y) = (2 + y)^{-2}$ (solid) and its recovered inverse (dotted). (top-right) The true latent states (solid) and the recovered latent states (dotted). (bottom-left) The inverse of the true observation function $f(y) = \sinh(3y)$ (solid) and the recovered inverse (dotted). (bottom-right) The true latent states (solid) and the recovered latent states (dotted). The inverse of the true observation function and the states are recovered accurately.

dynamics model yields similar results, but changes the scale of the mapping. Isomap performs poorly on this data set due to the low sampling rate on the manifold and the fact that the true mapping g is not isometric. KPCA chooses a linear projection that simply eliminates the first coordinate axis because this projection is smooth and the symmetry of the roll about the origin satisfies KPCA's constraint (5.18). ST-Isomap produced the best results when it was run in a mode where it emulated Isomap. In this mode, the only difference between Isomap and ST-Isomap is that the latter always includes temporal neighbors in the neighborhood of each point. There is some folding in the output of ST-Isomap, and the true underlying walk is not recovered. We found the optimal parameter settings for Isomap, KPCA, and ST-Isomap by grid search over the parameter space of each algorithm.

On a 1 Ghz Pentium III, each of these examples took about 3 minutes of CPU time. These low-dimensional examples show that when the true observation function is smooth and invertible, our algorithm recovers it accurately and quickly.

5.5.2 Comparison with the Algorithm of Roweis and Ghahramani

We evaluated the nonlinear system identification algorithm of Roweis and Ghahramani [26] on the data set of the previous section. Because it learns a mapping a from low-dimensional space to a high-dimensional space, its memory requirements prevent us from running it on higher dimensional data sets. Because it performs nonlinear state estimation in a hidden Markov chain with arbitrary smooth observation functions, the algorithm converges only when started at solutions that are sufficiently close to the true solution, and may become unstable. We also found that the algorithm, as described in [26] is significantly slower than our unsupervised learning algorithm. Our implementation of this algorithm seems to work well when the observations are low-dimensional, and the observation function is smooth and one-to-one.

Since in our setting, the dynamics of the latent states are known *a priori*, our implementations of this algorithm estimate only the most likely observation function in the generative model (5.29)-(5.30), with the latent states marginalized out. The algorithm of Roweis and Ghahramani takes approximate EM steps to perform the optimization $\arg\max_f \int_{\mathbf{Y}} p(\mathbf{X}|f,\mathbf{Y})p(\mathbf{Y})d\mathbf{Y}$. In the ith iteration of the algorithm, the M-step finds $f^{(i)} = \arg\max_f E_{q^{(i)}}[\log p(\mathbf{X},\mathbf{Y}|f)]$, where $q^{(i)}$ is a Gaussian approximation to $p(\mathbf{Y}|f^{(i-1)},\mathbf{X})$, and is calculated in the E-step of the iteration. The E-step finds the Gaussian approximation q by applying a nonlinear RTS smoother [41] to the observations $\mathbf{X}$, with the observation function $f^{(i-1)}$ estimated by the M-step of the previous iteration. The M-step finds $f^{(i)}$ by fitting a function to the observed $\mathbf{X}$'s and the estimated $q^{(i)}$. This step reduces to finding $\arg\min_f E_{q^{(i)}}[\sum_{t=1}^{T} \|f(y_t) - x_t\|^2]$. When adopting the RBF form $f(y) = \sum_j c_j k(y,\mu_j)$ with Gaussian kernels centered at fixed locations μ_j, this operation can be performed in closed form. Define $\mathbf{K_Y}$ as a kernel matrix whose jtth element is $k(\mu_j, y_t)$. The optimization of the M-step becomes $\arg\min_{\mathbf{C}} E_{q^{(i)}}[\|\mathbf{C}\mathbf{K_Y} - \mathbf{X}\|_F^2]$, which is optimized by $\mathbf{C}^* = \mathbf{X}E_{q^{(i)}}[\mathbf{K_Y}]'E_{q^{(i)}}[\mathbf{K_Y}\mathbf{K_Y}']^{-1}$. Expectations of a product of Gaussians under a Gaussian distribution can be calcu-

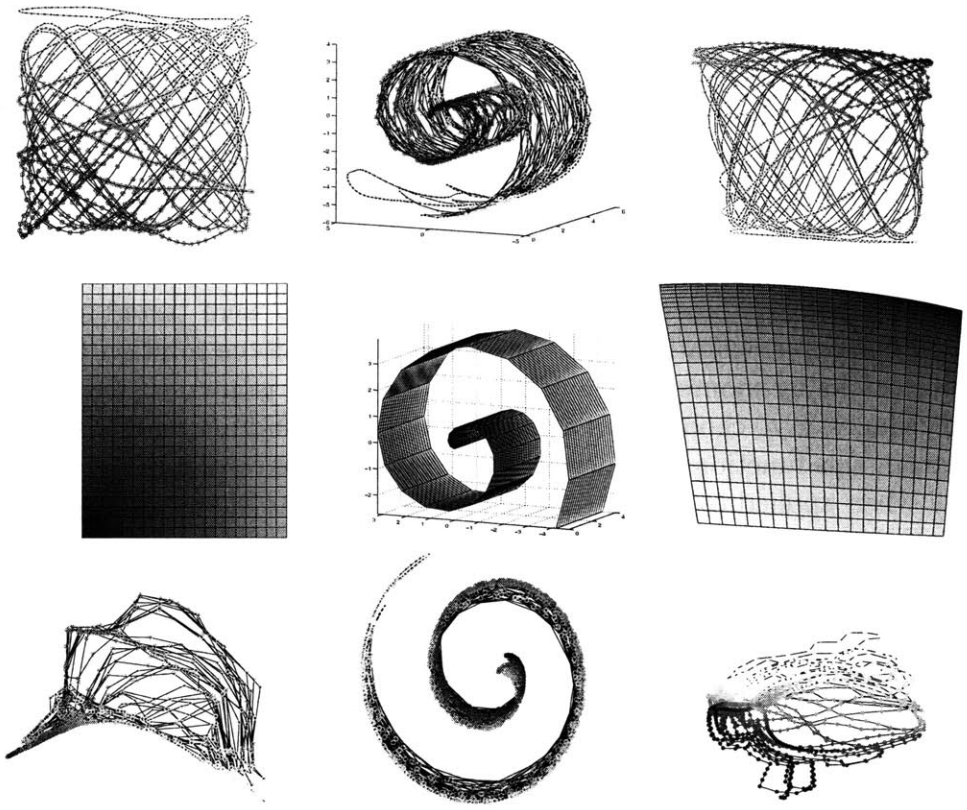


Figure 5-4: (top-left) Low-dimensional ground truth trajectory. Points are colored according to their distance from the origin in the low-dimensional space. (top-middle) Embedding of the trajectory. (top-right) Recovered low-dimensional representation using our algorithm. The original data in (top-left) is correctly recovered. To further test the recovered function g, we uniformly sampled a 2D rectangle (middle-left), lifted it using the true f (middle-middle), and projected the result to 2D using the recovered g (middle-right). g has correctly mapped the points near their original 2D location. Given only high-dimensional data, neither Isomap (bottom-left), KPCA (bottom-middle), nor ST-Isomap (bottom-right) find low-dimensional representations that resemble the ground truth. These figures are best viewed in color.

lated efficiently [26], yielding a closed form solution for $\mathbf{C}$ in the M-step.

Roweis and Ghahramani's algorithm requires that the RBF centers be placed manually, whereas our unsupervised learning algorithm simply places the RBF centers at the observed data points. Although the authors describe a way to automatically modify these centers by taking gradient steps, they show no experiments demonstrating the success of this technique. Computing these gradients involves evaluating expectations of Gaussians up to fourth order, which proves to be tedious. We therefore did not implement this feature.

We also experimented with a joint-max version of Roweis and Ghahramani's algorithm that found MAP estimates of the states as well as the observation function. Our joint-max algorithm uses a smoother to estimate the mode of $p(\mathbf{Y}|f, \mathbf{X})$, and fits an RBF f to the resulting mode $\mathbf{Y}$, iterating to a joint mode of $p(\mathbf{Y}, f|\mathbf{X})$ using coordinate ascent.

To ensure that our implementation of this algorithm was free of bugs, the smoother of the E-step and the function estimator of the M-step were tested separately. We compared our implementation of the M-step to a brute-force solution that calculated the expectations by monte carlo sampling. The comparison consisted of defining an arbitrary Gaussian q, choosing an arbitrary nonlinear function, and passing the mode of q through this function to generate observations. The results of the two algorithm matched when applied to this data, and looked similar to the ground truth functions that were used to generate the observations.

We compared various smoothers against each other to find the best smoother for the E-step and to ensure that our implementations were free of bugs. Four of the smoothers we tested were based on the RTS smoother, performing a forward filtering pass followed by a backward smoothing pass. Of these, two used an Unscented Kalman Filter [39] for the forward pass, and the other two used an Extended Kalman Filter. The backward pass of all the smoothers used an extended backward filter, and differed in whether they linearized about the states recovered by the backward pass, as is suggested in [26], or linearized about the smoothed estimate. The fifth smoother we tested used Newton-Raphson to estimate the mode of $p(\mathbf{Y}|f, \mathbf{X})$, and fitted a Gaussian at this mode using Laplace's method to obtain a Gaussian approximation to p. We tested these smoothers on a variety of observation functions, and found that they yielded similar results for very smooth, one-to-one observation functions. They all failed to recover the latent states accurately when the observation function was not one-to-one because such observation functions induce local optima in the state posterior $p(\mathbf{Y}|f, \mathbf{X})$. Only the smoother based on Newton-Raphson returned sensible results in regions where the derivative of the observation function was close to zero, so we used it in our implementation of the E-step. Our implementation of this smoother employs efficient linear algebra routines, and runs faster than Kalman filter-based smoothers by about 40 percent.

The RBF representation used 600 1-dimensional kernels with variance 0.5 with centers placed uniformly from $y = -3$ to $y = +3$. The parameters of the dynamical model were set to their ground truth values. The variance of the observation noise was set to a very small value (10^{-4}), since the data set is generated without observation noise. This effectively multiplies the driving noise of the dynamics model by 10^4.

Figure 5-5 plots the recovered functions for the data sets of the previous section. The figure also displays the most likely latent states given the estimated observation function. The joint-max version of the algorithm took about an hour to converge on a 1Ghz Pentium III, whereas our algorithm took only a few minutes. The version of the algorithm that marginalizes over states took three days to converge on a 3.2 Ghz P4, and returned similar results as the joint-max version.

The recovered functions are shrunk horizontally, and the function recovered for $f(y) = \tan^{-1}(10y)$ is curved the wrong way. Reducing the number of RBF centers results in linear estimates for f that exhibit no curvature at all, even though as few as 8 RBF centers are sufficient for representing the true f with high fidelity. Finally, the EM iterations were initialized with an RBF fit to the ground truth function. Randomly initializing the function, or initialing it at zero resulted in the smoother of the E-step to fail as explained above.

Figure 5-6 shows states recovered by this algorithm when applied to the swiss roll data set of Figure 5-4. The true observation function $f(x, y) = (x, y\cos(2y), y\sin(2y))$ is not recovered, because smoothing with the recovered function simply projects the observations without unrolling the roll.

The algorithm of Roweis and Ghahramani is framed as a MAP estimation of the parameters of the generative model. However its function representation requires too many parameters for us to apply it to subsequent data sets in this thesis. Our implementation of the M-step can solve for about 8000 parameters with 1 Gigabyte of RAM within 10 minutes per iteration on a 1Ghz PIII, whereas as explained earlier, we would require upwards of 1 million parameters to learn a mapping from states to 100×100 pixel frames of a 100 frame video. In addition, due to shortcomings in the smoothing operation of the E-step, it suffers from instabilities when the current iterate of the observation function is not one-to-one. This problem could be remedied by using a more sophisticated smoother based on monte carlo sampling, and performing an exact M-step, though this would further increase the computational complexity of the algorithm. Instead of performing approximate EM steps, Section 5.3 shows that our algorithm optimizes an approximation of the likelihood function and obtains reliable mappings.

The algorithm of Valpola et al. [90] is similar to that of Roweis and Ghahramani, with two differences. First, it uses an MLP representation for f rather than RBFs. This does not alleviate the requirement for the number of parameters to solve when learning a mapping to a high-dimensional space. Second, instead of searching for a point estimate for f, it uses EM to approximate a posterior over f by variational Bayes. Its M-step still relies on a smoother, so it will surfer from the same instability as Roweis and Ghahramani's algorithm.

