Learning Inverse Dynamics for Robot Manipulator Control

by

Joseph Sun de la Cruz

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2011

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

Model-based control strategies for robot manipulators can present numerous performance advantages when an accurate model of the system dynamics is available. In practice, obtaining such a model is a challenging task which involves modeling such physical processes as friction, which may not be well understood and difficult to model. Furthermore, uncertainties in the physical parameters of a system may be introduced from significant discrepancies between the manufacturer data and the actual system. Traditionally, adaptive and robust control strategies have been developed to deal with parametric uncertainty in the dynamic model, but often require knowledge of the structure of the dynamics. Recent approaches to model-based manipulator control involve data-driven learning of the inverse dynamics relationship, eliminating the need for any a-priori knowledge of the system model. Locally Weighted Projection Regression (LWPR) has been proposed for learning the inverse dynamics function of a manipulator. Due to its use of simple local, linear models, LWPR is suitable for online and incremental learning. Although global regression techniques such as Gaussian Process Regression (GPR) have been shown to outperform LWPR in terms of accuracy, due to its heavy computational requirements, GPR has been applied mainly to offline learning of inverse dynamics. More recent efforts in making GPR computationally tractable for real-time control have resulted in several approximations which operate on a select subset, or sparse representation of the entire training data set.

Despite the significant advancements that have been made in the area of learning control, there has not been much work in recent years to evaluate these newer regression techniques against traditional model-based control strategies such as adaptive control. Hence, the first portion of this thesis provides a comparison between a fixed model-based control strategy, an adaptive controller and the LWPR-based learning controller. Simulations are carried out in order to evaluate the position and orientation tracking performance of each controller under varied end effector loading, velocities and inaccuracies in the known dynamic parameters. Both the adaptive controller and LWPR controller are shown to have comparable performance in the presence of parametric uncertainty. However, it is shown that the learning controller is unable to generalize well outside of the regions in which it has been trained. Hence, achieving good performance requires significant amounts of training in the anticipated region of operation.

In addition to poor generalization performance, most learning controllers commence learning entirely from 'scratch,' making no use of any a-priori knowledge which may be available from the well-known rigid body dynamics (RBD) formulation. The second portion of this thesis develops two techniques for online, incremental learning algorithms which incorporate prior knowledge to improve generalization performance. First, prior knowledge

is incorporated into the LWPR framework by initializing the local linear models with a first order approximation of the prior information. Second, prior knowledge is incorporated into the mean function of Sparse Online Gaussian Processes (SOGP) and Sparse Pseudo-input Gaussian Processes (SPGP), and a modified version of the algorithm is proposed to allow for online, incremental updates. It is shown that the proposed approaches allow the system to operate well even without any initial training data, and further performance improvement can be achieved with additional online training. Furthermore, it is also shown that even partial knowledge of the system dynamics, for example, only the gravity loading vector, can be used effectively to initialize the learning.

Acknowledgements

I would like to thank my co-supervisors Dr. Dana Kulić and Dr. William Owen for providing their support, guidance, and expertise throughout the duration of this research. I am truly grateful for all the helpful meetings, presentation rehearsals, and paper-editing sessions. I am also indebted to them for funding my travels abroad where I was able to gain a great deal of exposure to the international robotics and controls community.

I would also like to thank Dr. Elizabeth Croft at the University of British Columbia CARIS lab for graciously providing the equipment that I needed for the experiments. Special thanks also go to Ergun Calisgan for assisting me in setting up, debugging and running numerous experiments.

Finally, I would like to thank my family and friends for their unwavering support and encouragement. This thesis would not have been possible without you all.

Table of Contents

List of Tables

List of Figures

Nomenclature

a_i	Link length
a_L	is the learning rate for gradient descent in LWPR
B_{eff}	Effective damping at joint
B_m	Motor damping constant
$\mathbf{C}^{GP}$	Covariance function parameter for GP
$\mathbf{C_f}$	Coulomb friction constant matrix
$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$	Coriolis and centripetal force vector
$\hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$	Estimated Coriolis and centripetal force vector
d_i	Link offset
$\mathbf{D}$	Distance parameter for LWPR
$\mathbf{e}_{N+1}$	Unit vector of length $N+1$
$\mathbf{e}$	Joint-space tracking error
$\mathbf{E}$	Inertia matrix estimation error
$\mathcal{E}$	Vector of servo error and derivative of error
$\bar{f}_*(\mathbf{x}^*)$	GP posterior mean evaluated at input $\mathbf{x}^*$
$\bar{\mathbf{f}}_N$	GP posterior mean function after observing N inputs
$\mathbf{f}_i$	Force exerted on link i by link $i-1$
$\mathbf{F}_i$	Net force at the COM of link i
$\mathcal{F}$	Forward Kinematics function
$\mathcal{F}^{-1}$	Inverse Kinematics relation
$\mathbf{g}$	Gravity vector
g_r	Gear reduction ratio
$\mathbf{G}(\mathbf{q})$	Gravity loading vector
$\hat{\mathbf{G}}(\mathbf{q})$	Estimated gravity loading vector
$\mathbf{H}(\mathbf{q}, \dot{\mathbf{q}})$	Coriolis and centripetal and gravity matrix
$\hat{\mathbf{H}}(\mathbf{q}, \dot{\mathbf{q}})$	Estimated Coriolis and centripetal and gravity matrix
$\mathbf{I}_i$	moment of inertia of link i about its COM
$\mathbf{I}_i^0$	moment of inertia of link i about the base frame
$\mathbf{I}_r$	$r \times r$ identity matrix
$\mathbf{I}_n$	$n \times n$ identity matrix
$\mathbf{J}(\mathbf{q})$	Manipulator Jacobian
J_{cost}	Cost function for LWPR optimization
J_{eff}	Effective inertia at joint
J_m	Rotational inertia of servo motor
$k(\mathbf{x}, \mathbf{x}')$	GP covariance function
k_b	Armature voltage constant

$\mathbf{k_d}$	Independent joint control derivative gain constant
$\mathbf{k_p}$	Independent joint control proportional gain constant
k_m	Motor torque constant
$k_* = k(\mathbf{X}, \mathbf{x}^*)$	Covariance vector for single input $\mathbf{x}^*$
K	is the number of linear models for LWPR
$K(\mathbf{X}, \mathbf{X})$	Covariance matrix
$K_0(\mathbf{x}, \mathbf{x}')$	Prior covariance function evaluated at $\mathbf{x}$ and $\mathbf{x}'$
$\hat{K}_0(\mathbf{x}, \mathbf{x}_{N+1})$	Projected covariance function
$\mathbf{K_d}$	Model-based control derivative gain constant
$\mathbf{K}_N$	Covariance matrix
$\mathbf{K_{NM}}$	Covariance matrix between data points and pseudo-inputs
$\mathbf{K_M}$	Pseudo-inputs covariance matrix
$\mathbf{K}_N^{SPGP}$	SPGP covariance matrix
$\mathbf{K_p}$	Model-based control proportional gain constant
$\mathcal{K}$	Kinetic energy
$\mathcal{L}$	Lagrangian
$\mathbf{m}$	Gradient descent parameter for LWPR
m_i	Mass of link i
m_{ii}	ii^{th} term of the inertia matrix, $\mathbf{M}$
M_s	Number of pseudo-inputs for SPGP
$\mathbf{M}(\mathbf{q})$	Inertia matrix
$\hat{\mathbf{M}}(\mathbf{q})$	Estimated inertia matrix
n	Number of Degrees of Freedom (DOF)
$\mathbf{n}_i$	Moment exerted on link i by link $i-1$
N	Training data set size
$\mathbf{N}_i$	Net moment at the COM of link i
$\mathbf{p}_i^*$	Vector from the origin of frame $i-1$ to frame i with respect to frame i
$\mathcal{P}$	Adaptive Controller Lyapunov function constant
$\mathbf{q}$	Vector of joint angles
$\mathbf{q_d}$	Vector of desired joint angles
q_i	Joint angle
q_{N+1}	SOGP update scalar
$\mathbf{Q}$	SPGP mean parameter
$Q(s)$	Joint angles in frequency domain
$Q_d(s)$	Desired joint angles in frequency domain
r_a	Armature resistance
$\mathbf{r_{c_i}}$	Centre of mass vector for link i
r_{N+1}	SOGP update scalar

$\mathbf{R}_i$	Rotation matrix defining frame i with respect to frame $i-1$
$\mathbf{s}_i$	Position vector of the COM
$\mathbf{T_n}$	Homogeneous Transformation matrix
T_{N+1}	Matrix dimension operator for SOGP
$\mathbf{u}$	Control signal
$\mathbf{u_{FB}}$	Feedback control signal
$\mathbf{u_{FF}}$	Feedforward control signal
U_{N+1}	Vector dimension operator for SOGP
$\mathbf{v}_c^0$	Linear velocity of the object's centre of mass (COM) wrt base frame
$\mathbf{v}_i$	Linear velocity of frame i
$\dot{\mathbf{v}}_i$	Linear acceleration of frame i
$\bar{\mathbf{v}}_i$	Linear velocity of the COM of link i
$\dot{\bar{\mathbf{v}}}_i$	Linear acceleration of the COM of link i
v_m	Motor voltage
$V(\mathbf{x}_*)$	GP posterior covariance
$\mathbf{V_f}$	Viscous friction constant matrix
$\mathcal{V}$	Potential energy
w_{ik}	Weight of the k^{th} local linear model of the i^{th} output dimension
w_{prune}	Threshold for removal of receptive field
w_{gen}	Threshold for addition of receptive field
$\mathbf{x}$	Input vector
x	Cartesian x-coordinate of end-effector
$\bar{\mathbf{X}}$	SPGP Pseudo-inputs
y	Cartesian y-coordinate of end-effector
$\mathbf{y}$	Output vector
z	Cartesian z-coordinate of end-effector
$\mathbf{z}_0$	Unit vector in the z direction
α	Mean function parameter for GPR
α_{PE}	Positive number for ACT convergence proof
α_i	i^{th} link twist parameter
$\boldsymbol{\varepsilon}$	zero-mean random noise term
$\boldsymbol{\eta}$	Dynamic model uncertainty
γ_{N+1}	Novelty measure of the current input

$\boldsymbol{\gamma}$	Adaptive gain matrix
$\Omega(s)$	Closed-loop characteristic polynomial of PD controlled joint
$\boldsymbol{\phi}_d$	Regressor matrix
ρ	Time interval
σ_n^2	Noise variance
$\boldsymbol{\tau}$	Joint torque vector
$\boldsymbol{\tau}_f$	Joint torque vector due to friction
τ_m	Motor torque
$\boldsymbol{\Theta}$	GP covariance hyperparameters
θ_m	Motor angle
θ_1	Hyperparameter of SE covaraiance function
θ_2	Hyperparameter of SE covaraiance function
$\hat{\boldsymbol{\theta}}$	Estimated dynamic parameters
$\boldsymbol{\omega}^0$	Angular velocity vector wrt base frame
$\boldsymbol{\omega}_i$	Angular velocity of link i
$\dot{\boldsymbol{\omega}}_i$	Angular acceleration of link i

ACT	Adaptive Computed Torque
BV	Basis Vector
CT	Computed Torque
CRS	CRS Robotics Corporation
DH	Denavit-Hartenberg
DOF	Degrees of Freedom
FF	Feedforward
FK	Forward Kinematics
GPR	Gaussian Process Regression
IK	Inverse Kinematics
ILC	Iterative Learning Control
LWR	Locally Weighted Regression
LWPR	Locally Weighted Projection Regression
NN	Neural Networks
RBD	Rigid Body Dynamics
RFWR	Receptive Field Weighted Regression
RTB	Robotics Toolbox
SE	Squared Exponential
SOGP	Sparse Online Gaussian Processes
SPGP	Sparse Pseudo-input Gaussian Processes

Chapter 1

Introduction

The use of robotics worldwide is most prevalent in the manufacturing industry where the environment is highly controlled and relatively constant. Since the introduction of the first robotic manipulator for industry use in the 1960s [45], simple decentralized control strategies such as independent joint PD control [45] have become ubiquitous for basic manipulation tasks such as pick-and-place motions. Unlike decentralized controllers, control strategies that are based on the dynamic model of the manipulator, known as model-based controllers, present numerous advantages such as increased performance during high-speed movements, reduced energy consumption, improved tracking accuracy and the possibility of compliance [35]. While effective under highly controlled conditions, these controllers do not easily adapt to sudden or even gradual changes to the dynamics of the system and often require an accurate model of the system dynamics to achieve good performance. Deriving an accurate model analytically is a difficult task, especially due to physical phenomena which are not well understood or difficult to model, such as friction and gear backlash. Furthermore, the lack of accurate dynamic parameters made available from the manufacturer [8] of robotic systems makes it difficult to obtain an accurate dynamic model. Even with the use of dynamic parameter estimation [24], unmodeled dynamics can still reduce the performance of model-based control systems. While adaptive controllers [14], [37], [45] are able to deal with gradual changes in parameter values that may occur from daily wear and tear, they are still unable to account for modeling errors or changes in the model structure.

