In: V. Graefe (Ed.): *Intelligent Robots and Systems 1994 (IROS 1994).* Amsterdam: Elsevier Science, 1995.

Robot Learning By Nonparametric Regression

Stefan Schaal and Christopher G. Atkeson

Department of Brain and Cognitive Sciences & The Artificial Intelligence Laboratory
Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139*

Abstract: We present an approach to robot learning grounded on a nonparametric regression technique, locally weighted regression. The model of the task to be performed is represented by infinitely many local linear models, i.e., the (hyper-) tangent planes at every point in input space at which a prediction is to be made, i.e., at every query point. Such a model, however, is only generated at the time of prediction and is not retained. This is in contrast to other methods using a finite set of linear models to accomplish a piecewise linear model. Architectural parameters of our approach, such as distance metrics, are also a function of the current query point instead of being global. Statistical tests are presented for when a local model is good enough such that it can be reliably used to build a local controller. These statistical measures also direct the exploration of the robot. We explicitly deal with the case where prediction accuracy requirements exist during exploration: By gradually shifting a center of exploration and controlling the speed of the shift with local prediction accuracy, a goal-directed exploration of state space takes place along the fringes of the current data support until the task goal is achieved. We illustrate this approach by describing how it has been used to enable a robot to learn a challenging juggling task: Within 40 to 100 trials the robot accomplished the task goal starting out with no initial experiences.

1. INTRODUCTION

Learning control means improving a motor skill by repeatedly practicing a task. We are exploring systems that learn by explicitly remembering their experiences. These systems will be useful in situations where the form or structure of the task to be performed is not known in advance. Previous work tested such methods by implementing learning for one-shot or static tasks, such as throwing a ball at a target (Aboaf et. al., 1988), and also repetitive or dynamic tasks, such as bouncing a ball on a paddle (Aboaf et. al., 1989) and hitting a stick back and forth (a form of juggling known as devil sticking) (Van Zyl, 1991). This experimental work has highlighted the importance of making sure the control paradigm used is robust to uncertainty, that the representational scheme 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 mod-

* New address of both authors: Georgia Institute of Technology, College of Computing, 801 Atlantic Drive, Atlanta, GA 30332, USA

els and controllers based on insufficient data are improved. In this paper we discuss the current stage of our nonparametric learning methods by describing some of their recent extensions with the help of an implementation of learning with the devil sticking robot. In the given context, we are using a form of indirect learning, where a model is learned and then control actions are chosen based on the model, rather than direct learning, where appropriate control actions are learned directly (Barto et. al., 1991); the demonstrated framework, however, is equally well suited to direct learning, too. Our starting point for modeling is that we do not know the structure or form of the system to be controlled. However, we assume we do know what constitutes a state of the system, and that we measure the complete state. Future work will discuss how to approach tasks in which complete measurements of the state are not available, or what constitutes a state is not even known. We adopted a recently developed statistical technique from nonparametric regression analysis, locally weighted regression (LWR (Cleveland, 1979; Farmer & Sidorowich, 1987; Atkeson, 1992), to learn models the system we are trying to control (Section 2). The LWR approach allows to efficiently estimate local linear models for different points in the state space. This approach is extended here to give information about the reliability of the predictions and local linearizations generated by locally weighted regression. Thus, as will be demonstrated in Section 3, a robot can monitor its own skill level and guide its exploratory behavior, and it can ensure that even in high dimensional state space good learning results are accomplished.

2. Locally Weighted Regression

The point of view acquired here 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. Models should incorporate the latest information. As they will be continuously updated with a stream of new training data, updating a model 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, a representation is needed that can quickly compute a local linear model of the represented transformation. Moreover, negative interference from learning new knowledge on previously stored information should be as minimal as possible.

As the most generic approximator that satisfies many of these criteria, we explore a version of nonparametric learning called locally weighted regression (LWR). LWR is a memory-based learning (MBL) system: it is trained just by storing the training data in a memory. This allows to achieve real-time learning and avoids interference between new and old data by retaining and using all the data to answer each query. Nonparametric learning systems like LWR approximate complex functions by 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 combines points near to the input point of a desired prediction, the query point, to estimate the appropriate output. Locally weighted regression uses a regression procedure to form the local model, and is thus more expensive than nearest neighbor and weighted average memory-based learning procedures. However, local linear models have been shown to have a favorable balance between bias and variance

(Cleveland et al., 1979). They also provide the first derivative at the query point which will be shown below to be very useful for control tasks. For each query LWR forms a new local model. The rate at which local models can be formed and evaluated limits the rate at which queries can be answered. At the end of this paper, it will be described how locally weighted regression was implemented for a real-time application.

