Control Systems Magazine, **14**, 1, pp.57-71.

Robot Juggling: An Implementation of Memory-based Learning

Stefan Schaal and Christopher G. Atkeson[*]

Abstract

This paper explores issues involved in implementing robot learning for a challenging dynamic task, using a case study from robot juggling. We use a memory-based local modeling approach (locally weighted regression) to represent a learned model of the task to be performed. Statistical tests are given to examine the uncertainty of a model, to optimize its prediction quality, and to deal with noisy and corrupted data. We develop an exploration algorithm that explicitly deals with prediction accuracy requirements during exploration. Using all these ingredients in combination with methods from optimal control, our robot achieves fast real-time learning of the task within 40 to 100 trials.

---

[*] Address of both authors: Massachusetts Institute of Technology, The Artificial Intelligence Laboratory & The Department of Brain and Cognitive Sciences, 545 Technology Square, Cambride, MA 02139, USA. Email: sschaal@ai.mit.edu, cga@ai.mit.edu. Support was provided by the Air Force Office of Scientific Research and by Siemens Corporation. Support for the first author was provided by the German Scholarship Foundation and the Alexander von Humboldt Foundation. Support for the second author was provided by a National Science Foundation Presidential Young Investigator Award. We thank Gideon Stein for implementing the first version of LWR on the i860 microprocessor, and Gerrie van Zyl for building the devil stick robot and implementing the first version of devil stick learning.

Introduction

Learning control means improving a motor skill by repeatedly practicing a task. There has been much progress in learning control research. But many projects test proposed algorithms only in simulation. We have found that actual implementation of learning control forces us to consider issues not adequately addressed in simulations. In this paper we describe which ingredients were needed to actually implement a learning algorithm on a robot for a complicated dynamic task.

We are exploring systems that learn by explicitly remembering their experiences in order to build models of the world. The learning community distinguishes between two different methods to represent a model, *parametric* and *nonparametric*. A *parametric* model consists of a certain mathematical function which possesses a finite set of free parameters that have to be determined to make the function fit the data. This function models *all* data simultaneously, which means that parametric models correspond to *global* function fitting. Parametric models and training methods often do not remember the data they were trained on. Standard linear regression, sigmoidal neural networks, radial basis function networks, etc., belong in this class of techniques. *Nonparametric* models also have an underlying function with a set of parameters which are to be adjusted. However, the number of the parameters can grow with the amount of data and the parameters can be recalculated whenever the model is used to generate an output from a new input (a process which is also called a lookup or query). This makes sense if not all data is taken into account to estimate the parameters but merely a subset, or if individual data points are weighted differently with respect to different query points. Common algorithms to choose the subset or the weighting are, for example, n-nearest neighbor methods (e.g., [12, 25]) or kernel regression (e.g., [19, 28, 37]). Common functions are (hyper-)planes or (hyper-)quadratic surfaces. By letting only a few data points contribute to forming the parameters, these types of nonparametric models correspond to *local* function fitting: they build a *local* model to fit a subset of data points with their function. As the word “local” implies, the model will be valid only in a restricted region. Due to the necessity of continuous recalculation of the parameters for each individual query, local nonparametric models have to memorize all data and are often called memory-based. Weighted averaging and nearest neighbor methods are presumably the best known nonparametric approaches.

We are investigating a recently developed nonparametric (memory-based) statistical technique, locally weighted regression (LWR), to model the system we are trying to control [11, 15, 16]. The LWR approach allows us to efficiently estimate local linear models for different points in the state space. LWR offers a variety of statistical tools to assess the reliability of lookups, to optimize the quality of a lookup, and also to cope with noise and corrupted data. This allows the robot to monitor its own skill level, and it provides the basis for an exploratory behavior that is almost entirely driven by the stream of incoming data from practicing the task.

Our starting point for modeling is that we assume knowledge of what constitutes a state of the system, i.e., the input/output representations, but the form of the dynamics equations of the task to be controlled is unknown. Past work tested our ideas by implementing learning for one-shot or static tasks, such as throwing a ball at a target [1], and also repetitive or dynamic tasks, such as bouncing a ball on a paddle [2] and hitting a stick back and forth (a form of juggling known as devil sticking) [40]. This as well as other experimental work (e.g., [32]) has highlighted the importance of making sure the control paradigm used is robust to uncertainty, that the robot is able to compute what is known about the task, and how well it is known, and that there is some process that generates exploration, so that models and controllers based on insufficient data are improved. All these points are addressed by the LWR learning algorithm. Using our work with the devil sticking robot as an example, this paper describes what was needed to implement real-time learning based on this algorithm.

The next section of this paper discusses a number of control approaches which make use of models and motivates the choice in our work. Locally weighted regression and some of its statistical tools are introduced afterwards. Exploration, a key feature for system identification of modeling approaches, receives attention in the fourth section where we introduce a goal-directed exploration algorithm which keeps explicit control over prediction accuracy during exploration. In the fifth section, the previously introduced methods are find application in a real-time implementation of learning how to juggle the devil stick.

Control Paradigms

Before discussing the details of our representational approach, it is useful to consider some of the alternative control paradigms that might make use of learned models.

Deadbeat Control

In considering repetitive or dynamic tasks, we will focus on nonlinear regulator design, assuming there is a desired state $\mathbf{x}_d$[1] to achieve. Since often the observations of system inputs and outputs occur at discrete time intervals and not all the derivatives of the state are typically measured, we restrict our analysis to discrete time models. The notation for the forward dynamics model of a discrete system is

---

[1] Out notation has the following conventions: scalars are denoted by lower cases letters in *italic* face (e.g., s), vectors are denoted by lower case letters in **bold** face (e.g., $\mathbf{v}$), matrices are denoted by upper case letters in **bold** face (e.g. $\mathbf{M}$), scalar valued function are in *italic* face (e.g., $f()$), vector valued function are in **bold** face (e.g., $\mathbf{f}()$), and $()^T$ denotes the transpose of a vector or matrix, whereby all vectors are originally column vectors. The ^ (caret) indicates models and predictions by models. Dots on top of variables indicate time derivatives.

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) \tag{2.1}$$

and attempts to perform the task generate experience vectors $(\mathbf{x}_k^T, \mathbf{u}_k^T, \mathbf{x}_{k+1}^T)^T$. A straightforward approach to improving performance on the task is to learn an inverse model

$$\hat{\mathbf{u}}_k = \hat{\mathbf{f}}^{-1}(\mathbf{x}_k, \mathbf{x}_{k+1}) \tag{2.2}$$

from the database of experiences and use the model to predict commands for later attempts of the task by replacing $\mathbf{x}_{k+1}$ in (2.2) by a desired state $\mathbf{x}_{k+1,desired}$. Another approach is to learn the forward model (2.1) and then search for a good command, minimizing the (by $\mathbf{Q}$ and $\mathbf{R}$ weighted) squared magnitude of the predicted state error and the command:

$$\min_{\mathbf{u}_k}\left[\left(\hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_d\right)^T \mathbf{Q}\left(\hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) - \mathbf{x}_d\right) + \mathbf{u}_k^T \mathbf{R} \mathbf{u}_k\right], \tag{2.3}$$

Eq.(2.2) and Eq.(2.3) with $\mathbf{R} = \mathbf{0}$ correspond to deadbeat control.

The deadbeat controllers above did not achieve satisfying robustness in our work since they attempt to cancel the plant dynamics entirely. A less aggressive nonlinear control approach is to locally linearize the system about the desired point, and then use one of the many linear controller design techniques, e.g., pole placement, linear quadratic (LQ), or H_∞. Such an approach is very successful if the system remains within the linear region.

Representing the Forward Model

Modeling approaches require model representations. If the nonlinear system has a particular structure, it can be globally linearized using nonlinear coordinate transformations and state feedback (feedback linearization) [36]. Any linear control design techniques may be used subsequently. Much of the recent work in adaptive controllers for nonlinear systems assumes some knowledge of the form of the nonlinearities and the plant's unknown parameters [30]. A common formulation requires the plant be representable accurately by a feedback linearizable model in which all unknown elements appear linearly as a parameter vector. A more black box approach to adaptive control [21] is to use a form of parametric Volterra series in the inputs and states. Single hidden layer perceptron-like neural network models essentially project the input data along a line given by the input weights, and then output a one dimensional function of the value of that projection. Radial basis function networks use centers of spherically symmetric contributions from parameterized one dimensional functions applied to the distance between each input and the center. All these approaches make implicit assumptions about the form of the system they are interacting with, which we want to avoid, as will be demonstrated in the next section.

Optimal Control Approaches

Learning approaches that do not commit to a particular representational form generate numerical representations, for which optimal control techniques provide natural methods to design control systems for nonlinear tasks. Dynamic programming [8, 9, 13] lays the basis for a general paradigm of nonlinear controllers. In our formulation of the regulation problem, a goal state $\mathbf{x}_d$ is given, which is typically an equilibrium state, so $\mathbf{x}_d = \mathbf{f}(\mathbf{x}_d, 0)$. A one step cost $L(\mathbf{x}, \mathbf{u})$ is defined over all states and controls. The criterion to be optimized is the infinite horizon sum of one step costs starting at the current time:

$$J = \sum_{k=1}^{\infty} L(\mathbf{x}_k, \mathbf{u}_k). \tag{2.4}$$

We typically require either a temporal discount factor or $L(\mathbf{x}_d, 0) = 0$ to ensure well defined solutions to the optimization problem. The value function $V(\mathbf{x})$ is the optimal cost created by solving (2.4) starting in state $\mathbf{x}$. At any point, a globally optimal control action can be chosen by the nonlinear controller by solving the local optimization problem:

$$\mathbf{u}^{opt} = \arg\min_{\mathbf{u}} \left[L(\mathbf{x}, \mathbf{u}) + V(f(\mathbf{x}, \mathbf{u})) \right]. \tag{2.5}$$

If one assumes a locally linear model of the plant,

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{f}}(\mathbf{x}_k, \mathbf{u}_k) \approx \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{c}, \tag{2.6}$$

a weighted locally quadratic model of the one step cost,

$$L(\mathbf{x}, \mathbf{u}) \approx \frac{1}{2}\mathbf{x}^T\mathbf{Q}\mathbf{x} + \frac{1}{2}\mathbf{u}^T\mathbf{R}\mathbf{u} + \mathbf{x}^T\mathbf{S}\mathbf{u} + \mathbf{t}^T\mathbf{u}, \tag{2.7}$$

and a locally quadratic model of the value function,

$$V(\mathbf{x}) \approx V_0 + V_x\mathbf{x} + \frac{1}{2}\mathbf{x}^T V_{xx}\mathbf{x}, \tag{2.8}$$

one can compute a locally optimal command analytically:

$$\mathbf{u}^{opt} = -\left(\mathbf{R} + \mathbf{B}^T V_{xx}\mathbf{B}\right)^{-1}\left(\mathbf{B}^T V_{xx}\mathbf{A}\mathbf{x} + \mathbf{S}^T\mathbf{x} + \mathbf{B}^T V_{xx}\mathbf{c} + V_x\mathbf{B} + \mathbf{t}\right). \tag{2.9}$$

Unfortunately, value functions are difficult to represent and to compute, even though this can be done off-line. Predictive control design techniques avoid using a value function, but are then merely locally optimal [10]. Value functions can also be approximated, e.g., with neural networks [42]. We are interested in exploring approximations to value functions that produce a locally quadratic model of $V(\mathbf{x})$ in a local neighborhood of $\mathbf{x}$.