The increasing complexity of robotic systems and their actuators, such as high degree-of-freedom (DOF) humanoid systems [8], hydraulically actuated manipulators [55], and series-elastic actuators increases the need for more complex forms of control strategies that often require knowledge of the dynamic structure of the system. As an alternative

to modeling the complex behaviour of these systems, machine learning algorithms for supervised learning can be applied to learn the dynamics. Recent developments in this area of learning control have already yielded promising results that enable robots to learn their own inverse dynamics function with no a-priori knowledge of the system [39],[43].

Locally Weighted Projection Regression (LWPR) is frequently applied [39],[43],[55] to learn the inverse dynamics of a manipulator, due to its use of simple local, linear models which allow online and incremental learning. However, due to its highly localized learning, the system must be first be trained in the expected regions of operation. Other forms of supervised learning, or regression techniques, have also been investigated for learning control. Gaussian Process Regression (GPR) has been applied [36] to learn the inverse dynamics function of a manipulator, but due to its heavy computational requirements, GPR has been limited mainly to offline learning. More recent efforts [16],[50] in making GPR computationally tractable for real-time control have resulted in several approximations which operate on a select subset, or sparse representation of the entire training data set.

While much progress has been made in the area of learning control for robot manipulators, there has not been much work in recent years to compare these new techniques with previous approaches such as adaptive control which were specifically developed to deal with parameter uncertainty. While learning control typically requires no a-priori knowledge of the system dynamics, adaptive controllers most often operate with the assumption that the structure of the dynamic model is known. With learning controllers, a common problem is that large amounts of relevant training data must first be provided to the algorithm before accurate results can be obtained. Realizing that this issue can be mitigated through the use of a-priori knowledge of the system, research has been done [32] to incorporate the a-priori known dynamics into the learning framework. However, due to the heavy computational load of the learning algorithm in [32], online and incremental updates to the system cannot be made.

This thesis will present a comparison between a fixed model-based control strategy, an adaptive controller and the widely-used LWPR learning controller. Simulations are carried out in order to evaluate the position and orientation tracking performance of each controller under varied end effector loading, velocities and inaccuracies in the known dynamic parameters. Both the adaptive controller and LWPR controller are shown to have comparable performance in the presence of parametric uncertainty. However, it is shown that the learning controller is unable to generalize well outside of the regions in which it has been trained. Hence, achieving good performance requires significant amounts of training in the anticipated region of operation.

The second portion of this thesis focuses on addressing the issues with learning control

which were encountered in the comparison work. Two techniques are developed for online, incremental learning algorithms which incorporate prior knowledge to improve generalization performance. First, prior knowledge is incorporated into the LWPR framework by initializing the local linear models with a first order approximation of the prior information. Second, prior knowledge is incorporated into the mean function of Sparse Online Gaussian Processes (SOGP) and Sparse Pseudo-input Gaussian Processes (SPGP), and a modified version of the algorithm is proposed to allow for online, incremental updates. It is shown that the proposed approaches allow the system to operate well even without any initial training data, and further performance improvement can be achieved with additional online training. Furthermore, it is also shown that even partial knowledge of the system dynamics, for example, only the gravity loading vector, can be used effectively to initialize the learning.

1.1 Thesis Contributions

1. A thorough comparison of standard model-based control, adaptive control and learning control is presented in this thesis to provide a better understanding of the relative strengths and weaknesses of each control strategy, and to identify areas in which improvements can be made.

2. Two types of online, incremental learning algorithms are developed which make use of the a-priori knowledge of the system to improve the system's initial performance and ability to generalize the learned model to areas in which it has not yet been trained. The two algorithms are validated in simulation and through experiments.

1.2 Thesis Outline

Chapter 2 provides the necessary background information on robot manipulator modeling. First, robot manipulator kinematics are presented, including the mathematical representation of the structure of robot manipulators, as well as the solution to the forward and inverse kinematics problem. Second, the dynamics of robot manipulators are presented through the derivation of the rigid body dynamics (RBD) equation as well as the recursive Newton-Euler algorithm for efficient computation of the dynamics. Third, basic control strategies for robot manipulators are outlined, including independent joint PD control and two variants of model-based control. The problem of model uncertainty is introduced in the context of control.

Chapter 3 reviews the related work in the literature which deals with parametric uncertainty in the control of robot manipulators. First, techniques based on knowledge of the dynamic model are presented, including robust and adaptive control strategies. Second, the newer approach of learning the dynamic model is reviewed. Two classes of supervised learning algorithms are presented - local learning techniques such as Locally Weighted Projection Regression (LWPR), and global techniques such as Gaussian Process Regression (GPR) and Support Vector Regression (SVR). Examples of the application of these learning algorithms to robot control are also presented.

Chapter 4 presents a comparison between standard model-based control, adaptive control and learning control with the LWPR algorithm. Simulations are carried out to evaluate the performance of these controllers under dynamic conditions such as varying trajectory speeds, end-effector loading, and parameter uncertainty in order to understand and identify the scenarios in which each controller is more suitable.

Chapter 5 proposes two types of novel learning controllers that incorporate a-priori knowledge to improve the generalization performance of the learning algorithms. A-priori knowledge in the form of the full RBD equation, or partial knowledge of just the gravity loading vector are used to initialize the LWPR algorithm as well as the Sparse Pseudo-Input Gaussian Process (SPGP) and Sparse Online Gaussian Process (SOGP) algorithms with modifications to allow for online, incremental learning of inverse dynamics. Simulation work is carried out to validate the proposed approaches.

Chapter 6 describes the experimental platform used to validate the proposed approaches in the previous chapter. Experimental and simulation results for the LWPR and SOGP controllers are presented and compared to standard model-based control techniques.

Chapter 7 reviews the results and contributions of the thesis, and also presents concluding remarks regarding the methods developed for online incremental learning of inverse dynamics. A discussion of the possible directions for future work is also presented.

Chapter 2

Background

This chapter overviews the modeling framework and basic algorithms for robot manipulator control. First, the mathematical modeling convention and kinematic structure of the manipulator are defined, and then the problem of forward and inverse kinematics is presented. Next, the dynamic model of a robot manipulator is presented, followed by a section on the basics of motion control.

2.1 Kinematics

Although robot manipulators are available in many configurations, this work assumes the use of serial-linked manipulators which are composed of a set of bodies, or links, connected by joints to form an open chain. This configuration is illustrated in Figure 2.1. It is also assumed that each joint enables movement along or about a single axis, contributing a single degree-of-freedom (DOF) to the entire manipulator. This is not a limiting assumption since multi-DOF joints can be represented by a series of single-DOF joints connected by links of zero length. For simplicity and clarity of notation, all results are given for revolute joints, but the work presented can be extended to manipulators with prismatic joints. A robot with n joints is referred to as an n DOF robot, and possesses $n+1$ links, with the 0^{th} link being the base, and the n^{th} link containing the end-of-arm tooling, or end-effector.

In order to describe the location of each link, coordinate frames are attached to each link, with a stationary base (or world) frame located along the z-axis of the first joint according to the Denavit-Hartenberg (DH) convention [17]. The DH convention stipulates that each joint i is actuated about the z_{i-1} axis, and allows the complete kinematic structure

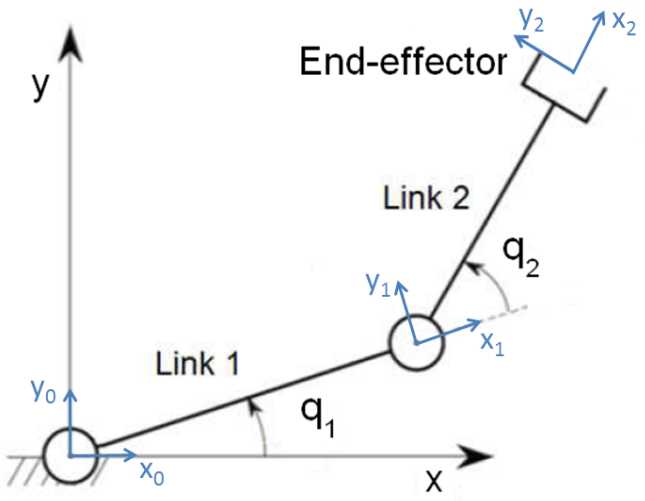

Figure 2.1: Serial-link Robot Manipulator

of a robot manipulator to be described by a set of link and joint parameters as described in Table 2.1.

link length	a_i	distance between z_{i-1} and z_i axes along the x_i axis
link twist	α_i	angle from z_{i-1} axis to the z_i axis about the x_i axis
link offset	d_i	distance from the origin of frame $i-1$ to x_i axis along z_{i-1} axis
joint angle	q_i	angle between x_{i-1} and x_i axes about the z_{i-1} axis

Table 2.1: DH Parameters

The 4×4 homogeneous transform matrix $\mathbf{T_i^{i-1}}$ specifies the position and orientation of the i^{th} frame with respect to the previous $(i-1^{th})$ coordinate frame:

$$\mathbf{T}_i^{i-1} = \begin{pmatrix} \mathbf{R}_i^{i-1} & \mathbf{p}_i \\ \mathbf{0} & 1 \end{pmatrix} \tag{2.1}$$

$$\mathbf{R}_i^{i-1} = \begin{pmatrix} \cos\theta_i & -\sin\theta_i \cos\alpha_i & \sin\theta_i \sin\alpha_i \\ \sin\theta_i & \cos\theta_i \cos\alpha_i & -\cos\theta_i \sin\alpha_i \\ 0 & \sin\alpha_i & \cos\alpha_i \end{pmatrix} \tag{2.2}$$

$$\mathbf{p}_i = \begin{pmatrix} a_i \cos\theta_i \\ a_i \sin\theta_i \\ d_i \end{pmatrix} \tag{2.3}$$

where $\mathbf{R_i^{i-1}}$ is the 3×3 rotation matrix defining the orientation of frame i with respect to frame $i-1$, and $\mathbf{p}_i$ is the position vector defining the location of frame i with respect to frame $i-1$. Thus, the homogeneous transform of link i with respect to the base frame is:

$$\mathbf{T}_i^0 = \mathbf{T}_{i-1}^0 \mathbf{T}_i^{i-1} \tag{2.4}$$

Although the rotation matrix, $\mathbf{R}$, contains nine elements, the orientation can also be minimally represented by three Euler angles (θ, ϕ, ψ) which describe the orientation of a reference frame in three successive rotations.

The forward kinematics (FK) of a robot specifies the position and orientation of the end-effector with respect to the base frame, given the set of robot joint positions. It is obtained by applying equation (2.4) along the entire link chain:

$$\begin{aligned} \mathbf{T_n} =& \mathbf{T}_1^0 \mathbf{T}_2^1 \dots \mathbf{T}_n^{n-1} \\ =& \mathcal{F}(\mathbf{q}) \end{aligned} \tag{2.5}$$

where $\mathbf{T_n}$ represents the Cartesian position and orientation of the end-effector with respect to the base frame, $\mathcal{F}$ is the FK function, and $\mathbf{q}$ is the $n \times 1$ vector of generalized coordinates consisting of the n joint angles for an n-DOF manipulator. Hence, the FK function can be seen as a transformation from the joint space coordinates of n joint angles to m task space coordinates of the end-effector. A task space dimension of $m = 6$ corresponds to Cartesian space, where three variables describe the end-effector's position and three variables describe its orientation. Hence, if $n = m$ then the robot possess the exact number of joints required to locate and orient the end-effector in the m-dimensional task space. Certain configurations of the robot may result in the effective loss of one of more degrees of freedom (e.g. when all the joints are in the same plane). These configurations are referred to as singularities and are formally defined in Equation 2.9. Excluding these configurations, manipulators with $n > m$ are referred to as redundant, whereas those with $n < m$ are underactuated.

While the FK relationship deals with positions, the manipulator Jacobian, $\mathbf{J}(\mathbf{q})$, is a differential relationship of the FK function which relates joint angle velocities to the end-effector velocity in Cartesian space:

$$\dot{\mathbf{x}}_m = \mathbf{J}(\mathbf{q}) \dot{\mathbf{q}} \tag{2.6}$$

where $\dot{\mathbf{x}}_m$ is an $m \times 1$ vector of end-effector velocities. For $m = 6$, $\dot{\mathbf{x}}_m$ consists of the 3×1 linear velocity vector $\mathbf{v}_n$ and the 3×1 angular velocity vector $\boldsymbol{\omega}_n$. Hence, $\mathbf{J}(\mathbf{q})$ is a matrix of dimension $m \times n$. Assuming all n DOF are revolute, the Jacobian can be calculated

geometrically by using the DH parameters:

$$\mathbf{J}(\mathbf{q}) = \begin{bmatrix} \cdots & \mathbf{z}_{i-1}^0 \times \mathbf{p}_{n,i-1}^0 & \cdots \\ \cdots & \mathbf{z}_{i-1}^0 & \cdots \end{bmatrix} \tag{2.7}$$

where $\mathbf{z}_{i-1}^0$ is obtained from the DH assignment and expressed with respect to the base frame, and $\mathbf{p}_{n,i-1}^0$ is the position vector from the $i-1^{th}$ frame to the n^{th} frame expressed in the base frame. Each i^{th} column of the Jacobian, J, describes the individual contribution of joint i to the motion of the end-effector.

