Locally Weighted Learning Model Predictive Control for Elastic Joint Robots

Christopher Lehnert & Gordon Wyeth
School of Electrical Engineering and Computer Science, Queensland University of Technology
(c.lehnert, gordon.wyeth)@qut.edu.au

Abstract

This paper proposes an efficient and online learning control system that uses the successful Model Predictive Control (MPC) method in a model based locally weighted learning framework. The new approach named Locally Weighted Learning Model Predictive Control (LWL-MPC) has been proposed as a solution to learn to control complex and nonlinear Elastic Joint Robots (EJR). Elastic Joint Robots are generally difficult to learn to control due to their elastic properties preventing standard model learning techniques from being used, such as learning computed torque control. This paper demonstrates the capability of LWL-MPC to perform online and incremental learning while controlling the joint positions of a real three Degree of Freedom (DoF) EJR. An experiment on a real EJR is presented and LWL-MPC is shown to successfully learn to control the system to follow two different figure of eight trajectories.

1 Introduction

As the robotics industry moves towards domestic applications, safety becomes vital when human interaction is involved. Popular solutions for safe robot human interaction are designs that have physical elastic compliance or are actively compliant [De Santis, *et al.*, 2008]. For this reason robots are becoming increasingly complex as compliant cables or elastic elements introduce nonlinear and time varying wear and fatigue effects that are both difficult to model and control on a real system [De Luca and Book, 2008]. These problems create extra costs and time associated with design and servicing of these types of robots. All of these difficulties could be accounted for if the robot has the ability to learn its own nonlinear and time varying dynamic model. Giving a robot the ability to learn its own model and subsequently using this to control itself allows a robot to account for its own nonlinear dynamics experienced in the real world, including internal changes from wear, fatigue and its environment. For this reason model based learning techniques are becoming increasingly popular in robotics, as they have shown to be a useful technique for learning dynamic models for control of complex and nonlinear

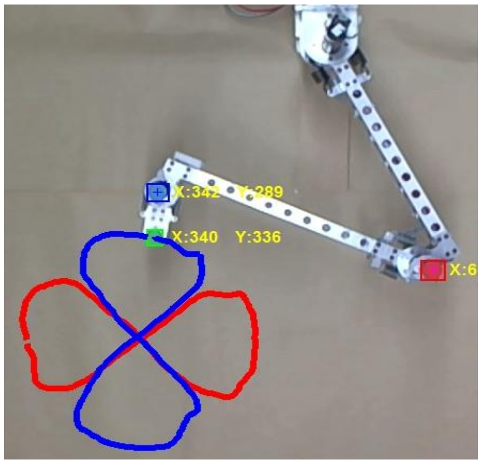


Figure 1 - Image of elastic joint robot arm with a trace of the end effector position after execution of our proposed LWL-MPC system. Two desired figure of eight trajectories are given to the learning controller. The red and blue lines are the end effector trajectory after the system has been trained on 20 iterations of both figure of eight's.

systems. One popular model based technique is Locally Weighted Regression (LWR) [Atkeson, *et al.*, 1997], a fast and efficient non-parametric regression technique that can globally approximate nonlinear functions by using independent local-linear models.

The majority of model learning for robot control techniques has been applied to rigid body dynamics using a computed torque control method. The inverse dynamics for this type of system is a one to one mapping of torque inputs to joint accelerations. Therefore, an inverse dynamics model can be directly learnt using supervised learning of inputs and state information, then used within a computed torque control scheme. However, EJR's are not strictly rigid body systems, which makes learning inverse dynamics increasingly difficult. [De Luca and Book, 2008] outlines that feedback linearization via computed torque control is not straightforward for EJR's and in some cases does not satisfy conditions for exact linearization. Therefore, more advanced control methods are required in order to use model learning techniques to control EJR's and other systems that cannot be controlled using a direct inverse dynamics model.

Model Predictive Control is a successful control method that has been applied to a variety of industrial and real world robotic systems. MPC uses a forward dynamics model combined with an optimization method to produce an optimal feedback control law for each control cycle. The advantages of MPC are its ability to explicitly handle system constraints and delays while being computationally feasible as an online control system. As MPC is based on a forward model it can be effectively applied within a model based learning technique, combining the advantages of the ability to learn its own model while also effectively computing an optimal feedback control law to stabilise the system.

This paper proposes the method of LWL-MPC which uses locally weighted learning in combination with model predictive control to produce an incremental and online system which is demonstrated to simultaneously learn and control a 3DoF Series Elastic Actuator (SEA) robot arm illustrated in Figure 1. This paper firstly gives a background of similar approaches for robot control, and an overview of model learning techniques for robot control. The paper then outlines the locally weighted learning technique of Receptive Field Weighted Regression (RFWR) used in our proposed system. An overview of general MPC is then given illustrating how a linear model can be learned within this method. LWL-MPC is then outlined for nonlinear systems and an experiment to evaluate this on a real EJR is proposed. The results are then given validating the system's ability to perform online learning and control. Finally a discussion of the system and results are given highlighting future directions and open questions of this research.

2 Background

Previous work has been done which investigates the use of model learning techniques with model predictive control. A Fuzzy Multi-Model predictive control (FMMPC) has been proposed in [Li, *et al.*, 2001] that uses Weighted Recursive Least Squares (WRLS) to adapt fuzzy local models. The system then applies model predictive control to the global fuzzy model. The system was demonstrated to model and control thermoplastic injection moulding. This method was found to not yield satisfactory control due to the time variance of the system process and difficulty with initialization of the learning rules of the fuzzy models. This system proposed a batch learning method in response to this result.

Another method uses Locally Weighted Projection Regression (LWPR) combined with a model predictive control method has been proposed for predictive force control for assistive beating heart surgery [Florez, *et al.*, 2011]. However this technique, similar to FMMPC applies the method of MPC to the global nonlinear function. Our proposed technique removes this computational cost by computing local linear MPC's inside the locally weighted framework. Other similar methods propose Neural networks as the learning method combined with model predictive control [Wan and Bogdanov, 2001].

A robust and stable Learning Based Model Predictive Control strategy has also been developed in [Aswani, *et al.*, 2011] which analyses the conditions of the learning system necessary to provide a robust and stable control system. This work generalises the problem and is a good framework for learning robust and stable model predictive control. However, the method relies on a good initial estimate of the model in order to ensure the robustness properties. This work has also been applied on a real quadrotor. The stability and robustness of this approach is based on a given linear model, where an additional function is added to learn the unmodeled dynamics of the quadrotor, such as ground effects [Aswani, *et al.*, 2012]. This system was demonstrated to robustly track an altitude set point.