In this paper we are working within an optimal control framework. We would like to design a fully nonlinear controller from a full computation of the optimal value function. This is currently too expensive to compute, so we use linear quadratic (LQ) regulator techniques to approximate the value function and design a corresponding controller. We make extensive use of local linear models of the system to be controlled. The linearized models are calculated on an as needed basis and are recalculated with each new piece of data to update the controller. All of this happens in real time as the robot is executing the task.

Locally Weighted Regression

The point of view explored in this paper is that the goal of a learning system for robots is to be able to build internal models of tasks during execution of those tasks. These models are multidimensional functions that are approximated from sampled data (the previous experiences or attempts to perform the task). The learned models are used in a variety of ways to successfully execute the task. We would like the models to incorporate the latest information. The models will be continuously updated with a stream of new training data, so updating a model with new data should take a short period of time. There are also time constraints on how long it can take to use a model to make a prediction. Because we are interested in control methods that make use of local linearizations of the plant model, we want a representation that can quickly compute a local linear model of the represented transformation. We would also like to minimize the negative interference from learning new knowledge on previously stored information.

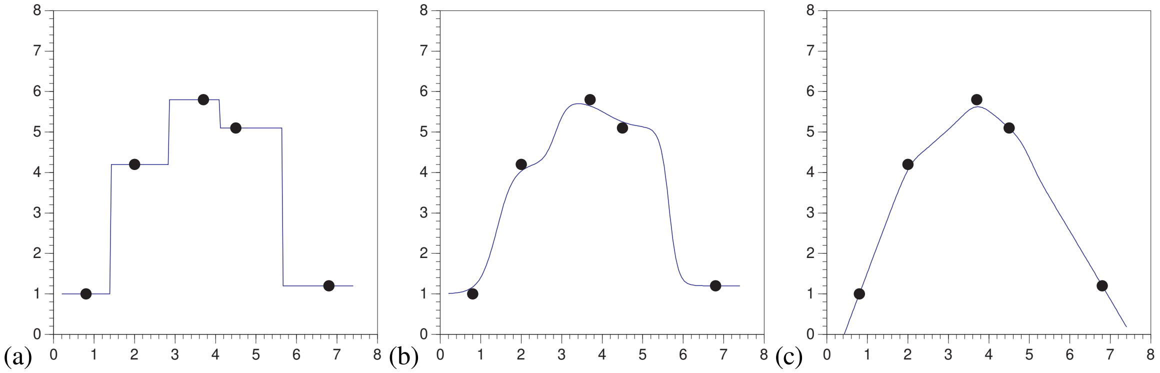


Figure 1: Characteristic performance of three different nonparametric function approximation techniques: (a) nearest neighbor; (b) weighted average; (c) locally weighted regression

As the most generic approximator that satisfies many of these criteria, we explore a version of memory-based learning techniques called locally weighted regression (LWR). [15, 16, 11, 6, 24, 14, 27]. A memory-based learning (MBL) system is trained by storing the training data in a memory. This allows MBL systems to achieve real-time learning. MBL avoids interference between new and old data by retaining and using all the data to answer each query. MBL approximates complex functions using simple local models, as does a Taylor series. Examples of types of local models include nearest neighbor, weighted average, and locally weighted regression. Each of these local models combine points near to a query point to estimate the appropriate output. Figure 1 shows typical curve fits for each of these methods.

Locally weighted regression uses a relatively complex regression procedure to form the local model, and is thus more expensive than nearest neighbor and weighted average memory-based learning procedures. For each query a new local model is formed. The rate at which local models can be formed and evaluated limits the rate at which queries can be answered. This paper describes how locally weighted regression can be implemented in real time.

An unweighted regression finds the solution to the equations:

$$\mathbf{y} = \mathbf{X} \cdot \beta \tag{3.1a}$$

by solving the normal equations:

$$\mathbf{X}^{\mathrm{T}}\mathbf{X}\beta = \mathbf{X}^{\mathrm{T}}\mathbf{y}, \tag{3.1.b}$$

where $\mathbf{X}$ is an $m \times (n+1)$ matrix consisting of m data points, each represented by its n input dimensions and a "1" in the last column, $\mathbf{y}$ is a vector of corresponding outputs for each data point, β is the $n+1$ vector of unknown regression parameters, and J is the sum of squared errors over all given data points (cf. Table 1, Appendix A). Solving for β yields

$$\beta = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{y}, \tag{3.2}$$

and a prediction of the outcome of a query point $\mathbf{x}_q$ becomes:

$$\hat{y}_q = \mathbf{x}_q^T\beta. \tag{3.3}$$

However, this gives distant points equal influence with nearby points on the ultimate answer to the query, for equally spaced data. To weight similar points more, locally weighted regression is used. First, a distance is calculated from each of the stored data points (rows in the $\mathbf{X}$ matrix) to the query point $\mathbf{x}_q$:

$$d_i^2 = \sum_{j=1}^{n} s_j \left(\mathbf{X}_{ij} - \mathbf{x}_{q_j}\right)^2. \tag{3.4}$$

The factor s_j reflects a positive weighting (distance metric) among the n input dimensions, either to normalize those or to give them different importance. The weight for each stored data point is a function of the distance (3.4):

$$w_i = f\left(d_i^2\right). \tag{3.5}$$

Each row i of $\mathbf{X}$ and $\mathbf{y}$ is multiplied by the corresponding weight w_i. A simple weighting function just raises the distance (3.4) to a negative power, which determines how local the regression will be (the rate of drop-off of the weights with distance):

$$w_i = \frac{1}{d_i^k}. \tag{3.6}$$

This type of weighting function goes to infinity as the query point approaches a stored data point which forces the locally weighted regression to exactly match that stored point. If the data is noisy, exact interpolation is not desirable, and a weighting scheme with limited magnitude is more appropriate. One such scheme, which we use in what follows, is a Gaussian kernel:

$$w_i = \exp\left(\frac{-d_i^2}{2k^2}\right). \tag{3.7}$$

The parameter k scales the size of the kernel to determine how local the regression will be. Such a weighting is used in Figure 1b and Figure 1c.

A potential problem is that the data points may be distributed in such a way as to make the regression matrix $\mathbf{X}$ singular. Ridge regression is used to prevent problems due to a singular data matrix. The following equation, with $\mathbf{X}$ and $\mathbf{y}$ already weighted, is solved for β:

$$\left(\mathbf{X}^T\mathbf{X} + \Lambda\right)\beta = \mathbf{X}^T\mathbf{y}, \tag{3.8}$$

where Λ is a diagonal matrix with small positive diagonal elements λ_i^2. Ridge regression is equivalent to adding fake data in each direction that has a small weight and a zero output value. The ridge regression constants can also be thought of as Bayesian priors on the variance of the estimated parameter vector β.

Assessing the computational cost

A lookup in a LWR model has three stages: forming weights, forming the regression matrix, and solving the normal equations. Let us examine how the cost of each of these stages grows with the size of the data set and dimensionality of the problem. We will assume a linear local model.

Forming and applying the weights involves scanning the entire data set, so it scales linearly with the number of data points in the database m. For each of n input dimensions there are a constant number of operations, so the number of operations scales linearly with the number of input dimensions. Note that we can eliminate points whose distance exceeds a threshold, reducing the number of points considered in subsequent computational stages.

Each element of $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$ is the inner (dot) product of two columns of $\mathbf{X}$ or $\mathbf{y}$. The architecture of digital signal processors is ideally suited for this computation, which consists of repeated multiplies and accumulates. The computation is linear in the number of rows m and quadratic in the number of columns $\left(n^2 + n*o\right)$, where o is the number of output dimensions.

Solving the normal equations is done using a LDL^T decomposition, which is cubic in the number of input dimensions, and independent of the number of data points. Other more sophisticated and more expensive decompositions, such as the singular value decomposition, are unnecessary since the ridge regression procedure guarantees well-conditioned normal equations.

The most straightforward parallel implementation of LWR would distribute the data points among several processors. Queries can be broadcast to the processors, and each processor can weight its data set and form its contribution to $\mathbf{X}^T\mathbf{X}$ and $\mathbf{X}^T\mathbf{y}$. These contributions can be summed and the full normal equations solved on a single processor. The communication costs are linear in the number of processors, quadratic in the number of columns $\left(n^2 + n*o\right)$, and independent of the total number of points.

We have implemented the local weighted regression procedure on a 33MHz Intel i860 microprocessor. The peak computation rate of this processor is 66 MFlops. We have achieved effective computation rates of 20 MFlops on a learning problem with $n = 10$ input dimensions and $o = 5$ output dimensions, using a linear local model. This leads to a lookup time of approximately 15 milliseconds on a database of $m = 1000$ points.

Tuning The Fit Parameters

In the past we have used off-line global cross validation ([41] to estimate reasonable values for the fit parameters: the distance metric s_j, the parameters that define the weighting function $w_i = f(d_i^2)$, and the ridge regression parameters λ_j. Since we are using a local model that is linear in the unknown parameters, we can compute derivatives of the cross validation error $e_i = \hat{y}_i - y_i$ with respect to the fit parameters:

$$\frac{\partial e_i}{\partial s_j}, \frac{\partial e_i}{\partial k}, \frac{\partial e_i}{\partial \lambda_j},$$

and minimize the sum of the squared cross validation error using a Levenberg-Marquardt (nonlinear least squares) procedure (MINPACK, NL2SOL).