In order to determine the joint angles which are required to obtain a desired end-effector location, the solution to the inverse kinematics (IK) problem is required:

$$\mathbf{q} = \mathcal{F}^{-1}(\mathbf{T_n}) \tag{2.8}$$

where $\mathcal{F}^{-1}$ is the IK relation. While the FK function uniquely maps $\mathbf{q}$ to a single end-effector position and orientation, this is not the case for IK. For manipulators with $n = m$, there may exist multiple solutions, and for $n > m$, or redundant manipulators, there is an infinite number of solutions. However, for all cases no solution may exist, if for example, the desired location is outside of the workspace of the robot.

For motion control and trajectory planning, IK is used to transform desired end-effector trajectories to a corresponding sequence of joint angles. While exact, closed-form solutions based on analytical geometry can often be derived for a specific manipulator geometry, there is no general solution. Numerical solutions are possible through the use of the differential kinematics. If J is square, i.e., $m = n$, rearranging Equation 2.6 and solving for $\dot{\mathbf{q}}$ gives the differential IK relationship,

$$\dot{\mathbf{q}} = \mathbf{J}^{-1}(\mathbf{q})\dot{\mathbf{x}}_m \tag{2.9}$$

where $\mathbf{J}^{-1}$ is the inverse of the Jacobian matrix. Certain configurations of $\mathbf{q}$ may result in $\mathbf{J}$ being rank deficient, meaning that there is no solution to the above equation. These configurations are referred to as singularities, and often exist near the boundary of the workspace of the manipulator.

To solve the IK problem for joint angles given an end-effector position, Equation 2.9 can be applied as an infinitesimal relationship in an iterative numerical procedure that starts from an initial guess of joint angles $\mathbf{q_{init}}$ and a desired end-effector position and orientation, $\mathbf{R_{ref}}$ and $\mathbf{p}_{ref}$. By applying the FK function (2.8), the end-effector location and orientation corresponding to the initial guess of joint angles can be obtained. The algorithm then applies Equation 2.9 iteratively to improve the initial guess until a certain accuracy tolerance is reached.

A widely-used pseudocode [45] for the numerical solution to IK is employed in this thesis, and is described in Algorithm 1. The required computation time for this numerical IK approach is sensitive to the initial guess of the joint angles and is generally much slower than an exact geometric IK approach. However, as trajectory planning is done offline in this thesis, the computational efficiency of the IK algorithm is not a significant concern.

Algorithm 1 Numerical Inverse Kinematics

1: **given:** $\mathbf{R}_{ref}, \mathbf{p}_{ref}, \mathbf{q} = \mathbf{q}_{init}$
2: $(\mathbf{p}, \mathbf{R}) \leftarrow \mathbf{T_n} = \mathcal{F}(\mathbf{q})$
3: $\Delta\mathbf{T_n} \leftarrow (\mathbf{\Delta p}, \mathbf{\Delta R}) = (\mathbf{p_{ref}} - \mathbf{p}, \mathbf{R^T R_{ref}})$
4: **if** $(\Delta\mathbf{p}, \Delta\mathbf{R}) < desired\ threshold$ **then**
5: **return** $\mathbf{q}$
6: **else**
7: $\delta\mathbf{q} = \mathbf{J}^{-1}(\mathbf{q})\Delta\mathbf{T_n}$
8: $\mathbf{q} = \mathbf{q} + \delta\mathbf{q}$
9: **go to** line 2
10: **end if**

2.2 Dynamics

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 can be obtained through the Lagrangian formulation [45], which states that:

$$\mathcal{L} = \mathcal{K} - \mathcal{V} \tag{2.10}$$

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial \mathcal{L}}{\partial \mathbf{q}} = \boldsymbol{\tau} \tag{2.11}$$

where $\mathcal{L}$ is the Lagrangian, $\mathcal{K}$ is the kinetic energy, $\mathcal{V}$ is the potential energy and $\boldsymbol{\tau}$ is the $n \times 1$ torque vector. Under the assumption that the links are rigid bodies and there are no other energy storage elements in the system, the only source of potential energy is gravity, and hence $\mathcal{V}$ can be expressed as a function of the joint angles alone. Applying this assumption to (2.11) results in:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{K}}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial \mathcal{K}}{\partial \mathbf{q}} + \frac{\partial \mathcal{V}}{\partial \mathbf{q}} = \boldsymbol{\tau} \tag{2.12}$$

The kinetic energy of any rigid object is dependent upon its inertial properties, as well as its linear and angular velocity, as shown in the following equation:

$$\mathcal{K} = \frac{1}{2} m \mathbf{v}_c^{0T} \mathbf{v}_c^0 + \frac{1}{2} \boldsymbol{\omega}^{0T} \mathbf{I}^0 \boldsymbol{\omega}^0 \tag{2.13}$$

where m is the mass of the object, $\mathbf{I}^0$ is the inertia matrix of the object, $\mathbf{v}_c^0$ is the linear velocity of the object's centre of mass (COM), and $\boldsymbol{\omega}^0$ is the object's angular velocity vector. The superscript in these expressions indicates that these quantities are expressed in the inertial frame of reference, i.e. the base frame. However, because the inertia matrix I is typically expressed in terms of the body frame, a similarity transform must be applied:

$$\mathbf{I}^0 = \mathbf{R}\mathbf{I}\mathbf{R}^T \tag{2.14}$$

where $\mathbf{R}$ is the rotation matrix which transforms coordinates from the body frame to the inertial frame. To compute the kinetic energy of a manipulator, it is necessary to derive expressions for the kinetic energy of each link and sum them to obtain the total kinetic energy. By using the Jacobian matrix in Equation 2.7, the linear and angular velocities of any point on a given link may be expressed as a function of the Jacobian and the joint angle velocity, $\dot{\mathbf{q}}$:

$$\mathbf{v}_{ci}^0 = \mathbf{J}_{v_{ci}}(\mathbf{q})\dot{\mathbf{q}} \tag{2.15}$$

$$\boldsymbol{\omega}_i^0 = \mathbf{J}_{\omega_i}(\mathbf{q})\dot{\mathbf{q}} \tag{2.16}$$

where $\mathbf{J}_{v_{ci}}$ and $\mathbf{J}_{\omega_i}$ are the linear and angular components of the Jacobian (2.7) for the COM of the i^{th} link. From equation (2.13), it then follows that the overall kinetic energy of the manipulator is equal to:

$$\mathcal{K} = \frac{1}{2} \dot{\mathbf{q}}^T \sum_{i=1}^{n} \left[m_i \mathbf{J}_{v_{ci}}^T \mathbf{J}_{v_{ci}} + \mathbf{J}_{\omega_i}^T \mathbf{I}_i^0 \mathbf{J}_{\omega_i} \right] \dot{\mathbf{q}} \tag{2.17}$$

where m_i is the mass of the i^{th} link and $\mathbf{I}_i^0$ is the inertia matrix of the i^{th} link in the inertial frame. Expressing the summation in Equation 2.17 in matrix form, the kinetic energy of the manipulator becomes:

$$\mathcal{K} = \frac{1}{2} \dot{\mathbf{q}}^{\mathbf{T}} \mathbf{M}(\mathbf{q}) \dot{\mathbf{q}} \tag{2.18}$$

where $\mathbf{M}(\mathbf{q})$ is the $n \times n$ inertia matrix. Assuming that the only source of potential energy is gravity, the potential energy of a manipulator can be computed as follows:

$$\mathcal{V} = \sum_{i=1}^{n} m_i \mathbf{g}^{\mathbf{T}} \mathbf{r}_{\mathbf{c_i}} \tag{2.19}$$

where $\mathbf{r}_{\mathbf{c_i}}$ is the location of the COM of the i^{th} link, and $\mathbf{g}$ is the gravity vector.

The first two terms on the left hand side of (2.12) represent the inertial forces on the arm. Substituting (2.18) into these terms results in:

$$\frac{d}{dt}\left(\frac{\partial \mathcal{K}}{\partial \dot{\mathbf{q}}}\right) - \frac{\partial \mathcal{K}}{\partial \mathbf{q}} = \mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.20}$$

where $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the $n \times 1$ centripetal and Coriolis torque vector, which is calculated as follows:

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \dot{\mathbf{M}}(\mathbf{q})\dot{\mathbf{q}} - \frac{\mathbf{1}}{\mathbf{2}}\left[\dot{\mathbf{q}}^{\mathbf{T}}\frac{\partial \mathbf{M}}{\partial \mathbf{q_1}}\dot{\mathbf{q}} \cdots \dot{\mathbf{q}}^{\mathbf{T}}\frac{\partial \mathbf{M}}{\partial \mathbf{q_n}}\dot{\mathbf{q}}\right]^{\mathbf{T}} \tag{2.21}$$

The third term in equation (2.12) represents the contribution of gravitational potential energy. Letting the gravity loading vector be defined as:

$$\mathbf{G}(\mathbf{q}) \;=\; \frac{\partial \mathcal{V}}{\partial \mathbf{q}} \tag{2.22}$$

The rigid body dynamics (RBD) equation of a robot manipulator is represented by:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{G}(\mathbf{q}) = \boldsymbol{\tau} \tag{2.23}$$

The RBD equation (2.23) is linear in terms of its parameters [45], and can thus be rearranged so that the dynamic parameters appear as linearly separated from the rest of the terms:

$$\boldsymbol{\tau} = \boldsymbol{\phi}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\hat{\boldsymbol{\theta}} \tag{2.24}$$

where $\boldsymbol{\phi}$ is an $n \times r$ regressor matrix which depends on the kinematics of the robot, and $\hat{\boldsymbol{\theta}}$ is an $r \times 1$ vector of the parameters. This property of the RBD equation allows for parameter identification techniques that are crucial for adaptive control strategies, as will be shown in Chapter 3.

Another property of the RBD (2.23) states that the mapping from joint torques to joint velocity, $\boldsymbol{\tau} \mapsto \dot{\mathbf{q}}$ is passive, i.e. that for some $\zeta > 0$ and for all finite time, t, the following inequality holds:

$$\int_0^t \dot{\mathbf{q}}^T \boldsymbol{\tau} dt \geq -\zeta \tag{2.25}$$

The passivity property of robot manipulators can be used to design a specific class of controllers which will be discussed further in Chapter 3.

Equation (2.23) represents the nonlinear and coupled dynamics of the robot manipulator, but does not include additional torque components caused by friction, backlash, actuator dynamics and contact with the environment. Coulomb and viscous friction can be modeled in equation (2.23) by the addition of the following dynamics:

$$\boldsymbol{\tau_f} = \mathbf{C}_f sign(\dot{\mathbf{q}}) + \mathbf{V_f}\dot{\mathbf{q}} \tag{2.26}$$

where $\boldsymbol{\tau_f}$ is the torque due to Coulomb and viscous friction, $\mathbf{C_f}$ is the $n \times n$ diagonal matrix containing n Coulomb friction constants, and $\mathbf{V_f}$ is the $n \times n$ diagonal matrix containing n viscous friction constants.

Ideally, the control signal, u, which is applied to the manipulator can be equated to the joint torque, τ. However, due to the dynamics of joint actuators, which are assumed to be servo motors, u is actually a motor voltage, v_m. The actuator dynamics relating the motor voltage and resulting torque can be modeled by the second-order system:

$$J_m\ddot{\theta}_m + \left(B_m + \frac{k_b k_m}{r_a}\right)\dot{\theta}_m = \left(\frac{k_m}{r_a}\right)v_m - \frac{\tau_m}{g_r} \tag{2.27}$$

where J_m is the rotational inertia of the servo motor, θ_m is the motor angle (before gearing), B_m is the motor damping constant, k_m is the torque constant, k_b is the voltage constant, g_r is the gear reduction ratio, r_a is the armature resistance, v_m is the motor voltage, and τ_m is the motor torque. The motor angle θ_m and the corresponding joint angle q_i are related through the gear reduction ratio:

$$\theta_m = g_r q_i \tag{2.28}$$

Applying this to Equation (2.27) results in:

$$g_r^2 J_m \ddot{q}_i + g_r^2\left(B_m + \frac{k_b k_m}{g_r}\right)\dot{q}_i = g_r\left(\frac{k_m}{r_a}\right)v_m - \tau_m \tag{2.29}$$

As with robot kinematics, there are two problems related to the dynamics of the robot - forward and inverse dynamics. Forward, or direct dynamics, solves for the joint acceleration $\ddot{\mathbf{q}}$ given the joint torques $\boldsymbol{\tau}$ as inputs. In order to compute the joint position and velocity,

the computed acceleration is integrated over time. The formulation of forward dynamics is used primarily in the simulation of robotic manipulators. Inverse dynamics is used to compute the joint torques as a function of the manipulator state $(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})$, and is used in various control methods, as described in Section 2.3 below.

