10th IFAC Symposium on Robot Control
International Federation of Automatic Control
September 5-7, 2012. Dubrovnik, Croatia

On-line Dynamic Model Learning for Manipulator Control ⋆

Joseph Sun de la Cruz * **Ergun Calisgan** ** **Dana Kulić** ***
William Owen **** **Elizabeth A. Croft** †

* *National Instruments, Austin, TX, USA (e-mail: josephsundelacruz@gmail.com)*
** *University of British Columbia, Vancouver, BC, Canada (email: erguncalisgan@gmail.com)*
*** *University of Waterloo, Waterloo, ON, Canada (e-mail: dkulic@ece.uwaterloo.ca)*
**** *University of Waterloo, Waterloo, ON, Canada (e-mail: bowen@uwaterloo.ca)*
† *University of British Columbia, Vancouver, BC, Canada (email: ecroft@mech.ubc.ca).*

Abstract: This paper proposes an approach for online learning of the dynamic model of a robot manipulator. The dynamic model is formulated as a weighted sum of locally linear models, and Locally Weighted Projection Regression (LWPR) is used to learn the models based on training data obtained during operation. The LWPR model can be initialized with partial knowledge of rigid body parameters to improve the initial performance. The resulting dynamic model is used to implement a model-based controller. Both feedforward and feedback configurations are investigated. The proposed approach is tested on an industrial robot, and shown to outperform independent joint and fixed model-based control.

Keywords: learning control, robot dynamics, robotic manipulators

1. INTRODUCTION

The great majority of current robot manipulators operating in industry are controlled via Proportional Integral Derivative (PID) controllers (Sciavicco and Scicliano (2000)). This control strategy assumes that each degree of freedom is independent and consists of a linear plant. In the case of revolute joint manipulators, this assumption is violated, as the system is non-linear and has coupling between the degrees of freedom. Multiple-input, multiple-output model based control can offer significant advantages over independent joint PID control, including increased performance during high-speed operation, reduced energy consumption and improved tracking accuracy (Nguyen-Tuong et al. (2009)). However, model based control requires that the dynamic model of the manipulator be known. Typically, classical rigid body dynamics is used to formulate the model (Sciavicco and Scicliano (2000)). The rigid body model is a function of the dynamic parameters of the system, such as the robot link masses and inertias. Dynamic parameters can be difficult to obtain, as the required data may not be available from the robot manufacturer, or may not be accurate. In this case, the dynamic parameters can be estimated from experiments. Two main classes of approaches have been used for dynamic parameter estimation: offline and on-line methods. In offline methods of dynamic parameter identification (Khosla (1989); Radkhah et al. (2007)), trajectories exciting the dynamics of the structure are executed, and a least squares formulation is used to find the dynamic parameter estimates. While offline methods can generate accurate estimates of the dynamic parameters, they are not suitable for cases when the parameters may be changing over time, for example due to varying loading or component wear. On-line methods, also known as *adaptive control* (Craig et al. (1987); Spong and Vidyasagar (1987)), use an outer control loop to update the parameter values incrementally based on the error between the predicted and observed performance. However, both dynamic parameter estimation and adaptive control methods assume a known dynamic model structure, and can be sensitive to error when the structure is not modeled accurately (de la Cruz et al. (2011a)).

Recently, data driven methods for learning the manipulator dynamics have been proposed (Schaal et al. (2002); Peters and Schaal (2008)). In this approach, the dynamic relationship is learned directly from the input and output data as function approximation, without relying on a known model structure. Several machine learning techniques have been used, including local linear learning (Schaal et al. (2002)), Gaussian Process Regression (Nguyen-tuong et al. (2008)) and Support Vector Regression (Nguyen-Tuong et al. (2009)). Locally Weighted Projection Regression (LWPR) is frequently applied (Peters and Schaal (2008); Schaal et al. (2002); Vijayakumar et al. (2005)) to learn the dynamic model of a manipulator, due to its use of simple local, linear models which allow

⋆ This work was supported in part by the Natural Sciences and Engineering Research Council of Canada

online and incremental learning. However, due to its highly localized learning, the system must be first be trained in the expected regions of operation, and is reliant on a large training corpus to achieve good performance. In addition, as LWPR uses local models, the model will not generalize well to regions of state space in which there are few training exemplars (de la Cruz et al. (2011a)). The LWPR model can be initialized through the use of motor babbling (Peters and Schaal (2008)), where the robot is controlled with independent joint control and small random movements are executed throughout the state space to generate training data. This approach provides a reliable initialization, but is time consuming, especially as the state space increases with the number of degrees of freedom of the manipulator.