However, it is clear that these parameters should depend on the location of the query point. In this section we describe new procedures that estimate local values of the fit parameters optimized for the site of the current query point. We want to demonstrate the differences between local and global fitting in an example where we only focus on the kernel width k of a Gaussian weighting function (3.7). In Figure 2a, a noisy data set of the function $y = x - \sin^3(2\pi x^3)\cos(2\pi x^3)\exp(x^4)$ was fitted by locally weighted regression with a globally optimized, i.e. constant, k. In the left half of the plot, the regression starts to fit noise because k had to be rather small to fit the high frequency regions on the right half of the plot. The prediction intervals, which will be introduced below, indicate high uncertainty in several places. To avoid such undesirable behavior, a local optimization criterion is needed. Standard linear regression analysis provides a series of well-defined statistical tools to assess the quality of fits, such as coefficients of determination, t-tests, F-test, the PRESS-statistic, Mallow's C_p-test ([23], confidence intervals, prediction intervals, and many more (e.g., [29]). These tools can be adapted to locally weighted regression. We do not want to discuss all possible available statistics here but rather focus on two that have proved to be useful.

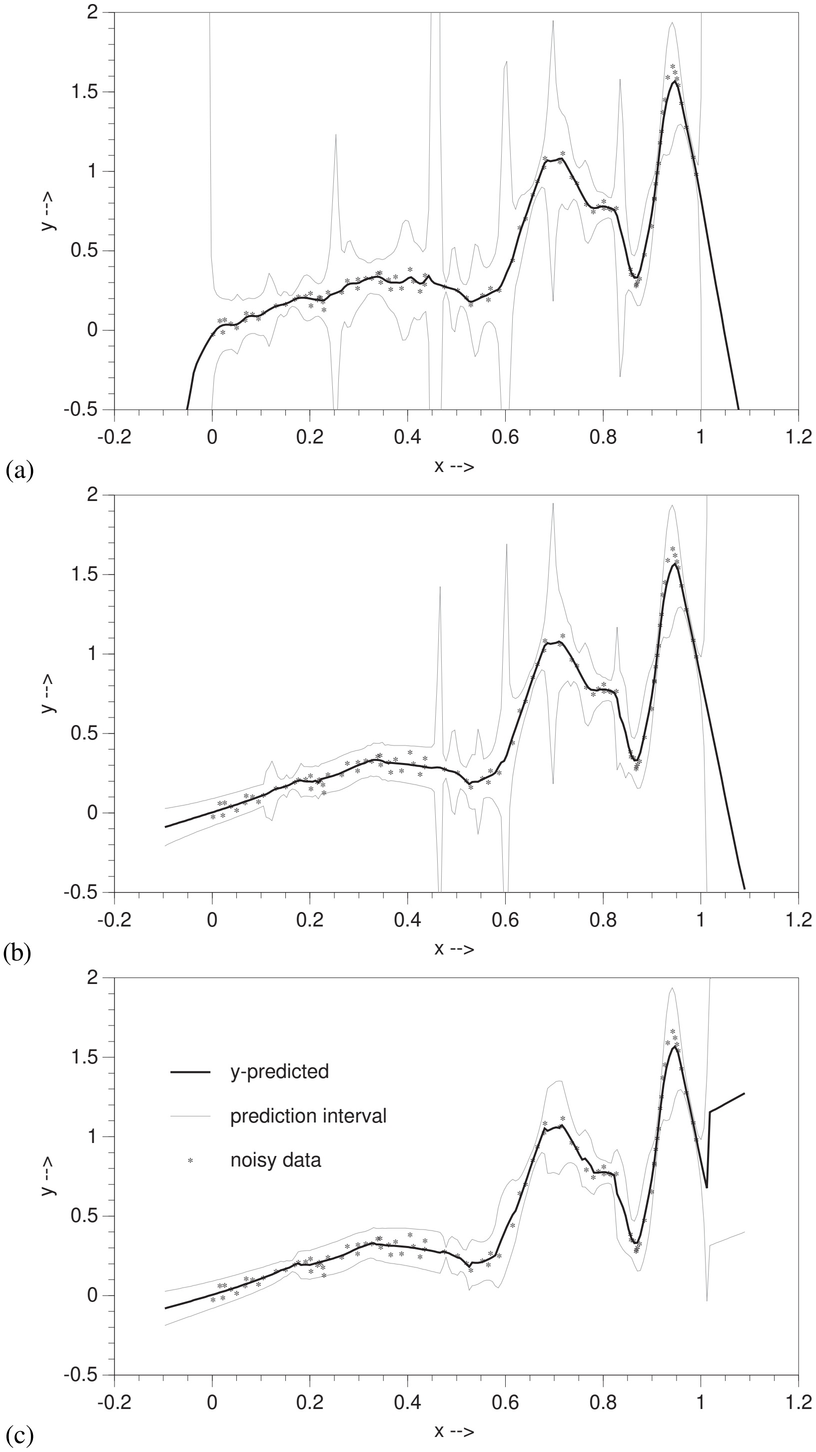


Figure 2 Optimizing the LWR fit using different measures: (a) global cross validation, (b) local cross validation, (c) local prediction intervals

Cross validation has a relative in linear regression analysis, the PRESS residual error. The PRESS statistic performs leave-one-out cross validation computationally very efficient by not requiring recalculation of the regression parameters for every excluded point. Table 1 in Appendix A shows how the PRESS residual can be expressed as a mean squared cross validation error MSE_{cross}. In Figure 2b, the same data as in Figure 2a was fitted by adjusting k to minimize MSE_{cross} *at each query point*. The outcome is much smoother than that of global cross validation, and also the prediction intervals are narrower. It should be noted that the extrapolation properties on both sides of the graph are quite appropriate (compared to the known underlying function), in comparison to Figure 2a and Figure 2c.

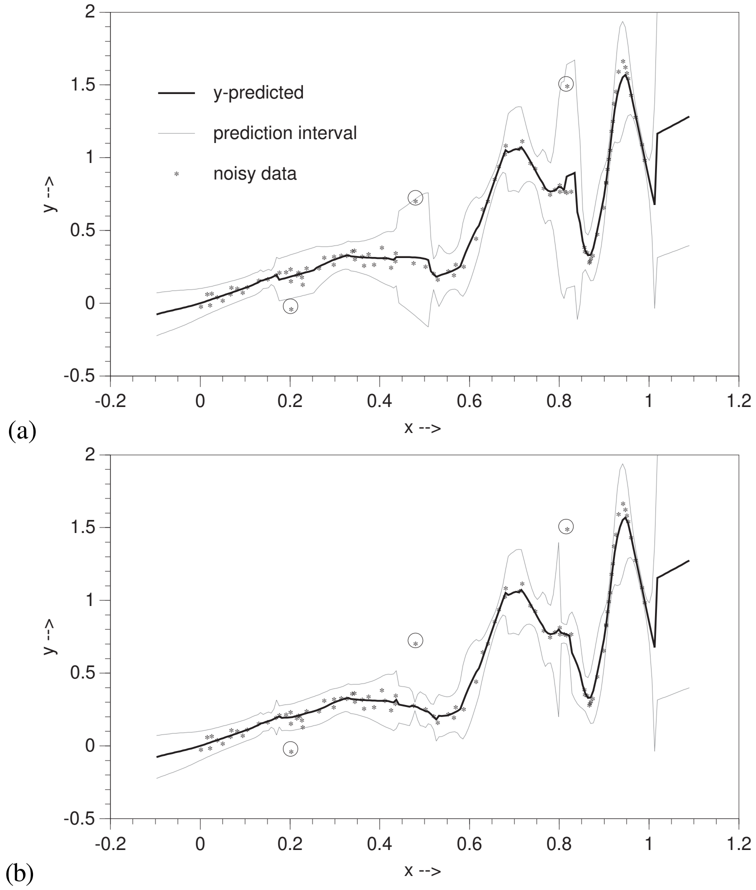


Figure 3: Influence of outliers on LWR: (a) no outlier removal, (b) with outlier removed

Prediction intervals I_q are expected bounds of the prediction error at a query point $\mathbf{x}_q$. Table 1 gives the appropriate definition for LWR; its derivation can be found in most text books on regression analysis (e.g., [29]). Besides using the intervals to assess the confidence in the fit at a certain point, they provide another optimization measure. Figure 2c demonstrates the result when applying this statistic for optimizing k at each query point.

Again, the fitted curve is significantly smoother than the global cross validation fit. A rather interesting and also typical effect happens at the very right end of the plot. When starting to extrapolate, the prediction intervals suddenly favor a global regression instead of the local regression, i.e., the k was chosen to be rather large. It turns out that in local optimization one always finds a competition between local and global regression. But sudden jumps from one mode into the other take place only when the prediction intervals are so large that the data is not trustworthy anyway.

Assessing The Quality of the Local Model

Both the local cross validation error MSE_{cross} and the prediction interval I_q may serve to assess the quality of the local fit:

$$Q_{fit} = \frac{\sqrt{MSE_{cross}}}{c} \quad or \quad Q_{fit} = \frac{I_q^+ - I_q^-}{c}.$$

The factor c makes Q_{fit} dimensionless and normalizes it with respect to some user defined quantity. In our applications, we usually preferred Q_{fit} based on the prediction intervals, which is the more conservative assessment.

Dealing with Outliers

Linear regression analysis is not robust with respect to outliers. This also holds for locally weighted regression, although the influence of outliers will not be noticed unless the outliers lie close enough to a query point. In Figure 3a we added three outliers to the test data of Figure 2 to demonstrate this effect; the charts in Figure 2 should be compared to Figure 2c. [27] applied the *median absolute deviation* procedure from robust statistics [18] to globally remove outliers in LWR. We would like to localize our criterion for outlier removal. The PRESS statistic can be modified to serve as an outlier detector in LWR. For this, we need the standardized individual PRESS residual $e_{i,cross}$ (see Table 1, Appendix A). This measure has zero mean and unit variance. If, for a given data point $\mathbf{x}_i$, it deviates from zero more than a certain threshold, the point can be called an outlier. A conservative threshold would be 1.96, discarding all points lying outside the 95% area of the normal distribution. In our applications, we used 2.57 cutting off all data outside the 99% area of the normal distribution. As can be seen in Figure 3b, the effects of outliers is reduced.

The Shifting Setpoint Exploration Algorithm

Learning algorithms which assume no a priori structure of the world often face the problem of sparse data in high dimensional spaces. Random exploration in order to build models of such worlds will take a very long time. Random exploration in an unknown world may also cause the system to enter unsafe or costly regions of operation. We want to develop an exploration algorithm which explicitly deals with such problems.