Unweighted regression finds the solution to the equations:

$$\mathbf{y} = \mathbf{X} \cdot \beta \tag{2.1a}$$

by minimizing

$$J = \frac{1}{2}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2 \tag{2.1b}$$

where $\mathbf{X}$ is an m∗(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 the corresponding outputs for each data point, β is the (n+1)-vector of regression parameters which are to be determined, and J is the sum of squared errors over all given data points ($\hat{y}_i$ is the predicted result after fitting the parameters β). The well-known result for β yields

$$\beta = (\mathbf{X}^{\mathbf{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathbf{T}}\mathbf{y} \tag{2.2}$$

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

$$\hat{y}_q = \mathbf{x}_q^T\beta \tag{2.3}$$

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

$$d_i^2 = \sum_{r=1}^{n}(x_q - x_{ir})^2 \tag{2.4}$$

The weight for each stored data point is a function of that distance:

$$w_i = f(d_i^2) \tag{2.5}$$

Each i-th row in $\mathbf{X}$ and $\mathbf{y}$ is multiplied by the corresponding weight w_i, and the regression problem is solved as before. In matrix notation, this introduces the diagonal m∗m weight matrix $\mathbf{W}$ in J:

$$J = \frac{1}{2}\sum_{i=1}^{m}(w_i(y_i - \hat{y}_i))^2 = \frac{1}{2}\left|\mathbf{W}(\mathbf{y} - \hat{\mathbf{y}})\right|^2 \tag{2.6}$$

and the local weighted regression parameters become:

$$\beta = \left[(\mathbf{WX})^{\mathbf{T}}\mathbf{WX}\right]^{-1}(\mathbf{WX})^{\mathbf{T}}\mathbf{Wy} \tag{2.7}$$

A simple weighting function just raises the distance to a negative power, which determines how local the regression will be (the rate of drop-off of the weights with distance).

$$w_i = \frac{1}{\left(d_i^2\right)^k} \tag{2.8}$$

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 desired. One such scheme, which we use in what follows, is a Gaussian kernel:

$$w_i = \exp\left(-\frac{d_i^2}{2k^2}\right) \tag{2.9}$$

As in (2.8), the parameter k scales the size of the kernel to determine how local the regression will be. A LWR lookup has three stages: forming the weights (Eqns. (2.4), (2.5)), forming the regression matrix $\mathbf{WX}$, and solving for the regression parameters as given in (2.7). A more thorough introduction of LWR can be found in Atkeson (1992) or Schaal and Atkeson (1994).

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 10 input dimensions and 5 output dimensions, using a linear local model. This leads to a lookup time of approximately 15 milliseconds on a database of 1000 points.

2.1. Tuning The Fit Parameters

LWR uses a few parameters which require tuning to optimize the quality of fit. Of most importance is the parameter k in (2.9) and, in multiple input systems, the distance metric $\mathbf{m}$ which determines the weighting of the input dimensions with respect to each other. The distance metric enters (2.4) and modifies this equation to:

$$d_i^2 = \sum_{r=1}^{n}\left[(x_q - x_{ir})m_r\right]^2 \tag{2.10}$$

In the past we have used off-line global cross validation to estimate reasonable values for these fit parameters. Since we are using a local model that is linear in the unknown parameters, derivatives can be computed of the cross validation error $e_i = \hat{y}_i - y_i$ with respect to the fit parameters k and $\mathbf{m}$ 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 (2.9). In Figure 1a, 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 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, demonstrate 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, confidence intervals, prediction intervals, and many more (e.g., Myers 1990). 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 quite helpful.