Other optimal control techniques have been combined with model learning methods. Iterative Linear Quadratic Gaussian with Learned Dynamics (ILQG-LD) [Li and Todorov, 2007] is an approach that iteratively computes local optimal control policies for a specified end goal state of a system. In this method a policy is the combined optimal path and feedback control law along the path. The method uses LWPR to compute a nonlinear dynamics model and uses this to iteratively compute an optimal policy. In this approach both the path planning and control law are iteratively computed for the given forward nonlinear dynamics model. ILQG-LD has been applied to control a single Variable Stiffness Actuator (VSA) [Mitrovic, *et al.*, 2011] and has been further shown to compute effective optimal controllers on simulated arms. A disadvantage with this method is the computational cost of iteratively computing the path and control law to the end goal until convergence is found. However, this approach is presently the most computationally efficient technique for computing a globally optimal path and feedback control law to move a system to a goal state.

3 Model Learning for Robot Control

Model learning methods aim to approximate functions of different model types from actual system data and are used within a robot's control architecture. Typically the model is generated using a supervised nonparametric and nonlinear regression method. 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. Three model types have been frequently used and are illustrated in Figure 2.

Each model type can then be incorporated into different learning control architectures for different advantages. 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}.

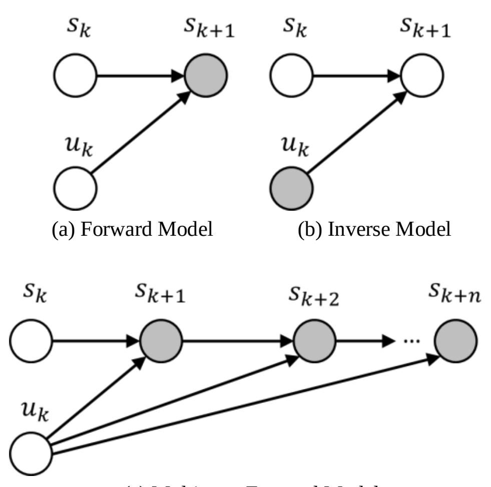


Figure 2 - Network Graphs of Forward and Inverse models. White nodes indicate observed information; grey nodes indicate information to be inferred.

A more in depth review of model learning techniques can be found within two recent surveys [Nguyen-Tuong and Peters, 2011; Sigaud, *et al.*, 2011]. Forward models predict the next state given the current action and state. They generally are well defined causal relationships of a system and can be straightforwardly learnt using regression techniques. An inverse model on the other hand predicts the action required to move the current state of the system to a given next state. An inverse model represents an anti-causal relationship and generally is not well defined due to the possible non-convexity of the inverse relationship.

An extension to the forward model is the Multi-step Forward Model which is useful as it can predict a sequence of future states. Multi-step Forward Model depending on the model type can be used to compute an action that optimises the prediction of the future state. This type of model can be used in Model Predictive Control which has been successfully applied to a large variety of systems. However, general MPC is based on a system model that is predefined, either theoretically or by offline system identification techniques.

If the relationship of the input and output is well defined each type of model can be generally learned using supervised regression techniques. A variety of nonlinear supervised learning techniques have been used. For this application we chose Locally Weighted Regression as it has been a popular technique that is computationally efficient and can be implemented online in an iterative algorithm useful for control applications.

3.1 Locally Weighted Regression

There are a few variants of the LWR algorithm, but the body of the work can be found in [Atkeson, *et al.*, 1997]. An iterative version was developed as RFWR [Schaal and Atkeson, 1998] which incrementally added and updated local linear models to produce an efficient online regression technique. A Partial Least Squares (PLS) solution was then incorporated as a dimensionality reduction method for high dimensional models in Locally Weighted Projection Regression (LWPR) [Vijayakumar and Schaal, 2000]. For this work the RFWR algorithm was investigated as the PLS in the LWPR introduces further complexities when combining the local models with MPC. The RFWR involves estimation of a nonlinear function in the form:

$$\mathbf{s} = \tilde{f}(\mathbf{z}) + \boldsymbol{\varepsilon} \tag{1}$$

where $\mathbf{z}$ is the input vector, $\mathbf{s}$ is the output vector and $\boldsymbol{\varepsilon}$ is a zero mean noise term. The RFWR 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

$$w_i(\mathbf{z}) = \exp\left(-0.5(\mathbf{z}-\mathbf{c}_i)^T \mathbf{D}_i (\mathbf{z}-\mathbf{c}_i)\right) \tag{2}$$

where $\mathbf{c}_i$ is the centre of the kernel within the input space of the i^{th} local model and $\mathbf{D}_i$ is a positive semi-definite matrix determining the shape and size of the Gaussian kernel. The function is then approximated using a WRLS rule which independently updates each locally linear regression weighted on the distance of the local model to the training point. The function can then be represented as a weighted average of each local model in the following form

$$\tilde{f}(\mathbf{z}) = \frac{1}{\mathbf{W}} \sum_{i=1}^{I} w_i(\mathbf{z}) \Phi_i(\mathbf{z}), \quad \mathbf{W} := \sum_{i=1}^{I} w_i(\mathbf{z}) \tag{3}$$

where $\mathbf{W}$ is sum of the all weights acting as the normalising factor, and Φ_i is the i^{th} local model represented by a linear equation in the form

$$\Phi_i(\mathbf{z}) = \boldsymbol{\beta}_i^T(\mathbf{z}-\bar{\mathbf{z}}) + \boldsymbol{\beta}_i^0 \tag{4}$$

The shape of the Gaussian kernels are then learnt online to minimise the prediction error using a gradient descent update rule. This rule includes two learning rates, one to update the diagonal of $\mathbf{D}_i$, representing the variance of each input dimension and a meta rate which updates the off diagonal terms representing the covariance between input dimensions. Lastly, a forgetting factor λ is also applied to the WRLS rule effectively down-weighting previous training data, allowing the system to adapt to changes in the model. RFWR is desirable as it is both computationally efficient in space and time as only local regression parameters are stored and each local model can be evaluated independently and recursively for each training point, allowing it to be computed in an online manner.

4 Model Predictive Control

The general concept of MPC is that by iteratively computing a control input based on a finite horizon optimization of a system model, the system can be efficiently stabilised to a desired set point or trajectory [García, *et al.*, 1989]. Only the next input is applied to the system and the optimization method is then repeated each control step. MPC therefore handles any accumulation of prediction error from the model by recomputing based on new measurements of the system at the next time step.

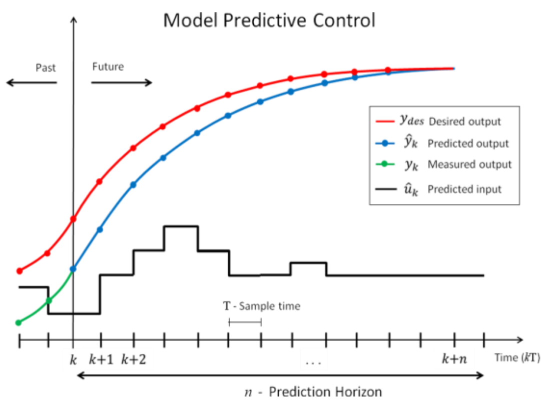


Figure 3 - Model Predictive Control diagram

The advantage of MPC is that the control input can computed each time step with a relatively short horizon using a numerical optimization technique, which can incorporate optimization constraints such as upper and lower bounds on the input or states of the system. This allows MPC to be deployed as an online sub-optimal feedback controller. Furthermore, the model of the plant can also incorporate real world values such as pure time delays and be effectively incorporated into the control method.