The shifting setpoint algorithm (SSA) attempts to decompose the control problem into two separate control tasks on different time scales. At the fast time scale, it acts as a nonlinear regulator by trying to keep the controlled system at some chosen setpoints. On a slower time scale, the setpoints are shifted to accomplish a desired goal. The SSA tries to explore the world by going to the fringes of its data support in the direction of the goal. It sets the setpoints in the fringes until statistically sufficient data has been collected to make a further step towards the goal. In this way the SSA builds a narrow tube of data support in which it knows the world. This data can be used by more sophisticated control algorithms for planning or further exploration.

We want to graphically illustrate the algorithm in a simple example of a mountain car (Figure 4) [26]. The task of the car is to drive at a given constant *horizontal* speed $\dot{x}_{desired}$ from the left to the right of the picture. $\dot{x}_{desired}$ need not be met precisely; the car should also minimize its fuel consumption. Initially, the car knows nothing about the world and cannot look ahead, but it has noisy feedback of its position and velocity. Commands, which correspond to the thrust F of the motor, can be generated at 5Hz.

The mountain car starts at its start point with one arbitrary initial action for the first time step; then it brakes and starts all over again, assuming the system can be reset somehow. The discrete one step dynamics of the car are modeled by an LWR forward model:

$$\hat{\mathbf{x}}_{next} = \hat{\mathbf{f}}(\mathbf{x}_{current}, F), \quad where \quad \mathbf{x} = (\dot{x}, x)^T. \tag{4.1}$$

After a few trials, the SSA searches the data in memory for the point $(\mathbf{x}_{current}^T, F, \mathbf{x}_{next}^T)_{best}^T$ whose outcome $\hat{\mathbf{x}}_{next}$ can be predicted with the smallest local confidence interval. Note that this does not imply that $\|\mathbf{x}_{next} - \hat{\mathbf{x}}_{next}\|$ is the smallest since we have noise in the data. This best point is declared the setpoint of this stage:

$$(\mathbf{x}_{S,in}^T, F_S, \mathbf{x}_{S,out}^T)^T = (\mathbf{x}_{current}^T, F, \hat{\mathbf{x}}_{next}^T)_{best}^T, \tag{4.2}$$

and its local linear model results from a corresponding LWR lookup:

$$\mathbf{x}_{S,out} = \hat{\mathbf{f}}(\mathbf{x}_{S,in}, F_S) \approx \mathbf{A}\mathbf{x}_{S,in} + \mathbf{B}F_S + \mathbf{c}. \tag{4.3}$$

Based on this linear model, an optimal LQ controller (e.g., [13]) can be constructed by minimizing the cost:

$$J = \sum_{k=1}^{\infty} \left((\mathbf{x}_k - \mathbf{x}_{S,in})^T \mathbf{Q} (\mathbf{x}_k - \mathbf{x}_{S,in}) + r\left(F_k - F_S\right)^2 \right) \tag{4.4}$$

of the regulator problem:

$$\mathbf{x}_{k+1} - \mathbf{x}_{S,out} = \mathbf{A}\left(\mathbf{x}_k - \mathbf{x}_{S,in}\right) + \mathbf{B}\left(F_k - F_S\right), \tag{4.5}$$

where $\mathbf{Q}$ and r are weight factors in matrix or scalar form, respectively. Solving this problem results in the control law:

$$F^{*} = -K\left(\mathbf{x}_{current} - \mathbf{x}_{S,in}\right) + F_S. \tag{4.6}$$

F^{*} is the optimal command under the cost J to go from the current state $\mathbf{x}_{current}$ to the setpoint $\mathbf{x}_{S,out}$ at this stage. This does not mean that the mountain car will actually reach $\mathbf{x}_{S,out}$ after applying F^{*}; the optimal control framework only guarantees a step towards the goal which reduces the magnitude of the value function. In the given problem it will trade speed accuracy for fuel consumption; the compromise between the two factors is reflected in the choice of $\mathbf{Q}$ and r. After these calculations, the mountain car learned one controlled action for the first time step. However, since the initial action was chosen arbitrarily, $\mathbf{x}_{S,out}$ will be significantly away from the desired speed $\dot{x}_{desired}$. A reduction of this error is achieved as follows. First, the SSA repeats to do one step actions with the LQ controller (which is updated with every new data point) until sufficient data was collected to reduce the size of the prediction intervals of LWR lookups for $\left(\mathbf{x}_{S,in}^{T}, F_S\right)^{T}$ (4.3) below a certain threshold. Then it shifts the setpoint towards the goal according to the procedure:

1) calculate the error of the predicted output state: $\mathbf{err}_{S,out} = \mathbf{x}_{desired} - \mathbf{x}_{S,out}$
2) take the derivative of the error with respect to the command F_S from a LWR lookup for $\left(\mathbf{x}_{S,in}^{T}, F_S\right)^{T}$ (cf. 4.3):

$$\frac{\partial \mathbf{err}_{S,out}}{\partial F_S} = \frac{\partial \mathbf{err}_{S,out}}{\partial x_{S,out}} \frac{\partial x_{S,out}}{\partial F_S} = -\frac{\partial x_{S,out}}{\partial F_S} = -\mathbf{B}, \tag{4.7}$$

   and calculate a correction ΔF_S from solving:

$$-\mathbf{B}\Delta F_S = \alpha\, \mathbf{err}_{S,out}\ , \tag{4.8}$$

   e.g., by singular value decomposition [31]; $\alpha \in [0,1]$ determines how much of the error should be compensated for in one step.
3) update F_S: $F_S = F_S - \Delta F_S$ and calculate the new $\mathbf{x}_{S,out}$ with LWR (4.3).
4) assess the fit for the updated setpoint with prediction intervals. If the quality is above a certain threshold, continue with 1), otherwise terminate shifting.

In this way, the output state of the setpoint shifts towards the goal until the data support falls below a threshold. Now the mountain car performs several new trials with the new setpoint and the correspondingly updated LQ controller. After the quality of fit statistics rise above a threshold, the setpoint can be shifted again. As soon as the first stage's setpoint reduces the error $\mathbf{x}_{desired} - \mathbf{x}_{S,out}$ to become close enough to zero, a new stage is created and the mountain car tries to move one step further in its world. The entire procedure is repeated for each new stage until the car knows how to move across the landscape along its line of setpoints with the associated LQ controllers. Figure 4b and Figure 4c show the thin band of data which the algorithm collected in state space and position-action space. These two pictures together form a narrow tube of knowledge in the input space of the forward model. The car never tried more than one exploration step into unknown territory and thus increased its probability of being safe to a high level.

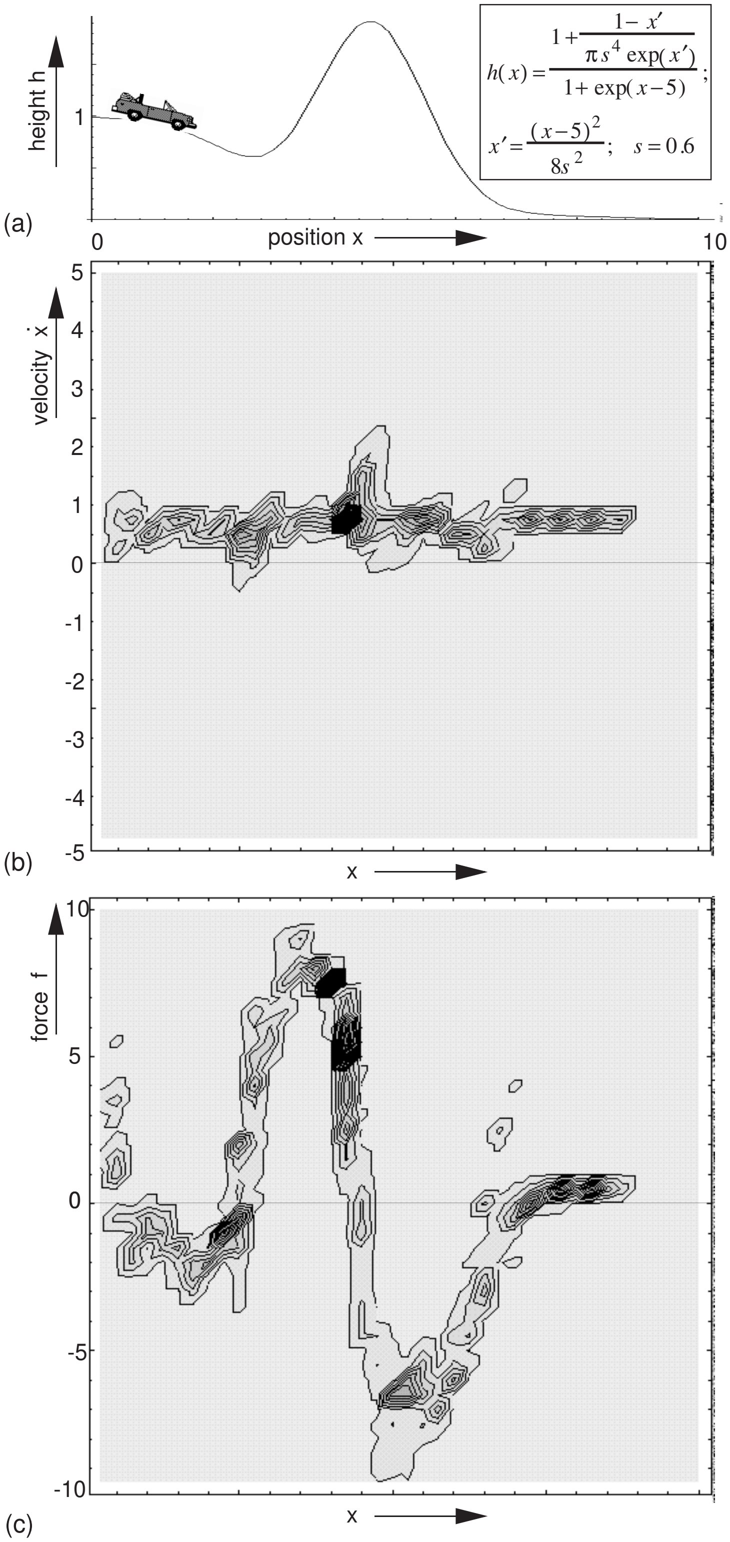


Figure 4: The mountain car: (a) landscape across which the car has to drive at constant velocity of 0.8 m/s, (b) contour plot of data density in phase space as generated by using multistage SSA, (c) contour plot of data density in position-action space

During the times where the setpoint statistics indicate insufficient data support to continue shifting, data collection is left to the randomness of the task dynamics. Thus the reduction of parameter uncertainty of the setpoint's local model also depends on this stochastic process. In order to identify the local model correctly, the stochastic process must provide data in all dimensions of the input and output space. If not, the regression problem (3.1a) may be ill-conditioned, resulting in bad estimates of the local model. Such situations were addressed by Fel'dbaum [17] as the dual control problem. In his formulations, the optimal command tries to minimize the cost and the uncertainty at the same time. So far, only expensive numerical solutions based on dynamic programming have been found to this problem [4, 7]. As an inelegant but effective way out, we add some small amount of random noise to the command F^*. The next section will demonstrate the importance of this measure.