This paper investigates the use of LWPR for modeling the dynamics of a robot manipulator. A technique for improving the initial learning performance by making use of full or partial knowledge of the rigid body model is applied (de la Cruz et al. (2011b)). The obtained dynamic model is used to implement a feedforward and feedback model based controller. The proposed approach is validated experimentally on a 3 degree of freedom (DOF) robotic platform.

2. DYNAMIC MODELING AND CONTROL

The dynamics of a manipulator characterizes the relationship between its motion (position, velocity and acceleration) and the joint torques that cause these motions. The closed-form solution for this relationship is

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} + \mathbf{V_f} \tag{1}$$

where q is the $n \times 1$ joint position vector, $\mathbf{M}(\mathbf{q})$ is the $n \times n$ inertia matrix, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the $n \times 1$ centripetal and Coriolis torque vector, $G(q)$ is the gravity loading vector, $\boldsymbol{\tau}$ is the torque vector, and $\mathbf{V_f}$ describes additional torque components due to forces not included in the rigid body model, such as friction and contact forces.

The simplest control approach is independent joint control. With independent joint control, or decentralized control (Sciavicco and Scicliano (2000)), each joint of the manipulator is treated as an independent system which is decoupled from the rest of the joints. Due to its simple structure, the computational load of this type of controller is extremely low and is easily scalable to systems with a large number of DOF. For example, the Proportional-Derivative control (PD) can be applied at each individual joint. The control signal generated by the PD controller for each joint i of the system, u_{FB_i}, is given by the following equation, where the index i is removed for clarity:

$$u_{FB} = k_p e - k_d \dot{q} \tag{2}$$

where k_p and k_d are proportional and derivative gains, $e = q_d - q$ is the joint space tracking error of the i^{th} joint, and $\dot{q}_d$ is the desired joint velocity for the i^{th} joint. The use of decentralized control does not explicitly account for coupled behaviour of the dynamics of the system in (1). Instead, these effects are treated as disturbances. For a constant disturbance such as gravity loading, the PD controller does not converge to zero tracking error.

2.1 Model-Based Control

Model-based controllers are a broad class of controllers that apply the joint space dynamic equation (1) to cancel the nonlinear and coupling effects of the manipulator. Two configurations can be used: feedforward (FF) and inverse dynamics (feedback) control, also known as computed torque (CT) control.

The goal of nonlinear feedforward control (Sciavicco and Scicliano (2000); An et al. (1989)) is to eliminate the nonlinearity and coupled behaviour in the dynamics according to equation (1) computed about the desired trajectory. With the linearized and decoupled system, a simple PD controller can be applied to achieve stability and disturbance rejection. The control signal $\mathbf{u}$ is thus composed of both the feedforward component $\mathbf{u_{FF}}$, as well as the feedback component $\mathbf{u_{FB}}$:

$$\mathbf{u} = \mathbf{u_{FB}} + \mathbf{u_{FF}} \tag{3}$$

$$\mathbf{u_{FB}} = \mathbf{K_p e} + \mathbf{K_d \dot{e}} \tag{4}$$

$$\mathbf{u_{FF}} = \hat{\mathbf{M}}(\mathbf{q_d})\ddot{\mathbf{q}}_\mathbf{d} + \hat{\mathbf{C}}(\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}) + \hat{\mathbf{G}}(\mathbf{q_d}) \tag{5}$$

where $\mathbf{q_d}$ represents the desired joint angles, $\mathbf{e} = \mathbf{q}_d - \mathbf{q}$, $\hat{\mathbf{M}}(\mathbf{q})$, $\hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$ and $\hat{\mathbf{G}}(\mathbf{q})$ denote the estimates of the actual values in (1), and $\mathbf{K}_p$ and $\mathbf{K}_d$ are proportional and derivative gain matrices.