The concept of MPC is shown in Figure 3 defining a finite prediction horizon n with a sequence of predicted outputs $\hat{\mathbf{y}}_k$, predicted inputs $\mathbf{u}_k$ and desired outputs $\mathbf{y}_{des}$. In this case the system has been discretised using the variable k to represent the number of time steps with a step size of T. The optimization cost function is then commonly defined as a linear quadratic cost in the form

$$J = \sum_{k=1}^{n} w_y (\mathbf{y}_k - \mathbf{y}_{des})^2 + \sum_{k=1}^{n} w_u \mathbf{u}_k^2, \qquad (5)$$

where w_y weights the tracking error of the system and w_u weights the magnitude of the control input. The selection of weights determines the effective trade-off between tracking performance and input energy. If the model of the system can be defined as a set of linear differential equations then the optimization problem can be solved using standard Quadratic Programming (QP) methods. However, if the system is nonlinear then nonlinear optimization methods are used, such as differential dynamic programming (DDP). A more in depth review of MPC can be found [Morari and H. Lee, 1999].

5 Learning Locally Weighted Model Predictive Control

This section outlines the method of combining model based learning to the framework of MPC to produce LWL-MPC. Firstly, it is shown how a linear model can be estimated and then used within MPC. The linear method is then extended within a locally weighted learning framework for use with nonlinear systems.

5.1 Learning Linear Model Predictive Control

For this method we model a linear system using a discrete state space representation defined as

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}\mathbf{x}_k + \mathbf{B}\mathbf{u}_k + \mathbf{K}\mathbf{e}_k \\ \mathbf{y}_k &= \mathbf{C}\mathbf{x}_k + \mathbf{D}\mathbf{u}_k + \mathbf{e}_k \end{aligned} \qquad (6)$$

where $\mathbf{u}_k \in \mathbb{R}^q$ is a vector of inputs, $\mathbf{y}_k \in \mathbb{R}^l$ is a vector of outputs and $\mathbf{x}_k \in \mathbb{R}^m$ is a vector of states. The matrices $\mathbf{A} \in \mathbb{R}^{m \times m}$, $\mathbf{B} \in \mathbb{R}^{m \times q}$, $\mathbf{C} \in \mathbb{R}^{l \times m}$, $\mathbf{D} \in \mathbb{R}^{l \times q}$ are the discrete state space matrices. $\mathbf{e}_k$ is a white noise disturbance and $\mathbf{K} \in \mathbb{R}^{l \times l}$ represents the Kalman gain of the process noise. MPC uses the future outputs and inputs to compute the optimal input to the system. In this case the future outputs and inputs are defined as

$$\mathbf{y}_k^f := [\mathbf{y}_k \quad \mathbf{y}_{k+1} \quad \dots \quad \mathbf{y}_{k+n}]^T,$$

$$\mathbf{u}_k^f := [\mathbf{u}_k \quad \mathbf{u}_{k+1} \quad \dots \quad \mathbf{u}_{k+n}]^T$$

where the superscript f denotes the future from the current time k to n steps later, where n defines the prediction horizon. The past inputs $\mathbf{u}_{k-n}^p$ and outputs $\mathbf{y}_{k-n}^p$ can also be similarly defined in this manner which represents a history of inputs and outputs from k-n to the current time step k.

The future outputs can further be defined with respect to the system matrices as a set of input output equations from k to n in the form of

$$\mathbf{y}_k^f = \mathbf{\Gamma}_n \mathbf{x}_k + \mathbf{H}_n^d \mathbf{u}_k^f + \mathbf{H}_n^s \mathbf{e}_k^f \qquad (7)$$

where,

$$\mathbf{\Gamma}_n = [\mathbf{CA} \quad \mathbf{CA}^2 \quad \dots \quad \mathbf{CA}^n]^T$$

$$\mathbf{H}_n^d = \begin{bmatrix} \mathbf{D} & 0 & \cdots & 0 \\ \mathbf{CB} & \mathbf{D} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{CA}^{n-1}\mathbf{B} & \mathbf{CA}^{n-2}\mathbf{B} & \cdots & \mathbf{D} \end{bmatrix} \quad \mathbf{H}_n^s = \begin{bmatrix} \mathbf{I}_l & 0 & \cdots & 0 \\ \mathbf{CK} & \mathbf{I}_l & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{CA}^{n-1}\mathbf{K} & \mathbf{CA}^{n-2}\mathbf{K} & \cdots & \mathbf{I}_l \end{bmatrix}$$

which are the input output matrices of this system, $\mathbf{\Gamma}_n$ is the extended observability matrix and $\mathbf{H}_n^d$ and $\mathbf{H}_n^s$ are lower block triangular Toeplitz matrix. The future output vector can also be chosen to start at k+1 in order for $\mathbf{H}_n^d$ to have **CB** along its diagonal improving the conditioning of the matrix as in most systems **D** may be zero. For this work we chose to start the output vector at k+1.

The advantage of representing the system in input output form allows the system to be easily estimated given a set of sample points of actual inputs and outputs. The assumption for this work is that the system is fully observable and we can generate measurements of the output and state of the system. In this case the linear model can be simply estimated using a least squares approach.

If r sample points of the future outputs, state and input are collected then an estimate of the input output matrices can be made by defining the regression matrices

$$\begin{aligned}\mathbf{Y}_f &:= [\mathbf{y}_k^f \quad \mathbf{y}_{k+1}^f \quad \cdots \quad \mathbf{y}_{k+r}^f]^T \\ \mathbf{U}_f &:= [\mathbf{u}_k^f \quad \mathbf{u}_{k+1}^f \quad \cdots \quad \mathbf{u}_{k+r}^f]^T \\ \mathbf{X}_f &:= [\mathbf{x}_k \quad \mathbf{x}_{k+1} \quad \cdots \quad \mathbf{x}_{k+r}]^T\end{aligned} \tag{8}$$

and then computing the least squares solution for the input output matrices as

$$[\hat{\mathbf{\Gamma}}_n \ \hat{\mathbf{H}}_n^d] = ([\mathbf{X}_f \ \mathbf{U}_f]^T[\mathbf{X}_f \ \mathbf{U}_f])^{-1}[\mathbf{X}_f \ \mathbf{U}_f]\mathbf{Y}_f\,. \tag{9}$$

