Learning Multidimensional Joint Control of a Robot using Receding Horizon Locally Weighted Regression

Chris Lehnert and Gordon Wyeth
School of Engineering Systems, Queensland University of Technology
2 George St, Brisbane QLD 4001
Email: {c.lehnert, Gordon.wyeth}@qut.edu.au

Abstract

In this paper we explore the ability of a recent model-based learning technique Receding Horizon Locally Weighted Regression (RH-LWR) useful for learning temporally dependent systems. In particular this paper investigates the application of RH-LWR to learn control of Multiple-input Multiple-output robot systems. RH-LWR is demonstrated through learning joint velocity and position control of a three Degree of Freedom (DoF) rigid body robot.

1 Introduction

With the increasing move from industrial robots to outdoor, service and domestic applications, modern robot designs are becoming increasingly complex. These complexities are introduced by changing environmental conditions, and a need for human interaction, requiring safer, lighter and more flexible designs. Taking all of these circumstances into account causes standard control models to become increasingly difficult to produce. Therefore, there is a growing need for control systems that can learn to compensate with these changing and nonlinear complexities associated with modern applications.

For this reason model based learning techniques are becoming increasingly popular in robotics, as they have been shown to be useful for learning models to control complex and nonlinear robots. One popular model based technique is Locally Weighted Regression (LWR) [Atkeson, Moore & Schaal, 1997a], a fast and efficient non-parametric regression technique that can globally approximate nonlinear functions by using independent local-linear models within a function's input space.

The current method to apply model-learning techniques to a robot for control involves the process of learning the inverse dynamics of the system for use in a computed torque control scheme. Model learning techniques have only been applied to a small set of robots, modelled as kinematic chains with standard rigid body links.

For the case of a rigid body robot, the inverse dynamics consist of a unique mapping from joint space to torque space and therefore can be directly learnt. However, there is a recent trend for modern robot designs to incorporate elastic joints for increased efficiency and safer human interaction. For the case of an elastic jointed robot the inverse dynamics are not as straightforward and require indirect inverse methods for use with computed torque control [De Luca & Book, 2008]. Attempting to control joint position or velocity of an elastic jointed robot is an example of a temporally dependent control system. This is because the control input (i.e actuator torque or voltage) does not instantaneously affect the desired output at the current time.

It was previously demonstrated in [Lehnert & Wyeth, 2011] that using current methods to apply model-based learning techniques for control of robots when the system is temporally dependent does not produce suitable controllers. This work introduced the concept of a *horizon of temporal dependence* which illustrates why current model-based learning techniques fail to learn controllers when acting outside of this horizon. A new approach was proposed, Receding Horizon Locally Weighted Regression (RH-LWR) which uses a receding horizon of future system outputs to learn controllers of temporally dependent systems. In this work an initial experiment was conducted successfully showing RH-LWR can learn to control a single Degree of Freedom (DoF) Series Elastic Actuator (SEA).

This paper further investigates the application of RH-LWR to multidimensional robots for learning temporally dependent control tasks. In particular this paper explores the ability of RH-LWR to learn temporally dependent control tasks of joint velocity and position control of a rigid body robot. These tasks where chosen as the control solution can be easily computed and evaluated using standard rigid body dynamic equations.

This paper firstly presents a short overview of model-based learning techniques for robot control, focusing on the popular LWR method. Section III outlines the RH-LWR strategy, highlighting how it can be applied to learn Multiple-Input Multiple-Output (MIMO) robotic systems. Section IV then outlines an experiment to apply RH-LWR to a MIMO rigid body robot. The results of this experiment are then presented and discussed in section V, followed by concluding observations and future work.

2 Model Learning for Robot Control

Model learning techniques differ from the field of adaptive control, by producing nonparametric models of a system without any prior knowledge of its structure, whereas adaptive control methods utilise system identification techniques to tune parameters of a given model specified by its mechanical design. The advantage of using model learning methods is that all of the complexities and nonlinear effects of the system associated with training data are inherently built into the nonparametric model.

Model learning methods aim to approximate functions of different model types that have different control applications. Each model is based on the notion that a dynamic system can be represented by a state s_k at the current time k, undergoing an action u_k which takes the system to its next state s_{k+1}. This process can then be observed as an output y_k, which is a function of the current state and action. This dynamic system can be represented in discrete form by the set of equations, analogous to a discrete state space system given as

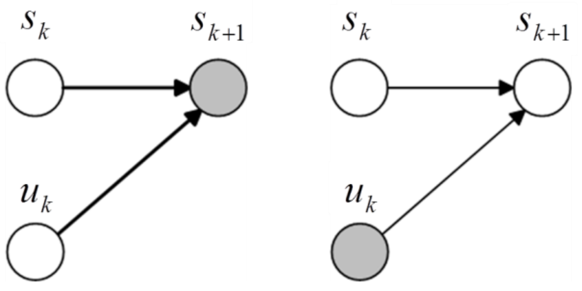
$$s_{k+1} = f(s_k, u_k) + \varepsilon_f \text{ , and} \quad (1)$$

$$y_k = h(s_k, u_k) + \varepsilon_y \text{ ,} \quad (2)$$

where f and h represent the state transition and output functions, ε_f and ε_y are the system and output noise components of the system.