5.5.3 Recovering Inverse Observation Functions for Image Sequences

Video is a very high-dimensional signal, yet a few smoothly varying degrees of freedom govern the appearance of many dynamic scenes. These latent degrees of freedom may

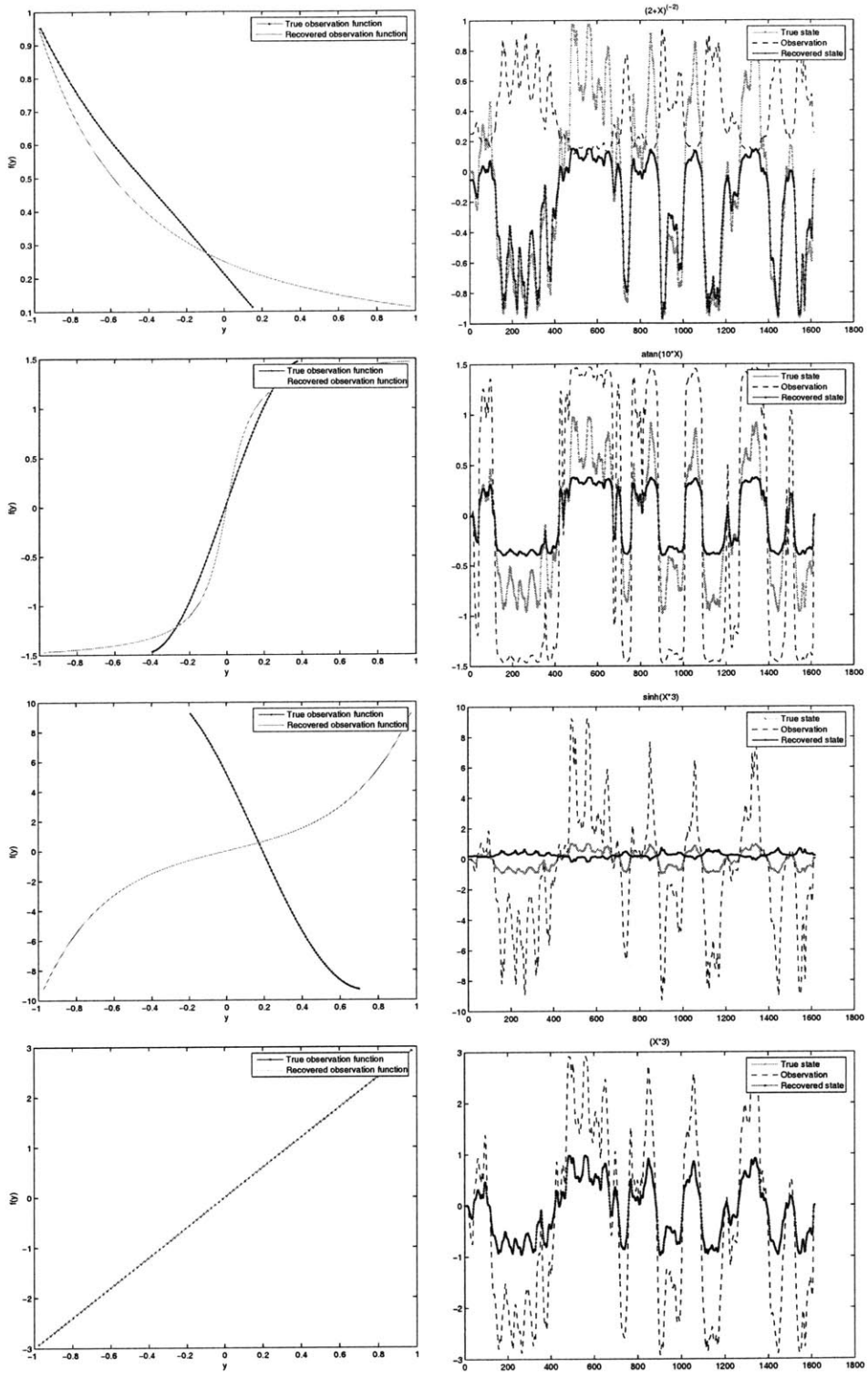


Figure 5-5: (left column) True observation function (solid green) and observation function recovered by the algorithm of Roweis and Ghahramani (dotted blue). (right column) 1D observations (dashed black) generated by observing the latent states (dotted green lines) through the true observation function, and recovered latent states estimated by the last E-step of the algorithm of Roweis and Ghahramani (dotted blue). The arctangent function is not correctly recovered. The other functions are similar to the ground truth observation functions, except for horizontal shrinkage. 5-3.

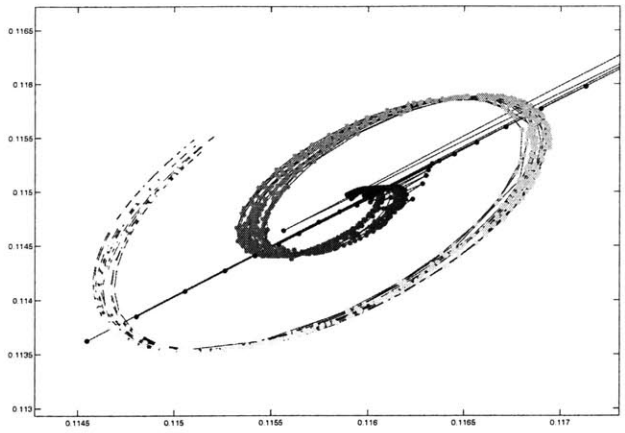
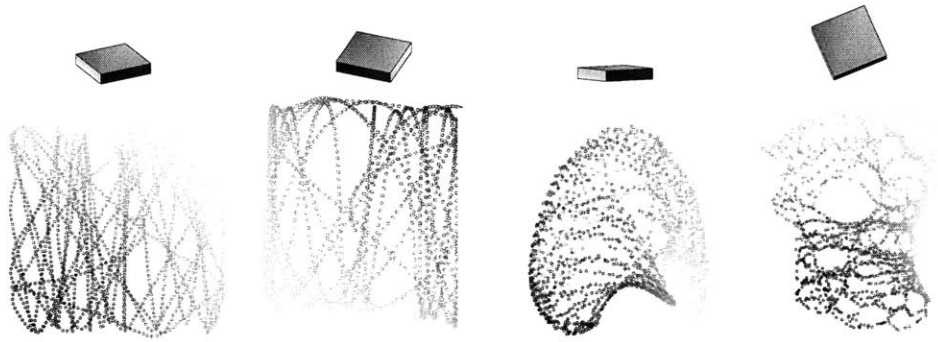


Figure 5-6: Low-dimensional states recovered for the swiss roll data set of Figure 5-4 by the algorithm of Roweis and Ghahramani [26]. The recovered function simply projects the 3-dimensional observations to 2 low-dimensional space without unrolling the roll.

correspond to the pose of objects, illumination conditions, or other physical states of the scene. Given a video sequence, we would like to learn a generative model of the appearance of the scene that takes into account these low-dimensional processes, to enable traditional time series operations on video sequences, such as predicting future frames, denoising, classification, anomaly detection, and dimensionality reduction. Our unsupervised learning algorithm can be applied to a video sequence of a rotating object to recover a function that maps the pixels of an image of the object to the the parameters that govern its appearance.

We generated a video sequence of a cube rotating along two axes. Figure 5-7 shows a few frames of this 2000 frame sequence. The goal in this experiment is to learn a function that maps an image of a cube to its 2-dimensional pose parameters. To show resilience to mismatch between the true dynamics and the assumed dynamics, different dynamics were used to generate the path of rotations than those specified to our algorithm, including reflections off of the boundary in the space of rotations. We used stable third order dynamics, with $\mathbf{A} = \begin{bmatrix} 0.9 & 0.2 & 0 \\ 0 & 0.9 & 0.2 \\ 0 & 0 & 0.9 \end{bmatrix}$, and Σ a diagonal matrix with a few orders of magnitude more noise in the acceleration components than the velocity or position components.

Figure 5-7 plots the low-dimensional coordinates recovered by our algorithm and various other unsupervised learning algorithms. The coordinates recovered by our algorithm are close to the true low-dimensional coordinates that were used to generate the sequence, so the recovered g is close to the inverse of the true observation function f. Note that the mapping between rotations and appearances is not isometric, since infinitesimal rotations produce different amounts of change in the appearance of the cube depending on its instantaneous rotation. This violation of Isomap's isometric assumption explains why its recovered poses are stretched in places. Both Isomap and KPCA pull together rotations that make the top face of the cube face the camera, because the appearance of the face is similar under all such poses. Use of dynamics



(a) (b) (c) (d)

Figure 5-7: (top row) A few frames from the synthetically generated rotating cube sequence. Only the azimuth and elevation of the cube are modified. (a) Shows the true elevation-azimuth trajectory, (b) the trajectory $\mathbf{Y}^*$ recovered by our algorithm, (c) by KPCA, and (d) by Isomap. Our algorithm recovers the true rotation up to a flip, but with very little distortion. Because appearance is not an isometric function of rotation, Isomap's trajectory is unevenly stretched.

allows our algorithm to disambiguate between these situations. KPCA also exhibits folding, which is absent from Isomap and our algorithm's output.

For this 2000 frame sequence, the algorithm runs in about 5 minutes on a 3 Ghz P4. In setting the parameters, we followed the guidelines of Section 4.5. Given only a video sequence of the rotating cube, a description of the dynamics of the latent states, and no labeled examples, our algorithm was able to recover a function that maps images of the cube under rotation to scaled versions of the underlying rotation. Note that the observations were raw $100\times$ 100 pixels, so this is a nonlinear system identification problem with 10000-dimensional observations.

5.5.4 Learning to Track in a Large Sensor Network

We consider an artificial distributed sensor network where many sensor nodes are deployed randomly in a field in order to track a moving target. The location of the sensor nodes is unknown, and the sensors are uncalibrated, meaning that it is not known *a priori* how they map the position of the target to the measurements they report. We will only assume that they maps the position of the moving to their reported measurement smoothly. Given only the measurements from the sensing nodes, we wish to track a target as it moves about in the sensing range of this sensor network. This situation arises when it is not feasible to calibrate each sensor prior to deployment, or when variations in environmental conditions affect each sensor differently. Assuming only that each sensor's observation function is smooth and memoryless, and that the target follows given dynamics, our unsupervised learning algorithm finds a transformation that maps observations from the sensor network to the position of the target. In our simulations, this transformation recovers the true position of the target with minor distortion, up to a scaling and flip, without the benefit of any labeled data.

The generative model described by Equations (5.29)-(5.30) can be used to model this situation. In this situation, a latent state y_t is the unknown position of the target at time t, f^d is the measurement function for the dth sensor, and x_t is the collection of measurements from the entire network at time t. To apply our unsupervised learning algorithm, we will assume that f is invertible. This assumption does not require the observation function of each sensor to be invertible, only that each pose of the target generate a unique set of observations from the ensemble of sensors. The individual measurement functions can be arbitrary as long as they are smooth. In general, we expect to be able to triangulate the position of the target from measurements when f is known. This implies that f is invertible in most places. The invertibility restriction may pose a minor practical problem if the target is allowed to exit the sensing range of the network, where all nodes produce quiescent observations. In such cases, many out-of-range positions result in the same quiescent observations. One could handle such ambiguities by ignoring quiescent measurements.

Figure 5-9(left) depicts the setup of the simulation. The network consists of 100 nodes spread randomly in a 2D field. To generate observations, we set the true observation function for each sensor to a decreasing function of its distance to the target:

$$x_t^d = f_0^d(y_t) = \alpha^d \exp(-\beta^d \|y_t - c^d\|_2), \tag{5.36}$$

where c^d is the true location of the node, and α^d and β^d are scalar calibration parameters for each sensor. In these experiments, the sensor locations c^d were uniformly drawn from a 2×2 area. The calibration parameters were set to the absolute value of iid Gaussian random variables. The target was made to follow a smooth trajectory that reflected from the boundaries of the sensor network. A 1500 sample time series of 100 measurements was generated under this setup. Figure 5-8 shows some the response functions of some of the nodes. Figure 5-9(middle) shows example measurements from the nodes as the target moves about the field. The only information supplied to the algorithm were

- 1500 time step of the 100-dimensional observations from the network.
- The parameters describing the dynamics of the target.

No functional form for f^d was supplied.

On a 1Ghz Pentium III, the unsupervised learning algorithm took 5.5 minutes to process this data. The recovered function implicitly performs all the triangulation necessary for recovering the position of the target, even though the position or characteristics of the sensors were not known *a priori*. Figure 5-9(right) shows the recovered trajectory for the unlabeled training data. To test the mapping further, we ran the target in a boustrophedonic (zigzag) pattern through the field, and passed the samples of the resulting measurements through g. Figure 5-10 shows the true trajectory of the target and the recovered trajectory. The position of the target is recovered with minor distortion, and up to a scale and 90 degree rotation. To show that the problem is not trivial, the figure also displays the output of KPCA when applied to the raw observations. KPCA displays significantly more distortion at the edges.

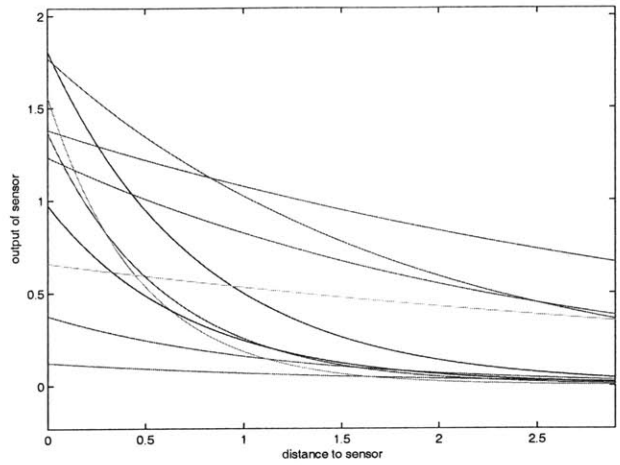


Figure 5-8: The response of some of the nodes in the network as a function of the euclidean distance of the target to the node.

As the variation in the sensor parameters is reduced, the identification improves. Figure 5-11 shows the trajectory recovered by our algorithm when the data is generated by sensors whose calibration parameters are equal, with $\alpha^d = \beta^d = 1$.

This setup is related to the ones explored by [62] and [77]. The method of [62] assigns to each node a high-dimensional coordinate consisting of all the measurements it has gathered. The authors show empirically that simply reducing the dimensionality of this data with Isomap, LLE, or HLLE recovers the 2D position of each sensor. Although not stated explicitly, under noise-free conditions, one can reverse-engineer the measurement model to be $x^d = f(y^d)$, where x^d is the collection of measurements made by the dth sensor, and y^d is the 2D location of the sensor. A manifold learning algorithm will therefore successfully recover the position from measurement data if its assumption about the lifting from low-dimensional to high-dimensional space matches the true measurement model. A more limited approach by [77] assumes that distances between nodes are observed, and uses MDS to recover the low-dimensional locations. Since the measurements in our experiments are governed by the position of the sensors as well as that of the target, these approaches are not applicable.