The residuals of this least squares problem is then the stochastic process of future errors

$$\mathbf{r}_f = \mathbf{H}_n^s\mathbf{E}_f\,, \tag{10}$$

where $\mathbf{E}_f$ is similarly defined as a matrix of future error vectors. This approach can also be solved using a recursive least squares algorithm by iteratively computing the regression for each sample point. An estimate of the future outputs can then be predicted given the query point $[\mathbf{x}_q, \ \mathbf{u}_q^f]$ as

$$\hat{\mathbf{y}}_k^f = \hat{\mathbf{\Gamma}}_n\mathbf{x}_k + \hat{\mathbf{H}}_n^d\mathbf{u}_k^f\,. \tag{11}$$

Using the estimated system parameters we can formulate a control policy that attempts to minimise both the output error and the inputs across the trajectory. This introduces the concept of MPC with an estimate linear system. For the linear case the cost can be defined as a linear quadratic cost, with respect to this system as

$$J = (\hat{\mathbf{y}}_k^f - \mathbf{y}_{des}^f)^T\mathbf{W}_y(\hat{\mathbf{y}}_k^f - \mathbf{y}_{des}^f) + \mathbf{u}_k^f\mathbf{W}_u\mathbf{u}_k^f \tag{12}$$

where $\mathbf{y}_{des}^f$ is the sequence of future desired outputs and $\mathbf{W}_y \in \mathbb{R}^{(l+n)\times(l+n)}$, $\mathbf{W}_u \in \mathbb{R}^{(q+n)\times(q+n)}$ are matrices of the weights defined in equation (5). Substituting equation (11) into the cost function and removing terms that are independent of the input, as in later steps the cost function is minimised with respect to the input. The optimization problem is then simplified to

$$\arg\min_{\mathbf{u}_k^f} J = \frac{1}{2}\mathbf{u}_k^{f\,T}\mathbf{\Psi}\mathbf{u}_k^f + \mathbf{u}_k^{f\,T}\mathbf{\Omega} \tag{13}$$

where,

$$\mathbf{\Psi} = \hat{\mathbf{H}}_n^{d\,T}\mathbf{W}_y\hat{\mathbf{H}}_n^d + \mathbf{W}_u \ \text{ and } \ \mathbf{\Omega} = \hat{\mathbf{H}}_n^{d\,T}\mathbf{W}_y(\hat{\mathbf{\Gamma}}_n\mathbf{x}_k - \mathbf{y}_{des}^f)\,.$$

In this case the cost function is represented in quadratic form. A QP method can now be used to solve the optimization problem, where constraints can be included, such as upper and lower bounds of the inputs and outputs. However in this work only the unconstrained solution is evaluated under the assumption that the desired trajectory implemented does not require spuriously large inputs to be constrained. In this case the unconstrained solution can be computed using the following approach. Taking the derivative of the cost function and setting it to zero gives

$$\frac{\partial J}{\partial \mathbf{u}_k^f} = \mathbf{\Psi}\mathbf{u}_k^f + \mathbf{\Omega} = 0. \tag{14}$$

Therefore the optimal set of future inputs is found to be

$$\mathbf{u}_k^f = -\mathbf{\Psi}^{-1}\mathbf{\Omega}, \tag{15}$$

and substituting for system parameters becomes

$$\mathbf{u}_k^f = (\hat{\mathbf{H}}_n^{d\,T}\mathbf{W}_y\hat{\mathbf{H}}_n^d + \mathbf{W}_u)^{-1}\hat{\mathbf{H}}_n^{d\,T}\mathbf{W}_y(\mathbf{y}_{des}^f - \hat{\mathbf{\Gamma}}_n\mathbf{x}_k). \tag{16}$$

This is the unconstrained linear feedback control law for the discrete state space system. It can be seen that the feedback law is a combination of the pseudoinverse of $\hat{\mathbf{H}}_n^d$ weighted by $\mathbf{W}_y$ with the regularisation term $\mathbf{W}_u$. Furthermore, the $(\mathbf{y}_{des}^f - \hat{\mathbf{\Gamma}}_n\mathbf{x}_k)$ term is an estimate of the error between the desired future output and a prediction of the future outputs based on the current state. This result effectively produces a state feedback controller using an estimate of the linear model parameters.

5.2 Learning Non-linear Locally Weighted Model Predictive Control

Locally weighted learning approximates a nonlinear model by separating the problem into locally linear regions using a weighted kernel function. The idea of LWL-MPC is to utilise this concept by simultaneously learning locally linear models and compute the MPC within the linear region. This method is based on the assumption that each region defined by the learning algorithm is approximately locally linear. The Receptive Field Weighting Regression method is chosen as the locally weighted learning algorithm for this work. The approach is formulated as follows. Given a nonlinear discrete state space equation in the form of

$$\begin{aligned}\mathbf{x}_{k+1} &= f(\mathbf{x}_k, \mathbf{u}_k) \\ \mathbf{y}_k &= g(\mathbf{x}_k, \mathbf{u}_k).\end{aligned} \tag{17}$$

Then RFWR can be used to approximate a model of this system in the form of

$$\hat{\mathbf{y}}_k^f = \frac{1}{\mathbf{W}}\sum_{i=1}^{I} w_i(\mathbf{x}_k, \mathbf{u}_k)\Pi_i(\mathbf{x}_k, \mathbf{u}_k^f) \tag{18}$$

where,

$$\Pi_i(\mathbf{x}_k, \mathbf{u}_k^f) = [\hat{\mathbf{\Gamma}}_n^i\mathbf{x}_k + \mathbf{H}_n^i\mathbf{u}_k^f]$$

which represents a linear model of the input output system defined in the previous section. It is noted that in the normal RFWR algorithm the input $\mathbf{z}$ given to the kernel function is also the same given to the linear regression. However, in this approach the RFWR is modified to pass the future inputs $\mathbf{u}_k^f$ into the linear model whereas only the current input $\mathbf{u}_k$ is given to the kernel function defining the local region of the model. Under the assumption that the system is locally linear the feedback control law can be computed from equation (16) to estimate a sequence of local control inputs

$$\hat{\mathbf{u}}_{k,i}^f = -\mathbf{\Psi}_i^{-1}\mathbf{\Omega}_i.$$

The estimated next input $\hat{\mathbf{u}}_{k+1}^i$ is then taken out of this sequence and combined as a weighted average across all models producing an estimate of the next control input as

$$\hat{\mathbf{u}}_{k+1} = \frac{1}{\mathbf{W}}\sum_{i=1}^{I} w_i(\mathbf{x}_k, \mathbf{u}_k)\hat{\mathbf{u}}_{k+1}^i. \tag{19}$$

The final LWL-MPC system is defined in the following diagram. Using the RFWR algorithm the models can be learnt online by introducing a delay term to build up a history of inputs, states and outputs from *k-n* to *k*. Every control step each local controller estimates a new input and is then averaged based on how close the local controller is to the current state and input of the system.