2.1 Model Types

Forward and inverse models are the two most common model types found in model-based learning for robot control, shown in a network graph form in Figure 1.



(a) Forward model (b) Inverse model

Figure 1 Network Graphs of a Forward and Inverse model. The white nodes indicate observed information, whereas the grey nodes indicate the information to be inferred. (a) Forward model: infers the next state given the current state and action. (b) Inverse model: infers the action required to move the system to the next state given the current state.

A forward model infers/predicts the next state and represents the causal relationship between state and action space. As forward models represent the physical properties of a dynamic systems they are relatively simple to learn [Nguyen-Tuong & Peters, 2011b] and can be useful to adjust actions based on a prediction future states, such as the technique of model reference adaptive control [Narendra & Annaswamy, 1989]. A forward model can be approximated by directly learning the functions given in Equations 1 and 2.

Inverse models on the other hand infer the action required to move the system from the current state to a desired future state and represent an anti-causal relationship. This means that they can be ill posed, depending on the type of the system. An inverse model is very useful for control applications as they can be directly used to compute the appropriate action to produce a desired change in state. In order to learn an inverse model the inverse function of the dynamic system is approximated as

$$u_k = f(s_{k+1}, s_k) \text{ ,} \quad (3)$$

Model learning can be defined as a supervised learning technique that is trained on input/output pairs provided by the supervisor, in this case the actual dynamic system that is being learnt. If the outputs are discrete then the problem is a classification system, whereas a continuous output becomes a regression problem. For the continuous case machine learning regression methods can then be employed.

Many regression algorithms have been used for model learning, ranging from Artificial Neural Networks (ANN's), Support Vector Machines (SVM's) to Locally Weighted Learning (LWR) methods. For further information two recently published surveys [Nguyen-Tuong et al., 2011b; Sigaud, Salaün & Padois] give an in depth overview of the current state of the art in model learning techniques for robot control. The current state of the art in online model learning for robot control is recent popular method that has been used in robotics is Locally Weighted Regression, a computationally efficient algorithm in space and complexity.

Other similar methods include Local Gaussian Process Regression (LGP) [Nguyen-Tuong & Peters, 2008] which uses a locally weighted transformation of the input data. In this case every local model stores local training data and is used to produce a probability density function of the output to produce an estimation and uncertainty of a prediction of a new output. This approach has the disadvantage of being computationally expensive as the number of training points increases, while also requiring a large matrix inversion.

2.2 Locally Weighted Regression

Locally Weighted Regression is a supervised learning technique that globally approximates nonlinear functions by a weighted average of multiple linear models locally spaced by weighted kernel functions. At its core LWR is built on the statistical weighted least squares regression technique and was primarily developed by[Atkeson et al., 1997a; Atkeson, Moore & Schaal, 1997b].

LWR has been applied as a robot learning framework for multiple robotic systems ranging from a 7 Degree of Freedom (DoF) SARCOS master arm to a 30 DoF humanoid [Vijayakumar, D'souza, Shibata, Conradt & Schaal, 2002]. LWR's versatility has also been extended to learning control within operational space of redundant systems [Peters & Schaal, 2008].

The LWR algorithm was also redesigned to operate online by incrementally building local models in the input space of the system at locations of given training points. The method incorporates the well-known recursive weighted least squares update rule [Ljung & Söderström, 1983] eliminating the requirement to store past training points. The incremental version of LWR is referred to as

Receptive Field Weighted Regression (RFWR).

An extension to the RFWR algorithm was proposed in [Vijayakumar & Schaal, 2000], which introduced a dimensionality reduction scheme using the Nonlinear Iterative Partial Least Squares algorithm (NIPALS). The extended version of LWR is referred to as Locally Weighted Projection Regression (LWPR) as it projects the input data to a lower dimension before performing the weighted regression. This method was developed in order to be useful for learning nonlinear models of robots with high DoF's.

The basic concept of the incremental version of RFWR, can be broken down into two main steps, composed of the training procedure (Algorithm 1) and the prediction procedure (Algorithm 2). The training procedure computes the weight of the training point for each local model and then performs the least squares update rule. If no local models exist within the space of the training point, designated by a weighted threshold, then a new local model is placed at the current training point. A forgetting factor, λ, is also incorporated into the algorithm in order to weight the previous data over time, allowing the algorithm to account for changes in the global function.