Exploration has many facets. Depending on the task to be solved, random exploration, exploration towards unknown state space regions, and exploration towards reduction of uncertainty, etc., have been suggested [39]. The SSA exploration algorithm is goal directed and uncertainty driven under the premise not to dare any aggressive exploration outside the current data support. It is targeted at working in high dimensional environments where aggressive exploration would spend too much time in inappropriate and possibly dangerous regions. It is well suited for a real machine for which experimentation is time consuming. The SSA requires the existence of explicit goals. However, it is not always necessary to know these goals in advance but rather let the goals develop out of the task definition, as will be shown in the next section. The SSA should be generally applicable to problems which allow a decomposition in a static exploitation and a slowly moving exploration time scale, which have one time differentiable forward dynamics, and where the noise does not exceed the capabilities of the LQ controllers.

A System For Learning Experiments: Robot Juggling

We have constructed a system for experiments in real-time motor learning [40]. The task is a juggling task known as "devil sticking". A center stick is batted back and forth between two handsticks (Figure 5a). Figures 5b,c show a sketch and photograph of our devil sticking robot. The juggling robot uses motor 1 and motor 2 to perform planar devil sticking. Hand sticks with springs and dampers are mounted on the robot to implement a passive catch: the center stick does not bounce when it hits the hand stick and requires an active throwing motion by the robot. For the time being, the problem is simplified by the center stick being constrained by a boom to move on the surface of a sphere (Figure 5b), and motor 3 is not used. For moderate amplitudes these movements are approximately planar. The boom also provides a way to measure the current state of the center stick. The task state is the predicted location at which the ballistic flight of the center stick intersects with the hand stick held in an arbitrary but fixed nominal position $\left(x_{h,nominal}, y_{h,nominal}\right)^T$. We chose $\left(x_{h,nominal}, y_{h,nominal}\right)^T$ to be the hand stick position of the "upright" robot as shown in Figure 5b. As soon as the center stick does not touch the throwing hand stick anymore, standard ballistics equations for the flight of the center stick are used to map flight trajectory measurements $\left(x(t), y(t), \theta(t)\right)$ into the 5-dimensional estimated task state vector, i.e., the impact state with the other hand stick held at $\left(x_{h,nominal}, y_{h,nominal}\right)^T$:

$$\mathbf{x} = \left(p, \theta, \dot{x}, \dot{y}, \dot{\theta}\right)^T. \tag{5.1}$$

p is the distance of the devil stick's center of mass to the impact point hand stick–devil stick (Figure 5b). The task command is given by a displacement $\left(x_h, y_h\right)^T$ of the hand stick from the nominal position $\left(x_{h,nominal}, y_{h,nominal}\right)^T$, a center stick angular velocity threshold to trigger the start of a throwing motion $\dot{\theta}_t$, and a throw velocity vector $\left(v_x, v_y\right)^T$ of the hand stick, measured at point where the hand stick is attached to the robot .

$$\mathbf{u} = \left(x_{h,} y_h, \dot{\theta}_t, v_x, v_y\right)^T. \tag{5.2}$$

The dynamics of throwing the devilstick are thus parameterized by 5 state and 5 task commands, resulting in a 10/5-dimensional input/output model for each hand. Every time the robot catches and throws the devil stick it generates an experience vector of the form:

$$\left(\mathbf{x}_k^T, \mathbf{u}_k^T, \mathbf{x}_{k+1}^T\right)^T, \tag{5.3}$$

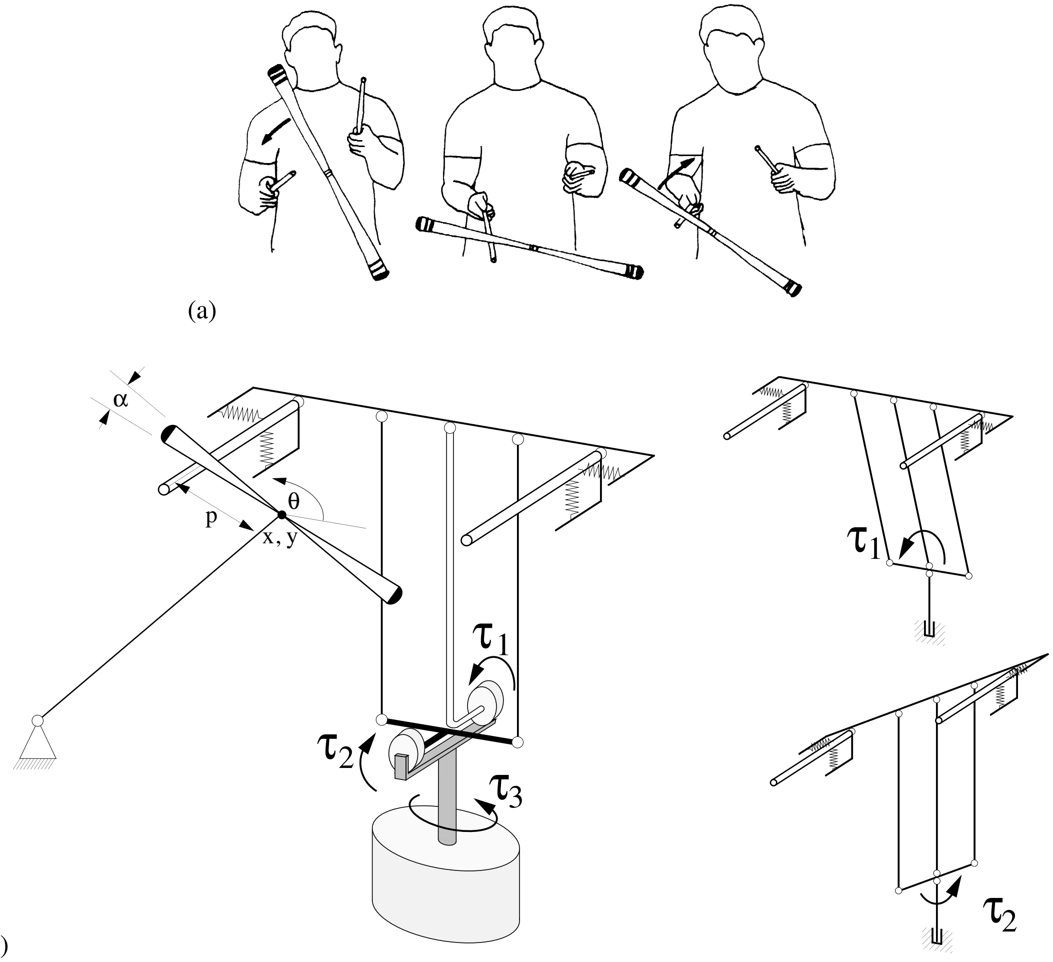
where $\mathbf{x}_k$ is the current state, $\mathbf{u}_k$ is the action performed by the robot, and $\mathbf{x}_{k+1}$ is the state of the center stick that results. Initially we explored learning an inverse model of the task, using nonlinear "deadbeat" control to eliminate all error on each hit. Each hand had its own inverse model of the form:

$$\hat{\mathbf{u}}_k = \hat{\mathbf{f}}^{-1}\left(\mathbf{x}_k, \mathbf{x}_{k+1}\right). \tag{5.4}$$

Before each hit, the system looked up a command with the expected impact state of the devilstick and the desired state:

$$\hat{\mathbf{u}}_k = \hat{\mathbf{f}}^{-1}\left(\mathbf{x}_k, \mathbf{x}_d\right). \tag{5.5}$$

Inverse model learning was successfully used to train the system to perform the devil sticking task. Juggling runs up to 100 hits were achieved. The system incorporated new data in real time, and used databases of several hundred hits. Lookups took less than 15 milliseconds, and therefore several lookups could be performed before the end of the flight of the center stick. Later queries incorporated more measurements of the flight of the center stick and therefore more accurate predictions of the state $\mathbf{x}_k$ of the task. However, the system required substantial structure in the initial training to achieve this performance. The system was started with a default command that was appropriate for open loop performance of the task. Each control parameter was varied systematically to explore the space near the default command. A global linear model was made of this initial data, and a linear controller based on this model was used to generate an initial training set for the memory-based system (approximately 100 hits). Learning with no initial data was not possible.



(c) INSERT PHOTO

Figure 5: (a) an illustration of devil sticking, (b) sketch of our devil sticking robot: the flow of force from each motor into the robot is indicated by different shadings of the robot links, and a position change due to an application of motor 1 or motor 2, respectively, is indicated in the small sketches; (c) photograph of robot

We also experimented with learning based on both inverse and forward models. After a command is generated by the inverse model, it can be evaluated using a memory-based forward model with the same data:

$$\hat{\mathbf{x}}_{k+1} = \hat{\mathbf{f}}\left(\mathbf{x}_k, \hat{\mathbf{u}}_k\right). \qquad (5.6)$$

Because it produces a local linear model, the LWR procedure generates estimates of the derivatives of the forward model with respect to the commands as part of the estimated parameter vector β (analog to 2.18 or 4.3). These derivatives can be used to find a correction to the command vector that reduces errors in the predicted outcome based on the forward model:

$$\frac{\partial \hat{\mathbf{f}}}{\partial \mathbf{u}} \Delta \hat{\mathbf{u}} = \hat{\mathbf{x}}_{k+1} - \mathbf{x}_d . \tag{5.7}$$

where the goal state $\mathbf{x}_d$ was calculated off-line from a comparison with human juggling. The pseudo-inverse of the matrix $\partial \hat{\mathbf{f}} / \partial \mathbf{u}$ is used to solve the above equation for $\Delta \hat{\mathbf{u}}_k$ in order to handle situations in which the matrix is singular or a different number of commands and states exists (which does not apply for devil sticking). The process of command refinement can be repeated until the forward model no longer produces accurate predictions of the outcome. This will happen when the query to the forward model requires significant extrapolation from the current database.