These experiments demonstrated the utility of a practical nonlinear system identification algorithm in a sensor network setting. Because it scales well with the number of dimensions, our algorithm can process the output of many sensor nodes. Furthermore, we have shown shown that restricting the observation function to be one-to-one can be an appropriate restriction. To show our algorithm works on real data, the following section applies it to the *Sensetable*'s outputs. The *Sensetable* setup is very similar to the setup of this section, in that each antenna acts as an uncalibrated sensor with a sinusoidal, smooth measurement function.

5.5.5 Learning to Track with the *Sensetable*

In Section 4.2, we recovered the inverse of the observation function of the *Sensetable* with only four labeled points. In this section, we show that this function can be recovered

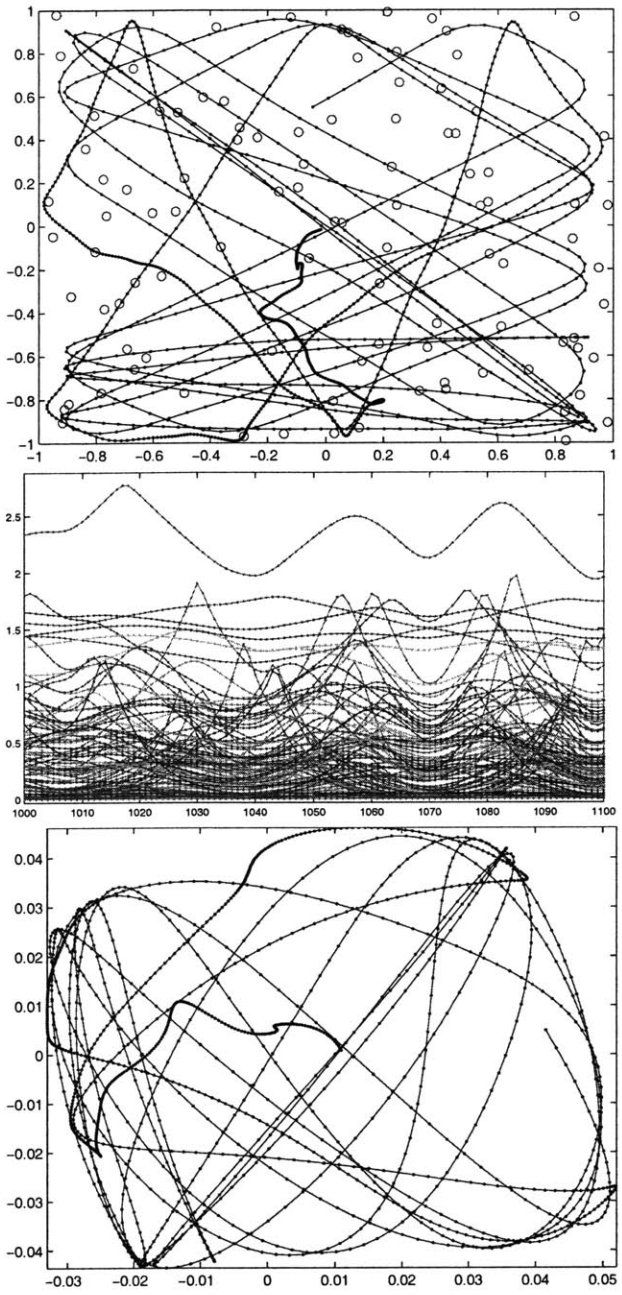


Figure 5-9: (top) A target followed a smooth trajectory (dotted line) in a field of 100 sensors (circle). The location and observation function of each sensor is unknown to the algorithm. (middle) Some measurements produced by the sensor network in response to the target's motion. Measurements were recorded for 1500 time steps. This plot shows time steps 1000 to 1100. (bottom) Target trajectory recovered by the unsupervised learning algorithm. The recovered trajectory is rotated by 90 degrees with respect to the true trajectory, but otherwise similar to it. See Figure 5-10 for an assessment.

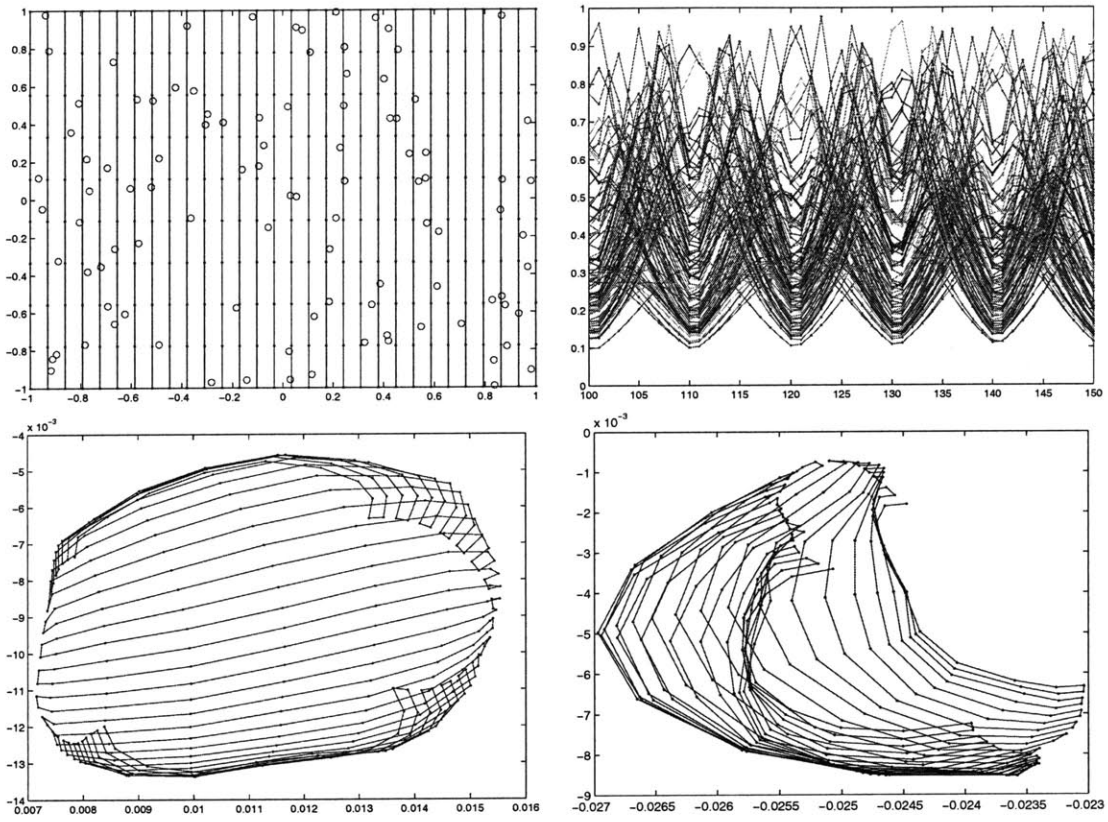


Figure 5-10: (top-left) To test the recovered mapping from measurements to positions, the target was made to follow a zigzag pattern over 300 time steps. (top-right) The measurements produced by the network in response to the target's zigzag motion, over 50 time steps. (bottom-left) The location of the target recovered by applying the estimated g to each sample of the measured time series. The resulting trajectory is similar to the ground truth zig-zag trajectory, up to scale, a 90 degree rotation, and some minor distortion. (bottom-right) The trajectory obtained by applying the mapping recovered by KPCA.

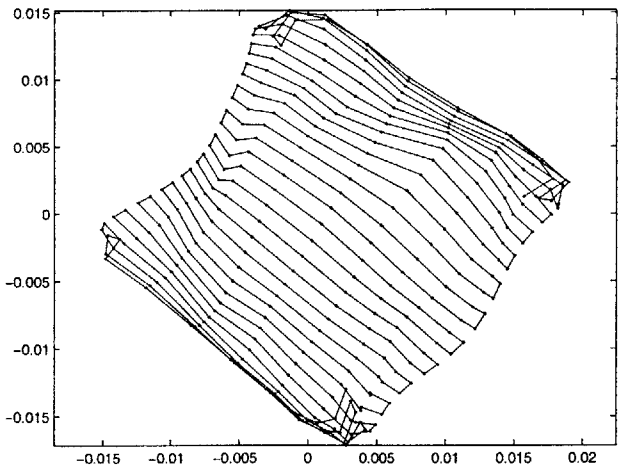


Figure 5-11: Recovered trajectory for an experiment where all sensor calibration parameters are the same. As the variation in these parameters is reduced, the recovered trajectory resembles the true trajectory more and more.

up to an affine transformation using only unlabeled data, so that labeled points only help to fix a coordinate system. This experiment also serves as a real-world instantiation of the sensor network setup of the previous section, where measurements are produced from the uncalibrated antennae of the *Sensetable* instead of the uncalibrated nodes of a sensor network. Recall from Section 4.2 that the true observation functions for the antennae of the *Sensetable* are sinusoidal, so this real-world experiment is more challenging than the simulated sensor network setup of the previous section.

The unsupervised learning algorithm was run on the unlabeled high-dimensional data set of the semi-supervised *Sensetable* experiment of Section 4.2. The parameter settings used in the semi-supervised experiment were used in this unsupervised experiment. Figure 5-12 shows the ground truth tag trajectory that generated this data set, and the trajectory recovered by our unsupervised algorithm given only the measurement time series from the *Sensetable*. Compared to the ground truth coordinates, these recovered coordinates are scaled down, and are flipped about both axes. There is also some additional shrinkage in the upper right corner, but the coordinates are otherwise recovered accurately. Note that there is no folding. By applying an affine transformation to the recovered trajectory, we can register it with the ground truth trajectory with a residual error of 2.1 cm per pixel. Registering the trajectory recovered by the semi-supervised algorithm on this data set against the ground truth results in 1.1 cm of residual error. Therefore, unlabeled data points are sufficient for recovering the latent process up to an affine transformation, and labeled points are needed only to fix an affine coordinate system, and to provide minor refinements that yield an additional 1 cm of accuracy.

Figure 5-13 shows the the result of KPCA, Isomap, ST-Isomap, and LLE on this data set. None of these algorithms recover low-dimensional coordinates that resemble the ground truth. For all the neighborhood sizes we tested LLE collapses the coordinates to one dimension (we tested all neighborhood sizes between 2 and 50 in

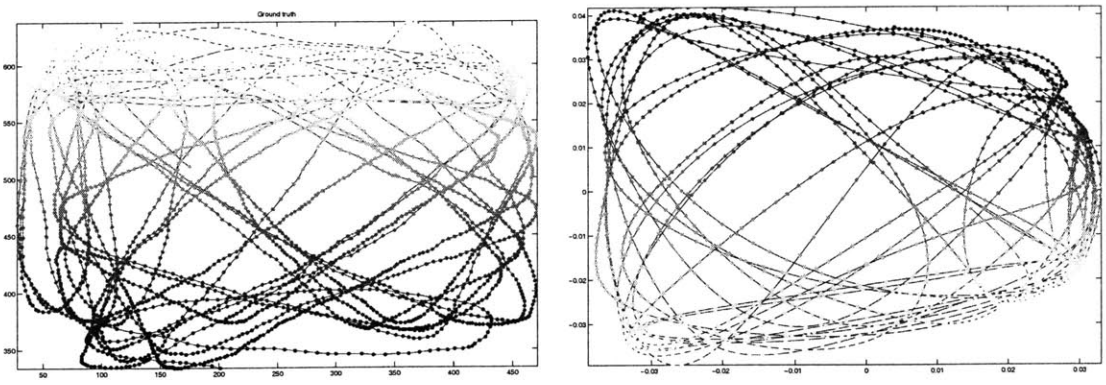


Figure 5-12: (left) The ground truth trajectory of the RFID tag. (right) The trajectory recovered by our algorithm is close to the ground truth trajectory: it is correct up to flips about both axes, a scale change, and some shrinkage along the edge. We showed in Chapter 3 that labeling the four corners was enough to fix these distortions.

increments of 5). Magnifying one of the coordinate axes by a factor of 10^8 reveals that this is not just an issue of scaling, but that there is severe folding in the recovered coordinates as well. From left to right, the affine registration errors were 8.5 cm, 10.4 cm, and 11.0 cm, for neighborhood sizes of 15, 30, and 50 respectively. KPCA also performed poorly, exhibiting folding and large gaps. Its affine registration error was 7.2 cm for the best setting of the kernel variance. Of these, Isomap performed the best, though it exhibited some folding and a large hole in the center. For the small neighborhood sizes shown in the figures, its affine residual error was 3.6 cm, 3.9 cm, and 3.4 cm, respectively. Larger neighborhood sizes up to 30 yielded registration errors as large as 4.5 cm. ST-Isomap performed similarly, with a best affine registration error of 3.1 cm. As before, ST-Isomap was run with every possible combination of the following parameter settings: temporal window size (1, 2, 3, 4, 8, 10, 15, 20, 30, 40), *catn* (1, 10, 50, 100, 200, 500, 1000, 5000, 10000), and k (2, 3, 4, 5, 10, 15, 20, 25, 30, 35) (see [36] for details). Larger window sizes produce worse results (varying from 8 cm to 14 cm of affine registration error for window sizes of 15 and up), as do larger neighborhood sizes (neighborhood sizes larger than 3 produced affine registration errors from 4 cm to 7cm). As in the synthetic experiments, a neighborhood size of 2 two 3, with a window size of 1 to 2 appears to provide the best performance. Using the prior on dynamics to smooth the output of these algorithms sometimes improves their accuracy by a few milimiters, but more often diminishes their accuracy by causing overshoots.