The RFWR algorithm involves estimation of a nonlinear function in the form:

$$y = f(x) + \varepsilon \tag{4}$$

where x is the input vector, y is the output vector and ε is a zero mean noise term. The LWR algorithm estimates this nonlinear function by introducing locally linear models across the input space and forms an output prediction, based on a weighted average of local predictions. The weights of an input, w_i, are computed for each local model using a kernel function; typically a Gaussian kernel in the form of (5) where c_i is the center of the kernel within the input space of the ith local model and D_i is a positive semi-definite matrix determining the shape and size of the Gaussian kernel. The training procedure is as follows:

Algorithm 1: RFWR training procedure

```
1. Given the training point [$x_k$ , $y_k$], where $k$ is the current iteration.
2. For each local model where $i$ =1, 2 ... $n$.
     Compute the local weight:
       $w_i = \exp\left(-\frac{1}{2}(x_k - c_i)^T D_i (x_k - c_i)\right).$   (5)
   3. Update the local means, given:
       $W_k = \lambda W_{k-1} + w_i$
       $\bar{x}_{i,k} = (\lambda W_{k-1}\bar{x}_{i,k-1} + w_i x_k)/W_k$
       $\bar{y}_{i,k} = (\lambda W_{k-1}\bar{y}_{i,k-1} + w_i y_k)/W_k$
   4. Update the inverse covariance matrix $P_i^k$ given zero mean inputs and outputs:
       $\tilde{x}_k = (x_k - \bar{x}_i)$ and $\tilde{y}_k = (y_k - \bar{y}_i)$
       $P_i^k = \frac{P_i^{k-1}}{\lambda} - \frac{P_i^{k-1}\tilde{x}_k\tilde{x}_k^T P_i^{k-1}}{\frac{\lambda^2}{w_i} + \tilde{x}_k^T P_i^{k-1}\tilde{x}_k}$   (6)
   5. Calculate the error vector
       $\varepsilon_k = \tilde{y}_k - \beta_k^T \tilde{x}_k$   (7)
   6. Update the local least squares parameters
       $\beta_k = \beta_{k-1} + w_i P_k \tilde{x}_k \varepsilon_k$   (8)
   end
7. If $w_{max} < w_{threshold}$, where
     $w_{\max} = \max\{w_0, w_1, \ldots, w_n\}$, Then add a new local model with center $c_{n+1} = x_j$
```

After sufficient training points have been given to the learning rule, the prediction rule in Algorithm 2 computes a global prediction $\hat{y}$ using a weighted average of the local predictions from each model.

Algorithm 2 RFWR prediction procedure

```
1. Given the query point $x_q$
2. For each local model where $i$ =1,2,...,$n$.
     Compute the local weight using equation 5 and Compute the local prediction
       $\hat{y}_i = (x_q - \bar{x}_i)\beta_i + \bar{y}_i$   (9)
   End For
3. Compute the global prediction
       $\hat{y} = \frac{\sum_{i=1}^{N} w_i \hat{y}_i}{\sum_{i=1}^{N} w_i}$   (10)
```

Overall, the incremental RFWR algorithm is O(j) in computational time, where j is the number of training points as no previous input values are stored [Vijayakumar et al., 2000], compared with a non-iterative least squares algorithm which has O(j^2) as it stores all or a subset of training points in the covariance matrices.

One additional advantage of LWR is that the algorithm does not suffer from problems of overfitting as more models or training points are added. This allows LWR to be straightforwardly applied as an online model learning algorithm.

2.3 Learning Computed Torque Control

One of the common approaches for learning control of a robot is to approximate the inverse model using an online regression method within a computed torque control scheme. This is possible as it utilises the unique inverse dynamics of a rigid body robot from joint accelerations to joint torques [Nguyen-Tuong et al., 2011b]. Both LWR and LGP have been shown to produce online learning and control of robot manipulators [Nguyen-Tuong & Peters, 2011a; Schaal, Atkeson & Vijayakumar, 2002].

In particular when learning the inverse dynamics, there is a feedforward relationship between the torque input and the acceleration output, with nonlinear forces (such as centripetal and Coriolis forces, and gravity) disturbing the system. The feedforward relationship can be seen in the general robot dynamic equation

$$u = M(q)\ddot{q} + f(q,\dot{q}), \qquad (11)$$

where the inputs are joint torques, u, and the output is the acceleration of the joints, $\ddot{q}$ and M is inertia matrix. In this case the feedforward term is therefore the inertia matrix. LWR has been shown to learn the inverse of this feedforward relationship by localizing the nonlinear terms with respect to the state of the system. In this case as the inertia matrix is well defined and invertible the inverse dynamics cancels the forward dynamics straightforwardly. The resulting global function to be approximated is simply given as

$$u = f(q,\dot{q},\ddot{q}). \qquad (12)$$

where predictions are made for a desired trajectory of joint accelerations, $\ddot{q}_d$, velocities, $\dot{q}_d$, and positions, q_d .

As the learnt system computes the required torque for a defined trajectory it is analogous to a computed torque controller and can be applied straightforwardly. By feeding back actual torques applied and measured joint positions, velocities and accelerations, the system can also update the model using an online training procedure. Figure 2 illustrates this control concept in a feedforward version of computed torque control. Furthermore, in order to compensate for errors in the learnt model an additional PD compensator was used in [Nguyen-Tuong et al., 2011a].