We investigated this method for incremental learning of devil sticking in simulations whose dynamics were adopted from the real machine. The outcome, however, did not meet expectations: without sufficient initial data around the setpoint, the algorithm did not work. Two main reasons can be held responsible:

i) Similar to the pure inverse model approach, the inverse-forward model acts as a one-step deadbeat controller. One-step deadbeat control applies large commands to correct for deviations from the setpoint. In the presence of errors in the model, this is detrimental since it magnifies the model errors. Additionally, the workspace bounds and command bounds of our devil sticking robot limit the size of the commands.

ii) Due to the nonlinearities in the dynamics of the robot, the 10-dimensional input space of the forward model suffers from the first symptoms of Bellman's "curse of dimensionality". Error reduction as described in (5.7) only works if sufficient data exists at the query sites. The inevitable model errors will make the robot explore randomly, leading to dispersed data, giving little chance for model improvements. Imagine we had to place data in a (hyper-)cube of normalized edge length 0.1. A 3-dimensional input space has 10^3 such cubes leaving some probability to finally arrive at the goal. A 10-dimensional state space, however, has 10^{10} such cubes – a prohibitive number for random exploration.

Thus, two ingredients had to be added to the devil sticking controller:

a) Control must start as soon as possible with the primary goal to increase the data density in the current region of the state-action space, and the secondary goal to arrive at the desired goal state.

b) Control actions must avoid deadbeat properties and must be planned to go to the goal in multiple steps.

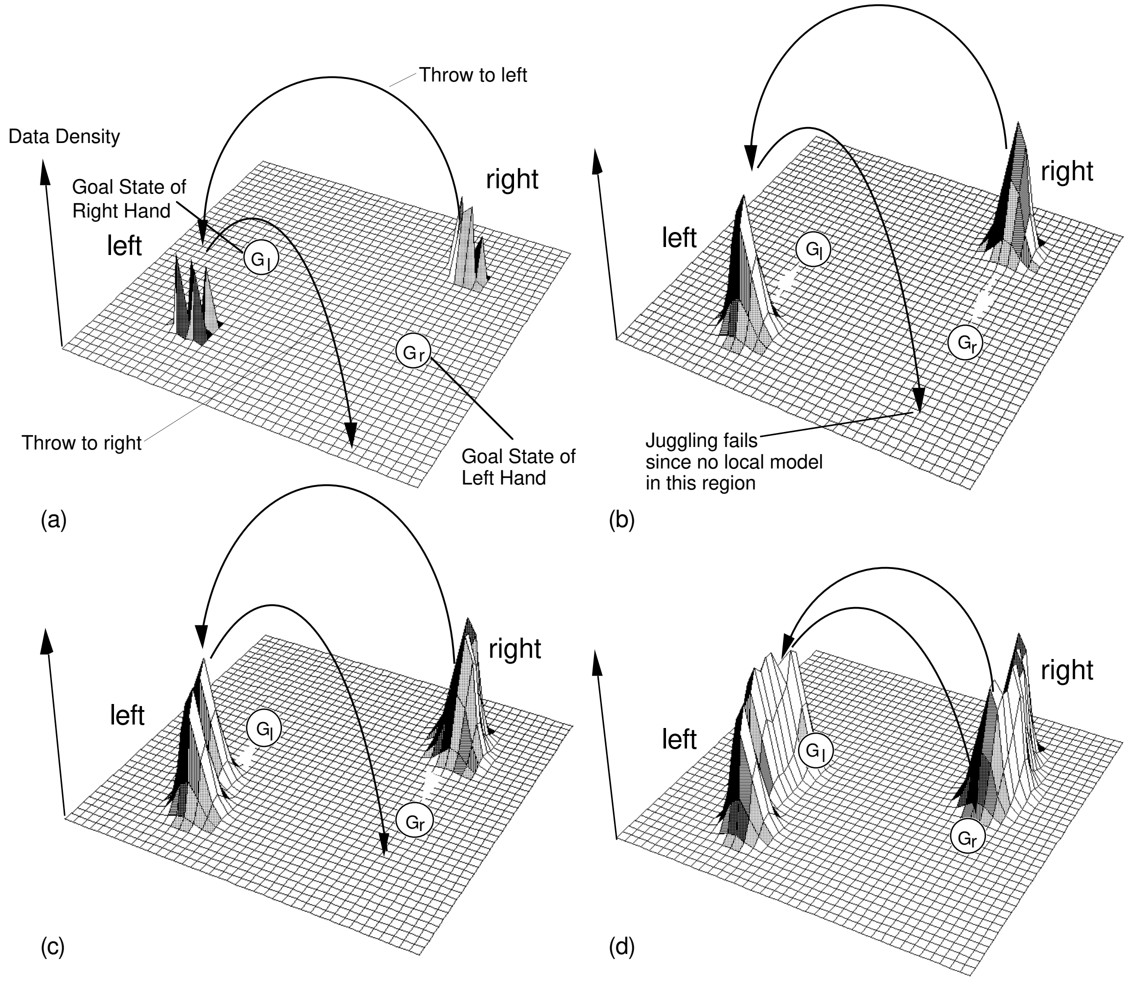


Figure 6: Abstract illustration how the SSA algorithm collects data in space: a) sparse data after the first few hits; b) high local data density due to local control in this region; c) increased data density on the way to the goals due to shifting of he setpoints; d) ridge of data density after the goal was reached

Both requirements are fulfilled by the shifting setpoint algorithm (SSA). Applied to devil sticking, the SSA proceeds as follows:

(1) Regardless of the poor juggling quality of the robot (i.e., at most two or three hits per trial), the SSA makes the robot repeat these initial actions with small random perturbations until a cloud of data was collected somewhere in state-action space of each hand. An abstract illustration for this is given in Figure 6a to 6b.

(2) Each point in the data cloud of each hand is used as a candidate for a setpoint of the corresponding hand by trying to predict its output from its input with LWR. The point achieving the narrowest local confidence interval becomes the setpoint of the hand and an LQ controller is calculated for its local linear model. By means of these controllers, the amount of data around the setpoints can quickly and rather accurately be increased until the quality of the local models exceeds a certain statistical threshold.

(3) At this point, the setpoints are gradually shifted towards the goal setpoints until the data support of the local models falls below a statistical value. Shifting occurs for both input state and output state of the setpoints (cf. Eq.4.2). After shifting, the kernel k (cf. Eq. (3.7)) is optimized by minimizing the local cross validation error MSE_{cross}. (In Figure 6, the goal setpoints are given explicitly, but they actually develop automatically from the requirement to throw the devilstick increasingly close to a place, in which the other hand has data support, i.e., $\mathbf{x}_{S,in,desired,left} = \mathbf{x}_{S,out,right}$, $\mathbf{x}_{S,out,desired,left} = \mathbf{x}_{S,in,right}$, and vice versa for the other hand)

(4) The SSA repeats itself by collecting data in the new regions of the workspace until the setpoints can be shifted again (Fig. 6c). The procedure terminates by reaching the goal, leaving a (hyper-) ridge of data in space (Figure 6d).

The LQ controllers play a crucial role for devil sticking. Although we statistically exploit data thoroughly, it is nevertheless hard to build good local linear models in the high dimensional spaces, particularly at the beginning of learning. LQ control has useful robustness even if the underlying linear models are imprecise.

We tested the SSA in a noise corrupted simulation and on the real robot. Learning curves are given in Figure 7. The learning curves are typical for the given problem. It takes roughly 40 trials before the setpoint of each hand has moved close enough to the other hand's setpoint. For the simulation (Figure 7a) a break-through occurs and the robot rarely loses the devilstick after that. In Figure 7b, the real robot learning curve is shown. The real robot takes more trials to achieve longer juggling runs, and its performance is not very consistent. This was due to the fact that the stochasticities of the robot did not sample the full state space sufficiently well during the data collection phases of the SSA. As pointed out in the dual control paragraph of the SSA section, we now added some random noise to the controls generated by the LQ controllers. Figure 7c shows the remarkable improvement in performance. On average, human beings need roughly a week of 1 hour practicing a day before they learn to juggle the devilstick. With respect to this, the robot learned very quickly. But the stability of our controllers is not global so far and will require future work.

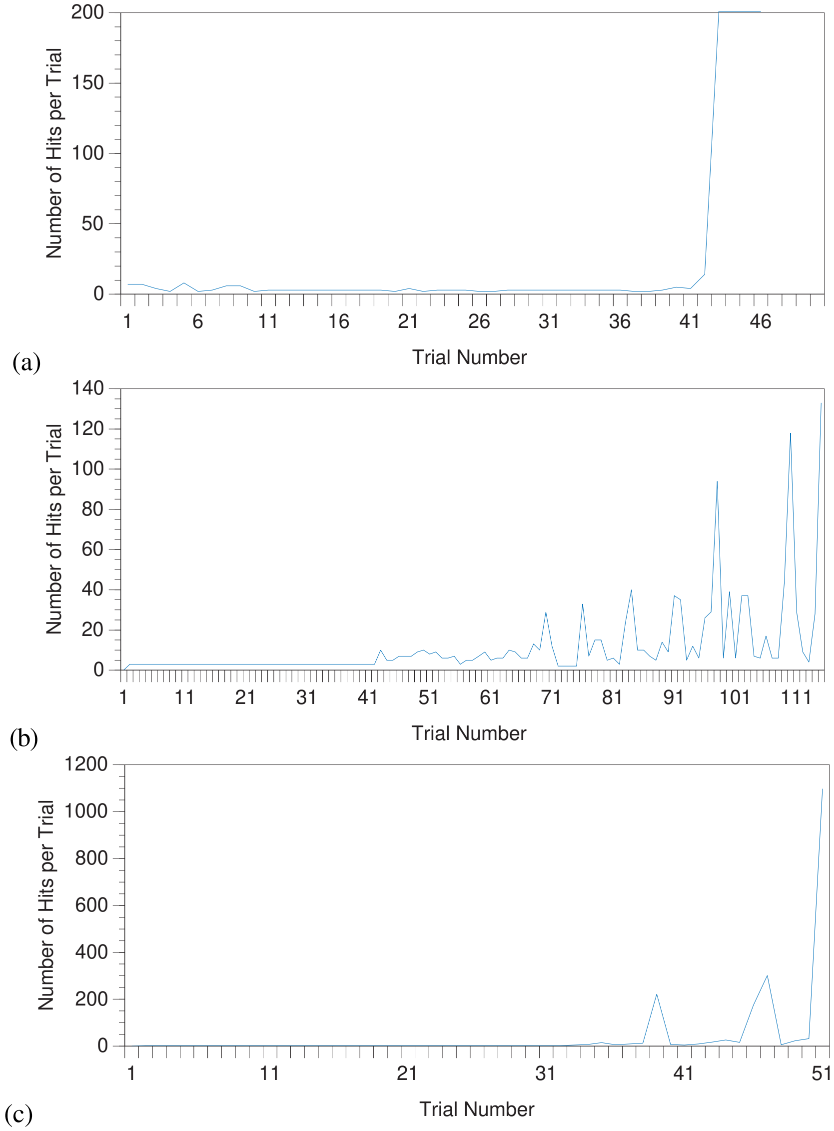


Figure 7: Learning curves of devil sticking using the SSA algorithm (a) simulation results (individual trials were stopped after 200 hits were reached), (b) real robot results; (c) real robot results with small amount of random noise added LQ controller commands

Discussion

In this paper we adopted a nonparametric approach to learning control. By means of locally weighted regression we built models of the world first, and exploited the models subsequently with statistical methods and algorithms from optimal control to design controllers. Despite the computational complexity of these methods, we demonstrated the usefulness of our algorithms in a real-time implementation of robot learning.