The feedback gains $\mathbf{K_p}$ and $\mathbf{K_d}$ are typically tuned such that the error equation is stable and a desired level of tracking performance is achieved (Sciavicco and Scicliano (2000)). The advantage of using this control scheme is that the feedforward compensation term $\mathbf{u_{FF}}$ can be computed offline since the desired trajectory, $\mathbf{q_d}$ is known a-priori. However, if the actual trajectory, $\mathbf{q}$, deviates from the desired trajectory, the cancelation of nonlinearities and coupling will be inaccurate.

Assuming that the links of the manipulator behave as rigid bodies,computed torque control (Craig et al. (1987); Nguyen-tuong et al. (2008)) is equivalent to the concept of feedback linearization used in non-linear controls (Sciavicco and Scicliano (2000)), and applies equation (1) to compensate for nonlinear and coupling effects. Unlike the feedforward approach, this compensation is computed about the measured joint position and velocity, as opposed to the desired trajectory. Hence, the control signal, $\mathbf{u}$ is computed as:

$$\mathbf{u} = \hat{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{q}}_\mathbf{r} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\mathbf{G}}(\mathbf{q}) \tag{6}$$

$$\ddot{\mathbf{q}}_\mathbf{r} = \ddot{\mathbf{q}}_\mathbf{d} + \mathbf{u_{FB}} \tag{7}$$

where $\mathbf{u_{FB}}$ is calculated as before in (4).

Again, the feedback gains $\mathbf{K_p}$ and $\mathbf{K_d}$ are typically tuned such that the error equation is stable and a desired level of tracking performance is achieved (Sciavicco and Scicliano (2000)). Whereas the feedforward approach allows a significant portion of computation to be performed offline, the inverse dynamics approach requires online computation of the rigid body dynamics (RBD) equation (1). Thus, the inverse dynamics approach has the potential to be more robust to cases in which the robot's actual trajectory deviates from the desired trajectory, assuming that accurate joint velocities are available. However, in practice, most manipulators are only equipped with joint encoders to

sense joint positions, and numerical differentiation must be applied to obtain joint velocity and acceleration values which result in noisy signals due to the quantization of the encoders. Hence, the application of inverse dynamics would require compensation for sensor noise through state estimation of the joint velocities and acceleration. This is not an issue with the feedforward approach, as the desired joint velocities and accelerations can be computed accurately offline.

A number of adaptive approaches have been proposed which adjust the parameters of the controller based on various criteria including system dynamics, passivity, and robustness to disturbance (Chung et al. (2008)). Alternatively others have proposed robust control frameworks for handling the unmodeled dynamics of the system, such as Quadratic Optimal Control, Nonlinear H∞ control (Chung et al. (2008)). Our proposed LWPR approach is aimed at supporting the implementation of model-based controllers by providing an adaptively learned dynamic model of the robot.

3. LOCALLY WEIGHTED PROJECTION REGRESSION

LWPR approximates a nonlinear function with a set of piecewise local linear models based on the training data that the algorithm receives. Formally stated, this approach assumes a standard regression model of the form

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) + \boldsymbol{\varepsilon} \tag{8}$$

where $\mathbf{x}$ is the input vector, $\mathbf{y}$ the output vector, and $\boldsymbol{\varepsilon}$ a zero-mean random noise term. For a single output dimension of $\mathbf{y}$, given a data point $\mathbf{x_c}$ and a subset of data close to $\mathbf{x_c}$, with an appropriately chosen measure of closeness, a linear model can be fit to the subset of data:

$$y_{ik} = \boldsymbol{\beta_{ik}}^T \mathbf{x} + \boldsymbol{\varepsilon} \tag{9}$$

where y_{ik} denotes the k^{th} subset of data close to $\mathbf{x_c}$ corresponding to the i^{th} output dimension and $\boldsymbol{\beta_{ik}}$ is the set of parameters of the hyperplane that describe y_{ik}. The region of validity, termed the receptive field (Vijayakumar et al. (2005)) is given by