Because the mapping recovered by our algorithm is close to the ground truth, it can be used it to track RFID tags without any additional labeled data, and without the affine correction. Figure 5-14 shows the output of the algorithm on the data set of Figure 4-6. The recovered shapes are similar to the ground truth shapes shown in Figure 4-6. Unlike the shapes of Figure 4-6, to allow for a fair comparison against KPCA, the output was not smoothed. Figure 5-15 shows the trajectories obtained by applying the mapping recovered by KPCA to the testing data set. The recovered trajectories do not at all resemble the true trajectories.

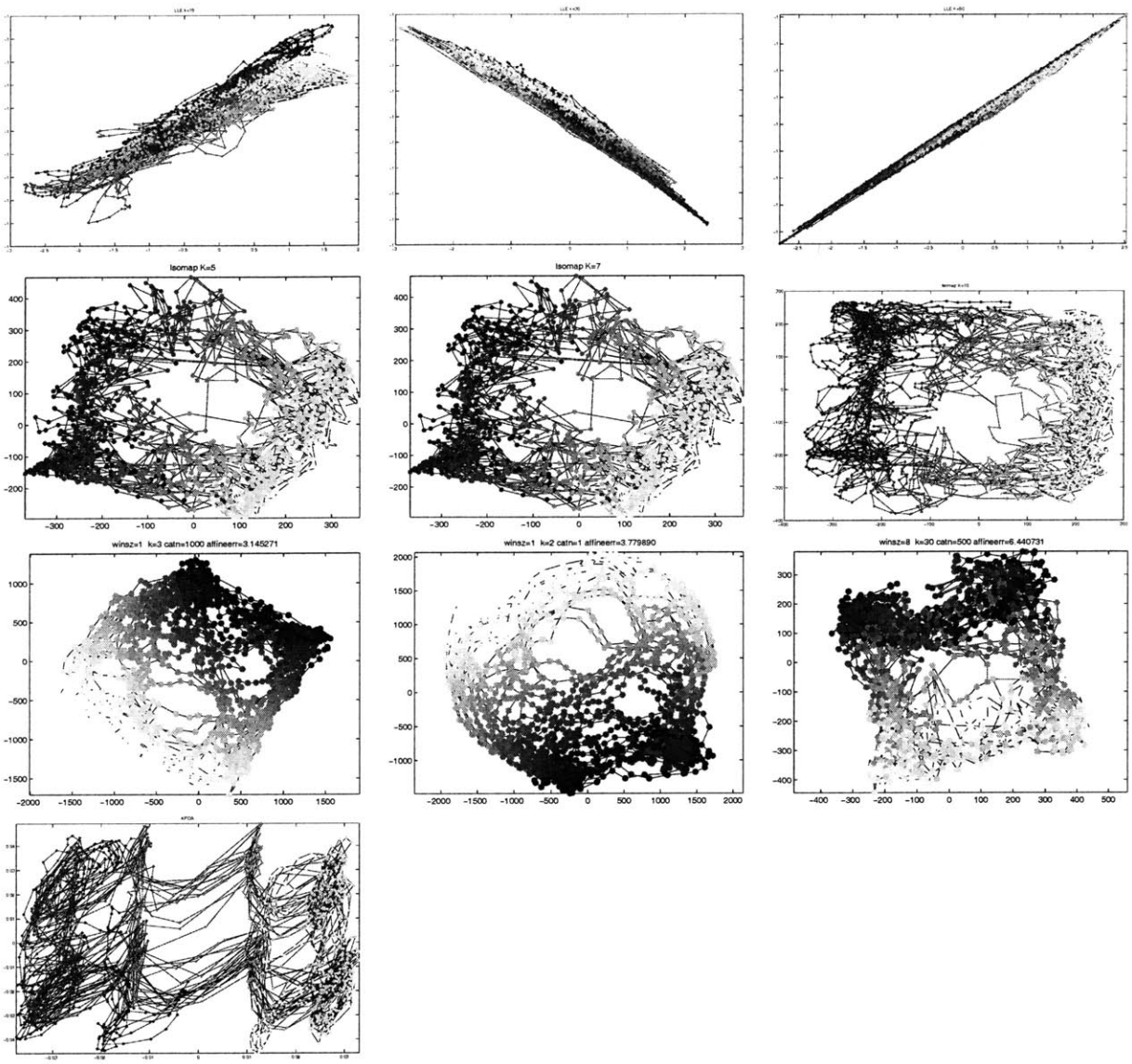


Figure 5-13: (top row) Tag trajectories recovered by LLE for a neighborhood sizes of 15, 30, and 50. The results are representation of all neighborhood sizes. (second row) Trajectory recovered by Isomap for a neighborhood sizes of 5, 7, and 10, also representative. (third row) ST-Isomap performed best with small window and neighborhood sizes. Shown from left to right are its output with its best setting, another similar good setting, and a typical bad setting with large window and neighborhood size. (bottom row) Trajectory recovered by the best setting for KPCA. All of these trajectories exhibit folding and severe distortions.

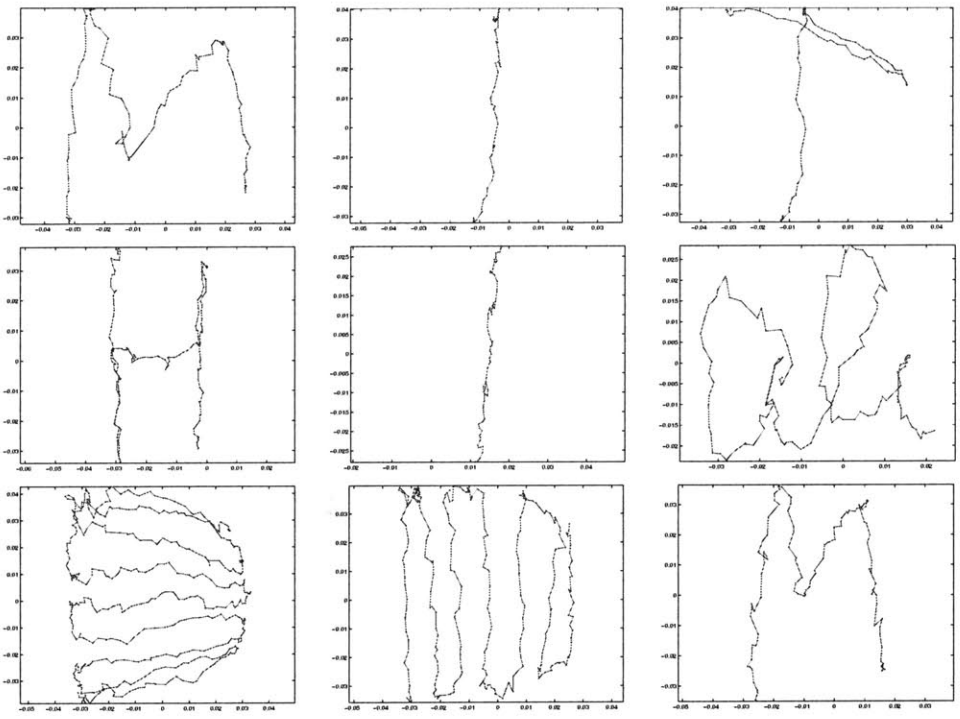


Figure 5-14: The mapping recovered by the unsupervised learning algorithm can be used to track RFID tags by applying it to each measurement sample from the *Sensetable*. Here, the mapping is applied to the data set of Figure 4-6. The recovered trajectories match the shapes traced by the tag.

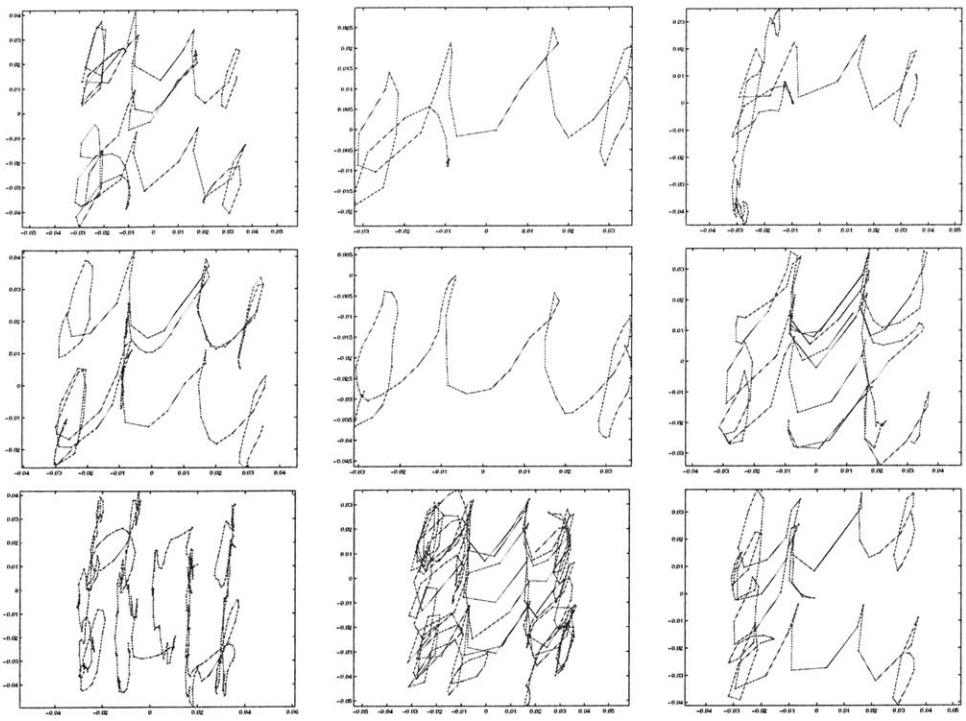


Figure 5-15: Applying the mapping recovered by KPCA to the test shape data does not recover the true trajectory of the RFID tag. The recovered trajectories do not match the shapes traced in Figure 4-6.

5.6 Conclusions and Future Work

The algorithm presented in this chapter is an unsupervised counterpart to the semi-supervised learning algorithm of Chapter 3. Given a time series, and the approximate dynamics of a latent representation of the time series, this algorithm learns a nonlinear transformation from the observed time series to the latent representation. The cost function of this unsupervised learning algorithm is similar to that of various manifold learning (LLE and KPCA) and nonlinear system identification algorithms. In turn, this shows that the algorithms of Chapter 3 are semi-supervised nonlinear system identification and manifold learning algorithms.

Our algorithm circumvents the storage and computational problems of state-of-the-art nonlinear system identification procedures, which learn an observation function that maps states to observations, by learning a nonlinear function that maps observations to states. When the true underlying mapping from states to observations is smooth and one-to-one, the mapping recovered by our algorithm is close to the inverse of the true observation function.

The storage and computational efficiency of our algorithm makes it a practical system identification procedure for high-dimensional data. I showed, for example, that in an uncalibrated sensor network setting where the measurement function is completely unknown, we can still learn to track if we take advantage of the knowledge about the dynamics of the target. In some cases, this unsupervised learning algorithm

can serve as a first pass before running a semi-supervised learning algorithm. Since in many tasks, the latent variables can be recovered up to a small distortion without labeled data, the output of the unsupervised algorithm can guide a user's selection of labeled points to refine its output. Finally, these algorithms may serve as knowledge discovery algorithms that uncover the underlying structure of high-dimensional data sets such as video sequences.

The cost functional behind this algorithm is very versatile. We have required that the output time series obey temporal dynamics, $p(\mathbf{Y})$ can enforce any Gaussian prior that factors over the dimensions of the latent states without any modifications to the algorithm. For example, one could capture *a priori* spatial dynamics between the latents states this way. It would be interesting to explore this direction.

Chapter 6

Localizing a Network of Non-Overlapping Cameras

Chapter 3 showed that dynamics can compensate for a dearth of labeled examples in regression problems. Chapter 5 showed that dynamics can compensate for a complete lack of labeled examples when uncovering the dynamical process underlying high-dimensional time series. In this chapter, we assume a more restricted functional form for the input-output mapping. Whereas in the previous chapters, an observation was available at every time step, this additional prior on the mapping provides enough structure to allow some of the unlabeled samples to be missing.

From observing a moving calibration target, we recover the position of cameras in a network of cameras whose fields of view do not overlap. Due to the lack of overlap, at any given time, the target appears in the field of view of at most one camera. That we can recover the position of the cameras from these observations may be a surprising result. Without making assumptions about the motion of the target, observing the target confers no information about the relative positions of the cameras. The gap between the cameras can be bridge by utilizing a prior on the physical dynamics of the target. This camera calibration problem is very similar to the unsupervised learning situation of the previous chapter. The camera calibration problem amounts to learning a function that maps a target's position on the ground plane to a pixel location on the camera's image plane. The only observations available in this setting are the pixel coordinates of the target when it appears in the field of view of each camera, and the objective is to estimate the projection function as well as the ground plane trajectory of the target.

There are two main differences between these experiments and those of the previous chapter. First, unlike the previous chapter, we do not use RBFs to represent the projection function. Instead, we assume that the mapping takes the form of a projective mapping, and search for the parameters of a pinhole camera model. Second, observations are only available when the target passes through the field of view of a camera, so for most time steps, observations are missing. This chapter shows that when enough structure is imposed on the function to be estimated, it is acceptable to have very few unlabeled data points.

This work was originally published at CVPR 2005 [68] and researched in collabo-

ration with Brian Dunagan, who helped with collecting data.

6.1 Introduction

Non-overlapping camera networks can cover a much wider area than overlapping configurations, but calibrating them is more challenging. We describe a method for recovering the extrinsic calibration parameters of non-overlapping cameras in a multi-camera network. To compensate for the lack of overlap, our approach assumes that the calibration target follows a smooth trajectory over time, as described by a stochastic dynamics model. Our algorithm then tracks the calibration target when it comes in the field of view of each camera, and searches for calibration parameters and target trajectories that are consistent with the observed tracks. To make calibration easier, we allow the calibration target to simply be a moving person. We demonstrate the algorithm with a network of indoor wireless cameras.