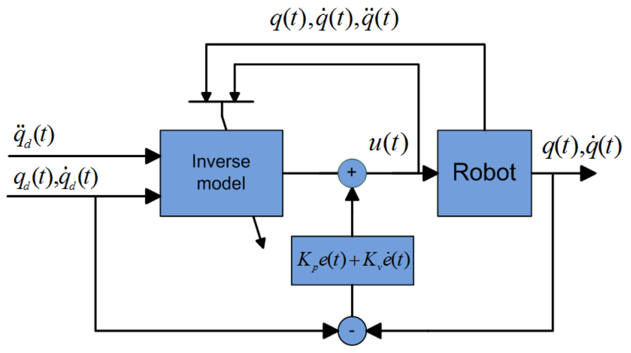


Figure 2 Feedforward Computed torque control using online model learning. The desired velocities and position ns are also used with actual joint values to compute a PD compensator added to the computed torque signal. Both the final torque command and the actual joint variables are then used to update and learn the inverse dynamic model. This figure was adapted from [Nguyen-Tuong et al., 2011a].

3 Receding Horizon Locally Weighted Regression

Recently it was shown in [Lehnert et al., 2011] that learning a model from torques to joint accelerations is a system which is temporally independent, as there exists a direct learnable relationship between the inputs and outputs at any given time. This is due to the fact that this system has a non-zero feedforward relationship. On the other hand if the system has no feedforward relationship, then the system can be defined as temporally dependent and a different approach is necessary in order to learn a suitable controller. The new approach, Receding Horizon Locally Weighted Regression was proposed and an overview of this strategy is given in this section.

3.1 Horizon of Temporal Dependence

In [Atkeson et al., 1997b] *temporal dependence* is defined as the property of a system where the future output is influenced by previous states; a simple example given is when the outcome is the next state defined as the state transition Equation 13. However, it's also possible for the input to not affect the current or next output but some future time step up to a possible *m+1* time steps, where *m* is the order of the system. An extension to this definition of temporal dependence was introduced as the notion of a horizon of temporal dependence, where the *horizon, n*, is defined by:

$$y_{k+n} = f(x_k, u_k) \qquad (13)$$

In other words, the horizon defines the number of time steps before a given input at the current time affects the output, starting at an initial state x_k.

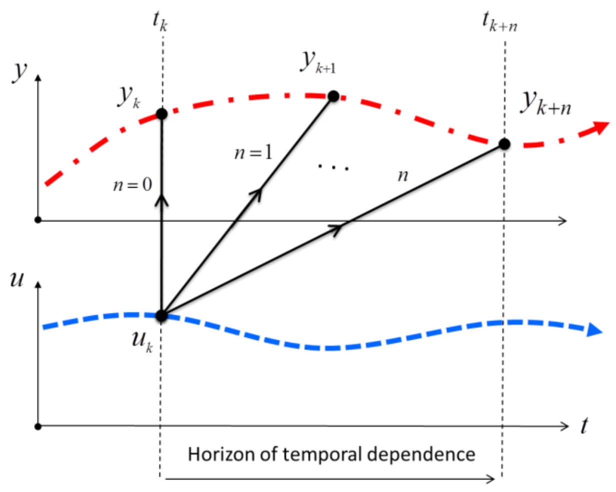


Figure 3 Diagram illustrating the discrete relationship of the current input u_k of a plant to its temporally dependent output for horizons of 0 to *n*.

For the case where there is no feedforward relationship of a system the effect of the input to the output is actually found in future time steps.

RH-LWR is framed around learning a discrete linear state space system in the form

$$x_{k+1} = Ax_k + Bu_k \qquad (14)$$
$$y_k = Cx_k + Du_k, \qquad (15)$$

where k is the current discrete time step, x_k is a vector of the state of the system, y_k is the output of the system and u_k is the input to the system. Lastly A, B, C and D are the discrete state space system matrices. For this application we assume the state space system is both controllable and observable in order to construct a relevant control policy.

The framework of using a receding horizon within LWR is based on a time response representation of the discrete state space system previously defined in [De Moor, 1988] which has had widespread use in system identification techniques [Van Overschee & De Moor, 1994]. The representation is often referred to as the *input output equations* of a state space system and has the form

$$\begin{pmatrix} y_k \\ y_{k+1} \\ \vdots \\ y_{k+n} \end{pmatrix} = \begin{pmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{pmatrix} x_k + \begin{pmatrix} D & 0 & \cdots & 0 \\ CB & D & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ CA^{n-2}B & CA^{n-3}B & \dots & D \end{pmatrix} \begin{pmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+n} \end{pmatrix}, \quad (16)$$

or in the simplified form:

$$Y_f = \Gamma x_k + HU_f \quad (17)$$

where Y_f and U_f are the vector of future time responses of the input and output respectively, evaluated up to the dimension of the system n, Γ is the extended observability matrix and H is an $n \times n$ lower block triangular Toeplitz matrix.

It can be seen from (16) that if the feedforward matrix D is zero then there is no relationship between the input and output at the same time step. Consequently, the current input u_k effects the output in the next time step by a factor of CB. However, if C and B are orthogonal then CB will be zero and the input would not affect the next output of the current time step but a subsequent two time steps is required until the input has an effect. This property is defined as a temporally dependent system with horizon, n, which has a zero D matrix, and the matrix formed by CA^rB equal to 0 for all $0 \le r \le n - 2$.