2.2.1 Recursive Newton-Euler

While equation (2.23) presents the dynamics in a compact, closed form, more computationally efficient methods of calculating the dynamics exist, which do not calculate the individual dynamic terms in (2.23), but rather, recursively iterate through the links applying the laws of classical mechanics. The recursive Newton-Euler (RNE) formulation [45] is an example of this approach, where a forward iteration propagates the kinematics from the base link to the end-effector. Then, a backward iteration propagates the forces and moments exerted on each link starting from the end-effector down to the base link. The RNE algorithm for a robot consisting of revolute joints is shown in Algorithm 2:

Algorithm 2 Recursive Newton-Euler

initialize: $\dot{\mathbf{v}}_0 \leftarrow \mathbf{g}, \mathbf{v}_0, \boldsymbol{\omega}_0, \dot{\boldsymbol{\omega}}_0 \leftarrow \mathbf{0}$
for $i = 1 \rightarrow n$ **do**
  $\boldsymbol{\omega}_{i+1} = \mathbf{R}_i^{i+1}(\boldsymbol{\omega}_i + \mathbf{z_0}\dot{q}_{i+1})$
  $\dot{\boldsymbol{\omega}}_{i+1} = \mathbf{R}_i^{i+1}[\dot{\boldsymbol{\omega}}_i + \mathbf{z_0}\ddot{q}_{i+1} + \boldsymbol{\omega}_i \times (\mathbf{z_0}\dot{q}_{i+1})]$
  $\mathbf{v}_{i+1} = \boldsymbol{\omega}_{i+1} \times \mathbf{p}_{i+1}^* + \mathbf{R}_i^{i+1}\mathbf{v_i}$
  $\dot{\mathbf{v}}_{\mathbf{i+1}} = \dot{\boldsymbol{\omega}}_{i+1} \times \mathbf{p}_{i+1}^* + \boldsymbol{\omega}_{i+1} \times (\boldsymbol{\omega}_{i+1} \times \mathbf{p}_{i+1}^*)\mathbf{R}_i^{i+1}\mathbf{v_i}$
  $\dot{\bar{\mathbf{v}}}_i = \dot{\boldsymbol{\omega}}_i \times \mathbf{s}_i + \boldsymbol{\omega}_i \times (\boldsymbol{\omega}_{i+1} \times \mathbf{s}_i) + \dot{\mathbf{v}}_i$
  $\mathbf{F}_i = m_i\dot{\bar{\mathbf{v}}}_i$
  $\mathbf{N}_i = \mathbf{I}_i\dot{\boldsymbol{\omega}}_i + \boldsymbol{\omega}_i \times (\mathbf{I}_i\boldsymbol{\omega}_i)$
end for
for $i = n \rightarrow 1$ **do**
  $\mathbf{f}_i = \mathbf{R}_{i+1}^i\mathbf{f}_{i+1} + \mathbf{F}_i$
  $\mathbf{n}_i = \mathbf{R}_{i+1}^i[\mathbf{n}_{i+1} + (\mathbf{R}_{i+1}^{i+1}\mathbf{p}_i^*) \times \mathbf{f}_{i+1}] + (\mathbf{p}_i^* + \mathbf{s}_i) \times \mathbf{F}_i + \mathbf{N}_i$
  $\boldsymbol{\tau_i} = (\mathbf{n}_i)^T(\mathbf{R}_{i+1}^i\mathbf{z}_0)$
end for

The variables used in the RNE algorithm are defined as the following:

i	link index
$\mathbf{g}$	gravity vector, $\mathbf{g} = [0\ 0\ -9.81]$
$\mathbf{I}_i$	moment of inertia of link i about its COM
$\mathbf{s}_i$	position vector of the COM of link i with respect to frame i
$\boldsymbol{\omega}_i$	angular velocity of link i
$\dot{\boldsymbol{\omega}}_i$	angular acceleration of link i
$\mathbf{v}_i$	linear velocity of the origin of frame i
$\dot{\mathbf{v}}_i$	linear acceleration of the origin of frame i
$\bar{\mathbf{v}}_i$	linear velocity of the COM of link i
$\dot{\bar{\mathbf{v}}}_i$	linear acceleration of the COM of link i
$\mathbf{n}_i$	moment exerted on link i by link $i-1$
$\mathbf{f}_i$	force exerted on link i by link $i-1$
$\mathbf{N}_i$	net moment at the COM of link i
$\mathbf{F}_i$	net force at the COM of link i
$\boldsymbol{\tau}_i$	torque experienced at joint i
$\mathbf{R}_i$	rotation matrix defining frame i with respect to frame $i-1$
$\mathbf{p}_i^*$	vector from the origin of frame $i-1$ to frame i with respect to frame i
$\mathbf{z}_0$	unit vector in the z direction, $\mathbf{z}_0 = [0\ 0\ 1]$

2.3 Motion Control

The control of robot manipulators is concerned with determining the necessary sequence of joint torque inputs to achieve a desired motion of the end-effector. A wide range of control methodologies exists depending on the given task and the physical design of the manipulator. As robots are being used to perform increasingly more difficult and complex tasks requiring high accuracy and speed, suitable control algorithms must also be used. This section first outlines the basic method of independent joint control and then covers the more advanced, model-based control.

2.3.1 Independent Joint Control

With independent joint control, or decentralized control [45], each joint of the manipulator is treated as an independent system which is decoupled from the rest of the joints. Hence, the calculation of the control input of a joint is based entirely on its own position, velocity and desired trajectory. 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.

The Proportional-Derivative control (PD) can be applied at each individual joint in order to track a desired trajectory. 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:

$$u_{FB} = k_p e - k_d \dot{q} \tag{2.30}$$

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 (2.23). Instead, these effects are treated as disturbances to the system. The resulting dynamics can be modeled through the modification of equation (2.29) as follows:

$$J_{eff}\ddot{q} + B_{eff}\dot{q} = g_r \left(\frac{k_m}{r_a}\right) v_m - d_k \tag{2.31}$$

where d_k is the disturbance to the system, modeled as a torque applied to the joint, and J_{eff} and B_{eff} are the effective inertia and damping values as seen by the joint:

$$J_{eff} = g_r^2 J_m + m_{ii} \tag{2.32}$$

$$B_{eff} = g_r^2 (B_m + k_b k_m / r_a) \tag{2.33}$$

where J_{eff} accounts for the inertia of the i^{th} link by adding the corresponding diagonal term of the inertia matrix $\mathbf{M}(\mathbf{q})$ (2.23), i.e. m_{ii}. Equating the voltage v_m in (2.31) to the PD control signal, u_{FB} in (2.30), and taking the Laplace transform results in the closed-loop system:

$$\Omega(s) = J_{eff}s^2 + (B_{eff} + kk_d)s + kk_p \tag{2.34}$$

$$Q(s) = \frac{kk_p}{\Omega(s)} Q_d(s) - \frac{g_r}{\Omega(s)} D_k(s) \tag{2.35}$$

where $\Omega(s)$ is the closed-loop characteristic polynomial, $k = k_m g_r / r_a$, and $Q(s)$ and $Q_d(s)$ are the actual and desired joint angles in the frequency domain. This system is illustrated in Figure 2.2. The closed loop system will be stable for all positive values of k_p and k_d and all bounded disturbances. The tracking error is given by the following:

$$E(s) = \frac{J_{eff}s^2 + (B_{eff} + kk_d)s}{\Omega(s)} Q_d(s) + \frac{1}{\Omega(s)} D_k(s) \tag{2.36}$$

During high-speed movements, the dynamic coupling between joints is more significant (i.e. $D_k(s)$ is high) causing higher tracking errors.

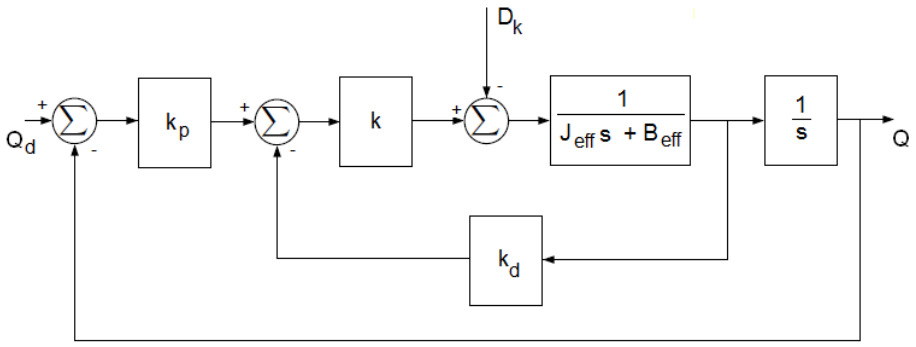

Figure 2.2: Independent Joint PD Control

2.3.2 Model-Based Control

Model-based controllers are a broad class of controllers which apply the joint space dynamic equation (2.23) to cancel the nonlinear and coupling effects of the manipulator. Model-based control can present numerous advantages over decentralized PD control such as increased performance during high-speed movements, reduced energy consumption, improved tracking accuracy and compliance [35].

Nonlinear Feedforward

The goal of nonlinear feedforward control [45],[4] is to eliminate the nonlinearity and coupled behaviour in the dynamics according to equation (2.23) 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 $\mathbf{u_{FB}}$:

$$\mathbf{u} = \mathbf{u_{FB}} + \mathbf{u_{FF}} \tag{2.37}$$

$$\mathbf{u_{FB}} = \mathbf{K_p e} + \mathbf{K_d \dot{e}} \tag{2.38}$$

$$\mathbf{u_{FF}} = \hat{\mathbf{M}}(\mathbf{q_d})\ddot{\mathbf{q}}_\mathbf{d} + \hat{\mathbf{C}}(\mathbf{q_d}, \dot{\mathbf{q}}_\mathbf{d}) + \hat{\mathbf{G}}(\mathbf{q_d}) \tag{2.39}$$

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 (2.23), and $\mathbf{K}_p$ and $\mathbf{K}_d$ are proportional and derivative gain matrices. This controller is illustrated in Figure 2.3. Assuming perfect knowledge of the parameters (i.e., $\hat{\mathbf{M}} = \mathbf{M}$, $\hat{\mathbf{C}} = \mathbf{C}$ and $\hat{\mathbf{G}} = \mathbf{G}$) and substituting (2.37) into the RBD

dynamics equation (2.23), the error dynamics for the system are calculated as:

$$\ddot{\mathbf{e}} + \mathbf{M}^{-1}\mathbf{K_d}\dot{\mathbf{e}} + \mathbf{M}^{-1}\mathbf{K_p}\mathbf{e} = \mathbf{0} \tag{2.40}$$

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 [45]. 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 too far from the desired trajectory, the cancelation of nonlinearities and coupling will be inaccurate.

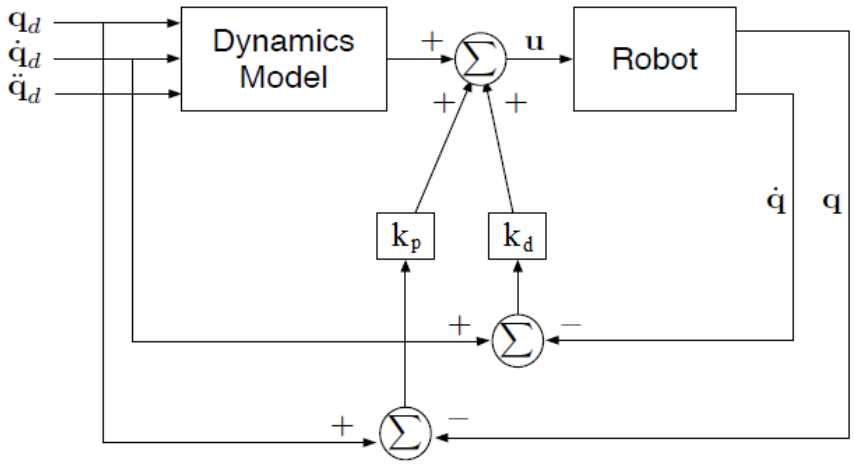

Figure 2.3: Feedforward Control

Inverse Dynamics

The inverse dynamics approach is shown in Figure 2.4 and is often referred to as computed torque control [14],[36]. Assuming that the links of the manipulator behave as rigid bodies, the inverse dynamics approach is equivalent to the concept of feedback linearization used in non-linear controls [45], and applies equation (2.23) 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:

$$\mathbf{u} = \hat{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{q}}_\mathbf{r} + \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\mathbf{G}}(\mathbf{q}) \tag{2.41}$$

$$\ddot{\mathbf{q}}_\mathbf{r} = \ddot{\mathbf{q}}_\mathbf{d} + \mathbf{u}_\mathbf{FB} \tag{2.42}$$

where $\mathbf{u_{FB}}$ is calculated as before in (2.38). Assuming perfect knowledge of the parameters as with Nonlinear Feedforward control (2.39), and substituting (2.41) into the RBD dynamics equation (2.23), the error dynamics for the system are represented by:

$$\ddot{\mathbf{e}} + \mathbf{K_d}\dot{\mathbf{e}} + \mathbf{K_p}\mathbf{e} = \mathbf{0} \tag{2.43}$$

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 [45]. Whereas the feedforward approach allows a significant portion of computation to be performed offline, the inverse dynamics approach requires online computation of the RBD equation (2.23). 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 effective 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.