Using models for control according to the certainty equivalence principle is nothing new and has been supported by many researchers in the last years (e.g., [3, 38, 22, 24]). Using memory-based or nonparametric models, however, has only recently received increasing interest. One of the favorable advantages of memory-based modeling lies in the least commitment strategy which is associated with it. Since all data is kept in memory, a lookup can be optimized with respect to the few open architectural parameters. Parametric approaches do not have this ability if they discard their training data; if they retained all the training data they essentially become memory-based. As we demonstrated in our LWR approach to nonparametric modeling, several established statistical methods may be adopted to assess the quality of a model. These statistics form the backbone of the SSA exploration algorithm. So far we have only examined some of the most obvious statistical tools which directly relate to regression analysis. Many other methods may be suitable as well, particularly in a Bayesian framework.

Training a memory-based model is computationally inexpensive, as the data is simply stored in a memory. Training a nonlinear parametric model typically requires an iterative search for the appropriate parameters. Examples of iterative search are the various gradient descent techniques used to train neural network models (e.g., [20]). Lookup or evaluating a memory based model is computationally expensive, as described in this paper. Lookup for a nonlinear parametric model is often relatively inexpensive. If there is a situation in which a fixed set of training data is available, and there will be many queries to the model after the training data is processed, then it makes sense to use a nonlinear parametric model. However, if there is a continuous stream of new training data intermixed with queries, as there typically is in many motor learning problems, it may be less expensive to train and query a memory-based model then it is to train and query a nonlinear parametric model.

A question that often arises with memory-based models is the effect of memory limitations. We have not yet needed to address this issue in our experiments. However, we plan to explore how memory use can be minimized based on several methods. One approach is to only store “surprises”. The system would try to predict the outputs of a data point before trying to store it. If the prediction is good, it is not necessary to store the point. Another approach is to forget data points. Points can be forgotten or removed from the database based on age, proximity to queries, or other criteria. Because memory-based learning retains the original training data, forgetting can be explicitly controlled.

That computational complexity does not necessarily limit real time applications was demonstrated with our successful devil sticking robot. We are able to do lookups for memory-based local models in less than 15ms for a thousand data points modeling a 10 to 5 mapping, and we are able to build on-line LQ controllers in another 5ms. The initial shortcomings of our deadbeat inverse or inverse-forward model controllers are not due to the

LWR learning algorithm but rather to the inherent problems of this kind of control. As has been pointed out by Jordan and Rumelhart [22], inverse models are not goal-directed and perform data sampling in action and not state space. They do not establish a connection between a certain sensation and a certain action but rather a connection between two sensations. Hence, they do not learn from bad actions. A forward model overcomes these problem. Pure forward model controllers, however, are still deadbeat controllers which try to cancel the plant dynamics in one step. This results in large commands if the system deviates only moderately from its desired goal and conflicts with the workspace bounds and command bounds of our robot. Additionally, modeling errors are strongly amplified by deadbeat control. Accurate data sampling, as it is necessary in high dimensional spaces, becomes thus rather difficult.

Due to the statistical properties of locally weighted regression, a simple exploration algorithm like the shifting setpoint algorithm is powerful enough to accomplish the desired task. Deadbeat control was replaced by LQ control which naturally blends into the LWR framework. By no means was the SSA algorithm intended to replace high-level controllers. Indeed, it remains to be explored in how far the chaining of individual LQ controllers is actually robust, and whether an approach from trajectory optimization [13] would not be more appropriate. In favor of the SSA algorithm stands its easy implementation for real-time systems.

Two crucial prerequisites entered our explanations on robot learning. First, we assumed we know the input/output representations of the task, and second, we were able to generate a goal state for the SSA exploration. A good choice of a representation is crucial in order to be able to accomplish the goal at all [33, 35], and we have very limited insight so far how to automate this part of the learning process. Of equal importance is a good choice of a goal state. In devil sticking, the goal state developed out of the necessity that the left and right hand have to cooperate. The initial action, however, which was given by the experimenter, clearly determined in which ballpark the juggling pattern would lie. Certain patterns of devil sticking are easier than others [34], and we picked an initial action of which we knew that it was favorable. One part of our future work will address these issues in more detail in that we search for good initial actions and strategies to approach a task [6].

Conclusions

The paper demonstrated that a real robot can indeed learn a non-trivial task. As pointed out above, by taking input/output representations and good learning goals as given, a large portion of the task was already solved in advance. Solving the remaining problems became practicable mainly because of the characteristics of the LWR learning method. The local linear models that this algorithm generated at every query point allowed us to make use of optimal control techniques which added useful robustness to the controllers. Since LWR is memory-based, the local linear models could be optimized with respect to statistical uncertainty measures. These measures also served as the basis of the SSA exploration algorithm. Such statistical tools are not generally available in learning. LWR is particularly suited to exploit statistics since it originates from a statistical method, and we could thus easily assure compatibility of the statistics and the learning algorithm. As a last point, LWR does not suffer from problems of interference when being trained on new data. Interference means a degradation of performance in one part of the model when training the model with data relevant for different parts. Such an effect could happen during SSA shifting if a parametric learning method were applied. But since lookups with LWR are affected only by a small cloud of data in the neighborhood of the query point, interference problems are greatly reduced.

Our future work will focus on extending LWR model-based control to multistage problems in the optimal control domain. Devil sticking should not only be stable within the validity of the local linear controllers but rather exhibit global stability. This requires nonlinear optimal control and planning techniques which we are currently exploring. Future work must furthermore address how we could approach tasks in which complete measurements of the states are not available, or what constitutes a state is not even known.

References

[1] Aboaf, E. W., Atkeson, C. G. ,Reinkensmeyer, D. J. (1988), "Task-Level Robot Learning", *Proceedings of IEEE International Conference on Robotics and Automation*, April 24-29, Philadelphia, Pennsylvania (1988).

[2] Aboaf, E.W., Drucker, S.M., Atkeson, C.G. (1989), "Task-Level Robot Learing: Juggling a Tennis Ball More Accurately", *Proceedings of IEEE Interational Conference on Robotics and Automation*, May 14-19, Scottsdale, Arizona (1989).

[3] An, C.H., Atkeson, C.G., Hollerbach, J.M. (1988), *Model-Based Control of A Robot Manipulator*, Cambridge, MA: MIT Press (1988).

[4] Åström, K.J., Wittenmark, B. (1989), *Adaptive Control*, Reading, MA: Addison-Wesley (1989).

[5] Atkeson, C.G. (1990), "Memory-Based Approaches to Approximating Continuous Functions", MIT, Cambridge, MA: The AI-Lab and the Brain and Cognitive Sciences Department (1990).

[6] Atkeson, C.G. (1994), "Using Local Trajectory Optimizers to Speed Up Global Optimization in Dynamic Programming", to appear in: Moody, J.E., Hanson, S.J., and Lippmann, R.P. (eds.) *Advances in Neural Information Processing Systems 6*, Morgan Kaufmann (1994).

[7] Bar-Shalom, Y. (1981), "Stochastic Dynamic Programming: Caution and Probing", *IEEE Transactions on Automatic Control*, vol.AC-26, no.5, October 1981 (1981).

[8] Bellman, R. (1957), *Dynamic Programming*. Princeton, N.J.: Princeton University Press (1957).

[9] Bertsekas, D.P. (1987), *Dynamic Programming*, Englewood Cliffs, NJ: Prentice-Hall (1987).

[10] Bitmead, R. R., Gevers, M., Wertz, V. (1990), *Adaptive Optimal Control*, Englewood Cliffs NJ: Prentice Hall (1990).

[11] Cleveland, W.S., Devlin, S.J., Grosse, E. (1988), Regression by Local Fitting: Methods, Properties, and Computational Algorithms. Journal of Econometrics 37, 87-114, North-Holland (1988).

[12] Duda, R.O., Hart, P.E. (1973), *Pattern Classification and Scene Analysis*, New York, NY: Wiley (1973).

[13] Dyer, P., McReynolds, S.R. (1970), *The Computation and Theory of Opitmal Control*, New York: Academic Press (1970).

[14] Fan, J., Gijbels, I. (1992), "Variable Bandwidth And Local Linear Regression Smoothers", *The Annals of Statistics*, vol.20, no.4, pp.2008-2036 (1992).

[15] Farmer, J.D., Sidorowich, J.J. (1987), "Predicting Chaotic Dynamics", Kelso, J.A.S., Mandell, A.J., Shlesinger, M.F., (eds.): *Dynamic Patterns in Complex Systems*, World Scientific Press (1987).

[16] Farmer, J.D., Sidorowich, J.J. (1988), "Exploiting Chaos to Predict the Future and Reduce Noise", Technical Report LA-UR-88-901, Los Alamos National Laboratory, Los Alamos, NM (1988).

[17] Fel'dbaum, A.A. (1965), *Optimal Control Systems*, New York, NY: Academic Press (1965).

[18] Hampbell, F., Rousseeuw, P, Ronchetti, E, Stahel, W. (1985), *Robust Statistics*, Wiley International (1985).

[19] Härdle, W. (1991), *Smoothing Techniques with Implementation in S*, New York, NY: Springer (1991).

[20] Hertz, J., Krogh, A., Palmer, R.G. (1991), *Introduction to the Theory of Neural Computation*, Redwood City, CA: Addison Wesley (1991).