Many security-critical areas such as airports and casinos are instrumented with thousands of non-overlapping cameras that remain uncalibrated. To track people in these environments and to visualize their trajectory in a globally consistent coordinate system, the location of these cameras must first be determined. In certain visual surveillance settings, the field of view of the cameras are made to overlap at least slightly, making it easier to track people as they move from the field of view of one camera to another, and in turn to recover the pose the cameras. We examine the problem of calibrating a camera network where the field of views of the cameras to not overlap, permitting a wider area to be instrumented using fewer cameras.

Various algorithms exist for recovering the topology of non-overlapping camera networks [53, 52, 40, 51]. By contrast, we wish to recover the relative location and orientation of non-overlapping cameras in order to provide a globally consistent coordinate system for tracking. After a single-camera calibration step involving a standard calibration target, our procedure requires a point calibration source to move along the ground plane in smooth paths between the field of view of the cameras. This path is not required to be straight, or to have constant velocity: we only assume that its evolution is consistent with a known stochastic dynamical model. Our main contribution is to show that a model of dynamics for the motion of the calibration target can compensate for the lack of overlap between the cameras.

When there is overlap between the fields of view of cameras, the appearance of the calibration target in a region of overlap provides information about the relative pose of the cameras. With enough such appearances, the relative orientation of the cameras can be recovered by taking advantage of stereopsis. Without information about the dynamics of the target, observing the calibration target where the cameras do not overlap only provides information about its pose relative to the camera that observed it, but not between the pose of two cameras.

For each camera, our algorithm recovers the two translation parameters and the rotation parameter in the ground plane. We rely on existing single-camera calibration techniques to recover the remaining extrinsic and intrinsic parameters before our algorithm is applied. The calibration task can be separated into these two tasks because

single-camera calibration procedures can recover the pose of the camera relative to a local patch on the ground plane, up to a rotation and translation in the ground plane, in addition to recovering the intrinsic camera parameters. After such a procedure, our algorithm can be applied to globally align the coordinate system of the patches corresponding to each camera.

To perform this global alignment step, on could require that the target moves at constant velocity and in a straight line when it leaves the field of view of a camera. An intuitive approach would then be to have each camera track the target when the target passes through the camera's field of view, and estimate the target's velocity as it leaves the field of view. The travel time between the fields of view of two cameras, along with the target's estimated velocity can be used to estimate the distance between the two fields of views. When enough such distances are obtained between pairs of cameras, they can be combined to recover the relative location and orientation of all the cameras in the network [34].

Our algorithm relaxes the assumption that the target moves in straight lines and at constant velocity. This allows the calibration target to be more casually carried by a person moving along curved paths. Our algorithm searches for camera calibration parameters and a smooth target trajectory that are consistent with the tracks observed by each camera while the target was in its field of view. This search is framed as finding the maximum *a posteriori* (MAP) camera calibration parameters and target trajectory, with a prior that prefers smooth target trajectories.

We have implemented our algorithm on a network of wireless PDAs equipped with cameras (see Figure 6-1). A real-time person tracker runs on each PDA which report trajectories to a central server that implements our algorithm.

6.2 Related Work

Many methods explicitly designed for calibrating networks of cameras rely on overlapping fields of view [81, 82, 43, 33]. Our method is most closely related to metric calibration algorithms proposed by [34], [24] and [40]. Javed et al. [34] employ an idea similar to velocity extrapolation to find the projection of the field of view lines of one camera onto the field of view of cameras. Knowledge of these projections is tantamount to recovering calibration parameters, but their method requires people to walk in a straight line and at constant velocity outside camera fields of view. Fisher [24] shows how to calibrate a network of non-overlapping cameras using distant objects (stars) to recover orientation, and nearby objects (airplanes) to recover relative position. Strong assumptions are made on the motion of these objects (the motion of stars is parabolic, nearby targets must move in a line with constant velocity). The metric calibration extension presented in [40] also requires targets to move in linear constant velocity paths outside the fields of view. By contrast, we allow our targets to move much more freely, as long as they exhibit smoothly varying acceleration

Various researchers have addressed the problem of maintaining consistent identity between multiple targets as they exit one field of view and enter another [42, 60, 35, 64] in non-overlapping multi-camera systems. Using such techniques, others have shown

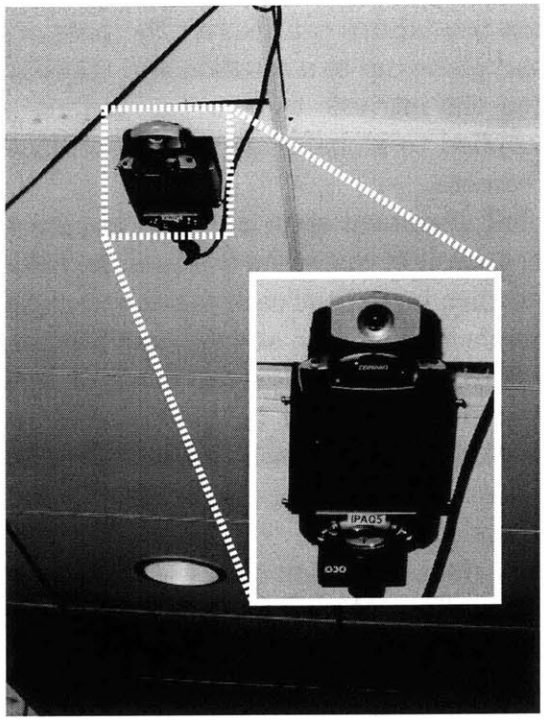


Figure 6-1: We use Compaq IPAQs as wireless camera nodes in our network. The IPAQs are mounted on the ceiling, with their camera image plane parallel to the ground plane.

how to recover a topological representation of the network of cameras, represented as the probability of the target transitioning from one field of view to another [53, 52, 40, 51]. These algorithms allows calibration to occur passively by observing moving crowds. By contrast, we use a point calibration target (in practice, a person walking about), and our technique attempts to recover a metric calibration for the network, where the distances and orientations of the cameras are recovered.

6.3 Single-Camera Calibration

Before running our algorithm, each camera is calibrated so that it can map the image coordinate of the target to a local coordinate system laid on the ground-plane. The homography required to perform this mapping can be estimated for each camera individually [88]. For example, by laying a calibration object of known size on the ground plane, the focal length, height, pitch, and yaw of the camera can be estimated. This local homography can be used to rectify the image location of the target in each camera so that the camera appears to be virtually fronto-parallel to the ground plane. This leaves a translation and roll in the ground plane to be estimated by our algorithm.

In the remainder of this paper, we presume that the homography for each camera has been recovered up to a rotation and a translation in the ground plane, and focus on aligning the local ground-plane coordinate system of each camera to the global

ground plane coordinate system. In our experiments, we used overhead cameras, so that only the height and the focal length of the cameras needed to be estimated during the single-camera calibration phase.

6.4 Global Alignment

To globally calibrate the network, we jointly find the maximum *a posteriori* (MAP) estimate of the remaining calibration parameters and the trajectory of a point calibration target that moves smoothly in the network. Each camera tracks the point calibration target as it passes through its field of view. MAP estimation amounts to searching for calibration parameters and a trajectory that are consistent with the observations made by each camera and a dynamics model for the motion of the target. Since these can only be recovered up to a global rotation and translation in the ground plane, we fix the parameters of one of the cameras.

We assume that the state evolves according to linear Gaussian Markov dynamics. Define x_t to be the state of the target at time t. This state contains information about the location, velocity, or any other dynamic state of the target, and evolves with:

$$x_{t+1} = \mathbf{A}x_t + \nu_t, \tag{6.1}$$

where ν_t is a zero-mean Gaussian random variable with covariance $\mathbf{\Sigma}_\nu$.

In our experiments, we let $x_t = [u_t; \dot{u}_t; v_t; \dot{v}_t]$, where (u_t, v_t) is the ground-plane location of the target and $(\dot{u}_t, \dot{v}_t)$ is its ground-plane velocity. We use the model parameters

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \tag{6.2}$$

so that each x_{t+1} adds the velocities in x_t to the positions in x_t, and corrupts the old velocities by Gaussian noise. We use a diagonal Σ_ν whose position components have much smaller value than its velocity components (by a few orders of magnitude), so that the velocities follow Brownian dynamics, and the resulting poses are corrupted only by a small amount of Gaussian noise. Equation (6.1) defines a prior $p(x)$ over a state trajectory $x = \{x_t\}_{t=1}^T$.

To extract the location components in x_t, we can multiply x_t by $\mathbf{C}$:

$$\mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}.$$

After taking into account the homography discussed in the previous section, camera i reports the location of the target in its own ground-plane coordinate system whenever the target is in its field of view. Let θ^i and p^i denote the unknown rotation and translation of camera i with respect to the global ground-plane coordinate system. Let the index set of observations $\mathcal{Z} = \{(t, i)\}$ be a collection of time index and camera pairs, where $(t, i) \in \mathcal{Z}$ iff camera i sees the target at time t. Let $\mu^i = [p^i; \theta^i]$ be the

parameters of camera i, and define $\mu = [\mu^1; \cdots; \mu^N]$ to be the collection of all the camera parameters. Letting $\mathbf{R^i}(\theta)$ denote the rotation matrix corresponding to a rotation of θ^i, the location reported by a camera when the target is in its field of view is:

$$y_t^i = \pi(x_t; \mu^i) + \omega_t = \mathbf{R}^i(\mathbf{C}x_t - p^i) + \omega_t, \tag{6.3}$$

where π is the mapping between global and local camera coordinate systems. We assume ω_t, the observation noise corrupting each measurement, is iid zero-mean and Gaussian with covariance $\sigma_y^2\mathbf{I}$. Equation (6.3) defines a likelihood $p(y|x, \mu)$ over trajectories and camera calibration parameters.

Any configuration of x and μ will produce the same tracking measurements as the same configuration arbitrarily rotated and translated [87]. This ambiguity can be fixed by setting the parameters of one of the cameras to some arbitrary value. We use a prior that favors configurations where the parameters of the first camera are 0.

Using the likelihood defined by Equation (6.3) and the prior over x defined by Equation (6.1) and the gauge-fixing prior for the first camera, the most *a posteriori* probable trajectory and calibration parameters can be obtained by performing a nonlinear least squares optimization over μ and x:

$$\begin{aligned}(x^*, \mu^*) = \arg\min_{x,\mu} & \sum_{(t,i)\in Z} \frac{1}{\sigma_y^2}\|y_t^i - \pi^i(x_t; \mu)\|^2 \\ & + \sum_{t=1}^{T} \|x_t - \mathbf{A}x_{t-1}\|_{\Sigma\nu}^2 + \|\mu^1\|^2.\end{aligned} \tag{6.4}$$

The first term in the cost function favors trajectories and parameters that would have generated the observed trajectory snippets. The second term favors trajectories that are consistent with the given dynamics by favoring trajectories where x_t can be predicted from x_{t-1} using the transition matrix of the dynamics model, and penalizing the prediction error by with a mahalanobis distance defined by the covariance of the driving noise. The third term fixes the gauge by setting the parameters of the first camera to zero.

To find the optimal μ and x, we use Newton-Raphson with a first order approximation to the Hessian. This involves it iteratively linearizing the nonlinearity inside the first term of Equation (6.4), transforming it to a a linear least squares problem. This least squares problem involves a minimization over $3 * N + 4 * T$ variables with $|Z|$ rows, where N is the number of cameras and T is the number of time steps in the trajectory to be estimated. It is also sparse, with $O(|Z|)$ nonzero elements, and so can be solved efficiently using standard sparse least squares solvers. The appendix derives the quantities needed in each Newton-Raphson step.

6.5 Synthetic Results

Our goal with this synthetic experiment was to see if a disparity between the dynamics model and actual target behavior had adverse effects on trajectory estimation and

calibration. We simulated a point target traveling in a square environment with elastic walls. The target's trajectory was generated by Brownian motion that deflected off of walls. The resulting trajectory was smoothed to yield the setup shown in Figure 6-2(a). The synthetic cameras were front-parallel to the ground plane, and measured the location of the target without noise within their coordinate system (i.e. up to a rotation and translation of the true location of the target).

To define the dynamics model, we used the $\mathbf{A}$ matrix of equation (6.2), and set $\Sigma_\nu = \begin{bmatrix} 10^{-4} & 1 & 10^{-4} & 1 \end{bmatrix}$. Notice that this dynamics model is different from the model used for generating the synthetic trajectory, as it does not model the bouncing behavior near walls, the final smoothing step, or the underlying Brownian motion.

The Newton-Raphson iterations were initialized with all trajectory points and camera parameters at the origin, pointing to the right. Figures 6-2(b-c) show an intermediate iteration and the converged estimate.

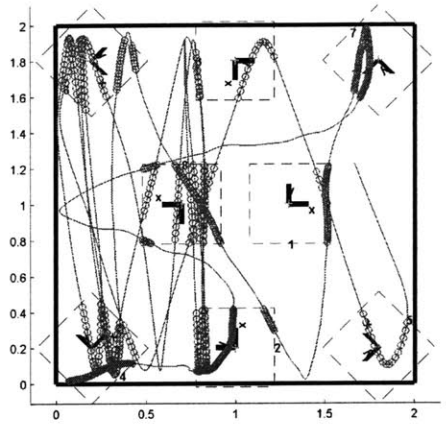
The estimated locations are wrong by an average of 0.03 size units, or 1.4% of the size of the environment. These experiments show that when there is no observation noise, the sensor parameters and trajectories are recovered very accurately, even if the target's true dynamics don't match those used in estimation.