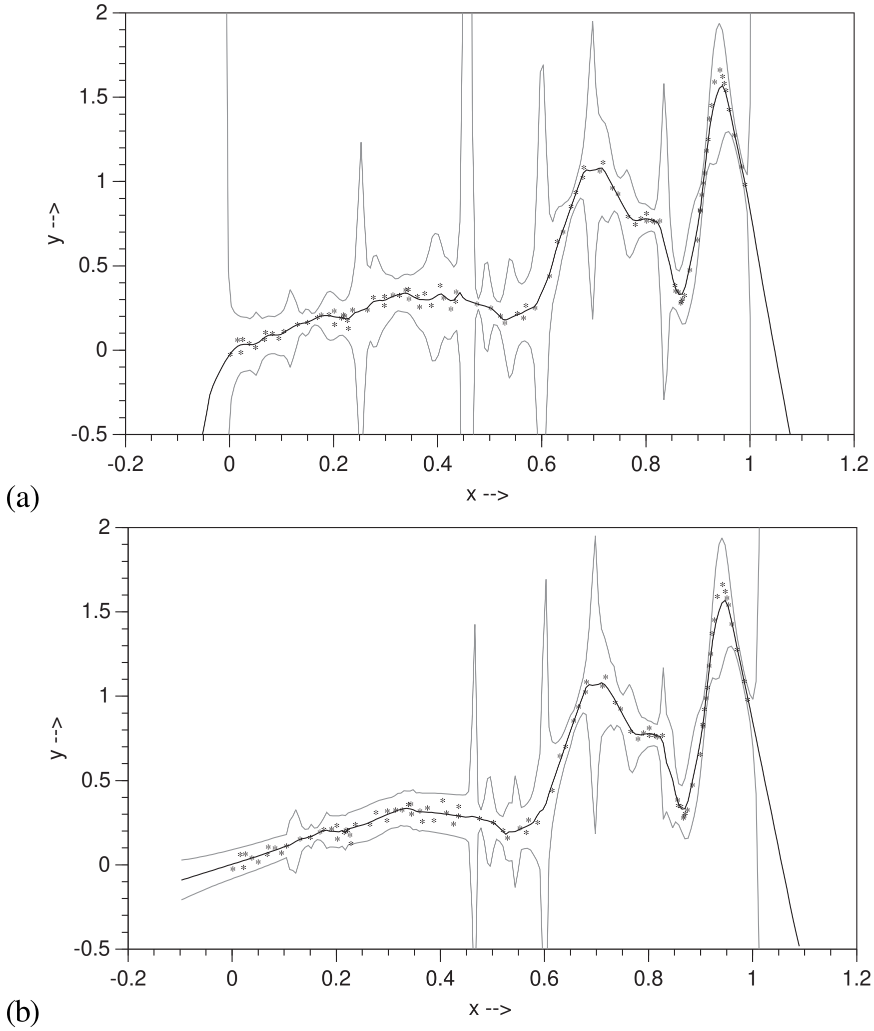


Figure 1: Optimizing the locally weighted regression fit using different measures: (a) global cross validation, (b) local cross validation

Cross validation has a relative in linear regression analysis called the PRESS residual error. The PRESS statistic performs leave-one-out cross validation, however, without the need to recalculate the regression parameters for every excluded point. This is computationally very efficient. Schaal and Atkeson (1994) show how the PRESS residual can be expressed as a mean squared cross validation error MSE_{cross}. In Figure 1b, the same data as in Figure 1a 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 1a.

Prediction intervals I_i are expected bounds of the prediction error at a query point $\mathbf{x}_i$. Their derivation can be found in most text books on regression analysis (e.g., Myers 1990), and an adaptation for LWR is straightforward (Schaal and Atkeson, 1994). Besides using the intervals to assess the confidence in the fit at a certain point, they provide another optimization measure. Using prediction intervals as optimization criterion for k, a plot similar to Figure 1b is obtained. The curves in Figure 2a,b are generated by using this optimization criterion.

2.2. Assessing The Quality of the Local Model

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

$$Q_{fit} = \frac{\sqrt{MSE_{cross}}}{c} \quad or \quad Q_{fit} = \frac{I_i^+ - I_i^-}{c} \tag{2.11}$$

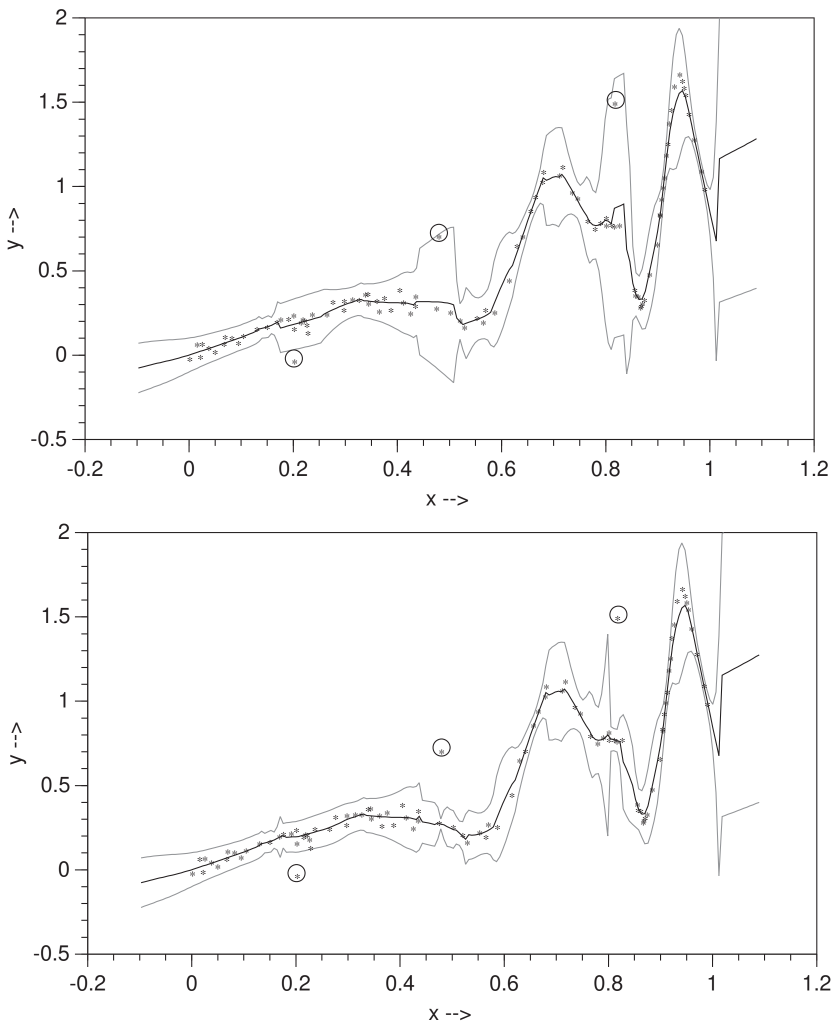
The factor c makes Q_{fit} dimensionless and normalizes it with respect to some user defined quantity. In our applications, we usually preferred Q_{fit} based on the prediction intervals, which is the more conservative assessment.