$$w_{ik} = exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x_{ck}})^\mathbf{T} \mathbf{D_{ik}} (\mathbf{x} - \mathbf{x_{ck}})) \tag{10}$$

where w_{ik} determines the weight of the k^{th} local linear model of the i^{th} output dimension (i.e. the ik^{th} local linear model), $\mathbf{x_{ck}}$ is the centre of the k^{th} linear model, $\mathbf{D_{ik}}$ corresponds to a positive semidefinite distance parameter which determines the size of the ik^{th} receptive field. Given a query point $\mathbf{X}$, LWPR calculates a predicted output

$$\hat{y}_i(x) = \sum_{k=1}^{K} w_{ik} \hat{y}_{ik} / \sum_{k=1}^{K} w_{ik} \tag{11}$$

where K is the number of linear models, $\hat{y}_{ik}$ is the prediction of the ik^{th} local linear model given by (9) which is weighed by weight w_{ik} (as computed by 10) associated with the ik^{th} receptive field. Thus, the prediction $\hat{y}_i(\mathbf{x})$ is the weighted sum of all the predictions of the local models, where the models having receptive fields centered closest to the query point are most significant to the prediction. This prediction is repeated i times for each dimension of the output vector $\mathbf{y}$.

The distance parameter, $\mathbf{D}$, (10) is learned for an individual local model through stochastic gradient descent:

$$\mathbf{m}^{n+1} = \mathbf{m}^n - a_L \frac{\partial J_{cost}}{\partial \mathbf{m}} \tag{12}$$

where a_L is the learning rate for gradient descent, $\mathbf{D} = \mathbf{m}^\mathbf{T}\mathbf{m}$ and J_{cost} is a penalized leave-one-out cross-validation cost function (Schaal et al. (2002)) given by:

$$J = \frac{1}{\sum_{i=1}^{L} w_i} \sum_{i=1}^{L} w_i (y_i - \hat{y}_{i,-i})^2 + \frac{\gamma}{n} \sum_{i,j=1}^{N} D_{ij}^2 \tag{13}$$

where L denotes the number of training points, N is the input dimensionality (three times the number of DOF), w_i is the weight associated with the i^{th} training data point calculated according to (10), y_i is the prediction of the local model, $\hat{y}_{i,-i}$ is the prediction of the local model by leaving out the i^{th} training data point, and γ is the penalty term constant. The first term of the cost function represents the mean of the leave-one-out cross-validation error of the local model which ensures proper generalization of the local model. The second term, referred to as the penalty term, ensures that the size of the receptive field does not shrink indefinitely which would cause an increase in the number of local models used by LWPR. Although this would be statistically accurate, the increasing computational and memory requirements would rapidly become too expensive for online computation. Hence, γ can be tuned to achieve the proper tradeoff between generalization and overfitting. However, in (Schaal et al. (2002)), it was found that the regression results are not very sensitive to this parameter and a value of $\gamma = 1.0^{-7}$ was found suitable for a wide range of experiments through empirical evaluation.

In addition to the adjustment of the size of the receptive fields, the number of receptive fields is also automatically adapted. A receptive field is created if for a given training data point, no existing receptive field possesses a weight w_i (10) that is greater than a threshold value of w_{gen}, which is a tunable parameter. The closer w_{gen} is set to one, the more overlap there will be between local models. Conversely, if two local models produce a weight greater than a threshold w_{prune}, the model whose receptive field is smaller is pruned.

Determining the set of parameters $\boldsymbol{\beta}$ of the hyperplane is done via regression, but can be a time consuming task in the presence of high-dimensional input data. To reduce computational effort, LWPR assumes that the data can be characterized by local low-dimensional distributions, and attempts to reduce the dimensionality of the input space $\mathbf{X}$ using Partial Least Squares regression (PLS). PLS fits linear models using a set of univariate regressions along selected projections in the input space which are chosen according to the correlation between input and output data (Schaal et al. (2002)). In addition to reducing computational complexity, this also eliminates statistically irrelevant input dimensions from the local model, thus adding numerical robustness by preventing singularities of the regression matrix due to redundant input dimensions (Schaal et al. (2002)).