6.6 Real Data

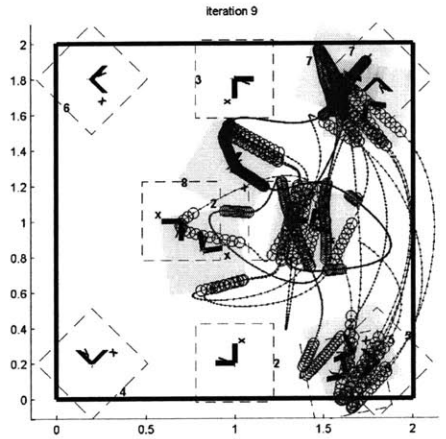
We implemented our algorithm on a network of wireless PDAs equipped with cameras. The PDAs were mounted on the ceiling in an indoor lab environment. The camera image planes are approximately parallel to the ground-plane so that computing the local ground-plane location of the person does not require any calibration beyond finding the focal length of the cameras. A real-time person tracker runs on each PDA and reports to a base station the time-stamped location of a person with respect to the camera's ground plane coordinate system every 250 ms. The person tracker uses background subtraction to extract the target and clusters the foreground pixels to compute the person's location. The individual trackers do not need to filter or smooth the data, as the dynamics model automatically regularizes trajectories during the optimization procedure.

In our first experiment, we installed 4 cameras in an open area in our building. The fields of view of the cameras were about 1.5 meters on each side, and the cameras were 3-4 meters apart. One person walked between the cameras at varying velocities and served as the calibration target. Figure 6-3(a) illustrates the setup. The ground truth camera parameters were found by measuring the location and orientation of every camera by hand. To render the trajectory of the target in this figure, we set μ to these ground truth camera parameters and optimized (6.4) for the trajectory only.

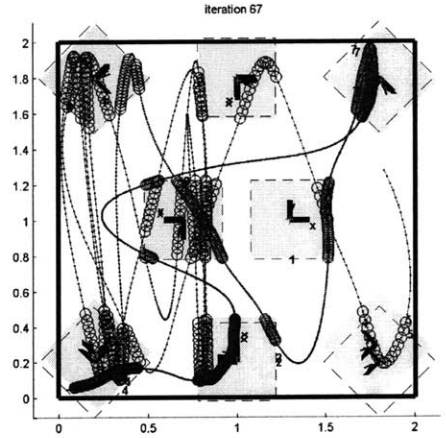
We used the same parameters for the dynamics model as in the synthetic case. The initial iterate for the optimizer was also the same as in the synthetic case. We set the observation noise σ_y^2 to be very small, a factor of 10^5 smaller than the driving noise of the velocity. Figures 6-3(b-c) show the recovered trajectory and sensor locations. On average, the sensor were misplaced by 28 cm from the locations measured by hand. Cameras `ipaq3` and `ipaq10` are off by 50 cm. The rotation of `ipaq3` is off by



(a)



(b)



(c)

Figure 6-2: (a) 2,000 steps of a synthetic trajectory. True camera fields of view are depicted as dashed squares. Circles along the trajectories indicate time steps in which a synthetic camera observes the target's location relative to its coordinate system. (b) After 9 iterations of the optimization procedure. The gauge is fixed by fixing sensor 1 at its true location and orientation. The blue (dark) path denotes the recovered trajectory. Gray squares are the recovered sensor fields of view. (c) Convergence after about 65 iterations. The sensor locations are estimated correctly.

8°. That of ipaq8 by 9° degrees. Ipaq10's rotation is off by 0.7°.

In the second experiment, we instrumented the hallway to the left of the open area with 2 additional cameras. The setup and results appear in Figure 6-4(a). The only way to reach ipaq5 and ipaq6 was to take a sharp right below ipaq7, outside its FOV. Because no camera ever witnessed this sharp right, and because straight trajectories are slightly favored by our dynamics model over turns, the estimated configuration did not recover the turn. Adding ipaq9 (Figure 6-4(b)) provided enough information to recover the turn. This is because ipaqs 7,9, and 5 form a triangle, and the angle 5-7-9 becomes constrained.

In contrast to the velocity extrapolation idea discussed earlier, our dynamics model allows curved trajectories. Figure 6-5 shows that our method works even when people take a sharp turn when entering the FOV of ipaq3. Using velocity extrapolation, the distance between ipaq7 and ipaq3 was found to be 518 cm, whereas in reality, it was only 423 cm. Our method recovered this distance more accurately to be 415 cm.

6.7 Optimization Procedure

This section derives the Newton-Raphson steps required to solve (6.4). We rewrite Equation (6.4) in nonlinear least squares form, linearize the nonlinearity, and describe one Newton-Raphson iteration.

The dynamics prior term $\sum_{t=1}^{T} \|x_t - \mathbf{A}x_{t-1}\|^2_{\Sigma_\nu}$ can be written as a quadratic form in terms of x, the column vector of stacked states. Denote the Cholesky factor of the inverse covariance $\mathbf{\Sigma}_\nu^{-1}$ of the driving noise in the dynamics model by $\mathbf{\Sigma}_\nu^{-\frac{1}{2}}$. [1]. Define the matrix $\mathbf{G}$ whose tth row is:

$$[\mathbf{G}]_t = \begin{bmatrix} 0 \cdots, & \mathbf{\Sigma}_\nu^{-\frac{1}{2}}\mathbf{A}, & -\mathbf{\Sigma}_\nu^{-\frac{1}{2}}, & 0 \cdots \end{bmatrix} \tag{6.5}$$

where the zeros pad the matrix to align its non-zero components with x_t and x_{t+1}. Then the dynamics prior term can be rewritten as $\sum_{t=1}^{T} \|x_t - \mathbf{A}x_{t-1}\|^2_{\Sigma_\nu} = x^\top \mathbf{G}^\top \mathbf{G} x$.

Equation (6.4) can now be rewritten as

$$(x^*, \mu^*) = \arg\min_{x,\mu} r(x,\mu)^\top r(x,\mu),$$

with

$$r(\mu, x) = \begin{bmatrix} r_y \\ \hline r_x \\ \hline r_\mu \end{bmatrix} = \begin{bmatrix} \sigma_y^{-1}\left(\mathbf{R}^i(\mathbf{C}x_t - p^i) - y_t^i\right) \\ \vdots \\ \hline \mathbf{G}x \\ \hline (\mu^1 - \mu_0) \end{bmatrix}.$$

The column vector $r(\mu, x)$ is partitioned into three sections, corresponding to the

[1] The Cholesky factor of a positive semidefinite matrix $\mathbf{A}$ is an upper triangular matrix $\mathbf{C}$ such that $\mathbf{A} = \mathbf{C}^\top\mathbf{C}$.

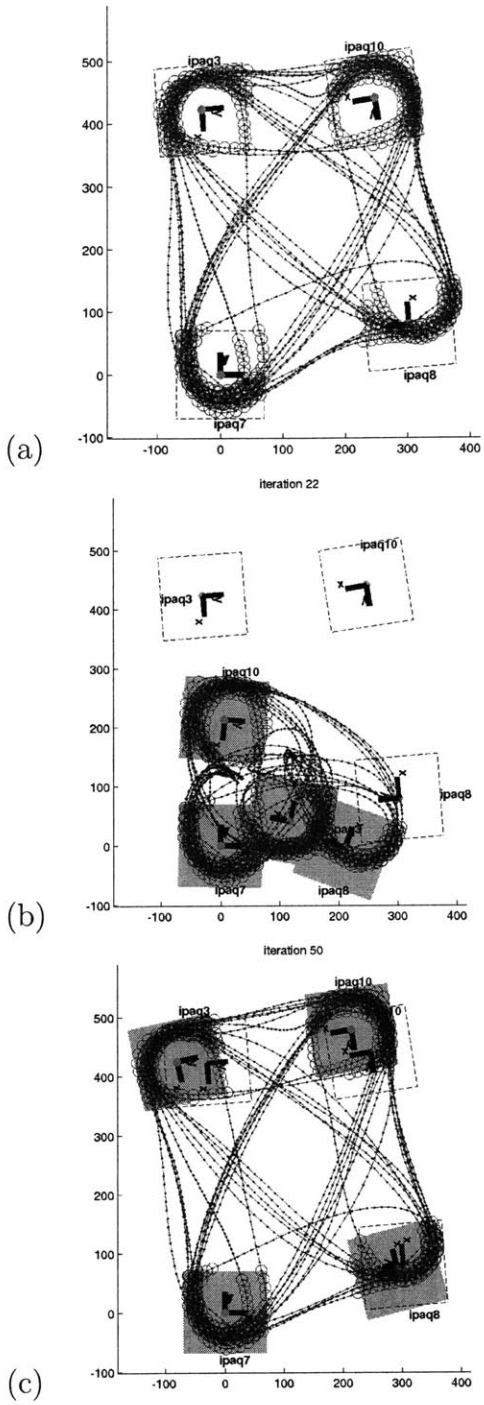


Figure 6-3: (a) The trajectory for the first experiment. Axes are labeled in centimeters. The coordinate system is fixed on `ipaq7`. (b) After 22 iterations. (c) Convergence after about 50 iterations.

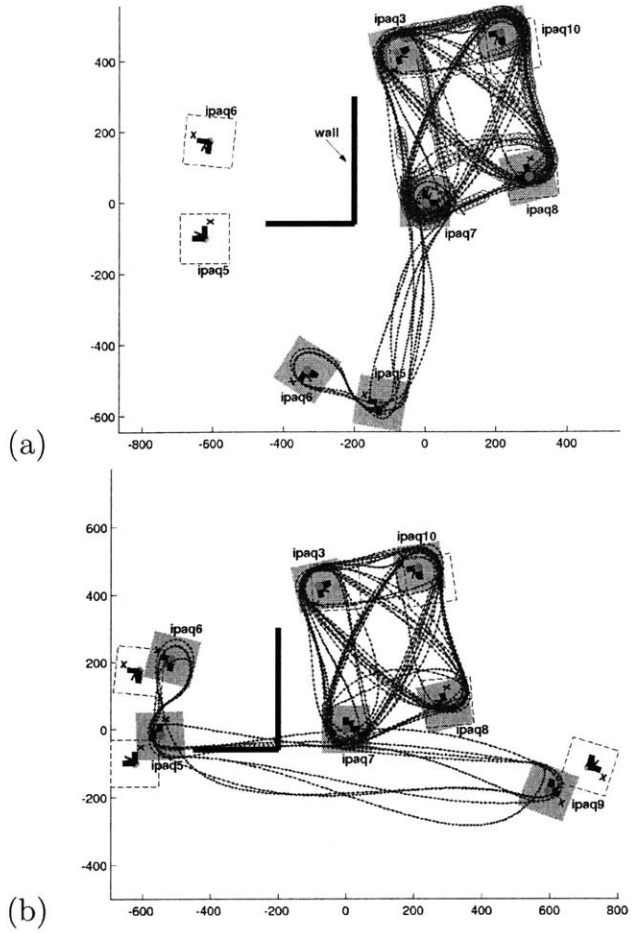


Figure 6-4: (a) A wall forces people to take a sharp turn outside the FOV of `ipaq7`. This turn is never witnessed by any camera, so the algorithm does not deduce its existence. (b) Although `ipaq9` doesn't observe the turn directly, its presence provides enough information to determine that `ipaq7` and 6 cannot be below `ipaq7`.

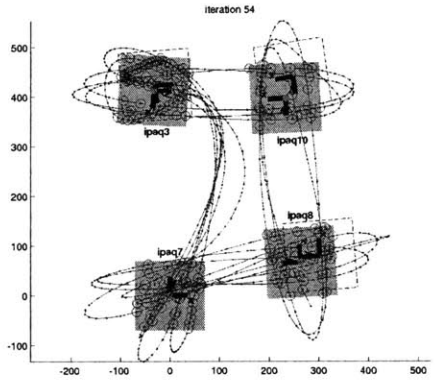


Figure 6-5: Recovering curved trajectories.

three terms in (6.4). Each measurement $(t, i) \in \mathcal{Z}$ introduces two elements into r_y.

To optimize a nonlinear least squares cost function such as $r^\top r$, Newton-Raphson requires the Jacobian $\mathbf{J}$ of r. Newton-Raphson maps an iterate $\chi^{(t)} = [\mu^{(t)}; x^{(t)}]$ to the next iterate by solving a linear least-squares problem:

$$\chi^{(t+1)} = \chi^{(t)} - \arg\min_{\delta} \|\mathbf{J}\delta - r\|^2. \tag{6.6}$$

The process of computing r, $\mathbf{J}$, and applying equation (6.6) is repeated until until χ converges to a fixed point.

For completeness, we derive $\mathbf{J}$ here. It is block structured and very sparse:

$$\mathbf{J} = \nabla r(\mu, x) = \left[\begin{array}{c|c} \mathbf{J}_\mu & \mathbf{J}_{\mu\mathbf{x}} \\ \hline \mathbf{0} & \mathbf{J}_\mathbf{x} \\ \hline \mathbf{I}\ \mathbf{0} & \mathbf{0} \end{array}\right]. \tag{6.7}$$

The left block column of $\mathbf{J}$ corresponds to differentiating r with respect to μ, and its right block column corresponds to differentiating it with respect to x.

If the zth measurement in r_y came from camera i, the derivative of r_y with respect to parameters p^i and θ^i of this camera introduces a block row into $\mathbf{J}_\mu$ at location (z, i).

$$[J_\mu]_{z,i} = -\sigma_y^{-1} \left[\ \mathbf{R}^i,\ \ ((\mathbf{C}x_t - p^i)^\top \otimes \mathbf{I})\, \tfrac{d\vec{\mathbf{R}}}{d\theta}\ \right], \tag{6.8}$$

where we have used the identity $\mathbf{X}\vec{\mathbf{Y}} = (\mathbf{Y}^\top \otimes \mathbf{I})\vec{\mathbf{X}}$ [55], where $\otimes$ is the Kronecker product, $\vec{\cdot}$ stacks elements of a matrix into a column, and $\mathbf{I}$ is the 2×2 identity matrix.