2.3. 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 2a we added three outliers to the test data of Figure 1 to demonstrate this effect. Again, 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 $e_{i,cross}$ (Schaal and Atkeson, 1994). 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 2b, the effects of the outliers is reduced.

2.4. Data Compression

Despite that the original version of LWR is purely memory-based and does not attempt any kind of data compression, it is straightforward to add this property. The “infinity” of local linear models can be converted into a finite set of piecewise linear models. An efficient way to do this is by means of a recursive tessellation of the inputs space, another nonparametric technique well-known from decision tree induction (Breiman et al., 1984). K-d tree partitioning is the most common of these methods (Friedman et al., 1977). K-d trees partition the input space into hyper-rectangles by recursively splitting the partitions along one input dimension. There are many different ways when and where to split, and we will sketch only what was used in our algorithms.



(a)

(b)

Figure 2: Influence of outliers on LWR: (a) no outlier removal, (b) with outlier removal

For building piecewise linear models, our method starts with one partition which includes all the data. Local linear models are fit to the center of the partition and all its vertices. by LWR. If the local linear model at the center is capable of predicting the outputs of the LWR lookups at the vertices within a certain error margin, the partition is not split anymore, and, if the prediction intervals of the lookup at the center are narrow enough, the partition will be marked valid. The outputs of all query points falling into this partition can be predicted with the linear model at the center.

If the linear model of the center of the partition fails to predict the vertices well enough, the partition is split at the center along the dimension where the greatest mismatch between predicted value and real value had been encountered. Then the process repeats with the newly created partitions. The algorithm terminates when no more splitting needs to be done. Figure 3 shows the result of this method for the function introduced in Figure 1.

As Figure 3 illustrates, the piecewise linear fit of the data represents the given curve very well. At locations where the data has high frequencies, more partitions are allocated than at the more linear regions. By changing the error margin when to accept a partition and its local linear model, the piecewise fit could become coarser or finer. One advantage of this method over other piecewise linear partitioning techniques is that the partitioning of the input space is only done in order to find candidate centers for the linear models. For the local linear fits, *all data* are always included in the LWR lookup. Thus, there is no hard but rather a soft partitioning of the input space since each local linear models also includes data from the neighboring partitions. Due to space limitations, this data compression method cannot be discussed in detail. It can be applied in conjunction with all the methods mentioned so far.

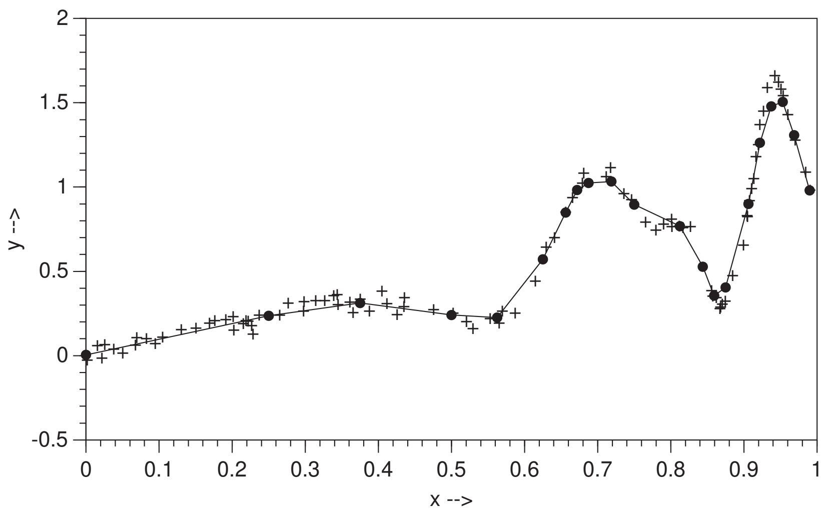


Figure 3: Piecewise linear fit of the data from Figure 1; the black dots denote the boundaries between two neighboring partitions.