4. LWPR INITIALIZATION

Although LWPR has the ability to learn in an online, incremental manner due to its local learning approach, performance deteriorates quickly as the system moves outside of the region of state space it has been trained in (de la Cruz et al. (2011a)). In order to improve the generalization performance of LWPR, an algorithm for incorporating a-priori knowledge from the RBD model (1) into the LWPR algorithm is applied (de la Cruz et al. (2011b)). This is done by initializing the receptive fields in the LWPR model with a first order approximation of the available RBD model:

$$\boldsymbol{\beta} \leftarrow \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{x}}\Big|_{x=\mathbf{x}^*} \tag{14}$$

where $\mathbf{x}^*$ is a new query point for which no previous training data has been observed and $\tau(\mathbf{x}^*)$ is the known or partially known dynamics. Equation 14 represents the linearization of the known system dynamics about a particular state of the system known as the query point, $\mathbf{x}^*$. Although expressions for this linearization can be derived analytically, due to the complexity of the system dynamics, linearization is performed through numerical perturbation about the query point $\mathbf{x}^*$:

$$\beta_{ik} = \frac{\tau_i(\mathbf{x}^*_{k-tol}) - \tau_i(\mathbf{x}^*)}{tol} \tag{15}$$

where β_{ik} is the slope of the linearized model relating the i^{th} output dimension to the k^{th} input dimension, $\mathbf{x}^*_{k-tol}$ is the perturbed query point where tol is subtracted from the k^{th} element of the query vector $\mathbf{x}^*$, and tol is the perturbation tolerance, which is set to a small value for accuracy, i.e. $tol = 10^{-5}$.

For a query point $\mathbf{x}^*$ and the known RBD equation $\boldsymbol{\tau}(\mathbf{x})$, the algorithm shown in Algorithm 1 for initializing the LWPR model is added to the standard prediction procedure of LWPR:

Algorithm 1 LWPR with Prior Knowledge

```
if there is no existing RF centered near x* then
    Compute (β) according to (14)
    Initialize a new RF centered at x* with hyperparameters β
else
    Compute prediction with standard LWPR procedure (10)
end if
```

The determination of whether an existing RF is centered near $\mathbf{x}^*$ is made in the same way as determining whether a new RF should be added, as described in Section 3, i.e., if there is no existing RF that produces a weight (10) greater than w_{gen} given $\mathbf{x}^*$, the initialization procedure is applied.

By evaluating the partial derivative of the RBD equation (1) at the query point, a first-order linear approximation of the system dynamics is obtained, and is used to initialize the model at that point. The size of the receptive field is initially set as a tunable parameter, and is eventually learned optimally through gradient descent.

5. EXPERIMENTS

5.1 Experimental Platform

All experimental work was carried out on a CRS A460 robotic manipulator with a custom designed open architecture control platform (Ho (2003)) developed and fabricated in the UBC CARIS lab. Use of the CARIS open-architecture controller was necessary to bypass the default factory PID control architecture, allowing voltage commands to be issued to the DC servo motors actuating the arm. The Ardence RTX real-time kernel (Ardence) is used to allow RTX processes to take priority over Windows processes during CPU scheduling, thus allowing real-time performance of the control system. Quanser's Wincon 5.0 software (Quanser) allows Simulink® [1] -coded controllers to be compiled into C code which can then be executed on a Wincon client running as an RTX kernel process. The controller sampling time is set to 1 ms, which is the lowest setting achievable by the Wincon system.

For the LWPR experiments, updates to the learned model are processed at a frequency of 100 Hz, or at every 10th sample of the global 1ms sample time. These were the maximum update rates and settings achievable by the controller hardware.

5.2 Dynamic Parameters

The dynamic parameters for the CRS arm (link mass, location of centre of mass and inertia matrix values) were obtained from the results presented in (Radkhah et al. (2007)). The identified parameters were manually checked and corrected to ensure that they made physical sense. For example, the location of the centre of mass location for link 2 was identified in (Radkhah et al. (2007)) as being outside the volume of the link itself. Thus, the identified value was replaced with the approximate location of the volumetric centroid of the link, as physically measured by the author. These parameters were then used to calculate the RBD model (1) and the gravity vector for initialization of LWPR. The motor parameters (rotor inertia, damping and torque constant) for the joint motors were obtained directly from the manufacturer specification sheet.