Differentiating the zth element of r_y with respect to the observed trajectory element x_t introduces a block into $\mathbf{J}_{\mu\mathbf{x}}$ at location (z, t):

$$[\mathbf{J}_{\mu\mathbf{x}}]_{z,t} = \sigma_y^{-1}\mathbf{R}^i\mathbf{C}, \tag{6.9}$$

where i is the number of the camera that made the zth observation.

Differentiating r_x with respect to μ yields 0. Differentiating it with respect to x yields $\mathbf{J}_x = \mathbf{G}$.

r_μ is only a function of the parameters of the first sensor. Its derivative with respect to μ^1 is $\mathbf{I}$, where $\mathbf{I}$ is the 3×3 identity matrix.

6.8 Conclusion

In this chapter, we have shown how to calibrate a network of non-overlapping cameras by learning the parameters of a function that maps the ground plane position of a target to its pixel coordinates on the image plane of a camera. Because the form of the projection function is known *a priori*, no overlap was necessary between the fields of view of the cameras, allowing the target to remain invisible during much of its trajectory. Our solution is framed as a joint MAP estimation camera pose

parameters and the trajectory of a calibration target. The prior on the trajectory is a linear Gaussian Markov chain, which allows for both nonlinear paths and speed changes when the target is outside the field of view of the cameras. Information about the target's dynamics allows the system to reason about the behavior of the target when it is not visible, compensating for the lack of overlap between the cameras. We demonstrated our system with a network of battery-operated cameras that are easy to deploy.

Chapter 7

Conclusion

I have presented an algorithm that can learn a mapping between two time series. The algorithm takes as input examples of how to map individual samples of the input time series to corresponding samples of the output time series. Because it assumes that the output time series follows known dynamics, it can also take advantage of unlabeled input samples. Our algorithm searches for a smooth memoryless 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. I showed its effectiveness by using it to learn trackers.

This work augments previous work on example-based tracking with a prior on the dynamics of the output. Utilizing this prior can significantly reduce the number of examples required to learn trackers by making it possible to leverage unlabeled data. When learning visual trackers, we were able to recover the pose of articulated bodies, and the contour of deformable shapes with very few labeled frames. The algorithm was also able to learn the complicated mapping between signal strength measurements induced by an RFID tag in a set of antennae to the position of an RFID tag. For these tasks, this algorithm required significantly fewer examples than fully-supervised regression algorithms or semi-supervised learning algorithms that do not take the dynamics of the output time series into account.

An unsupervised version of this algorithm elucidated its relationship with various nonlinear system identification and manifold learning techniques. This unsupervised learning algorithm approximately learns the inverse of the observation function in a hidden Markov model with known linear-Gaussian dynamics when the observation function is invertible. Adopting a nearest-neighbors representation for the mapping learned by the algorithm reveals that our algorithm is similar to the LLE algorithm, minus the initial fitting step of LLE, and with the addition of a prior that favors low-dimensional coordinates that exhibit temporal coherency.

The unsupervised version of the algorithm can learn to track in some cases without recourse to labeled data. We were able to recover the rotation of a rigidly rotating object in a synthetically generated video sequence. The algorithm assumes only that the mapping between the appearance of the object and its pose was smooth, and that the pose of the object evolved smoothly over time. We also recovered the mapping

between signal strength measurements and the position of the RFID tag using only these smoothness assumptions, without recourse to any labeled points. Finally, we showed how to track a smoothly moving target in a network of sensors where the measurement function of the nodes were unknown arbitrary smooth functions of the position of the target.

The success of our algorithms on these tracking problems points out some features of these problems:

1. The relationship between poses and observations can be approximated well by a smooth function.

2. The domain-specific knowledge required to track can be captured by a few labeled examples, and some unlabeled examples.

3. Linear-Gaussian dynamics and unlabeled data convey the domain-specific knowledge about the tracking problem. Sometimes, labeled examples are not necessary at all.

It would be interesting to take a more comprehensive inventory of the field and identify tracking problems beyond those presented in this thesis that adhere to these traits, as these are tracking problems can be solved with little effort by supplying a few labeled examples.

7.1 Future Work

Several interesting directions remain to be explored. I would like to apply the semi-supervised learning algorithm to more application areas, and to extend this work by exploring various kernels that could potentially 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 were the only factors governing the appearance of the target. This allowed us to use a simple Gaussian kernel to compare observations. To address the complications which arise in video sequences with dynamic backgrounds or other distractors, Agarwal and Triggs [1] summarized images by a list of salient features and their descriptors. It would be interesting to pursue a similar path with our algorithm. Along these lines, I would like to use the algorithms of Grauman et al. [29] and those referenced within to compute the similarity matrix. These may provide more invariance to distractors than afforded by the Gaussian kernel.

We have also assumed that *a priori* the components of the output evolve independently of each other. This was an adequate approximation in the cases we explored, but in some settings, such as when tracking articulated objects in 3D with a weak-perspective camera, the *a priori* correlation between the outputs becomes an important cue. It would be interesting to examine the benefits of correlated priors on the output.

Section 4.5 showed that although the semi-supervised learning algorithm is stable under large variations of its parameters, its output is very sensitive to the choice of labeled examples. We provided some guidelines for choosing these labeled examples, but it would be interesting to treat this as a pool-based active learning problem [71, 72, 95]. Given an initial set of labeled data points, these algorithms prompt a teacher to label points that would lower an expected loss over the remaining unlabeled data. The techniques used in [71, 72, 95] show that this loss can be approximated using the current learner. The appendix defines a probability distribution whose peak is found by our semi-supervised learning algorithm. The active learning loss in our case could be defined by the probability or marginal variances assigned to the recovered labels. Further empirical studies are needed to determine if selecting points to label according to this loss would indeed improve results on our data sets.

In the future, I hope to explore many other application areas. For example, we can 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. The semi-supervised learning algorithm can be used as a rapid-prototyping environment for building signal processing applications. This would allow designers and engineers to quickly evaluate the difficulty or feasibility of a problem by first attempting to solve it with this tool. It would also let end-users who do not have advanced engineering know-how to easily build and customize their own signal processing applications. Some of these possibilities are described in more detail here.

The arm tracking experiments showed that it is possible to learn mappings from images to graphical primitives. Extending this idea, one could learn a mapping from a video sequence of the gestures of an animator to a graphical representation of a cartoon puppet. This would allow, for example, an animator to animate the face of a cartoon character by performing hand gestures that are evocative of various emotional states. Defining such natural mappings programmatically is very difficult [56], but I hope that allowing an animator to specify them with examples will prove to be much more intuitive. Along the same lines, one can also cartoonify video sequences, converting frames of arbitrary scenes to cartoons drawings. Existing tooning techniques [2, 3] base their transformation on the contours of objects in the scene. I hope that the semi-supervised learning algorithm will allow animators to define almost arbitrary mappings from video sequences to cartoons, allowing for tooning that is not tied to object contours.

We could also learn mappings from images to images, or from images to deformations of images. For example, we might be able to learn a function that maps images of someone's face to a vector field that transforms these faces to the face of another character. The input to the algorithm would be an input video sequence, and a few frames labeled with the parameters of an active appearance model [16] or some other image transformation.

Semi-supervised learning can also be used to learn appearance models of large environments. The problem of Simultaneous Localization and Mapping (SLAM) is to build a map of the environment from images acquired from a roving robot [15, 79, 4, 48, 47, 30]. This map can be used to localize the robot at a later time from only a few snapshots of the scene. Given a video sequence of the robot roaming the

environment, and a few ground truth poses for some of these locations, it may be possible to learn a function that maps images of the environment to the robot's pose.

Semi-supervised learning is a promising approach for learning how to track, especially when combined with existing methods for preprocessing images. It may provide a similar contribution to computer animation and robotics.

Appendix A

Probabilistic Interpretations

There are two Bayesian interpretations for the optimization (3.2). In the general case, we can say that the optimization finds joint MAP estimates of g and $\mathbf{Y}$ in a conditional random field that directly specifies the distribution $p(\{y_i\}_\mathcal{I}, g|\{x_i\}_\mathcal{I})$. If certain conditions hold, the optimization (3.2) can also be interpreted as MAP estimates in a generative model for the observed inputs.

The conditional random field interpretation is obtained by directly modeling the joint conditional distribution over the variables of interest, and letting it factorize according to:

$$p(\{y_i\}_{1\cdots T}, \{z_i\}_\mathcal{L}, g|\{x_i\}_\mathcal{I}) = \frac{1}{Z}\phi(g)\phi(\{y_i\}_{1\cdots T})\prod_{i=1}^{T}\phi_i(y_i, g; x_i)\prod_{i\in\mathcal{L}}\phi_i(z_i, g; x_i), \quad \text{(A.1)}$$

with the identification

$$\phi_i(y_i, g; x_i) = \prod_d \exp\left(-w_i V(g(x_i), y_i)\right) \quad \text{(A.2)}$$

$$\phi(g) = \prod_d \exp(-\lambda_k \|g^d\|_k^2) \quad \text{(A.3)}$$

$$\phi(\{y_i\}_{1\cdots T}) = \exp\left(-\lambda_s \mathcal{S}(\{y_i\}_{1\cdots T})\right), \quad \text{(A.4)}$$

where $w_i = \lambda_l$ for $i \in \mathcal{L}$, and $w_i = 1$ otherwise. The normalizing constant Z depends only on $\{x_i\}_\mathcal{I}$, and on the parameters of the model. Figure A-1 depicts the independence relations in the random field corresponding to this cost function.

For our application, the interesting quantity is the mode of $p(\{y_i\}_{\{1\cdots T\}}, g|\{x_i\}_\mathcal{I}, \{z_i\}_\mathcal{L})$, which can be found by optimizing over the missing labels and g in Equations (A.1-A.4). Taking logs and eliminating constants reveals that this is the operation being performed by (3.2).

This random field model may be useful for defining error bars on estimates, or for justifying various Bayesian operations, such as maximizing over one of the variables and marginalizing over the other. For example, if we are only interested in the parameters of the mapping g, we should seek the MAP g while marginalizing over $\mathbf{Y}$. On the other hand, if we are only interested in labeling this a particular sequence

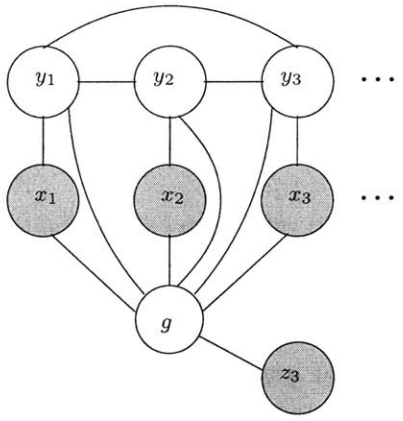


Figure A-1: A conditional random field interpretation of the cost functional in (3.2). A function g forces agreement between the input samples x_t and the outputs y_t. The output must evolve according to known dynamics. Some y_t's are labeled.

and g is of no further interest, we should optimize over $\mathbf{Y}$ with g marginalized out. When the loss V is quadratic, as is the case throughout this thesis, g and $\mathbf{Y}$ are jointly Gaussian *a posteriori* under the conditional random field interpretation, so the joint maxima we obtain in (3.2) will be identical to the maxima we would obtain by marginalizing over one of the variables and maximizing over the other. When the loss V is not quadratic, the joint maxima will will in general differ from from the maximum over one variable when the other is marginalized.

While it provides justification for certain operations, the random field model provides no insight into the process that generated the observations. The potential $\phi(y, g; x)$ favors compatibility between y and g, but it cannot be interpreted as a model for how x's are generated from y's, or even vice-versa. Generative models, on the other hand, make explicit assumptions about the way observations are generated, thus exposing falsifiable assumptions about the model, which in turn allows us to fine-tune the mode if these assumptions turn out to be unreasonable. Furthermore, generative models permit a wider set of operations, such drawing samples from the input time series $\mathbf{X}$ if we so desired.

In some cases, we may interpret the optimization (3.2) as computing a joint MAP estimate of $\mathbf{Y}$ and g in a generative model. The joint distribution over $\mathbf{Y}$, g, and $\mathbf{X}$ factorizes according to

$$p(\{y_i\}_{1\cdots T}, \{z_i\}_{\mathcal{L}}, \{x_i\}_{\mathcal{I}}, g) = \prod_{i=1}^{T} p(x_i|g, y_i) \prod_{i\in\mathcal{L}} p(x_i|g, z_i) p(g) p(\{y_i\}_{\mathcal{I}}). \tag{A.5}$$

The peak of $p(\mathbf{Y}, g|\{x_i\}_{\mathcal{I}}, \{z_i\}_{\mathcal{L}})$ can then be obtained by optimizing (A.5) over $\mathbf{Y}$ and g. One might suspect that the following identification results in the optimization

(3.2):

$$p(x_i|y_i, g) \propto \exp(-w_i V(g(x_i), y_i)), \tag{A.6}$$

$$p(g) \propto \prod_d \exp(-\lambda_k \|g^d\|_k^2) \tag{A.7}$$

$$p(\{y_i\}_{1\cdots T}) \propto \exp\left(-\lambda_s \mathcal{S}(\{y_i\}_{1\cdots T})\right) \tag{A.8}$$

where $w_i = \lambda_l$ for $i \in \mathcal{L}$, and $w_i = 1$ otherwise. But this factorization is problematic because in general $p(x_i|g, y_i)$ has a normalization constant that depends on g and y_i. When estimating the MAP of g and $\mathbf{Y}$, this normalization term does not vanish, introducing terms that do not appear in (3.2).