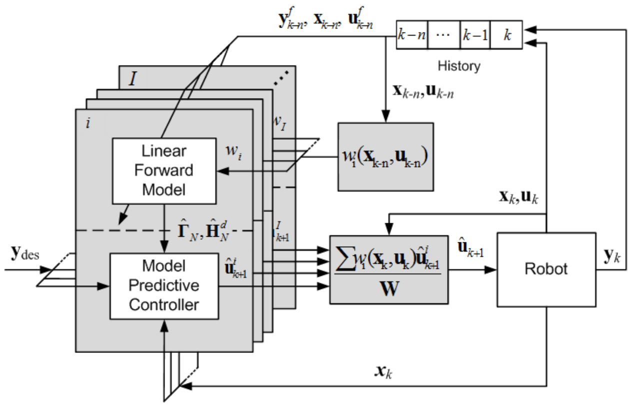

$\mathbf{y}^f_{k-n}, \mathbf{x}_{k-n}, \mathbf{u}^f_{k-n}$
$k-n$ ⋯ $k-1$ k
History
$\mathbf{x}_{k-n}, \mathbf{u}_{k-n}$
I
i
Linear Forward Model
w_i
v_I
$w_i(\mathbf{x}_{k-n}, \mathbf{u}_{k-n})$
$\mathbf{x}_k, \mathbf{u}_k$
$\hat{\Gamma}_N, \hat{\mathbf{H}}^d_N$
$\mathbf{y}_{des}$
Model Predictive Controller
$\hat{\mathbf{u}}^i_{k+1}$
$\sum w_i(\mathbf{x}_k, \mathbf{u}_k)\hat{\mathbf{u}}^i_{k+1}$
$\mathbf{W}$
$\hat{\mathbf{u}}_{k+1}$
Robot
$\mathbf{y}_k$
$\mathbf{x}_k$

Figure 4 - Illustration of our LWL-MPC control scheme. Each model is represented by a page that incorporates a linear forward model and a shared local controller. A history of past inputs, outputs and states are given to update each linear model. The current state is also given to each controller to estimate local inputs. The current input and state then defines the weighted average of each local controller to produce an estimate of the next input applied to the robot.

6 Experiments

The performance of LWL-MPC was investigated by applying the system to a 3DoF EJR arm shown in Figure 5. The challenge of this experiment was to learn joint position control of the arm undergoing significant non-linear effects using motor voltages as inputs and state information with no prior knowledge of the system model.

The EJR arm is setup in a planar arm configuration and consists of three Series Elastic Actuators (SEA) for joints. Each joint is composed of a geared DC motor with encoder feedback and a spring element with an attached magnetic encoder that measures the spring deflection. The measurement of spring deflection provides both a measurement of the torque acting on the joint and is combined with the motor encoders to estimate the joint position after the spring.

The robot arm is controlled from a PC via a National Instruments Multifunction Data Acquisition (NI-DAQ) card. The NI-DAQ card generates PWM signals to low level analog motor drivers providing control over motor voltages as inputs; timers to perform quadrature decoding on all three encoder signals; and includes an ADC port to read in the analog signal from the magnetic encoders for spring deflection measurements. The NI-DAQ card is controlled through MATLAB and compiled using the Real Time Workshop to produce a real time system.

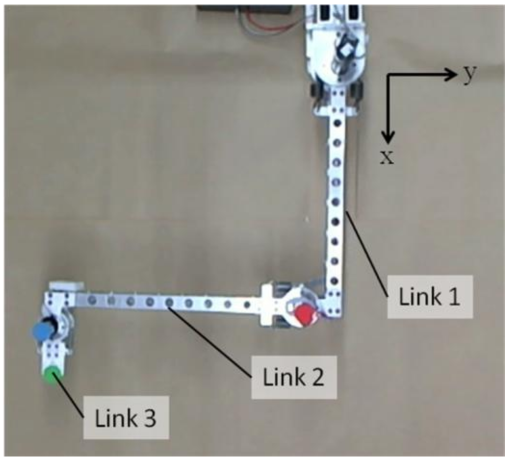
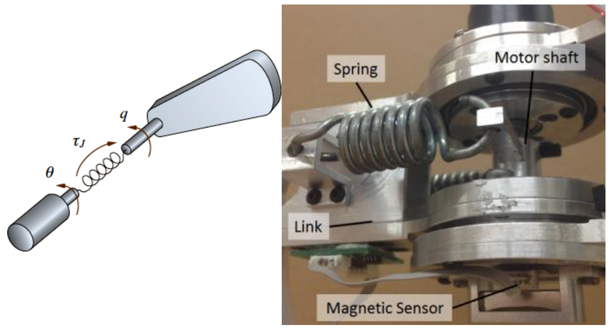


Figure 5 - Top down view of 3DoF elastic joint robot arm. The arm consists of three brushed DC motors, three spring elements and three links connected in series forming a planar arm configuration.



(a) Concept Model of an SEA (b) SEA joint on robot arm

Figure 6 - Model and image of series elastic joint. (a) Concept model shows the motor angle θ is connected in series to the joint angle q through the spring element. (b) Shows the implementation of the SEA on the real robot arm, highlighting all the SEA elements.

6.1 Experimental Setup and Procedure

The experiment can be separated into two main steps, firstly the training phase which allows the learning algorithm to generate an initial model of the system. The second step is the control phase where the system is given a desired trajectory and simultaneously performs online learning while attempting to track the given trajectory.

For both parts of the experiment the LWL-MPC system is setup with the state defined as

$$\mathbf{x} := [\dot{\boldsymbol{\theta}} \quad \boldsymbol{\tau}_j \quad \mathbf{q} \quad \dot{\mathbf{q}}],$$

which is a vector of motor velocities, joint torques, joint positions and joint velocities respectively for each degree of freedom. The inputs to the system were given as the motor voltages

$$\mathbf{u} = [v_1 \quad v_2 \quad v_3],$$

for each joint. Lastly the output of the system was defined as the joint positions

$$\mathbf{y} := [q_1 \quad q_2 \quad q_3].$$

Other parameters of the LWL-MPC system include the size of the prediction horizon *n*, the step size *T* defining the control rate, and the initialisation parameters for the RFWR algorithm. The prediction horizon was chosen as a balance between having a large enough look ahead window and small enough to decrease the computational time in computing the control law every control cycle. For this system a window size of 25 was experimentally selected. For this work the LWL-MPC controller was split up into three MISO sub systems for each joint, but still incorporating all the states of the full system. This allowed this system to be computed independently across three separate threads improving computational cost of the system. This allowed the system to achieve a current control rate of 100Hz.

The learning rates, kernel parameters, forgetting factor and initial size for the local models of the RFWR were experimentally chosen which produced stable results while performing online updates of the model.