5.3 Experiments

The independent joint PD, fixed model FeedForward (FF), fixed model Computed Torque (CT) and LWPR controllers were compared on a figure-8 trajectory in the horizontal XY plane. The LWPR controller was initialized through Motor Babbling (MB), rigid body dynamics (RBD) and gravity loading information in order to evaluate their ability to deal with uncertainty in the dynamic model. Due to the limited processing power of the control PC for the CRS arm, only the first three degrees of freedom of the manipulator were used.

The theoretical derivation assumes that the commands to joint actuators were issued directly as torques. However, with the open-architecture controller, motor commands are issued in voltages and not torques. Thus, the LWPR controller was used to learn both the dynamics of the arm

[1] Simulink® is a registered trademark of the MathWorks Inc.

(1) as well as the motor dynamics. The mapping to be learned by LWPR is given by:

$$(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_\mathbf{d}) \mapsto \mathbf{v_m} \tag{16}$$

where $\mathbf{v}_m$ is the vector of motor voltages for each joint.

Feedforward vs Computed Torque Both the feedforward (FF) and Computed Torque (CT) controllers described in Section 2 were implemented experimentally to determine which fixed control strategy is the most effective. For the FF controller, desired joint velocities and accelerations were computed offline through numerical differentiation of the desired joint angles. For the CT controller, state estimation (Ortega and Spong (1988); Yu and Lloyd (1995)) was required to determine the joint velocity and acceleration signals given the joint angles as measured by the encoders. Numerical differentiation was also used for the CT controller. However, due to the quantization of the joint encoders, significant amounts of noise were introduced into the differentiated signals. To solve this, a second-order Butterworth filter was used to smooth out the noise caused by numerical differentiation. As seen in Figure 1 and Table 1, the FF controller is able to outperform the CT controller. Part of the reason for this discrepancy in performance is likely due to the use of the causal Butterworth filter, which injects a significant delay into the resulting filtered signals of the desired velocity and acceleration. If the experimental system was able to operate at a higher frequency, or if sufficient computational resources were available to implement a non-linear observer (Ortega and Spong (1988)), the CT controller may have been able to outperform the FF controller, as the cancelation of the nonlinear dynamics would be more accurate about the actual, rather than the desired trajectory. As the FF controller achieved superior performance to the CT controller, the FF controller is used in subsequent sections for comparison with the LWPR controller.

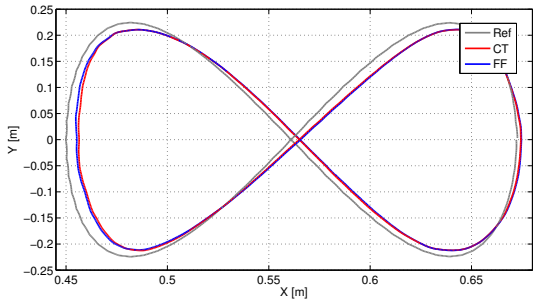

Fig. 1. FF vs CT Tracking Performance

Table 1. FF vs CT RMS Joint Space Error (deg)

Joint	1	2	3	Avg
FF	1.75	0.88	1.51	1.38
CT	1.82	1.05	1.78	1.55

Initialization Technique The effects of different initialization strategies on LWPR tracking performance are shown in Figure 3. For this experiment, LWPR was initialized either through motor babbling, or through full or partial knowledge of the RBD equation. The same motor