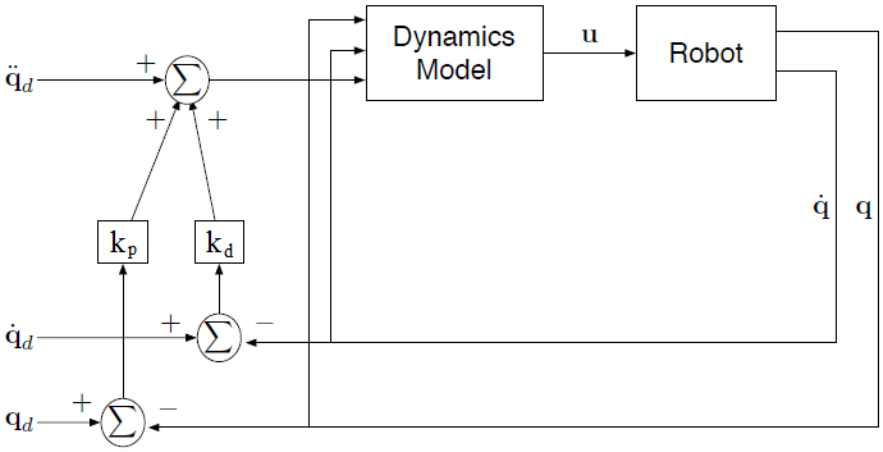

Figure 2.4: Inverse Dynamics Control

2.3.3 Model Uncertainty

While model-based approaches can provide superior performance to independent joint control [56],[35], this is contingent upon the assumption that the dynamic model (2.23) closely matches the actual system, both in the values of the parameters and the structure of the dynamics. In practice, obtaining such a model is a challenging task which involves modeling physical processes that are not well understood or difficult to model, such as friction [6] and backlash. Thus, assumptions concerning these effects are often made to simplify the modeling process, leading to inaccuracies in the model. Furthermore, uncertainties in the physical parameters of a system may be introduced from significant discrepancies between the manufacturer data and the actual system [8]. Changes to operating conditions can also cause the structure of the system model to change. These inaccuracies in the dynamic model cause imperfect cancelation of the nonlinearities and coupling in (2.23) when model-based control is applied.

To represent the model uncertainty in the computed torque control law, (2.41) the control signal u can be rewritten as:

$$\mathbf{u} = \hat{\mathbf{M}}(\mathbf{q})\ddot{\mathbf{q}}_\mathbf{r} + \hat{\mathbf{H}}(\mathbf{q}, \dot{\mathbf{q}}) \tag{2.44}$$

$$\hat{\mathbf{H}}(\mathbf{q}, \dot{\mathbf{q}}) = \hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}}) + \hat{\mathbf{G}}(\mathbf{q}) \tag{2.45}$$

where $\hat{\mathbf{M}}(\mathbf{q})$, $\hat{\mathbf{C}}(\mathbf{q}, \dot{\mathbf{q}})$ and $\hat{\mathbf{G}}(\mathbf{q})$ denote the estimates of the actual values in (2.23). Thus, the modeling error can be described by:

$$\mathbf{\Delta M} := \hat{\mathbf{M}}(\mathbf{q}) - \mathbf{M}(\mathbf{q}) \tag{2.46}$$

$$\mathbf{\Delta H} := \hat{\mathbf{H}}(\mathbf{q}) - \mathbf{H}(\mathbf{q}) \tag{2.47}$$

By substituting the control signal (2.44) into the RBD equation (2.23) and dropping the arguments $(\mathbf{q})$ for clarity, the system becomes:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{H} = \hat{\mathbf{M}}\ddot{\mathbf{q}}_\mathbf{r} + \hat{\mathbf{H}} \tag{2.48}$$

Thus, the joint accelerations can be written as:

$$\begin{aligned} \ddot{\mathbf{q}} =& \mathbf{M^{-1}}\hat{\mathbf{M}}\ddot{\mathbf{q}}_\mathbf{r} + \mathbf{M^{-1}\Delta H} \\ =& \ddot{\mathbf{q}}_\mathbf{r} + \mathbf{E}\ddot{\mathbf{q}}_\mathbf{r} + \mathbf{M^{-1}\Delta H} \end{aligned} \tag{2.49}$$

where $\mathbf{E} = \mathbf{M^{-1}}\hat{\mathbf{M}} - \mathbf{I}$, with $\mathbf{I}$ representing the identity matrix. Applying PD control defined in Equation 2.42 results in the error dynamics:

$$\ddot{\mathbf{e}} + \mathbf{k_d}\dot{\mathbf{e}} + \mathbf{k_p}\mathbf{e} = \eta \tag{2.50}$$

where $\mathbf{e} = \mathbf{q_d} - \mathbf{q}$, and the mathematical notion of model uncertainty is represented by η in the following:

$$\begin{aligned} \boldsymbol{\eta} =& \mathbf{E}\ddot{\mathbf{q}}_\mathbf{r} + \mathbf{M^{-1}}\boldsymbol{\Delta}\mathbf{H} \\ =& \mathbf{E}(\ddot{\mathbf{q}}_\mathbf{d} - \mathbf{k_d}\dot{\mathbf{e}} - \mathbf{k_p}\mathbf{e}) + \mathbf{M^{-1}}\boldsymbol{\Delta}\mathbf{H} \end{aligned} \tag{2.51}$$

The presence of the $\mathbf{E}, \mathbf{M}$ and $\Delta\mathbf{H}$ terms in the error dynamics indicates that the system is still nonlinear and coupled due to the model uncertainty. Hence, the application of PD control (2.30) cannot necessarily be tuned to achieve stability and convergence of tracking error.

Since the introduction of model-based control, much effort has been devoted to developing techniques for model estimation and finding ways of compensating for model uncertainty. Techniques such as system identification, as well as adaptive and robust control have been proposed. Many of these approaches for dealing with model uncertainty rely on the knowledge of the structure of the dynamics and treat the parameters as unknowns to be identified. Newer approaches to manipulator control involve data-driven learning of the inverse dynamics relationship of a manipulator, thus eliminating the need for any a-priori knowledge of the system model. Unlike the adaptive control strategies, these approaches do not assume an underlying structure but rather attempt to infer a model which describes the observed data as closely as possible. These techniques, as well as the model-based approaches are summarized in the following chapter.

Chapter 3

Related Work

This chapter gives a brief summary of the existing research on model uncertainty in robot manipulator control. To begin with, the approaches of robust and adaptive control are presented, followed by various learning control approaches which originate from the machine learning community.

3.1 Model-based Control

Following the introduction of model-based control and the method of computed torque control (2.41) described in Chapter 2, researchers began to address the issue of model uncertainty and how to effectively control the system under this uncertainty. In a broad sense, two strategies have evolved - robust control and adaptive control. Robust controllers have a fixed structure and are designed to have low sensitivity to parameter variations, disturbances and unmodeled dynamics [45]. Adaptive controllers, which are time-varying, employ online parameter estimation techniques to compensate for uncertainty in the dynamic model [45]. Similar to the concept of adaptive control, given the structure of the system, parameter identification can be used to estimate the unknown parameters. However, unlike adaptive control, parameter identification is an offline procedure that processes batches of data collected from the system and applies regression techniques such as least squares [8], [24], [40] to determine the parameter values. Because the identification procedure is carried out offline, the trajectories can be optimized specifically to excite the dynamics to a sufficient extent in order to yield more accurate results. However, due to its offline nature, this procedure is not well-suited to deal with systems in which the parameters may vary over time.

3.1.1 Robust Control

Whereas adaptive controllers deal with model uncertainty by attempting to identify a more accurate model of the system through parameter update laws, robust control schemes focus on the development of the control strategies which can satisfy a given performance criteria over a range of uncertainty.

Various strategies have been proposed for robust control. Anticipating that the inexact linearization and decoupling due to model uncertainty will introduce nonlinearities into the error dynamics (2.43), well-known multivariable non-linear control techniques such as the total stability theorem [19], Youla parameterization and H^∞ [51], are used to design compensators which guarantee convergence and stability of the system error for a given set of nonlinearities. However, the application of these non-linear multivariable techniques can often result in high-gain systems [3].

An alternative solution to dealing with model uncertainty is the application of variable-structure controllers such as sliding mode control [48]. The main feature of such controllers is that the nonlinear dynamic behaviour of the system is altered through the use of discontinuous control signals which drive the system dynamics to 'slide' across a surface where the system can be approximated by an exponentially stable linear time invariant system. Hence, asymptotic stability of the tracking error can be achieved [3] even in the presence of model uncertainty. Despite this advantage, due to the discontinuous control signals sliding mode systems are susceptible to control chattering, which may result in the excitation of high-frequency dynamic behaviour [3].

3.1.2 Adaptive Control

Adaptive control is a broad class of time-varying controllers which deal with model uncertainty by attempting to identify a more accurate model of the system through parameter update laws. Craig et al. [14] present an adaptive version of the computed torque control method named Adaptive Computed Torque (ACT) control where the parameter update law is driven by the tracking error of the system. A Lyapunov function is specified as:

$$\boldsymbol{v}(\mathbf{E}, \hat{\boldsymbol{\theta}}) = \boldsymbol{\mathcal{E}^T \mathcal{P} \mathcal{E}} + \hat{\boldsymbol{\theta}}^{\boldsymbol{T}} \boldsymbol{\gamma^{-1}} \hat{\boldsymbol{\theta}} \tag{3.1}$$

where $\mathcal{E} = [\mathbf{e}\ \dot{\mathbf{e}}]^T$ is the vector of the filtered servo error and its derivative, $\mathcal{P}$ is a semi-positive definite constant matrix, $\hat{\boldsymbol{\theta}}$ is the estimate of the unknown dynamic parameters, $\boldsymbol{\gamma}$ is an $r \times r$ gain matrix, and r is the number of parameters. The equilibrium point is given by $\mathcal{E} = 0$. By making use of the property of linearity in the parameters (2.24), the

following gradient-based update law is chosen to estimate the parameters online:

$$\dot{\hat{\boldsymbol{\theta}}} = \boldsymbol{\gamma}\boldsymbol{\phi}^{\boldsymbol{T}}(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}})\hat{\boldsymbol{M}}^{-\mathbf{1}}\boldsymbol{e} \tag{3.2}$$

where $\boldsymbol{\phi}$ is the regressor matrix (2.24), $\hat{\mathbf{M}}$ is the estimated inertia matrix. By using the adaptive law in Equation 3.2, the requirements for Lyapunov stability are met, and hence asymptotic convergence of the tracking error is guaranteed. However, convergence of the estimated parameters to their true values is not guaranteed and relies on a condition termed persistence of excitation (PE), given by the inequality:

$$\alpha_{PE}\mathbf{I_r} \leq \int_{\mathbf{t_0}}^{\mathbf{t_0}+\rho} \boldsymbol{\phi}_{\mathbf{d}}^{\mathbf{T}}\boldsymbol{\phi}_{\mathbf{d}}\mathbf{dt} \tag{3.3}$$

where α_{PE} is a positive number, ρ is a time interval, $\mathbf{I_r}$ is the $r \times r$ identity matrix, and $\boldsymbol{\phi}_d$ is the regressor matrix $\boldsymbol{\phi}$ evaluated along the desired trajectory instead of the actual trajectory. This is a valid simplification due to the convergence of tracking error. It is shown that if this inequality is met for all time from t_0 to $t_0 + \rho$ and some $\alpha > 0$, parameter errors will converge to a bounded region near zero.