A desired trajectory of a figure of eight was chosen to evaluate the performance of the controller. A figure of eight was chosen as this smooth trajectory requires a range of dynamic movements which moves the state of the system through a generous slice of the state space. The joint trajectories where generated using the inverse kinematics of the real 3DoF planar arm, using actual measurements of the link dimensions. The inverse kinematics was computed using the robotics toolbox [Corke, 1996] in MATLAB.

In order to keep the states of the system continuously smooth the figure of eight trajectory is concatenated with a smooth start up trajectory that joins the robot arm at the initial rest position (shown in Figure 5) to a point along the figure of eight which has zero velocity. This step is taken so no initial jerk in accelerations is experienced during start up. The figure of eight was selected to take 7.5 seconds to execute.

Training Phase

A PI controller was chosen to produce the initial training data on the real robot arm. In other learning control methods the training phase typically involves active motor babbling [Saegusa, *et al.*, 2009] and can also be applied to this system. However, using a simple PI controller instead of motor babbling was selected for two reasons; firstly it produces initial training data, where the state of the system should be close to the solution of the desired trajectory; and secondly, it produces a base line for comparing the final performance of the LWL-MPC system. This allows the model learning algorithm to make a good initial approximation of the model close to the space of the final solution. The PI controller was given the same figure of eight trajectory that is given during the control phase.

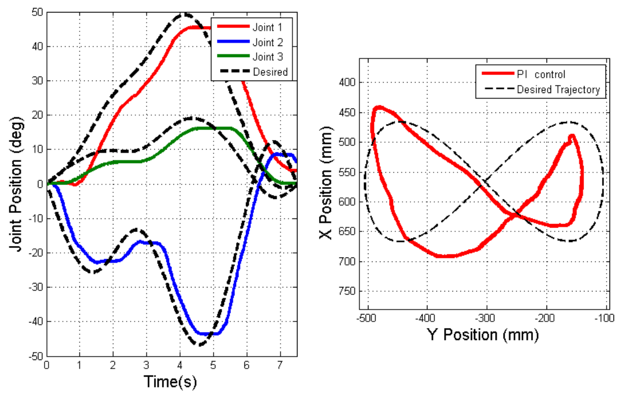
Control Phase

Once the offline training is complete the initial model is used within the previously outlined LWL-MPC system and applied to the real robot arm. The controller is then given the desired figure of eight trajectory. The controller performs both an online update of the forward model from a history of the inputs and outputs, and then uses the updated model with the MPC to produce a prediction of the next input required for the current desired trajectory. After applying the predicted input the update and prediction steps are continuously repeated every sample time over the whole trajectory. In order to demonstrate the online learning capabilities of this system 20 iterations of the figure of eight are executed. After the first 20 iterations are complete a second figure of eight is given to the controller that has been rotated by 90 degrees and was not given in the training phase. This demonstrates the capabilities of the learning system when the robot arm explores new areas of the learning space.

7 Results

7.1 Training Results

The training phase was executed for a total of 4 iterations of the figure of eight, taking 30 seconds to complete. The training data generated a total of 3000 sample points. Figure 7 shows a plot of the joint positions and resulting end effector position for the last iteration of the figure of eight for the PI controller. The performance of this controller was tuned to achieve the minimum joint position error without creating an unstable closed loop system. Figure 7a indicates that there is a significant phase delay and error between the actual and desired trajectory.



(a) Joint position vs desired (b) End effector position vs desired

Figure 7 - Plot of the arm and the resulting end effector trajectory versus the desired figure of eight.

It was found that if the gains of the PI controller were increased to improve this phase lag the system started to become unstable. This is a result of the elastic properties of each joint of the system having complex poles (defining the resonance of the elastic joint) which are close to the real axis. Increasing the gain to reduce the steady state error moves the closed loop poles over to the negative real axis. Therefore, in order to achieve better performance further system analysis to identify the open loop complex poles for each joint would be required. After sufficient system identification is complete an advanced controller would then be required to ensure the complex poles do not move onto the negative real axis.

This generally is a complicated and time consuming task.

The states of the system during the training phase are shown in Figure 8, illustrating the range of the state space that was explored. In particular it can be seen that some nonlinear friction and dead-zones appear in the motor and joint velocity states around zero.

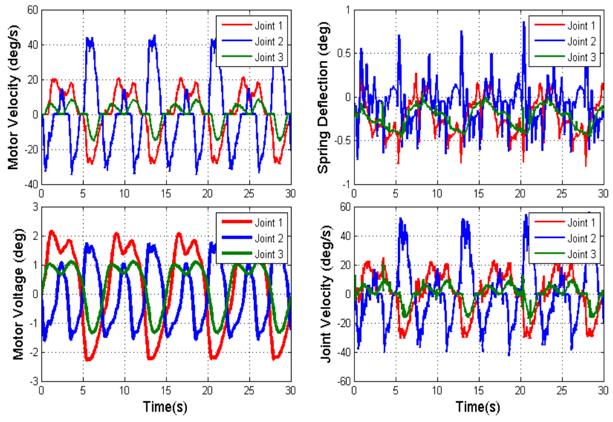


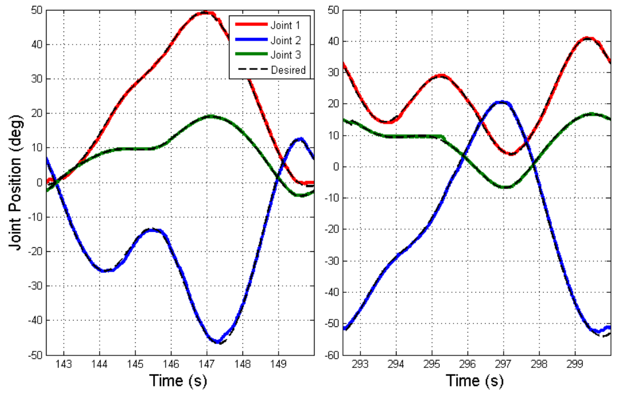
Figure 8 - Plot of system states during the training phase, illustrating the coverage of the state space with the training data.

Applying the learning algorithm offline using training data produced a total of 10 local models for each joint for the initial learning parameters selected. During this offline training the learning rates were adjusted in order to provide stable gradient descent updates of the local models without any over estimation of the gradient causing unstable updates.