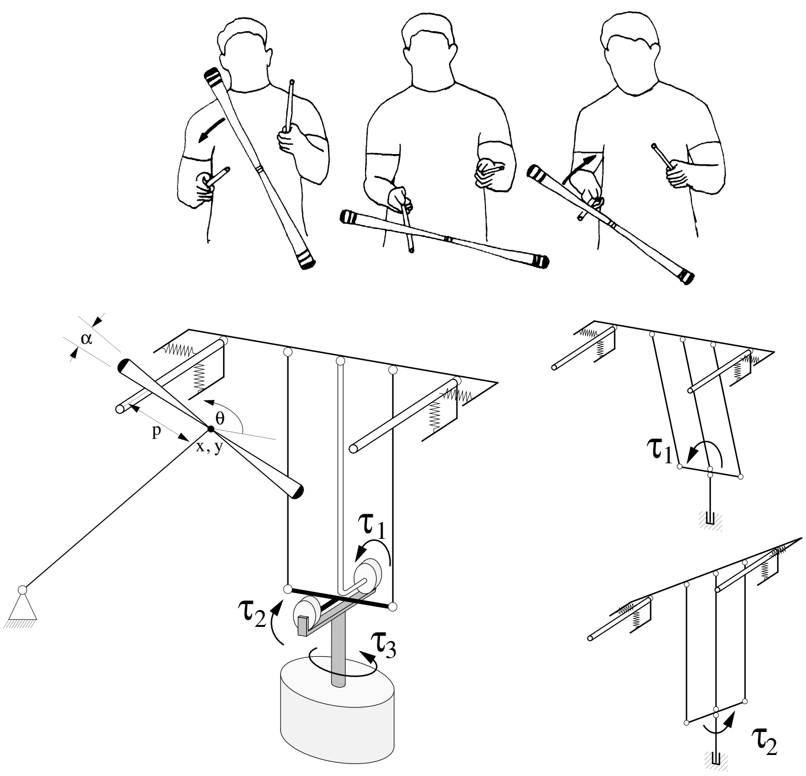
3. REAL-TIME LEARNING OF ROBOT JUGGLING

We have constructed a system for experiments in real-time motor learning (van Zyl 1991). The task is a juggling task known as “devil sticking”. A center stick is batted back and forth between two handsticks (Figure 4a). Figure 4b shows a sketch of our devil sticking robot. The juggling robot uses its top two joints to perform planar devil sticking. Hand sticks are mounted on the robot with springs and dampers. This implements a passive catch. The center stick does not bounce when it hits the hand stick, and therefore requires an active throwing motion by the robot. To simplify the problem the center stick is constrained by a boom to move on the surface of a sphere. For small movements the center stick movements are approximately planar. The boom also provides a way to measure the current state of the center stick. Ultimately we want to be able to perform unconstrained three dimensional devil sticking. The task state is the predicted location of the center stick when it hits the hand stick held in a nominal position. Standard ballistics equations for the flight of the center stick are used to map flight trajectory measurements $(x(t), y(t), \theta(t))$ into a task state. The dynamics of throwing the devilstick are parameterized by 5 state and 5 action variables, resulting in a 10/5-dimensional input/output model for each hand:

$$\mathbf{x} = \left(p, \theta, \dot{x}, \dot{y}, \dot{\theta}\right) \tag{3.1}$$

The task command is given by a displacement of the hand stick from the nominal position (x_h, y_h), a center stick angular velocity threshold to trigger the start of a throwing motion $\dot{\theta}_t$, and a throw velocity vector (v_x, v_y).

$$\mathbf{u} = \left(x_{h,} y_h, \dot{\theta}_t, v_x, v_y\right) \tag{3.2}$$



(a)

(b)

Figure 4: (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

Every time the robot catches and throws the devil stick it generates an experience of the form:

$$\left(\mathbf{x}_k, \mathbf{u}_k, \mathbf{x}_{k+1}\right) \tag{3.3}$$

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:

$$\mathbf{u}_k = \hat{f}^{-1}\left(\mathbf{x}_k, \mathbf{x}_{k+1}\right) \tag{3.4}$$

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

$$\mathbf{u}_k = \hat{f}^{-1}\left(\mathbf{x}_k, \mathbf{x}_d\right) \tag{3.5}$$

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.

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:

$$\hat{\mathbf{x}}_{k+1} = \hat{f}\left(\mathbf{x}_k, \hat{\mathbf{u}}_k\right) \tag{3.6}$$

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 $\boldsymbol{\beta}$. 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:

$$\frac{\partial \hat{f}}{\partial \mathbf{u}} \Delta \hat{\mathbf{u}} = \hat{\mathbf{x}}_{k+1} - \mathbf{x}_d \tag{3.7}$$

The pseudo-inverse of the matrix $\partial \hat{f} / \partial \mathbf{u}$ 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. 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 unreasonably high 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 (3.7) only works if sufficient data exists at the query sites. The inevitable model errors will make the robot explore randomly, leading to dispersed data. This gives little chance for model improvements. Imagine we had to place data in a (hyper-)cube of normalized edge length 0.1. In a 3-dimensional input space there are 10^3 such cubes which leaves some probability to finally arrive at the goal. In a 10-dimensional state space, there are 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.

Both requirements are fulfilled by a simple exploration method, the shifting setpoint algorithm (SSA) (Schaal and Atkeson, 1994). The SSA attempts to decompose the control problem into two separate control tasks on different time scales. At the fast time scale, it generates controls as a nonlinear regulator by trying to keep the controlled system at the 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 extends its knowledge 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 how to control. This data can be used by more sophisticated control algorithms for planning or further exploration.