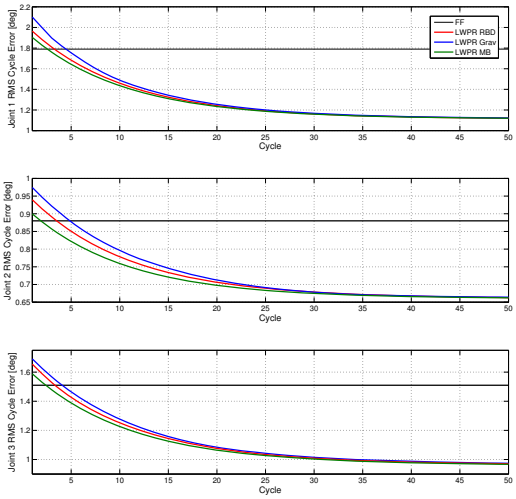

Fig. 2. LWPR Initialization - Joint Space Per Cycle Average Error

babbling strategy as in (de la Cruz et al. (2011a)) was applied for this experiment, resulting in roughly 50,000 initial training points from motor babbling. For LWPR, the full 50,000 training points were used for initialization. Initially, the performance of LWPR with motor babbling initialization is consistently better than both RBD and gravity vector initialization for all three joints. This is due to the inaccuracy in the dynamic parameters used to initialize the models, whereas the motor babbling initialization is based on data collected from the physical system. Initialization with the RBD model consistently outperforms gravity-only initialization, as expected due to the initial lack of compensation for the inertia, $\mathbf{M}(\mathbf{q})$, and Coriolis/centripetal, $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$, terms in 1. However, after repeating the trajectory for 50 cycles, the performance of all three initialization cases converges.

Table 2. LWPR Initialization - Task Space Error

RMS Error (mm)	x	y	z	Avg
RBD Initial	5.21	19.45	0.51	8.39
RBD Final	4.17	14.33	0.35	6.28
Grav Initial	6.55	19.91	0.78	9.08
Grav Final	4.21	15.12	0.37	6.57
MB Initial	4.78	16.54	0.52	7.28
MB Final	4.20	14.25	0.33	6.26

Although the same RBD model is used for the FF controller and for initializing LWPR, the FF controller initially outperforms LWPR with RBD initialization. This can be explained by the fact that LWPR is initialized with linear approximations of the RBD equation and hence, cannot outperform the nonlinear RBD function in the FF controller. LWPR initialization with motor babbling data gives the best initial performance, as the motor babbling data is collected from the actual system. However, the FF controller with an inaccurate system model still outperforms LWPR initially. This is due to the local learning nature of LWPR, whereby the model learned from motor babbling cannot be completely generalized to the necessary regions of state space required to track the figure-8 trajectory.

Table 2 gives the initial and final tracking error of the end-effector in Cartesian space. Most notably, the error in the y-direction is much higher than the other directions. This can be explained by noting that the frequency of the sinusoid for the y-direction is twice as fast as that of the x-direction, thus requiring the robot to move up to twice as fast in the y-direction compared to the x-direction.

The eventual improvement of tracking error over the FF controller illustrates the impact of inaccuracies in the dynamic model, and also highlights the ability of learning algorithms to account for these inaccuracies. Lastly, it should be noted that the initial performance of the learning controller initialized through motor babbling is heavily dependent upon the trajectory used for motor babbling. Although the FF controller initially outperforms LWPR with motor babbling, it is expected that if enough relevant training data is seen by the LWPR, it could also outperform the FF controller from the start.

Comparison with PD Control Figure 3 compares the performance of LWPR initialized with the full RBD model and the performance of a PD controller while tracking a figure-8 trajectory with a period of 8 seconds ($T_c = 8$). As seen in the ZY graph of the figures, the PD controller performance is worst in the z-direction, which is expected as the system is not gravity compensated. Because the LWPR controller is initialized with the full RBD model, its initial performance in the z-direction is much better than the PD controller, indicating that gravity is being compensated for. However, when viewed from the XY plane, it is evident that the initial LWPR performance is not noticeably better than the PD controller. This is due to two factors which cause poor initial performance. First, the use of inaccurate dynamic parameters of the CRS arm in initializing the LWPR system, and second, the use of first-order approximations of the dynamics. The issue of inaccurately known parameters is compensated for by further training of the system for an additional 50 cycles. This allows the LWPR system to obtain 40,000 training points to improve the tracking performance.