3.2 Learning a Temporally Dependent system

A temporally dependent system can be learnt by approximating the inverse of (17) as

$$U_f = H^{\dagger} Y_f - H^{\dagger} \Gamma X \quad (18)$$

where † represents the Moore-Penrose pseudoinverse, included because temporally dependent systems will have a singular H matrix. In order to incorporate this representation into a LWR framework the least squares problem for this system can be defined as finding the parameter β to estimate future inputs $\hat{U}_f$ given the relationship

$$\hat{U}_f = \beta\, [Y_f \mid X]. \quad (19)$$

Therefore, β can be found by using past training points and constructing the input output matrices in the form of

$$[Y_{p|j} \mid X_{p|j}] = \left(\begin{array}{cccc|c} y_k & \cdots & & y_{k+n} & x_k \\ y_{k+1} & \cdots & & y_{k+n+1} & x_{k+1} \\ \vdots & \ddots & & \cdots & \vdots \\ y_{k+j} & y_{k+j+1} & & y_{k+n+j} & x_{k+j} \end{array}\right), \text{ and}$$

$$U_{p|j} = \begin{pmatrix} u_k & u_{k+1} & \cdots & u_{k+n} \\ u_{k+1} & u_{k+2} & \cdots & u_{k+n+1} \\ \vdots & \vdots & \ddots & \vdots \\ u_{k+j} & u_{k+j+1} & \dots & u_{k+n+j} \end{pmatrix},$$

where j is the number of training points and the subscript p stands for the past. Then the least squares solution to this problem is

$$\beta = ([Y_{p|j} \mid X_{p|j}]^T [Y_{p|j} \mid X_{p|j}])^{-1} [Y_{p|j} \mid X_{p|j}]^T U_{p|j}.$$

By using this time response representation, the appropriate relationships between the current input and future outputs are incorporated into the inverse model. This essentially allows the model to estimate the appropriate input given a set of future outputs, such that the system moves to the desired output given its current state. Using the regression parameters the future inputs, $\hat{U}_f$, can be estimated given a set of desired outputs, $\tilde{y}_d$ and the current state x_k in the form

$$\hat{U}_f = \beta\, [\tilde{y}_d \mid x_k] = \beta\, [y_d \quad y_{d+1} \quad \dots \quad y_{d+n} \mid x_k]$$

As this system incorporates a linear least squares regression problem it can straightforwardly be applied to a nonlinear system within a LWR method.

3.3 RH-LWR state feedback control

In order to control a temporally dependent output of an unknown nonlinear plant RH-LWR can be applied in series with the plant while supplying a feedback of the current state of the system and a set of desired future outputs, generally available from the path planning system.

The RH-LWR algorithm is then given a horizon vector of future outputs and the current state

$$\tilde{u}_f = f(\tilde{y}_f, x_k) \quad (20)$$

This control law can be interpreted as having two aspects: a feedforward component generated from the desired inputs; and a state feedback component of the system at the current time.

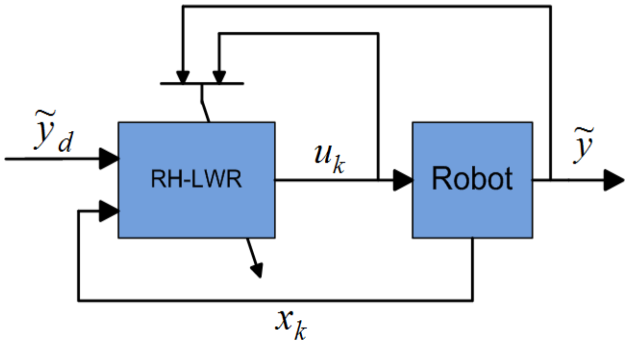


Figure 4 Control block diagram illustrating how RH-LWR can be used to control a nonlinear plant. The architecture is analogous to a state feedback controller which computes an n-step dead beat control law.

This control scheme is similar to the computed torque controller illustrated in Figure 2, but instead of feeding back all the joint variables including acceleration terms, only state feedback is required to compute a feedforward control term given the vector of future desired outputs. By giving the learning scheme future vectors of actual inputs and outputs the system can then update the regression parameters to improve the control predictions. As the RH-LWR algorithm is based on LWL techniques it can also be performed online without any overfitting problems.

4 Experiments

In previous work RH-LWR was shown to successfully learn position control of a single DoF SEA. This control application was identified as a temporally dependent system from the actuator inputs (motor voltages) to its output position. This work also confirmed that using a standard LWR approach fails to learn a suitable controller, with no noticeable appropriate inputs generated across a trajectory of desired output positions.

This control experiment was only applied to a single input single output system. This experiment therefore aimed to control a Multiple Input Multiple Output system. This experiment was performed using a simulation of a three DoF rigid body planar arm using MATLAB and the robotics toolbox by [Corke, 1996]. The model was based on a real three DoF elastic jointed planar arm shown in Figure 5, but for simplicity was modelled with rigid joints.