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 is collected somewhere in state-action space of each hand. An abstract illustration for this is given in Figure 5-top-left→5-top-right.

(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 $(\mathbf{x}_S, \mathbf{u}_S)$ of the hand and an LQ controller is calculated for its local linear model:

$$\begin{aligned} &\mathbf{x}_s = \mathbf{A}\mathbf{x}_s + \mathbf{B}\mathbf{u}_s + \mathbf{c}, \quad or \\ &\mathbf{x}_{s,out} = \mathbf{x}_s - \mathbf{c} = \mathbf{A}\mathbf{x}_s + \mathbf{B}\mathbf{u}_s \end{aligned} \tag{3.8}$$

As in (3.7), the linear model is estimated by LWR. 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.

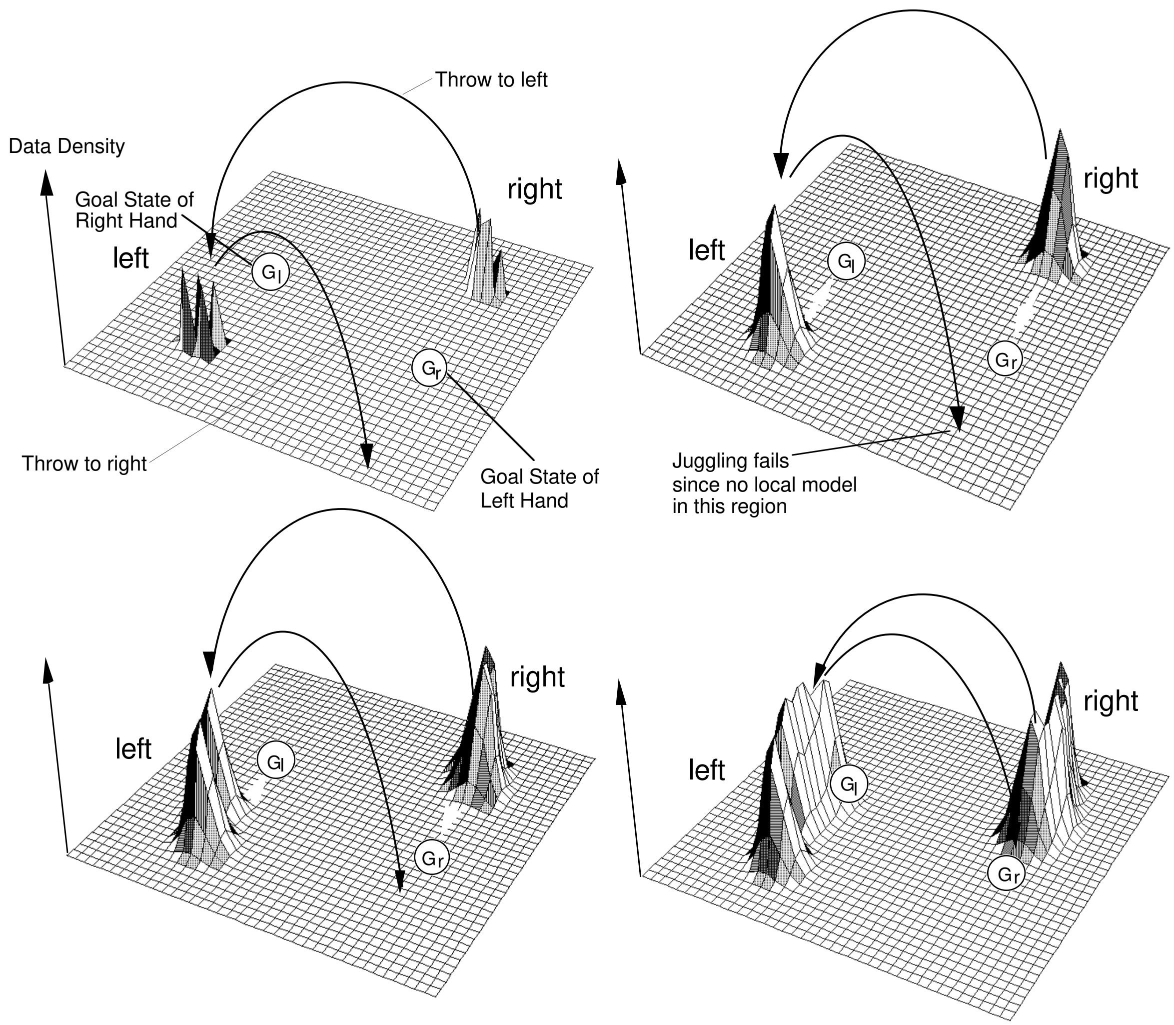


Figure 5: Abstract illustration how the SSA algorithm collects data in space: top-left: sparse data after the first few hits; top-right: high local data density due to local control in this region; bottom-left: increased data density on the way to the goals due to shifting the setpoints; bottom-right: ridge of data density after the goal was reached