The adaptive approach presented in [14] makes two major assumptions - firstly, that the inverse of the estimated inertia matrix is always bounded, and secondly, that measurements of joint acceleration are available. To satisfy the first assumption, parameter estimates are restricted to lie within a-priori known bounds of the actual parameters [14]. Spong and Ortega [51] propose using fixed a-priori estimates of the dynamics and an additional outer loop control component which is chosen adaptively to compensate for the inaccurate dynamics. The same update law as (3.2) is used, and hence the measurement of joint accelerations is still required for parameter estimation.

Middleton et al. [28] also build upon the ACT controller [14] but eliminate the need for joint acceleration to perform the parameter estimates by filtering the linear parameterization equation (2.24) such that the regressor matrix $\boldsymbol{\phi}$ is a function strictly of joint angles and velocity. Other researchers address this issue through the use of nonlinear observers to perform state estimation [11],[57] of the joint accelerations.

A number of adaptive laws have been proposed [26],[47] based on the preservation of passivity properties of the RBD model (2.25). Although they differ from the class of controllers originating from [14], the motivation for these schemes is also to eliminate the need for joint acceleration measurement [37].

Despite the numerous advancements, the adaptive methods presented thus far are still reliant upon adequate knowledge of the structure of the dynamic model and are thus particularly susceptible to the effects of unmodeled dynamics. In an attempt to account

for this, the RBD equation (2.23) used in adaptive control laws has been extended to include additional dynamic effects such as actuator dynamics [42],[57] and joint friction [12]. However, in many cases, such as with friction [6], simplified dynamic models are often used to approximate physical processes which, in reality, are complex and highly nonlinear. Consequently, dealing with the effects of unmodeled dynamics remains an open research area within adaptive control, and some researchers have combined adaptive control with robust control techniques [3],[11] and even learning control [38],[22].

3.2 Learning Model-based Control

Newer approaches to manipulator control involve data-driven learning of the inverse dynamics relationship of a manipulator, thus eliminating the need for a-priori knowledge of the system model. Research stemming from the machine learning community often approaches this problem through function estimation, or supervised learning [9]. By analyzing the input torques and resulting motion of the manipulator, an approximation of the dynamic equation of the manipulator can be obtained. Unlike the adaptive control strategies, many of these approaches do not assume an underlying structure but rather attempt to infer a model which describes the observed data as closely as possible. Thus, it is possible to encode nonlinearities whose structure may not be well-known.

Rather than learning the underlying model structure of the system through supervised learning, Iterative Learning Control (ILC) [5], [10] incorporates information from error signals in previous iterations to directly modify the control input for subsequent iterations. However, ILC is limited primarily to systems which track a specific repeating trajectory and are subject to repeating disturbances [10], whereas the model learning approaches discussed below can be incrementally trained to deal with non-repeating trajectories.

While supervised learning is carried out through the provision of exemplar data of the model being learned, another branch of machine learning, namely Reinforcement Learning (RL) [53], is concerned with learning in situations where such exemplar data is not available. The goal of reinforcement learning is to determine the set of actions, or *policy* that must be taken in order to maximize a cumulative notion of *reward*. While reinforcement learning has been successfully applied in many areas of robotics including motor primitive learning [25], motion planning [23], and locomotion [54],[30], for the problem of learning an internal model of the system where exemplar data is readily available in the form of joint encoder data and commanded motor torques, RL would be much slower, especially in high-dimensional spaces [53]. Thus, the following sections will focus on reviewing the relevant work in supervised learning, which can be broadly categorized into two types [55] - global methods

such as Neural Networks (NN), Gaussian Process Regression (GPR) [41] and Support Vector Regression (SVR) [35], and local methods such as Locally Weighted Projection Regression (LWPR) [43]. Regardless of the approach, it is desirable that learning can be achieved in real-time by processing large amounts of data in an online, incremental manner as opposed to offline batch processing. Furthermore, with online learning, the training data could potentially cover an increasingly large portion of the input space over time. Hence, the learning algorithm should also be able to adapt its models to account for the changing input space.

3.2.1 Local Learning

Local learning approaches fit nonlinear functions with spatially localized models, usually in the original input space of the training data [55]. Typically, simple local models (linear or low-order polynomial) are used for the local models, and the learning algorithm automatically adjusts the complexity (i.e., number of local models and their locality) to accurately account for the nonlinearities and distributions of the target function.

Atkeson presents an early approach to local learning [7] termed Locally Weighted Regression (LWR) which approximates a continuous nonlinear function by storing samples of the function's inputs and outputs. When queried for a prediction, LWR builds a linear model in a local region around the query point through weighted least squares regression. Although LWR is shown to exhibit superior prediction performance compared to neural networks [4], the entire training data set is stored in memory, and must be exhaustively searched to perform a prediction. Thus, the memory and computational requirements are highly unfavourable, especially if the algorithm receives a large stream of incoming training data, which is a characteristic of real-time control of robotic systems.

Building upon LWR, Receptive Field Weighted Regression (RFWR) [44] allows for incremental, online learning by continuously constructing locally linear models in the region of input space that data is being observed in. Thus, a set of piecewise local models are stored in memory rather than the entire training data set, and a prediction can be performed by taking the weighted average of the predictions of all the local models. The region of validity of each local model, termed as the 'receptive field', is learned through the use of gradient descent with a cross validation cost function.

Although RFWR eliminates the need for storage of the entire training data set, it still suffers from the curse of dimensionality, in that increasing the dimension of the data set results in an increase in computational requirements. This is a significant problem for high-DOF robot manipulators which generate training points with an input dimensionality of

$3n$ (angle, velocity and acceleration of each joint).

To address this issue, the technique of Locally Weighted Projection Regression (LWPR) is introduced in [43], which incorporates the principles of RFWR but also incorporates methods for dealing with high-dimensional systems. As LWPR will be the basis of much of the work in Chapters 4, 5, and 6, the complete algorithm is explained in detail in the following paragraphs.

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

$$\mathbf{y} = \mathbf{f}(\mathbf{x}) + \boldsymbol{\varepsilon} \tag{3.4}$$

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:

$$y_{ik} = \boldsymbol{\beta_{ik}}^T \mathbf{x} + \boldsymbol{\varepsilon} \tag{3.5}$$

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 [55] is given by

$$w_{ik} = exp(-\frac{1}{2}(\mathbf{x} - \mathbf{x_{ck}})^{\mathbf{T}} \mathbf{D_{ik}} (\mathbf{x} - \mathbf{x_{ck}})) \tag{3.6}$$

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

$$\hat{y}_i(x) = \sum_{k=1}^{K} w_{ik} \hat{y}_{ik} / \sum_{k=1}^{K} w_{ik} \tag{3.7}$$

where K is the number of linear models, $\hat{y}_{ik}$ is the prediction of the ik^{th} local linear model given by (3.5) which is weighed by weight w_{ik} (as computed by 3.6) 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 $\mathbf{y}$. An example of LWPR approximating a nonlinear function with a

single dimension output and input is given in Figure 3.1. In this figure, the data points represent noisy samples taken from a decaying sinusoidal function. Each linear model is represented by a single line which is fit to a subset of the data, and its corresponding receptive field is illustrated by an ellipse which illustrates the region of validity of the local model.

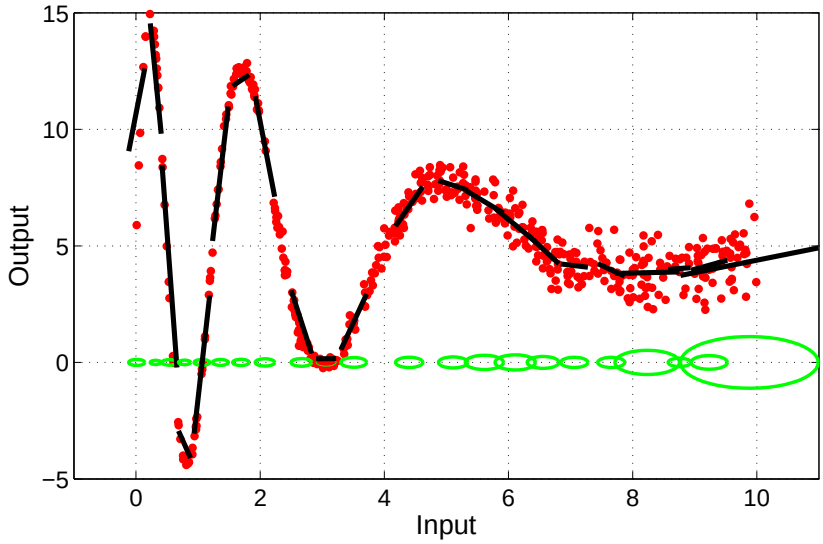

Figure 3.1: Single dimension input and output LWPR example

The distance parameter, $\mathbf{D}$, (3.6) is learned for an individual local model through stochastic gradient descent given by

$$\mathbf{m}^{n+1} = \mathbf{m}^n - a_L \frac{\partial J_{cost}}{\partial \mathbf{m}} \tag{3.8}$$

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 [43] given by:

$$J = \frac{1}{\sum_{i=1}^{L} w_i} \sum_{i=1}^{L} w_i (y_i - \hat{y}_{i,-i})^2 + \frac{\gamma}{n} \sum_{i,j=1}^{N} D_{ij}^2 \tag{3.9}$$

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 (3.6), 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 [43], it was found that the regression results are not very sensitive to this parameter and a value of $\gamma = 1e-7$ 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 (3.6) 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 [43]. 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 [43].

3.2.2 Global Learning

Global learning methods fit nonlinear functions globally, typically by transforming the input space with predefined or parameterized basis functions and subsequent linear combinations of the transformed inputs. An early approach to the problem of global function approximation is the Neural Network (NN) [20], which consists of a group of interconnected neurons, or nodes representing mathematical transformations. Collectively, the entire network defines a function mapping, $f\colon \mathcal{R}^{3n} \rightarrow \mathcal{R}^n, \mathbf{x} \mapsto \mathbf{y}$, i.e. from a $3n \times 1$ input vector $\mathbf{x}$ to the corresponding $n \times 1$ output vector $\mathbf{y}$. In [27], NN approaches are used to approximate the inverse dynamics function of a simple planar manipulator. Neural networks have

also been used in conjunction with adaptive control strategies in [38] to compensate for unmodeled dynamics. However, the NN approach employs a fixed parametric structure, as the number of nodes as well as their connections must be fixed prior to training [20],[27]. Thus, for applications in online control where the input space of the training data may continuously change, NNs may not be suitable.

A more recent regression technique is Support Vector Regression (SVR) [49]. By using a kernel mapping function, SVR projects the training data to a high-dimensional space where a linear regression model can be used to describe the training data. The model parameters are learned by solving a constraint optimization problem which sweeps through the entire training data set. Consequently, SVR is inherently an offline algorithm which does not support incremental updates to the model. Attempts to make this algorithm computationally tractable for online control involve sparsification of the training data set, i.e. representing the entire training data set with a smaller subset of key data points. In [18], a test of linear independence is applied such that only data points which cannot be approximated with the existing data are added to the model. However, the sparse data set is unbounded, and is allowed to become arbitrarily large during online learning. To deal with this, a framework is developed in [35] for the insertion or deletion of key data points into the sparse representation of the entire data set, often referred to as a 'dictionary.'

3.2.3 Gaussian Processes Regression (GPR)

Gaussian Process Regression (GPR) [41] is another global supervised learning technique which employs Gaussian process (GP) models to formulate a Bayesian framework for regression. A GP is characterized by its mean and covariance functions which provide the prior for Bayesian inference. After updating the GP prior with training data to yield the posterior GP, the parameters of the mean and covariance function, known as the hyperparameters, are updated according to the training data. This is typically done by selecting hyperparameters that maximize the probability of observing the training data. As with LWPR, GPR and its variants will be the focus of Chapters 4, 5, and 6, and hence, the algorithms are explained in detail in the following paragraphs.

Given the standard regression model (3.4), the goal of GPR is to find the function f which maps the inputs $\mathbf{X}$ to their target output values $\mathbf{y}$. The single observed output can be described by

$$y \sim \mathcal{N}(0, K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_n) \tag{3.10}$$

where σ_n^2 is the noise variance, $\mathbf{I}_n$ is the $n \times n$ identity matrix, and $K(\mathbf{X}, \mathbf{X})$ is the covariance

matrix which is composed of the covariances, $k(\mathbf{x}, \mathbf{x}')$, evaluated at all pairs of the training points. A widely used covariance function is given by the squared exponential (SE) form as

$$k(\mathbf{x}, \mathbf{x}') = \theta_1 \exp(-\theta_2 \|\mathbf{x} - \mathbf{x}'\|^2) \tag{3.11}$$

where θ_1 and θ_2 are the two hyperparameters of the SE covariance term which control the amplitude and characteristic length-scale respectively. The hyperparameters of the Gaussian process can be learned for a particular data set by maximizing the log marginal likelihood using optimization procedures such as gradient-based methods.