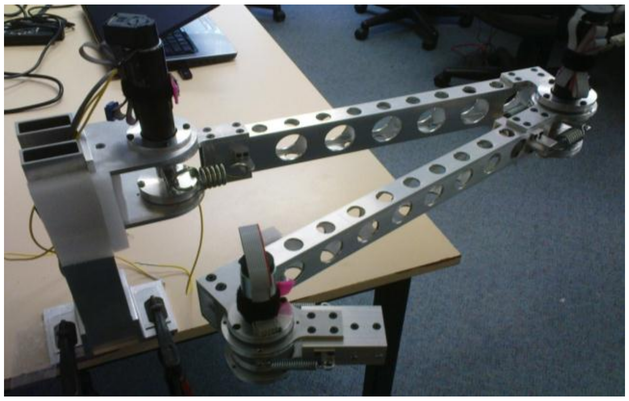


Figure 5 Three DoF planar series elastic jointed robot arm. The arm consists of three brushed DC motors and three spring mechanisms linking the actuators in series to each link.

This experiment was performed on two different temporally dependent systems, firstly to learn joint velocity control, and secondly joint position control. Both of these tasks can be defined as temporally dependent with respect to the input torques to the system. The experiment used the RH-LWR method with a receding horizon of three for the velocity task and a receding horizon of four for the position task, selected to incorporate the horizon of temporal dependence. Notably these control tasks can generally be learnt using model-based methods by defining there higher order derivatives in joint accelerations. However, the idea of this experiment is to show that RH-LWR incorporates this step within the future time steps of the desired output.

4.1 Experimental Setup and Procedure

The simulation of the real three DoF arm incorporated the system state as a vector of three joint positions and velocities

$$x = [q_1 \quad q_2 \quad q_3 \mid \dot{q}_1 \quad \dot{q}_2 \quad \dot{q}_3],$$

and the input was chosen as the set of three actuator torques for each joint as

$$u = [\tau_1 \quad \tau_2 \quad \tau_3].$$

Finally the output of the MIMO system for the velocity and position tasks were defined as

$$y_v = [\dot{q}_1 \quad \dot{q}_2 \quad \dot{q}_3], \text{ and } y_p = [q_1 \quad q_2 \quad q_3].$$

In order to produce an inverse model of the robot arm a training phase was firstly used to generate an initial model of the system. The procedure commonly used to produce the training data in this phase can be collected using a motor babbling method [Saegusa, Sakka, Metta & Sandini, 2010]. However, for simplicity the training data was pre-computed for the desired trajectory that would be used to control. This was done using the Newton Euler inverse dynamics function found in the robotic toolbox. The training data was generated at a rate of 100 Hz for a desired joint position trajectory from zero to $[\pi/2 \quad \pi/4 \quad -\pi/4]$ over a time frame of 5 seconds.

The second phase of the experiment used the trained inverse model to control the desired trajectory of velocities and positions. The controller was run at the same sampling rate as the generated training data and if required the control task was repeated to allow further training of the inverse model eliminating small prediction errors.

5 Results

5.1 Training Results

Figure 6 shows the results from the training phase of the velocity controller. The training results show that the inverse model can accurately reproduce input torques of a MIMO robot arm that was generated using the Newton Euler inverse dynamic equations. The trained RH-LWR system only required a total of 15 local models in the space of the given velocity trajectory in order to reproduce it.

In particular the plot of errors between the actual input torques and the reproduced ones illustrate regions of high nonlinearities therefore producing fluctuations in prediction error for constant local kernel spacing. Similar results were also found with the training procedure of the position controller. However, it was found that in order to produce a similar prediction result the size D_i of the local kernel function needed to be decreased. This is due to the fact that the position controller has a larger horizon of temporal dependence and therefore incorporates higher nonlinearities when trying to reproduce the joint torques. This change in kernel size increased the required number of local models from 15 to 102.

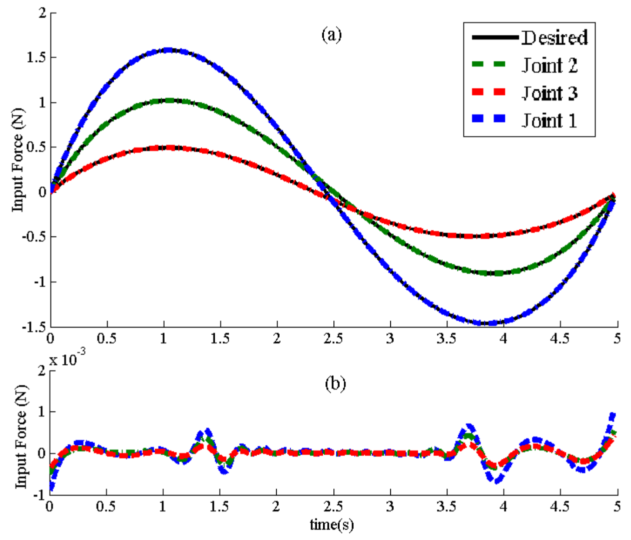


Figure 6 Training performance of learning a velocity controller for 5 seconds of training with a horizon of 3. (a) Shows the plot of the desired inputs in black versus the reproduced signal from the learnt model in blue, green and red, for each DoF respectively. (b) Shows the error of the actual signal to the predicted one for each DoF.