(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. The shifting algorithm executes as follows:
   1) calculate the error of the predicted output state: $\mathbf{err}_{S,out} = \mathbf{x}_{desired} - \mathbf{x}_{S,out}$
   2) solve the equation:
   $$\frac{\partial \mathbf{x}_{S,out}}{\partial \mathbf{u}_s} \Delta \mathbf{u}_s = \mathbf{B} \Delta \mathbf{u}_S = \alpha \ \mathbf{err}_{S,out}$$
   for $\Delta \mathbf{u}_s$, e.g., by singular value decomposition (Press et al, 1989); $\alpha \in [0,1]$ determines how much of the error should be compensated for in one step.
   3) update $\mathbf{u}_S$: $\mathbf{u}_S = \mathbf{u}_S + \Delta \mathbf{u}_S$ and calculate new predicted output setpoint state $\mathbf{x}_{S,out}$ with a LWR lookup.
   4) assess the fit for the updated setpoint with the help of prediction intervals. If the quality is above a certain threshold, continue with 1), otherwise terminate shifting.

   Shifting occurs for both $\mathbf{x}_S$ and $\mathbf{x}_{S,out}$. After shifting, the kernel k (cf. Eqn. (2.9) is optimized by minimizing the local cross validation error MSE_{cross}. (In Figure 5, the goal setpoints are given explicitly, but they can also develop automatically from the requirement to throw the devilstick increasingly close to a place, in which the other hand has data, i.e., $\mathbf{x}_{desired,left} = \mathbf{x}_{S,right}$ and $\mathbf{x}_{desired,right} = \mathbf{x}_{S,left}$.)

(4) The SSA repeats itself by collecting data in the new regions of the workspace until the setpoints can be shifted again (Fig. 5-bottom-left). The procedure terminates by reaching the goal, leaving a (hyper-) ridge of data in space (Figure 5-bottom-right).

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

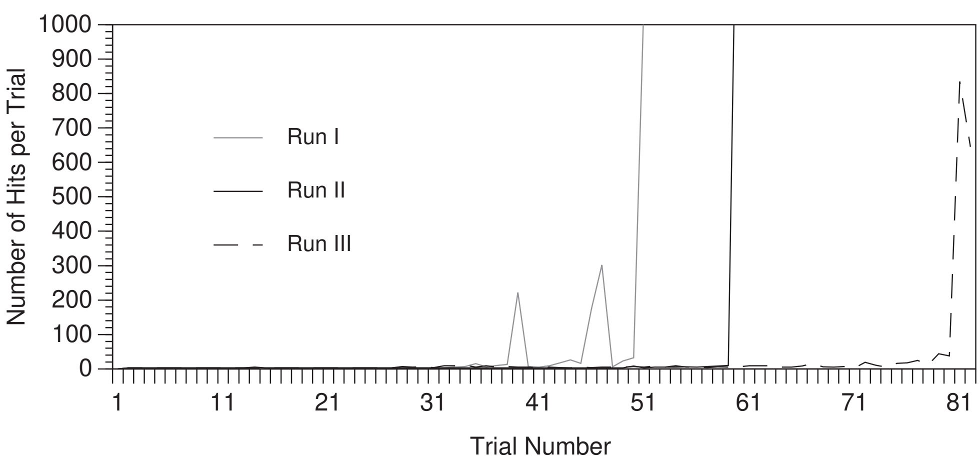


Figure 6: Learning curves of devil sticking for three runs.

We tested the SSA in a noise corrupted simulation and on the real robot. Learning curves of the real robot are given in Figure 6. 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 a break-through occurs and the robot rarely loses the devilstick after that.

The real robot (Figure 6) takes more trials to achieve longer juggling runs. In our first experiments, its performance was not very consistent. Applying LWR and a LQ controller in the above described form turned out to undersample certain directions in control space. Regression analyses need full rank data matrices to achieve good results. By undersampling one direction, the regression matrix becomes close to singular. To avoid this problem some small amount of uniformly distributed noise was added to the controls. The performance of three runs is shown in Figure 6 (note the scale of the ordinate). On average, humans need roughly a week of 1 hour practicing a day before they learn to juggle the devilstick. With respect to this, the robot demonstrated rather nice learning performance, but the controller is still local and could not correct large perturbations, and due to the boom the task is slightly simplified. Future work will attempt to further improve learning rate and robustness.

4. 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., An et al., 1988; Sutton, 1990; Jordan, 1992; Moore, 1991). Using memory-based or nonparametric models, however, has only recently received increasing interest. One of the 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, too, particularly from Bayesian statistics.