To make a prediction $\bar{f}_*(\mathbf{x}^*)$ given a new input vector $\mathbf{x}^*$, the joint distribution of the output values and the predicted function is given by

$$\begin{bmatrix} y \\ \bar{f}_*(\mathbf{x}^*) \end{bmatrix} \sim \mathcal{N}(0, \left(\begin{bmatrix} K(\mathbf{X}, \mathbf{X}) + \sigma_n^2 \mathbf{I}_n & k(\mathbf{X}, \mathbf{x}^*) \\ k(\mathbf{x}^*, \mathbf{X}) & k(\mathbf{x}^*, \mathbf{x}^*) \end{bmatrix} \right) \tag{3.12}$$

Thus, the conditional distribution gives the predicted mean value $\bar{f}_*(\mathbf{x}^*)$ with variance $V(\mathbf{x}^*)$

$$\begin{aligned} \bar{f}_*(\mathbf{x}^*) &= k_*^T (K + \sigma_n^2 I)^{-1} y = k_*^T \alpha, \\ V(\mathbf{x}^*) &= k(\mathbf{x}^*, \mathbf{x}^*) - k_*^T (K + \sigma_n^2 I)^{-1} k_* \end{aligned} \tag{3.13}$$

where $k_* = k(\mathbf{X}, \mathbf{x}^*)$, $K = K(\mathbf{X}, \mathbf{X})$ and α is the prediction vector.

A training data set of size N requires the inversion of the $N \times N$ matrix in (3.13). This is achieved through Cholesky factorization [41], and results in a computational complexity of $\mathcal{O}(N^3)$ for training with the standard GPR. Once this inversion is done, prediction of the mean and variance are $\mathcal{O}(N)$ and $\mathcal{O}(N^2)$ respectively.

The GPR framework has been applied in several previous works [29],[33],[36],[41] to learn the inverse dynamics function of a robot manipulator. However, due to the computational complexity of GPR which scales cubically with the number of training points ($\mathcal{O}(N^3)$), the learning is performed entirely offline. Attempts to alleviate this problem have been proposed both by the robotics [34] and machine learning community [16],[41],[50]. In [34], online learning is achieved by combining GPR with the approach of local learning, such that each local GP sees only a fraction of the total training data and hence remains computationally tractable. Other researchers rely on sparse representations of the full GP which typically involves representing the full training data set of size N with a subset of data consisting of M_s elements, where $M_s << N$ [16],[41],[50]. A common method of choosing this subset of the data is based on an information gain criterion [50].

3.2.4 Sparse Pseudo-input Gaussian Processes (SPGP)

A recent example of an approximation of the full GPR procedure is the Sparse Pseudo-input Gaussian Processes (SPGP) [50]. SPGP introduces a method of simultaneously finding an active set of point locations for 'pseudo-inputs', denoted by $\bar{\mathbf{X}}$ while learning the hyperparameters of the Gaussian process in a smooth joint optimization scheme. These pseudo-inputs can be viewed as a parametrization of an approximation of the GP covariance function (3.11)[50]. Unlike other sparse approximations, the pseudo-inputs are not a fixed subset of the training data but are treated as parameters to be optimized to yield a better fit to the data. The SPGP covariance function is given by

$$\begin{aligned} \mathbf{K}^{SPGP}(\mathbf{x}_N, \mathbf{x}_N) &= \mathbf{K_{NM}}\mathbf{K_M^{-1}}\mathbf{K_{MN}} + \mathbf{\Lambda} \\ \mathbf{\Lambda} &= diag(\lambda) \\ \lambda_n &= K(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{K}_{NM}\mathbf{K}_M^{-1}\mathbf{K}_{MN} \end{aligned} \tag{3.14}$$

where the $N \times M$ matrix $\mathbf{K_{NM}}$ is composed of the covariances between the N data points and M_s pseudo-inputs given by $K(\mathbf{x}_N, \bar{\mathbf{x}}_M)$ and $\mathbf{K_M}$ is composed of the covariance of the pseudo-inputs given by $K(\bar{\mathbf{x}}_M, \bar{\mathbf{x}}_M)$. By calculating (3.14) for each data point, the SPGP covariance matrix, $\mathbf{K}_N^{SPGP}$ is formed. Similar to (3.10), the marginal likelihood is given by

$$p(y|(\mathbf{X}, \bar{\mathbf{X}}, \mathbf{\Theta}) = \mathcal{N}(\mathbf{y}|0, \mathbf{K}_N^{SPGP} + \sigma^2\mathbf{I}_n) \tag{3.15}$$

Both the hyperparameters, $\mathbf{\Theta}$, and the pseudo-inputs, $\bar{\mathbf{X}}$ are learned jointly by maximizing the likelihood using gradient ascent. With standard GP regression, this step requires inverting $\mathbf{K}_N$ which is composed of the complete training data set, thus requiring $\mathcal{O}(N^3)$ complexity for training. With SPGP, the equivalent step involves inverting $\mathbf{K}_N^{SPGP}$ which is not only smaller in dimension compared $\mathbf{K}_N$ but consists of a low rank part and a diagonal part (3.14) [50]. This allows the inversion in $\mathcal{O}(M_s^2 N)$ time for training.

Similar to GP regression, the predictive distribution for a new point $\mathbf{x}^*$ can then be computed through the following

$$\begin{aligned} \bar{f}(\mathbf{x}^*) &= \mathbf{k}_*^T Q_M^{-1}\mathbf{K}_{MN}(\mathbf{\Lambda} + \sigma_n^2\mathbf{I}_n)^{-1}y \\ V(\mathbf{x}^*) &= K(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{k}_*^T(\mathbf{K}_M^{-1} - \mathbf{Q}_M^{-1})\mathbf{k}_* + \sigma_n^2\mathbf{I}_n, \end{aligned} \tag{3.16}$$

where $\mathbf{Q} = \mathbf{K_M} + \mathbf{K_{MN}}(\mathbf{\Lambda} + \sigma^2\mathbf{I})^{-1}\mathbf{K_{NM}}$. Hence, the predictive mean and variance can be computed in $\mathcal{O}(M_s)$ and $\mathcal{O}(M_s^2)$ respectively after the initial $\mathcal{O}(M_s^2 N)$ training time.

3.2.5 Sparse Online Gaussian Processes (SOGP)

Similar to SPGP, Sparse Online Gaussian Processes (SOGP) [16],[15] is another sparse approximation of the full GPR framework which reduces the computational load of GPR by keeping track of a sparse set of inputs named Basis Vectors (BV). SOGP also incorporates a method of incrementally processing data by treating the posterior mean $\bar{\mathbf{f}}_N$ and covariance functions $K(\mathbf{X}, \mathbf{X})$ as linear combinations of the prior covariance functions $K_0(\mathbf{x}, \mathbf{x}')$:

$$\bar{\mathbf{f}}_N = \bar{\mathbf{f}}_0 + \sum_{i=1}^{N} K_0(\mathbf{x}, \mathbf{x}_i)\alpha_N(i) = \bar{\mathbf{f}}_0 + \boldsymbol{\alpha}_N^T \mathbf{k}_x \tag{3.17}$$

$$\begin{aligned} K(\mathbf{X}, \mathbf{X}) &= K_0(\mathbf{x}, \mathbf{x}') + \sum_{i,j=1}^{N} K_0(\mathbf{x}, \mathbf{x}_i) C_n(ij) K_0(\mathbf{x}_j, \mathbf{x}') \\ &= K_0(\mathbf{x}, \mathbf{x}') + \mathbf{k}_x^T \mathbf{C}_N^{GP} \mathbf{k}_{x'} \end{aligned} \tag{3.18}$$

where $\boldsymbol{\alpha}_N = [\alpha_N(1), ..., \alpha_N(t)]^T$, $\mathbf{C}_N^{GP} = \{C_N(ij)\}_{i,j=1}$ are the mean and covariance parameters, and $\mathbf{k}_x = [K_0(\mathbf{x}_1, \mathbf{x}), .., K_0(\mathbf{x}_N, \mathbf{x})]^T$ is the vector of kernel functions centered on each combination of training points. The parameters $\boldsymbol{\alpha}$ and $\mathbf{C}^{GP}$ can be iteratively updated [16] through the following:

$$\begin{aligned} \boldsymbol{\alpha}_{N+1} &= T_{N+1}(\boldsymbol{\alpha}_N) + q_{N+1}(\mathbf{s}_{N+1}) \\ \mathbf{C}_{N+1}^{GP} &= U_{N+1}(\mathbf{C}_N^{GP}) + r_{N+1}(\mathbf{s}_{N+1}\mathbf{s}_{N+1}^T) \\ \mathbf{s}_{N+1} &= T_{N+1}(\mathbf{C}_N^{GP}\mathbf{k}_{N+1}) + \mathbf{e}_{N+1} \end{aligned} \tag{3.19}$$

where $\mathbf{e}_{N+1} = [0, .., 1]^T$ is a unit vector of length $N+1$, and the vector $\mathbf{s}_{N+1}$ is introduced for clarity of the equations. Operators T_{N+1} and U_{N+1} extend an N-dimensional vector and $N \times N$ dimensional matrix to an $N+1$ vector and $(N+1) \times (N+1)$ matrix respectively by appending a zero to the end of the vector, and a zero row and column to the matrix. The scalars q_{N+1} and r_{N+1} are computed as follows:

$$\begin{aligned} q_{N+1} &= \frac{\delta}{\delta \bar{f}_N(\mathbf{x}_{N+1})} ln\langle P(y_{N+1}|f_N(\mathbf{x}_{N+1})\rangle_N \\ r_{N+1} &= \frac{\delta^2}{\delta \bar{f}_N^2(\mathbf{x}_{N+1})} ln\langle P(y_{N+1}|f_N(\mathbf{x}_{N+1})\rangle_N \end{aligned} \tag{3.20}$$

where $\langle\rangle_N$ denotes the average with respect to the GP at the N^{th} iteration. As seen in Equation 3.19, the dimensions of $\boldsymbol{\alpha}$ and $\mathbf{C}^{GP}$ increase with each data point added, which is problematic for computational tractability when dealing with large data sets. However,

if the new input $\mathbf{x}_{N+1}$ can be represented by a linear combination of covariance functions constructed about the existing N data points, the dimensions of $\boldsymbol{\alpha}$ and $\mathbf{C}^{GP}$ do not need to be increased. This is expressed in the following:

$$K(\mathbf{x}, \mathbf{x}_{N+1}) = \sum_{i=1}^{N} \hat{\mathbf{e}}_{N+1}(i) K_0(\mathbf{x}, \mathbf{x}_i) \tag{3.21}$$

If the vector $\hat{\mathbf{e}}_{N+1}$ can be found, then the updated GP could be represented using only the first N inputs. However, for most covariance functions and inputs $\mathbf{x}_{N+1}$, Equation 3.21 cannot be satisfied for all x [16]. Instead, an approximation of the solution can be achieved by minimizing the error measure:

$$\left\| K_0(\mathbf{x}, \mathbf{x}_{N+1}) - \sum_{i=1}^{N} \hat{\mathbf{e}}_{N+1}(i) K_0(\mathbf{x}, \mathbf{x}_i) \right\|^2 \tag{3.22}$$

where $\|\cdot\|^2$ is a suitably defined norm, which is selected in [16] as a kernel Hilbert space norm. The solution of the minimization problem in Equation 3.22 is given by:

$$\hat{\mathbf{e}}_{N+1} = \mathbf{K}_N^{-1} \mathbf{k}_{N+1} \tag{3.23}$$

where $\mathbf{K}_N = \{K_0(\mathbf{x}_i, \mathbf{x}_j)\}_{i,j=1}$. Applying this solution to Equation 3.21 results in:

$$\hat{K}_0(\mathbf{x}, \mathbf{x}_{N+1}) = \sum_{i=1}^{N} \hat{\mathbf{e}}_{N+1}(i) K_0(\mathbf{x}, \mathbf{x}_i) \tag{3.24}$$

where $\hat{K}_0(\mathbf{x}, \mathbf{x}_{N+1})$ can be seen as the orthogonal projection of $K_0(\mathbf{x}, \mathbf{x}_{N+1})$ on the linear span of the functions $K_0(\mathbf{x}, \mathbf{x}_i)$.

To determine whether or not the current input $\mathbf{x}_N$ will be included in the BV set, the residual error vector from the projection in (3.24) is calculated:

$$\gamma_{N+1} = K_0(\mathbf{x}_{N+1}, \mathbf{x}_{N+1}) - \mathbf{k}_{N+1}^T \mathbf{K}_N^{-1} \mathbf{k}_{N+1} \tag{3.25}$$

where γ_{N+1} can be treated as a measure of the 'novelty' of the current input. An input resulting in a high value of γ indicates that the new data cannot be described well through linear combinations of the existing data points, and hence should be added to the BV set. The overall SOGP algorithm is described in the following algorithm.