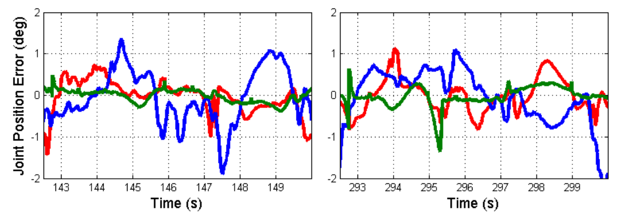
7.2 Control Results

After sufficient offline training, LWL-MPC was applied to the real robot arm. The controller was executed for a total of 5 minutes which included 20 iterations of the first figure of eight and a subsequent 20 iterations of the second figure of eight. The performance of the controller was simultaneously recorded during the experiment, including overhead video footage at 20 frames per second to produce a reasonable ground truth of the end effector. The LWL-MPC successfully learned to control the joint position of the robot arm and incrementally improved its performance every iteration of the trajectory. Figure 9 illustrates the performance of the controller during the last iterations of both types of figure of eight's. This figure shows that the controller produces a significant improvement when compared to the PI controller shown in Figure 7.

Figure 9b further shows that not only can LWL-MPC learn to control one type of trajectory it can perform online exploration of new areas in its learning space when attempting to track a different trajectory. This shows the ability of RFWR to generalise and apply the learnt dynamics to new trajectories.



(a) Initial figure of eight (b) Rotated figure of eight



(c) Error for Initial figure of eight (d) Error for Rotated figure of eight

Figure 9 - Plot of control performance for the last iteration of the control phase for both types of figure of eight's after 20 iterations of each trajectory.

By using the forward kinematics of the robot arm and the measured joint positions the resulting end effector trajectory is constructed in x and y coordinates. Figure 11 shows the resulting trajectory of the end effector for the last iterations of both types of figure of eight's. The results show LWL-MPC learns to track the desired trajectory with a mean distance error of 50mm ± 17mm for the first figure of eight. Similar results are also produced for the second figure of eight with a mean distance error of 50mm ± 28mm. For this result a total of 171 models were added to approximate the system across both figure of eight trajectories.

It can be seen that even after 20 iterations there are still some points along the trajectory that prove to be difficult for the controller to improve. These points are found to lie on parts of the trajectory where the joints exhibit changes in direction, explicitly points where the velocity of the joints are zero. This may be related to the underlying performance of the RFWR to identify and learn the nonlinearities of the forward model.

The learning curves of both figure of eight's for the experiment is shown in Figure 10. The learning curves show the mean and standard deviation of Euclidean distance between the end effector position and the desired position for each iteration of the controlled trajectories.

The learning curves show that LWL-MPC incrementally improves the performance of the system every iteration. After approximately 20 iterations the curves indicate that the performance plateaus to an approximate steady state for both figure of eight trajectories.

A video recording of the experiment was also used as an estimate of ground truth to further validate the results. The ground truth was used only for validation of the results and was not used as an input to the learning algorithm. A trace of the end effector from the video frames was generated by post-processing the image and performing colour tracking of the coloured dots that were placed on the robot arm.

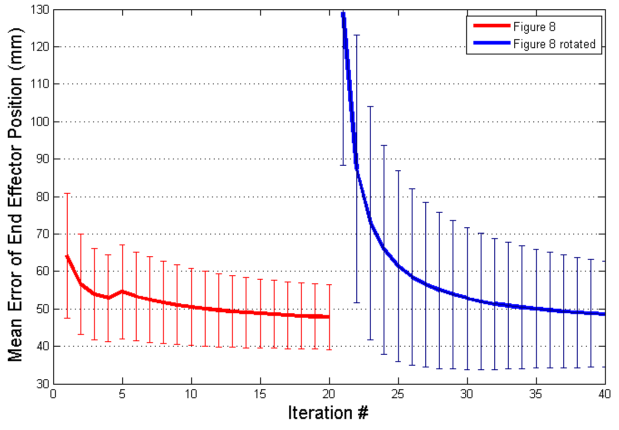


Figure 10 – Mean error per iteration of figure of eight, demonstrating the learning performance of the system. Error bars show one standard deviation above and below the mean.

Figure 12 shows a sequence of images for specific iterations of the experiment. The images validate that the traces of the end effector indeed follow a figure of eight pattern. The figure of eight pattern in the image generated for 20 iterations also correlates with the trajectory shown in Figure 11.

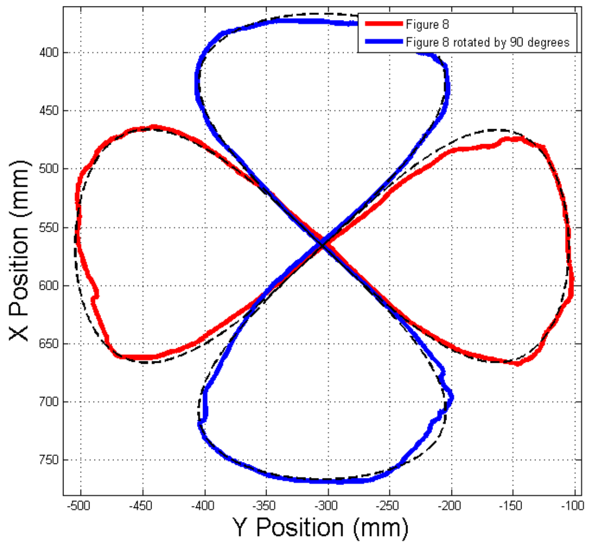


Figure 11 - Plot of end effector trajectory of last two figure of eight's after 20 iterations of each trajectory

Furthermore, the images show the incremental improvement of the figure of eight trajectories after a set amount of iterations.

8 Discussion and Future Work

8.1 Discussion

The results show a marked improvement in control performance for the EJR when LWL-MPC is applied compared to the basic PI controller. Small areas of error