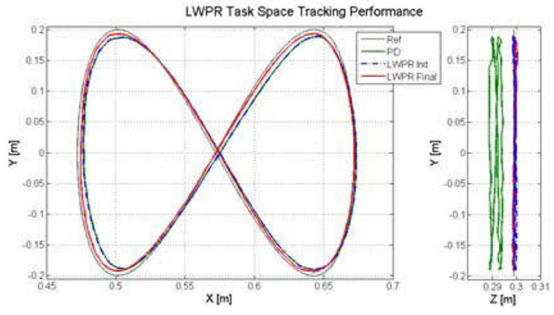

Fig. 3. PD vs LWPR with RBD Init. - Task Space

6. CONCLUSION

This paper presents an approach for learning the nonlinear dynamic model of a robot manipulator during online operation. The relationship between the kinematic variables and the robot command torque is modeled as a weighted sum of local linear models, and the model parameters are learned via linear regression using LWPR. To improve performance during the initial learning phase, when little training data is available, available knowledge from the rigid body model can be used to initialize the models. The proposed approach is tested on a robotic platform; the experiments show that LWPR outperforms independent joint and fixed model control, especially when there are inaccuracies in the fixed model.

REFERENCES

An, C., Atkeson, C., Griffiths, J., and Hollerbach, J. (1989). Experimental evaluation of feedforward and computed torque control. *IEEE Trans Robotics and Automation*, 5(3), 368 –373.

Ardence (2012). Ardence RTX, http://www.intervalzero.com/.

Chung, W., Fu, L.C., and Hsu, S.H. (2008). Motion control. In B. Siciliano and O. Khatib (eds.), *Springer Handbook of Robotics*, 133–159. Springer.

Craig, J., Hsu, P., and Sastry, S. (1987). Adaptive control of mechanical manipulators. *Int J of Robotics Research*, 6(2), 16–28.

de la Cruz, J.S., Kulić, D., and Owen, W. (2011a). A comparison of classical and learning controllers. In *IFAC World Congress*, 1102–1107.

de la Cruz, J.S., Kulić, D., and Owen, W. (2011b). Online incremental learning of inverse dynamics incorporating prior knowledge. In *Int Conf on Autonomous and Intelligent Systems*.

Ho, J. (2003). Open architecture controller for the crs a465 robot arm. Technical report.

Khosla, P. (1989). Categorization of parameters in the dynamic robot model. *IEEE Trans on Robotics and Automation*, 5(3), 261 –268.

Nguyen-Tuong, D., Scholkopf, B., and Peters, J. (2009). Sparse online model learning for robot control with support vector regression. In *IROS*, 3121–3126.

Nguyen-tuong, D., Seeger, M., and Peters, J. (2008). Computed torque control with nonparametric regression models. In *American Control Conference*, 212–217.

Ortega, R. and Spong, M. (1988). Adaptive motion control of rigid robots: a tutorial. In *IEEE Conf on Decision and Control*, 1575–1584.

Peters, J. and Schaal, S. (2008). Learning to Control in Operational Space. *Int J of Robotics Research*, 27(2), 197–212.

Quanser (2005). Wincon 5.0 user's guide, quanser consulting. URL `http://www.quanser.com/`.

Radkhah, K., Kulic, D., and Croft, E. (2007). Dynamic parameter identification for the CRS A460 robot. In *IROS*, 3842 –3847.

Schaal, S., Atkeson, C., and Vijayakumar, S. (2002). Scalable techniques from nonparametric statistics for real time robot learning. *Appl. Intelligence*, 16, 49–60.

Sciavicco, L. and Scicliano, B. (2000). *Modelling and Control of Robot Manipulators*. Springer, 2nd edition.

Spong, M. and Vidyasagar, M. (1987). Robust linear compensator design for nonlinear robotic control. *IEEE Trans Robotics and Automation*, 3(4), 345–351.

Vijayakumar, S., D'souza, A., and Schaal, S. (2005). Incremental online learning in high dimensions. *Neural Computation*, 17(12), 2602–2634.

Yu, H. and Lloyd, S. (1995). Adaptive control of robot manipulators including motor dynamics. In *American Control Conf*, 3803 –3807.