The MAP estimate under this model and the optimum of (3.2) can be made to coincide by suitably modifying both models. For example, Let be V be quadratic, and constrain g to be a diffeomorphism (a differentiable and one-to-one mapping, with a differentiable inverse) and locally volume preserving, so that the determinant of its partial, $|\frac{\partial g}{\partial x}|$, is constant for all x. This is true, for example, if g is constrained to be isometric [19], with $g : \mathcal{R}^N \rightarrow \mathcal{R}^N$, or mapping an N-dimensional manifold embedded in $\mathcal{R}^M$ to $\mathcal{R}^N$. The normalization constant for distribution of the form $p(z_i|y_i) = \kappa \exp(-w_i\|z_i - y_i\|^2)$ is $\kappa = (w_i/\pi)^{\frac{N}{2}}$, which is independent of z_i^d. A change of variable via the map $z_i = g(x_i)$ results in $p(x_i|y_i, g) = \kappa \left|\frac{\partial g}{\partial z}\right|^{-1} \exp\left(-w_i\|g(x_i) - y_i\|^2\right)$ [58]. Because g is locally volume-preserving, we get $p(x_i|y_i, g) = \kappa' \exp(-w_i\|g(x_i) - y_i\|^2)$, where κ' is again constant with respect to g and x_i. Thus, under the quadratic penalty, when g is a volume-preserving diffeomorphism, finding the MAP $\mathbf{Y}$ and g in the generative model (A.5), with the identification (A.6-A.8), is equivalent to the optimization (3.2).

After inferring $\mathbf{Y}$ and g, the Bayesian interpretations provide guidance towards transforming a new sequence $\mathbf{X}^{new}$ to an output sequence $\mathbf{Y}^{new}$. Ideally, to transform $\mathbf{X}^{new}$, we would find the peak of $p(\mathbf{Y}|\mathbf{X}^n, \{x_i\}_\mathcal{I}, \{z_i\}_\mathcal{L})$ according to either the random field model or the generative model. If we assume that the posterior over g obtained from the training phase is sharp and not perturbed by the addition of new unlabeled data (i.e. $p(g|\mathbf{X}^{new}, \{x_i\}_\mathcal{I}, \{z_i\}_\mathcal{L}) \approx \delta(g - g^*)$), this peak coincides with $\arg\max_\mathbf{Y} p(\mathbf{Y}, g^*|\mathbf{X}^{new})$, which is exactly the operation implemented by the smoother of Equation (3.14).

In summary, we can always describe the optimization in (3.2) as finding a MAP estimate of $\mathbf{Y}$ and g in a conditional random field defined by Equation (A.1). Under certain conditions, we can also interpret (3.2) as finding MAP estimates in a generative model.

Bibliography

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

[2] A. Agarwala. Snaketoonz: A semi-automatic approach to creating cel animation from video. In *International Symposium on Non-Photorealistic Animation and Rendering*, 2002.

[3] A. Agarwala1, A. Hertzmann, and S. M. Seitz D. H. Salesin1. Keyframe-based tracking for rotoscoping and animation. In *SIGGRAPH*, 2004.

[4] N. Ayache and O. Faugeras. Maintaining representations of the environment of a mobile robot. *IEEE Tran. Robot. Automat.*, 5(6):804–819, 1989.

[5] P. Bahl and V. N. Padmanabhan. Radar: An in-building rf-based user location and tracking system. In *INFOCOM*, volume 2, pages 775–784, 2000.

[6] M. Balasubramanian, E. L. Schwartz, J. B. Tenenbaum, V. de Silva, and J. C. Langford. The isomap algorithm and topological stability. *Science*, 295(5552), 2002.

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

[8] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[9] M. Belkin, P. Niyogi, and V. Sindhwani. Manifold regularization: a geometric framework for learning from examples. Technical Report TR-2004-06, University of Chicago CS Technical Report, 2004.

[10] Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel pca. *Neural Computation*, 10(16):2197–2219, 2004.

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

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

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

[14] M. Brand. Charting a manifold. In *Neural Information Processing Systems (NIPS)*, 2002.

[15] J. A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tards. The SPmap: A probabilistic framework for simultaneous localization and map building. *IEEE Transactions on Robotics and Automation*, 15:948–953, October 1999.

[16] T.F. Cootes, G.J. Edwards, and C.J. Taylor. Active appearance models. *Pattern Analysis and Machine Intelligence (PAMI)*, 23(6):681–684, June 2001.

[17] M.R. Cowper, B. Mulgrew, and C.P. Unsworth. Nonlinear prediction of chaotic signals using a normalised radial basis function network. *Signal Processing*, 82:775–789, 2002.

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

[19] D.L. Donoho and C. Grimes. Hessian eigenmaps: new locally linear embedding techniques for highdimensional data. Technical report, TR2003-08, Dept. of Statistics, Stanford University, 2003.

[20] G. Doretto, A. Chiuso, and Y.N. Wu S. Soatto. Dynamic textures. *International Journal of Computer Vision (IJCV)*, 51(2):91–109, 2003.

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

[22] S. Elanayar and Y. C. Shin. Radial basis function neural network for approximation and estimation of nonlinear stochastic dynamic systems. *IEEE Transactions on Neural Networks*, 5(4), 1994.

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

[24] R. B. Fisher. Self-organization of randomly placed sensors. In *European Conference on Computer Vision (ECCV)*, volume 4, pages 146–160, 2002.

[25] D.R. Fokkema, G.L.G. Sleijpen, and H.A. van der Vorst. Jacobi-davidson style qr and qz algorithms for the reduction of matrix pencils. *SIAM J. Sc. Comput.*, 20(1):94–125, 1998.

[26] Z. Ghahramani and S. Roweis. Learning nonlinear dynamical systems using an em algorithm. In *Neural Information Processing Systems (NIPS)*, pages 431–437, 1998.

[27] G. Golub and C.F. Van Loan. *Matrix Computations.* The Johns Hopkins University Press, 1989.

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

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

[30] D. Hähnel, D. Fox, W. Burgard, and S. Thrun. A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In *Proceedings of the Conference on Intelligent Robots and Systems (IROS)*, 2003.

[31] J.H. Ham, D.D. Lee, and L.K. Saul. Learning high dimensional correspondences from low dimensional manifolds. In *ICML*, 2003.

[32] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[33] B. K. P. Horn. Relative orientation. *International Journal of Computer Vision (IJCV)*, 4:59–78, January 1989.

[34] O. Javed, Z. Rasheed, O. Alatas, and M. Shah. $KNIGHT^{TM}$: A real time surveillance system for multiple overlapping and non-overlapping cameras. In *International Conference on Multimedia and Expo*, July 2003.

[35] O. Javed, K. Shafique, and M. Shah. Appearance modeling for tracking in multiple non-overlapping cameras. In *Computer Vision and Pattern Recognition (CVPR)*, 2005.

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

[37] T. Joachims. Transductive inference for text classification using support vector machines. In *International Conference on Machine Learning*, pages 200–209, 1999.

[38] 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*, 31(12):1725–1750, 1995.

[39] S. Julier and J. Uhlmann. A new extension of the kalman filter to nonlinear systems. In *Int. Symp. Aerospace/Defense Sensing, Simul. and Controls*, 1997.

[40] W. E. L. Grimson K. Tieu, G. Dalley. Inference of non-overlapping camera network topology by measuring statistical dependence. In *International Conference on Computer Vision (ICCV)*, 2005.

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

[42] V. Kettnaker and R. Zabih. Counting people from multiple cameras. In *ICMCS*, volume 2, pages 267–271, 1999.

[43] S. Khan and M. Shah. Consistent labeling of tracked objects in multiple cameras with overlapping fields of view. *Pattern Analysis and Machine Intelligence (PAMI)*, 25(10), October 2003.

[44] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conf. on Machine Learning*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[45] A. LaMarca, J. Hightower, I. Smith, and S. Consolvo. Self-mapping in 802.11 location systems. In *Ubicomp*, 2005.

[46] N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in Neural Information Processing Systems (NIPS)*, 2004.

[47] J. J. Leonard and P. M. Newman. Consistent, convergent, and constant-time SLAM. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1143–1150, 2003.

[48] Y. Liu, R. Emery, D. Chakrabarti, W. Burgard, and S. Thrun. Using EM to learn 3D models of indoor environments with mobile robots. In *IEEE International Conference on Machine Learning (ICML)*, 2001.

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

[50] J. Ting-Ho Lo. Synthetic approach to optimal filtering. *IEEE Transactions on Neural Networks*, 5(5), 1994.

[51] D. Makris, T. Ellis, and J. Black. Bridging the gaps between cameras. In *Computer Vision and Pattern Recognition (CVPR)*, 2004.

[52] D. Marinakis and G. Dudek. Topology inference for a vision-based sensor network. In *Canadian Conference on Computer and Robot Vision*, 2005.

[53] D. Marinakis, G. Dudek, and D. Fleet. Learning sensor network topology through monte carlo expectation maximization. In *International Conference on Robotics and Automation*, 2005.

[54] T.P. Minka. Expectation-maximization as lower bound maximization. Technical report, Media Lab, `http://www.media.mit.edu/~tpminka/papers/em.html`, 2001.

[55] T.P. Minka. Old and new matrix algebra useful for statistics. Technical report, Media Lab, `http://www.media.mit.edu/~tpminka/papers/matrix.html`, 2001.

[56] P. Nemirovsky. Control of signal processing algorithms using the matrix interface. In *Proceedings of the 114th Audio Engineering Society Convention*, 2003.

[57] V. Otsason, A. Varshavsky, A. La Marca, and Eyal de Lara. Accurate gsm indoor localization. In *UbiComp*, 2005.

[58] A. Papoulis. *Probability, random variables, and stochastic processes.* McGraw-Hill, New York, third edition, 1991.

[59] A. G. Parlos, S. K. Menon, and A. F. Atiya. An adaptive state filtering algorithm for systems with partially known dynamics. *Transactions of the ASME*, 124, 2002.

[60] H. Pasula, S. J. Russell, M. Ostland, and Y. Ritov. Tracking many objects with many sensors. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1160–1171, 1999.

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

[62] N. Patwari and A. O. Hero. Manifold learning algorithms for localization in wireless sensor networks. In *International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2004.

[63] R. Pless and I. Simon. Using thousands of images of an object. In *CVPRIP*, 2002.

[64] F.M. Porikli. Inter-camera color calibration by cross-correlation model function. In *International Conference on Image Processing (ICIP)*, volume 2, September 2003.

[65] J. Principe, D. Xu, and J. Fisher. *Unsupervised Adaptive Filtering*, chapter Information Theoretic Learning. Wiley, 1999.

[66] J. C. Principe, A. Rathie, and J. M. Kuo. Prediction of chaotic time series with neural networks and the issue of dynamic modeling. *Int. J. of Bifurcation and Chaos*, 2(4):989–996, 1992.

[67] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[68] A. Rahimi, B. Dunagan, and T. Darrell. Simultaneous calibration and tracking with a network of non-overlapping sensors. In *Computer Vision and Pattern Recognition (CVPR)*, pages 187–194, 2004.

[69] 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*, 190, 2003.

[70] S. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

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

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

[73] B. Schölkopf, R. Herbrich, A.J. Smola, and R.C. Williamson. A generalized representer theorem. Technical Report 81, NeuroCOLT, 2000.

[74] B. Schölkopf, A. Smola, and K-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998.

[75] M. Seeger. Gaussian processes for machine learning. *Int. J. Neural Syst.*, 14(2):69–106, 2004.

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

[77] Y. Shang, W. Ruml, Y. Zhang, and M. Fromherz. Localization from mere connectivity. In *MobiHoc*, 2003.

[78] R. H. Shumway and D. S. Stoffer. *Time Series Analysis and Its Applications.* Springer Texts in Statistics, 2005.

[79] R. Smith, M. Self, and P. Cheeseman. Estimating uncertain spatial relationships in robotics. In *Uncertainity in Artificial Intelligence*, 1988.

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

[81] C. Stauffer and K. Tieu. Automated multi-camera planar tracking correspondence modeling. In *Computer Vision and Pattern Recognition (CVPR)*, volume 1, June 2003.

[82] G. P. Stein, R. Romano, and L. Lee. Monitoring activities from multiple video streams: Establishing a common coordinate frame. Technical Report AIM-1655, MIT AI Lab, 1999.

[83] M. Szummer and R. W. Picard. Temporal texture modeling. In *International Conference on Image Processing (ICIP)*, pages 823–826, 1996.

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

[85] S. Tekinay. Wireless geolocation systems and services. *Special Issue of the IEEE Communications Magazine*, April 1998.

[86] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[87] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustment – a modern synthesis. In W. Triggs, A. Zisserman, and R. Szeliski, editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag, 2000.

[88] R.Y. Tsai. Multiframe image point matching and 3-D surface reconstruction. *Pattern Analysis and Machine Intelligence (PAMI)*, 5(2):159–173, March 1983.

[89] H. Valpola. Nonlinear independent component analysis using ensemble learning theory. In *Proceedings of the International Workshop on Independent Component Analysis and Blind Signal Separation*, pages 351–356, 2000.

[90] H. Valpola and J. Karhunen. An unsupervised ensemble learning method for nonlinear dynamic state-space models. *Neural Computation*, 14(11):2647–2692, 2002.

[91] P. vanOverschee and B. DeMoor. *Subspace Identification of Linear Systems: Theory, Implementation, Applications.* Kluwer Academic Publishers, 1996.

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

[93] G. Wahba. Spline models for observational data. *SIAM*, 59, 1990.

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

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

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