[21] Isermann, R., Lachmann , K.-H., Matko, D. (1992), *Adaptive Control Systems*, New York, NY: Prentice Hall, (1992).

[22] Jordan, I.M. , Rumelhart, D.E.(1990), "Forward Models: Supervised Learning with a Distal Teacher", *MIT Center for Cognitive Science Occasional Paper 40* , Cambridge, MA: MIT(1990).

[23] Mallows, C.L. (1966), "Choosing a Subset Regression", unpublished paper presented at the annual meeting of the American Statistical Association, Los Angles (1966).

[24] Moore, A. (1991), "Fast, Robust Adaptive Control by Learning only Forward Models", in: Moody, J.E., Hanson, S.J., and Lippmann, R.P. (eds.) *Advances in Neural Information Processing Systems 4*, Morgan Kaufmann (1991).

[25] Moore, A.W. (1990), *Efficient Memory-based Learning for Robot Control*. PhD. Thesis, Technical Report No. 229, Computer Laboratory, University of Cambridge, October 1990 (1990).

[26] Moore, A.W. (1991), "Knowledge of Knowledge and Intelligent Experimenttion for Learning Control", *Proceedings of the 1991 International Joint Conference on Neural Networks*, Seattle (1991).

[27] Moore, A.W., Atkeson, C.G (1993), "An Investigation of Memory-based Function Approximators for Learning Control", submitted to *Machine Learning* (1993).

[28] Müller, H.-G. (1988), *Nonparametric Regression Analysis of Longitudinal Data*, Lecture Notes in Statistics Series, vol.46, Berlin: Springer (1988).

[29] Myers, R.H. (1990), *Classical And Modern Regression With Applications*, Boston, MA: PWS-KENT (1990).

[30] Narendra, K.S., Dorato, P. (eds.) (1991), *Advances in Adaptive Control*, Piscataway, NJ : IEEE Press (1991).

[31] Press, W.P., Flannery, B.P., Teukolsky, S.A., Vetterling, W.T. (1989), *Numerical Recipes in C – The Art of Scientific Computing*, Cambridge, MA: Press Syndiacate University of Cambridge (1989).

[32] Rizzi, A.A., Whitcomb, L.L., Koditschek, D.E. (1992a), "Distributed Real Time Control of a Spatial Robot Juggler", *IEEE Computer*, 25 (5) (1992).

[33] Schaal, S., Atkeson, C.G. (1993b), "Open Loop Stable Control Strategies for Robot Juggling", *Proceedings IEEE International Conference on Robotics and Automation*, vol.3, pp.913-918, Georgia, Atlanta (1993).

[34] Schaal, S., Atkeson, C.G., Botros, S. (1992b), “What Should Be Learned?”, In: *Proceedings of Seventh Yale Workshop on Adaptive and Learning Systems*, pp.199-204, New Haven (1992).

[35] Schaal, S., Sternad, D., (1993), “Learning Passive Motor Control Strategies with Genetic Algorithms”, in: Nadel, L. & Stein, D. (eds.): *1992 Lectures in Complex Systems 1992*, Addison-Wesley (1993).

[36] Slotine, J.-J.E., Li, W. (1991), *Applied Nonlinear Control*, Englewood Cliffs, NJ: Prentice Hall (1991).

[37] Specht, D.F. (1991), “A General Regression Neural Network”, *IEEE Transactions on Neural Networks*, vol.2, no.6, Nov.1991 (1991).

[38] Sutton, R.S. (1990), “Integrated Architectures for Learning, Planning, and Reacting Based on Approximating Dynamic Programming”, *Proceedings of the International Machine Learning Conference* , pp.212-218 (1990).

[39] Thrun, S.B. (1992a), “Efficient Exploration in Reinforcement Learning”, Technical report CMU-CS-92-102, Pittsburgh, Pennsylvania: Carnegie-Mellon University (1992).

[40] Van Zyl, G. (1991), “Design and control of a robotic platform for machine learning”, MS thesis, MIT Dept. of Mechanical Engineering.

[41] Wahba, G., Wold, S. (1975), “A Completely Automatic French Curve: Fitting Spline Functions By Cross-Validation”, *Communications in Statistics*, 4(1) (1975).

[42] White, D.A, Sofge, D.A. (1992), *Handbook of Intelligent Control : Neural, Fuzzy, and Adaptive Approaches*, New York, NY: Van Nostrand Reinhold (1992).

Appendix A

The conversion from unweighted linear regression to locally weighted regression analysis is done mostly by inserting the matrix **W** at the appropriate sites. The definitions of the parameters n’ and p’, however, need some explanation. Imagine the weighting function is not a soft-weighting function (e.g., a Gaussian) but rather a hard-weighting function clipping off points beyond a certain threshold: $w_i = 1 \quad if \quad d^2 < \varepsilon, \quad otherwise \quad w_i = 0$. Redefining n to n’ accommodates such a k-nearest neighbor weighting and transforms the n-point regression problem to an k-point regression. If p stays unchanged, all statistics would correspond to unweighted regression. Subtraction of p from n to calculate variances aims at achieving unbiased estimators. For LWR, it is easily possible to find the case where n’<p. In the above mentioned k-nearest neighbor example, this would mean that we do not have sufficient data support for the regression model. For soft-weighting functions, the problem cannot be resolved so clearly; we could always imagine scaling up all weights by a constant factor so that $\sum_{i=1}^{n} w_i^2 = n$ which would not change the regression variables. LWR weights data with respect to each other and not absolutely. Applying n’ instead of n to calculate variances or mean squared errors makes such measures invariant towards mere weight scaling; this can easily be verified by setting $w_i = w_i \cdot const.$. Defining p’ in the given way avoids problems when n’<p. The bias introduced this way should diminish with an increasing number of data points in memory. One could argue in favor of neglecting p entirely, but incorporating it in the given way makes the statistics stay on the more pessimistic side which seems reasonable.

The only statistics which see a direct influence of a weight scaling are the prediction intervals and the standardized individual PRESS residuals. Restricting the weights to the range [0,1] and requiring that the weight of a point with zero Euclidean distance from the lookup point equals 1 resolves this problem. The dependence of the t-value on n'-p' in the prediction intervals increases the t-value and thus the prediction intervals with diminishing n'. The smallest value t may acquire is for n=n', i.e., unweighted regression. The standardized individual PRESS residual $e_{i,cross}$ is proportional to the magnitude of the i-th weight. As we pointed out in Section x3, this measure has zero mean and unit variance. With increasing distance from the current query point, it will be weighted down, i.e., the likelihood of being an outlier diminishes even if $e_{i,cross}$ is rather large. As the weights cannot be larger than 1, $e_{i,cross}$ cannot assume larger values as for unweighted regression.

It must be pointed out that statistics literature provides much more sophisticated and mathematically exact statistics for locally weighted regression [19, 28, 14, 11]. However, most of these measures require the estimation of Hessians and/or data densities which for high dimensional problems are not readily adapted without numerical problems. Our LWR statistics are used to tune fit parameters and need not give precise statistical assessments.

Table 1:

	Unweighted Linear Regression	**Locally Weighted Regression**
number of data in regression	n	n, we define: $n' = \sum_{i=1}^{n} w_i^2$, $w_i \in [0,1], \quad w_i = 1 \quad if \quad \mathbf{x}_i = \mathbf{x}_q$, $\mathbf{x}_q$ is the current query point
number of regression parameters	p	p, we define: $p' = \frac{n'}{n} p$
matrix and vector definitions	$\mathbf{x}_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{p-1} \\ 1 \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}$	$\mathbf{x}_i = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{p-1} \\ 1 \end{bmatrix}, \ \mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix}, \ \mathbf{W} = \begin{bmatrix} w_1 & & & \mathbf{0} \\ & w_2 & & \\ & & \ddots & \\ \mathbf{0} & & & w_n \end{bmatrix}$
regression model	$\mathbf{X}\beta = \mathbf{y}$	$\mathbf{WX}\beta = \mathbf{Wy}$
sum squared error = residual error	$SSE_{res} = (\mathbf{X}\beta - \mathbf{y})^T (\mathbf{X}\beta - \mathbf{y})$	$SSE_{res} = (\mathbf{X}\beta - \mathbf{y})^T \mathbf{W}^T \mathbf{W} (\mathbf{X}\beta - \mathbf{y})$
normal equations	$(\mathbf{X}^T\mathbf{X})\beta = \mathbf{X}^T\mathbf{y}$	$(\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X})\beta = \mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{y}$

solution for regression parameters	$\beta = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$	$\beta = \left(\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{y}$
variance of residuals	$s^2 = \dfrac{(\mathbf{X}\beta-\mathbf{y})^T(\mathbf{X}\beta-\mathbf{y})}{n-p}$	$s^2 = \dfrac{(\mathbf{X}\beta-\mathbf{y})^T\mathbf{W}^T\mathbf{W}(\mathbf{X}\beta-\mathbf{y})}{n'-p'}$
mean squared cross validation error (mean PRESS residual)	$MSE_{cross} = \dfrac{1}{n-p}\sum_{i=1}^{n}\left(\dfrac{y_i-\mathbf{x}_i^T\beta}{1-\mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{x}_i}\right)^2$	$MSE_{cross} = \dfrac{1}{n'-p'}\sum_{i=1}^{n}\left(\dfrac{w_i\left(y_i-\mathbf{x}_i^T\beta\right)}{1-w_i\mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{x}_i w_i}\right)^2$
prediction intervals	$I_i = \mathbf{x}_i^T\beta \pm t_{\alpha/2,n-p}\, s\sqrt{1+\mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{x}_i}$ where $t_{\alpha/2,n-p}$ is Student's t-value of n-p degrees of freedom for a 100(1-α)% prediction bound	$I_i = \mathbf{x}_i^T\beta \pm t_{\alpha/2,n'-p'}\, s\sqrt{1+\mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{x}_i}$ where $t_{\alpha/2,n'-p'}$ is Student's t-value of n'-p' degrees of freedom for a 100(1-α)% prediction bound
standardized individual PRESS residual	$e_{i,cross} = \dfrac{y_i-\mathbf{x}_i^T\beta}{s\sqrt{1-\mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{x}_i}}$	$e_{i,cross} = \dfrac{w_i\left(y_i-\mathbf{x}_i^T\beta\right)}{s\sqrt{1-w_i\mathbf{x}_i^T\left(\mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X}\right)^{-1}\mathbf{x}_i w_i}}$