5.2 Controller Performance

After the controllers where learnt using the generated training data they were applied to the modelled robot arm in order to control the desired velocity and position trajectories. The RH-LWR algorithm was applied using the control block diagram previously outlined in Figure 4.

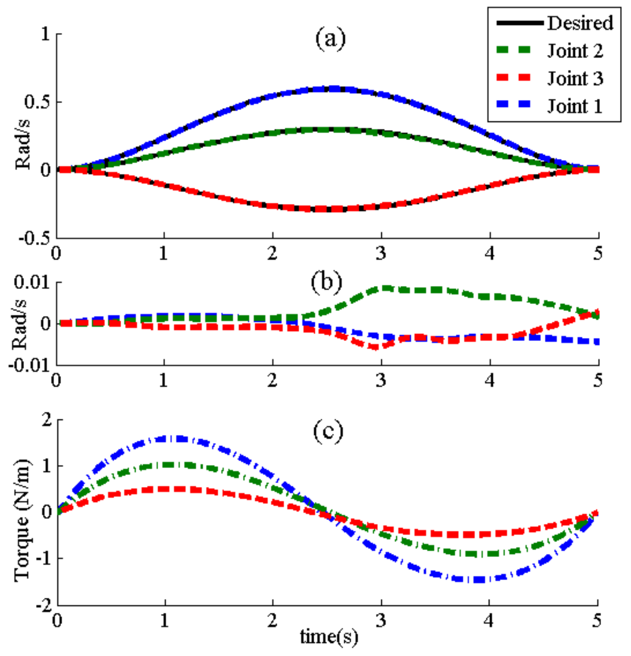


Figure 7 Performance of joint velocity control for a joint velocity trajectory over 5 seconds with a horizon of 3. (a) Shows desired joint velocities in black versus actual in blue, green and red, for each DoF. (b) Is a plot of error of actual joint velocities to desired. (c) Computed control input from the learnt inverse model using RH-LWR

Figure 7 shows the results of the velocity control task. This result shows that the learnt model succesfully estimated torques that control the MIMO robot arm to follow the desired trajectory. Specifically, it can be seen in Figure 7c that the function of torque matches the inverse dynamic torques produced by the robot toolbox in the training procedure.

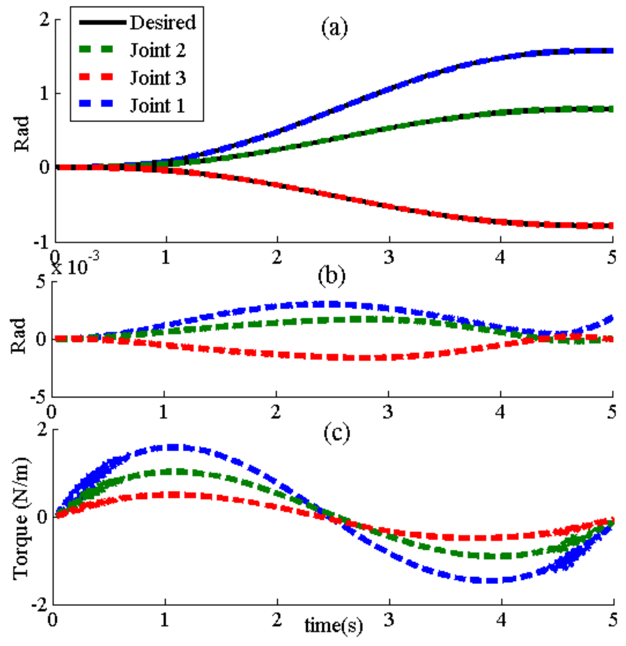


Figure 8 Performance of joint position control for a position trajectory of 0 to π/2, π/4 and –π/4 over 5 seconds for a horizon of 4. (a) Shows desired joint velocities in black versus actual in blue, green and red, for each DoF. (b) The plot of errors of actual joint velocities to desired. (c) plot of computed torques from the learnt inverse model.

Figure 8 shows that by increasing the receding horizon within the learning algorithm by one time step the system can learn position control of a MIMO robot arm. It can also be seen from Figure 8c that the system learns the same function of input torques, given the equivalent trajectory in joint velocities or position. This result demonstrates the ability of using a receding horizon within LWR to learn temporally dependent systems, requiring only a set of future outputs and the current state of the system.

In order to verify that the RH-LWR is truly learning a unique solution to the inverse dynamic model the same training and control procedure for position control was attempted using a receding horizon of 0, therefore evaluating the standard LWR model based approach. Figure 9 shows that the learnt controller fails to learn a suitable position controller for the temporally dependent task. It can be seen that the prediction of torque exceeds a predefined control threshold when the system enters a region that requires control from outside of the receding horizon of 0.