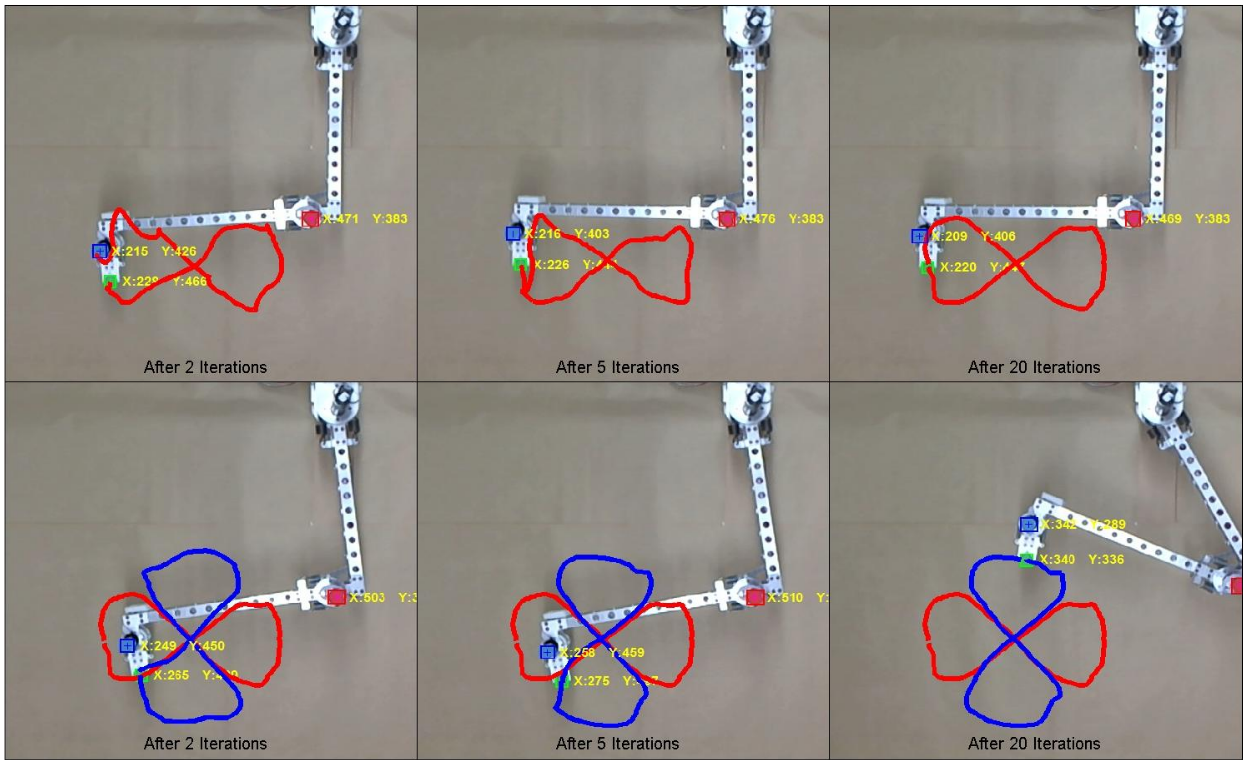


Figure 12 - Sequence of images with end effector traces overlayed. The images show the performance of the controller as the number of iterations increases. The red line indicates the initial figure of eight for each iteration and the blue line indicates the rotated figure of eight after each iteration.

were observed causing distortions in the final tool point trajectory. It was noticed that these areas correspond to zero velocity points along the trajectory. In this case the highly nonlinear stiction and dead-zones of the real system could be the cause of the larger errors in this area. Further investigation is required to look at ways the learning system can be improved to better account for these areas. One possible way is to investigate some of the trade-offs between learning rates and penalty terms of the RFWR which limit the minimum size of the locally linear regions.

The LWL-MPC system is widely applicable in robotic applications. In this experiment the system rapidly learned the raw motor voltages required to control a compliant multi-degree of freedom arm. The only caveat to its deployment in other robotic applications is the initial selection of hyper parameters in the learning algorithm, and MPC parameters, such as cost function weights and horizon window size.

8.2 Future Work

LWL-MPC has the ability to extend its use to more complicated control tasks. One extension that will be investigated is the ability to build local models of a system which incorporate task or context dependent variables where underlying changes in system dynamics occurs. The problem could be as simple as controlling an EJR while manipulating objects of different masses, causing a change in system dynamics. This could simply be implemented by introducing the context variable within the RFWR kernel function allowing the system to distribute models along the context dimension. Further work is also required to analyse the stability and robustness of the controller, including developing methods to ensure convergence of the learning system while performing stable control. Work has been done in this area and could be used to extend LWL-MPC to ensure stability and robustness [Aswani, *et al.*, 2011]. Other work we wish to perform is to investigate the selection of control parameters such as the weights defined in the cost function and selection of the horizon window to produce the optimal performance.

References

[Aswani, et al., 2011] A. Aswani, H. Gonzalez, S.S. Sastry and C. Tomlin. Provably safe and robust learning-based model predictive control. Arxiv preprint arXiv:1107.2487, 2011

[Aswani, et al., 2012] A. Aswani, P. Bouffard and C. Tomlin. Extensions of Learning-Based Model Predictive Control for Real-Time Application to a Quadrotor Helicopter. Proc. American Control Conference (ACC), pages (Montreal, Canada), 2012

[Atkeson, et al., 1997] CG Atkeson, AW Moore and S Schaal. Locally weighted learning. Artificial intelligence review, 11(1):11-73, 1997

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

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

[De Santis, et al., 2008] A. De Santis, B. Siciliano, A. De Luca and A. Bicchi. An atlas of physical human–robot interaction. Mechanism and Machine Theory, 43(3):253-270, 2008

[Florez, et al., 2011] J. M. Florez, D. Bellot and G. Morel. LWPR-model based predictive force control for serial comanipulation in beating heart surgery. Advanced Intelligent Mechatronics (AIM), 2011 IEEE/ASME International Conference on, pages 320-326, 2011

[García, et al., 1989] Carlos E. García, David M. Prett and Manfred Morari. Model predictive control: Theory and practice—A survey. Automatica, 25(3):335-348, 1989

[Li, et al., 2001] M. Li, Y. Yang, F. Gao and F. Wang. Fuzzy multi-model based adaptive predictive control and its application to thermoplastic injection molding. The Canadian Journal of Chemical Engineering, 79(2):263-272, 2001

[Li and Todorov, 2007] W. Li and E. Todorov. Iterative linearization methods for approximately optimal control and estimation of non-linear stochastic system. International Journal of Control, 80(9):1439-1453, 2007

[Mitrovic, et al., 2011] D. Mitrovic, S. Klanke and S. Vijayakumar. Learning impedance control of antagonistic systems based on stochastic optimization principles. The International Journal of Robotics Research, 30(5):556-573, 2011

[Morari and H. Lee, 1999] Manfred Morari and Jay H. Lee. Model predictive control: past, present and future. Computers & Chemical Engineering, 23(4–5):667-682, 1999

[Nguyen-Tuong and Peters, 2011] D. Nguyen-Tuong and J. Peters. Model learning for robot control: a survey. Cognitive Processing, 1-22, 2011

[Saegusa, et al., 2009] R. Saegusa, G. Metta, G. Sandini and S. Sakka. Active motor babbling for sensorimotor learning. Robotics and Biomimetics, 2008. ROBIO 2008. IEEE International Conference on, pages 794-799, 2009

[Schaal and Atkeson, 1998] S. Schaal and C.G. Atkeson. Constructive incremental learning from only local information. Neural Computation, 10(8):2047-2084, 1998

[Sigaud, et al., 2011] Olivier Sigaud, Camille Salaün and Vincent Padois. On-line regression algorithms for learning mechanical models of robots: A survey. Robotics and Autonomous Systems, 2011

[Vijayakumar and Schaal, 2000] S. Vijayakumar and S. Schaal. Locally weighted projection regression: An O (n) algorithm for incremental real time learning in high dimensional space. American Control Conference, pages 288–293, 2000

[Wan and Bogdanov, 2001] E.A. Wan and A.A. Bogdanov. Model predictive neural control with applications to a 6 DOF helicopter model. pages 488-493 vol. 481, 2001