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 (Hertz et al., 1991). 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 nonparametric 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. 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. Additionally, we also presented a method for data compression. Data compression on basis of k-d tree partitioning will be useful when sufficient data was collected in a region such that there is enough confidence to build a local parametric model. The piecewise local models which are generated by our method can be used in form of a lookup table which quickly provides predictions and derivatives at a query point. Smoothing across partition can be done by interpolation and will guarantee continuous first derivatives. The LWR-k-d-tree can also be built on-line and in combination with the regular LWR learning. If the k-d tree does not provide a valid partition for a query, the system falls back to the normal LWR mode, otherwise the result is generated by the LWR-k-d-tree.

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 15 milliseconds 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 (1992), 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 (Dyer and McReynolds, 1970) 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 (Schaal and Atkeson, 1993, Schaal and Sternad, 1993), 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 (Schaal et al., 1992), and we picked an initial action of which we knew that it was in the right ballpark. 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.

5. 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 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.

ACKNOWLEDGMENTS

Support was provided by the Air Force Office of Scientific Research, by the Siemens Corporation, the German Scholarship Foundation and the Alexander von Humboldt Foundation to Stefan Schaal, and a National Science Foundation Presidential Young Investigator Award to Christopher G. Atkeson. We thank Gideon Stein for implementing the first version of LWR on a DSP board, and Gerrie van Zyl for building the devil sticking robot and implementing the first version of learning of devil sticking.

REFERENCES

Aboaf, E. W., Atkeson, C. G., & Reinkensmeyer, D. J. (1988). "Task-level robot learning." In: *Proceedings of the IEEE International Conference on Robotics and Automation*, April 24-29, Philadelphia, Pennsylvania.

Aboaf, E. W., Drucker, S. M., & Atkeson, C. G. (1989). "Task-level robot learing: Juggling a tennis ball more accurately." In: *Proceedings of IEEE Interational Conference on Robotics and Automation*, May 14-19, Scottsdale, Arizona.

An, C. H., Atkeson, C. G., & Hollerbach, J. M. (1988). *Model-based control of a robot manipulator*. Cambridge, MA: MIT Press.

Atkeson, C. G. (1992). "Memory-based approaches to approximating continuous functions." In: Casdagli, M., & Eubank, S. (Eds.), *Nonlinear Modeling and Forecasting*, pp.503-521. Redwood City, CA: Addison Wesley.

Barto, A. G., Bradtke, S. J., & Singh, S. P. (1991). "Real-time learning and control using asynchronous dynamic programming." Technical Report 91-57, University of Massachusetts at Amherst, Amherst: Univ. of Massachusetts.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Belmont, CA: Wadsworth International Group.

Cleveland, W. S. (1979). "Robust locally weighted regression and smoothing scatterplots." *Journal of the American Statistical Association*, **74**, pp.829-836.

Dyer, P., & McReynolds, S. R. (1970). *The computation and theory of opitmal control*. New York: Academic Press.

Farmer, J. D., & Sidorowich (1987). "Predicting chaotic time series." *Phys. Rev. Lett.*, **59 (8)**, pp.845-848.

Friedman, J. H., Bentley, J. L., & Finkel, R. A. (1977). "An algorithm for finding best matches in logarithmic expected time." *ACM Transactions on Mathematical Software*, **3 (3)**, pp.209-226.

Hertz, J., Krogh, A., & Palmer, R. G. (1991). *Introduction to the theory of neural computation*. Redwood City, CA: Addison Wesley.

Jordan, I. M. (1992). "Supervised learning with a distal teacher." In: *Cognitive Science*, **16**, 3, pp.307-354.

Mallows, C. L. (1966). "Choosing a subset regression." unpublished paper presented at the annual meeting of the American Statistical Association, Los Angles, .

Moore, A. (1991a). "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*. San Mateo, CA: Morgan Kaufmann.

Myers, R. H. (1990). *Classical and modern regression with applications*. Boston, MA: PWS-KENT.

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.

Schaal, S., Atkeson, C. G., & Botros, S. (1992b). "What should be learned?." In: *Proceedings of Seventh Yale Workshop on Adaptive and Learning Systems*, p.199-204, New Haven, CT.

Schaal, S., & Atkeson, C. G. (1993b). "Open loop stable control strategies for robot juggling." In: *IEEE International Conference on Robotics and Automation*, **3**, pp.913-918, Georgia, Atlanta.

Schaal, S., & Sternad, D. (1993c). "Learning passive motor control strategies with genetic algorithms." In: Nadel, L., & Stein, D. (Eds.), *1992 Lectures in complex systems*. Redwood City, CA: Addison-Wesley.

Schaal, S., & Atkeson, C. G. (1994a). "Robot juggling: An implementation of memory-based learning." *Control Systems Magazine*, **14**, 1, pp.57-71.

Sutton, R. S. (1990). "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming." In: *Proceedings of the International Machine Learning Conference*.

van Zyl, G. (1991). "Design and control of a robotic platform for machine learning." MS Thesis, Cambridge, MA: Massachusetts Institute of Technology, Dept. of Mech. Engineering.

Wahba, G., & Wold, S. (1975). "A completely automatic french curve: Fitting spline functions by cross-validation." *Communications in Statistics*, **4 (1)**, .