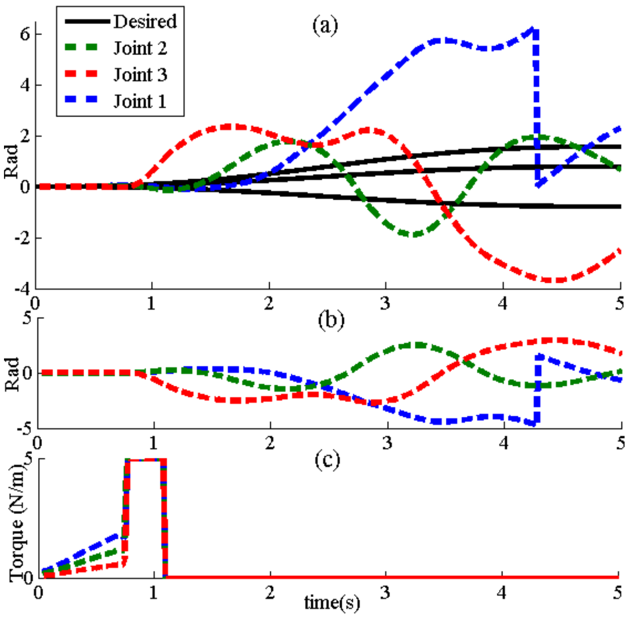


Figure 9 Performance of a learnt position controller using a receding horizon of 0. (a) Shows the actual joint positions in black versus the desired in blue, green and red, for each DoF. (b) The plot of errors of actual joint positions to desired. (c) Plot of computed torques from the learnt inverse model.

7 Discussion and Future Work

7.1 Discussion

It can be seen in Figure 8c that small oscillations of the predicted input occur at the beginning and end of the position trajectory. This result could be caused by RH-LWR attempting to learn the control law during locally flat trajectories when there is a limited amount of information available in the horizon. A possible solution to this problem could be found by ensuring there is sufficient excitation in the training procedure allowing the RH-LWR to converge to a better solution.

However, despite these small oscillations the results show that RH-LWR can be used to learn robot control of temporally dependent MIMO systems. The results also show that for different temporally dependent control tasks the inverse model that is produced by RH-LWR is consistent with actual inverse dynamics of the modeled three DoF robot arm.

7.2 Future Work

The simulated model of the MIMO robot arm was based on a real elastic jointed robot. Future work will be aimed at applying RH-LWR to the real system in order to learn the temporally dependent tasks associated with elastic joint robots. Furthermore we are also intend to investigate the continous on-line learning capabilities of the RH-LWR method.

8 References

Atkeson, C., Moore, A., & Schaal, S. (1997a). Locally weighted learning. Artificial intelligence review, 11(1), 11-73.

Atkeson, C. G., Moore, A. W., & Schaal, S. (1997b). Locally weighted learning for control. Artificial intelligence review, 11(1), 75-113.

Corke, P. I. (1996). A robotics toolbox for MATLAB. Robotics & Automation Magazine, IEEE, 3(1), 24-32.

De Luca, A., & Book, W. (2008). Robots with flexible elements. Handbook of Robotics, B. Siciliano and O. Khatib, Eds., Springer, 287–319.

De Moor, B. (1988). Mathematical concepts and techniques for modelling of static and dynamic systems.

Lehnert, C., & Wyeth, G. (2011). Adding a Receding Horizon to Locally Weighted Regression for Learning Robot Control. Paper presented at the International Conference on Intelligent Robots and Systems. San Francisco, California: IEEE.

Ljung, L., & Söderström, T. (1983). Theory and practice of recursive identification.

Narendra, K. S., & Annaswamy, A. M. (1989). Stable adaptive systems: Prentice-Hall, Inc.

Nguyen-Tuong, D., & Peters, J. (2008). Local gaussian process regression for real-time model-based robot controlIntelligent Robots and Systems (pp. 380-385). IEEE.

Nguyen-Tuong, D., & Peters, J. (2011a). Incremental online sparsification for model learning in real-time robot control. Neurocomputing

Nguyen-Tuong, D., & Peters, J. (2011b). Model learning for robot control: a survey. Cognitive Processing, 1-22.

Peters, J., & Schaal, S. (2008). Learning to control in operational space. The International Journal of Robotics Research, 27(2), 197.

Saegusa, R., Sakka, S., Metta, G., & Sandini, G. (2010). Autonomous Learning Evaluation toward Active Motor Babbling.

Schaal, S., Atkeson, C., & Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. Applied Intelligence, 17(1), 49-60.

Sigaud, O., Salaün, C., & Padois, V. (2011). On-line regression algorithms for learning mechanical models of robots: A survey. Robotics and Autonomous Systems, In Press, Accepted Manuscript

Van Overschee, P., & De Moor, B. (1994). N4SID: Subspace algorithms for the identification of combined deterministic-stochastic systems* 1. Automatica, 30(1), 75-93.

Vijayakumar, S., D'souza, A., Shibata, T., Conradt, J., & Schaal, S. (2002). Statistical learning for humanoid robots. Autonomous Robots, 12(1), 55-69.

Vijayakumar, S., & Schaal, S. (2000). Locally weighted projection regression: An O (n) algorithm for incremental real time learning in high dimensional spaceAmerican Control Conference (pp. 288–293). Citeseer.