Algorithm 3 Sparse Online Gaussian Process Regression

For a new data point, $(y_{N+1}, \mathbf{x}_{N+1})$
Compute q^{N+1}, r^{N+1} (3.20), $\hat{\mathbf{e}}_{N+1}$ (3.23), and γ_{N+1} (3.25)
if $\gamma_{N+1} < \epsilon_{tol}$ **then**
    Compute $\boldsymbol{\alpha}$ and $\mathbf{C}^{GP}$ without extending their size:
    $\boldsymbol{\alpha}_{N+1} = (\boldsymbol{\alpha}_N) + q_{N+1}(\mathbf{s}_{N+1})$
    $\mathbf{C}^{GP}_{N+1} = \mathbf{C}^{GP}_N + r_{N+1}\mathbf{s}_{N+1}(\mathbf{s}^T_{N+1})$
    $\mathbf{s}_{N+1} = \mathbf{C}^{GP}_N \mathbf{k}_{N+1} + \hat{\mathbf{e}}_{N+1}$ where $\hat{\mathbf{e}}_{N+1}$ is computed according to (3.23)
else
    Compute $\boldsymbol{\alpha}$ and $\mathbf{C}^{GP}$ according to (3.19)
    Add current input $\mathbf{x}_{B+1}$ to the BV set
end if
if Size of BV set $> M_{BV}$ **then**
    Compute the novelty, γ, of each BV and remove the lowest scoring one
end if

3.2.6 Incorporating Prior Knowledge

The learning approaches introduced thus far assume that there is no prior knowledge of the system dynamics, and thus learning commences entirely from 'scratch'. Furthermore, adequate performance of these algorithms requires sufficient training in the regions of state space that the system is expected to operate in. Thus, the performance of these learning algorithms is poor outside of the regions in which they have trained, especially in the case of local learning. Instead of assuming that there is no a-priori knowledge of the system dynamics, recent work in learning control aims to incorporate the RBD model (2.23), to provide a global characterization of the dynamics. In [32], the RBD model is incorporated into the GPR algorithm to improve its performance in terms of generalization and prediction accuracy, however the high computational requirements of GPR still prohibit incremental online updates from being made, and hence the system must be trained offline. Thus, the use of the RBD model as prior knowledge has not yet been applied to online learning algorithms.

Chapter 4

Comparison of Model-Based and Learning controllers

While much progress has been made in the field of adaptive control and the more recent learning approaches to dealing with model uncertainty, little work has been carried out to compare the two. Burdet and Codourey [11] present a comparison between various learning approaches (such as Neural Networks and and ILC) and model-based adaptive controllers. Since then, there has not been any work in comparing the newer generation of regression-based learning (LWPR, SVM, GPR) techniques to the classical model-based control strategies. This chapter [1] focuses on evaluating Locally Weighted Projection Regression (LWPR) as an alternative to the model-based techniques such as computed torque (CT) or adaptive computed torque (ACT) control. Simulations are carried out in order to evaluate the position and orientation tracking performance of each controller while varying trajectory velocities, end-effector loading and errors in the known parameters.

In order to evaluate the performance of the LWPR learning controller, two model-based controllers were also implemented: the computed torque (CT) controller in (2.41) and the adaptive computed torque (ACT) controller as described in Equation 3.2. The performance of these controllers was evaluated in simulation using MATLAB/Simulink, the Robotics Toolbox (RTB) [13] and the open source LWPR code [55]. LWPR was used to learn the joint space dynamics of a standard 6 DOF Puma 560 with the kinematic and dynamic parameters obtained from the RTB. The control loop executed at 1ms while the model was updated every 5ms for both the LWPR and ACT algorithms.

[1] A version of this chapter has been published. J. Sun de la Cruz, D. Kulić and W. Owen, A Comparison of Classical and Learning Controllers, World Congress of the International Federation of Automatic Control, pp. 1102-1107, 2011.

4.1 Trajectories

In order to properly assess the performance of the model-based controllers, a trajectory which excites the dynamics of the system caused by inertia, gravity and Coriolis/centripetal effects in (2.23) must be tracked. The figure-8 trajectory [31] is used, as it includes both straight and curved sections which induce significant torques from Coriolis/centripetal effects. The figure-8 trajectory, as seen in Figure 4.1, is described by the following task space equations:

$$\begin{aligned} x_d &= \frac{w_8}{2}\sin(\frac{2\pi}{K_c}t) + x_i \\ y_d &= \frac{l_8}{2}\cos(\frac{4\pi}{K_c}t) + y_i - \frac{l_8}{2} \\ z_d &= z_i \end{aligned} \tag{4.1}$$

where $\mathbf{p_d} = [x_d\ y_d\ z_d]^T$ is the vector of the desired end-effector task space position, $[x_i\ y_i\ z_i]$ is the desired Cartesian initial position, $2\pi/K_c$ is the desired duration of one cycle of the figure-8 in seconds, and w_8 and l_8 are the width and length of the trajectory. The task-space trajectory in Equation 6.1 is converted to a joint space trajectory at a sampling rate of 1ms through the inverse kinematics algorithm in Chapter 2. Desired velocity and acceleration terms are generated through numerical differentiation also with a 1ms time step using Matlab. All trajectory generation was carried out offline.

In addition to the figure-8 trajectory, a persistently exciting (PE) trajectory [14] is used to provide sufficient excitation of the dynamics. This trajectory was designed directly in the joint space as a linear combination of sinusoids given by:

$$q_i = a_i + b_i \sum_{l=1}^{3}(\sin(w_{PE}t) + cos(2w_{PE}t)) \tag{4.2}$$

where q_i is the i^{th} joint angle, a_i and b_i are constant scalars, and w_{PE} is proportional to the fundamental frequency. The resulting task space trajectory is a cardioid-like shape, as seen in Figure 4.6.

Simulation I

The first simulation compares the performance of the LWPR controller to the fixed parameter computed torque controller at varying trajectory velocities. The figure-8 trajectory

in Equation (4.1) is tracked with parameters $w_8 = l_8 = 0.2$, $[x_i\ y_i\ z_i] = [-0.3, 0.1, 0]$. To vary the trajectory velocities, values of $K_c = 1.57$ and $K_c = 2.09$ were used, corresponding to frequencies of 0.25 and 0.3Hz. To evaluate orientation control, a sinusoidal signal was input as the desired orientation of the end-effector. The LWPR controller was trained for 60s on the 0.25Hz trajectory, after which training was stopped and tracking performance was evaluated. The system was then allowed to train for another 120s. Next the trajectory frequency was increased to 0.3Hz, and the system was trained for an additional 30s, after which tracking performance was evaluated. These results are then compared against the CT controller. The durations of training were determined by observing the mean squared error (MSE) of the predicted torques from the LWPR controller. Training was stopped when asymptotic convergence of the MSE was observed. Since no parameter perturbation was introduced yet, only the CT and LWPR controllers are compared in the first simulation.

Simulation II

The second simulation evaluates tracking of the figure-8 pattern with varied end-effector loads, thereby introducing model parameter errors. The trained system from the first simulation (0.25Hz and 180s training) was used to track the figure-8 trajectory, but with additional end-effector masses of 0.5 and 1kg. The same conditions were repeated on the ACT and CT controller. After 30s of training time for the LWPR controller and ACT controller, the performance of all three controllers was evaluated.

Simulation III

The third simulation adds varying amounts of error to the inertia parameters of the model while observing the resulting performance of the three controllers when tracking the figure-8 trajectory. Training of these models was done in the same manner as above.

Simulation IV

The fourth simulation introduces friction in addition to inertia parameter uncertainty while tracking the persistently exciting (PE) trajectory in Equation (4.2) This trajectory is used for two reasons. Firstly, by using a trajectory with significant frequency content, the effect of persistence of excitation on the tracking and performance of ACT can be evaluated. Secondly, by shifting the operating range of the manipulator away from that of the figure 8

trajectory, the generalization performance of LWPR is tested. Furthermore, both Coulomb and viscous friction are introduced into the simulation. In order to assess the ACT controller's ability to cope with unmodeled dynamics, two cases are tested: one in which both Coulomb and viscous friction (2.26) are modeled, and one in which only Coulomb friction is modeled in the controller, but both types of friction are modeled in the plant. The friction constants were obtained from the defaults for the Puma 560 in the RTB.

4.2 Parameter Tuning and Initialization

Adaptive Computed Torque

The stability of the ACT controller was found to be highly sensitive to the adaptive gain parameter, γ (3.2). While a higher value of γ generally results in faster adaptation time, it increases the system's sensitivity to noise and numerical errors from integration of the time derivative of the estimated parameters (3.2). An adaptive gain of 0.01 was found to be the best tradeoff.

LWPR

Although LWPR incorporates many algorithms which enable the system to automatically adjust its parameters for optimal performance, initial values of these parameters can significantly impact the convergence rate. The initial value for the distance parameter $\mathbf{D}$ (3.6) dictates how large a receptive field is upon initialization. Too small a value of $\mathbf{D}$ (corresponding to large receptive fields) tends to delay convergence while a larger value of $\mathbf{D}$ results in overfitting of the data [55]. This parameter was generally tuned through a trial-and-error process which involved monitoring the MSE of the predicted values during the training phase. The initial performance of the LWPR controller is also highly dependent upon the data sets that are used to train the LWPR model. Because LWPR is a local learning approach, it must be trained in the region(s) of input space that the manipulator will be operating in. In order to train the model, a low-gain PD controller was used to track the desired trajectory while the LWPR model obtained training data. The initial value of the distance parameter $\mathbf{D}$ was set to 0.05 for each input dimension.

4.3 Results

Simulation I

The LWPR model was trained on the figure-8 trajectory at 0.25Hz, enabling it to predict the necessary torques for tracking at frequencies near 0.25Hz. As seen in Figure 4.1 and Table 5.1, after an additional training period of 40s, the LWPR controller compensated for the 0.3Hz trajectory, allowing it to perform nearly as well as the ideal CT controller. This illustrates the ability of the LWPR controller to rapidly adjust to changes in its operating conditions. However, frequencies greater than 0.3Hz were sufficient to push the system far enough from the trained region of input space, eventually resulting in LWPR predicting of zero for all the joint torques, causing the system to rely entirely on the feedback component of the control law. This result highlights the sensitivity of the LWPR controller to the initial training set. A potential solution is a more complete initial training of the LWPR model. If the initial training set were to include a larger subset of the input space obtained through motor babbling [39] for example, it is expected that the LWPR controller would be able to handle larger perturbations.

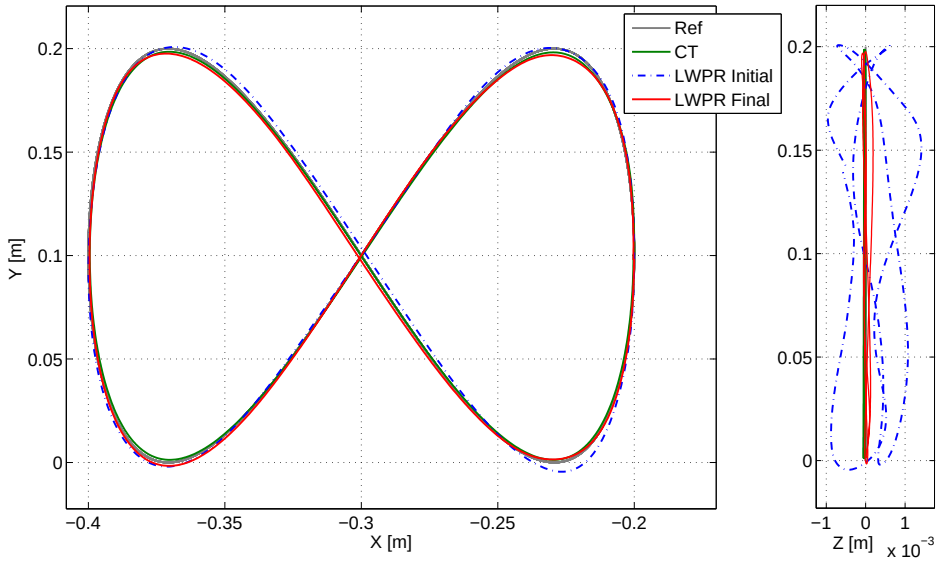

Figure 4.1: Position Tracking at 0.25Hz

Frequency	0.25Hz		0.3Hz	
	[mm]	[deg]	[mm]	[deg]
PD	4.07	0.89	5.54	2.89
CT	1.80	0.15	2.03	0.22
LWPR Initial	3.56	1.67	4.68	2.54
LWPR, 60s	2.71	1.31	/	/
LWPR, 180s	1.95	0.25	2.20	0.42

Table 4.1: Frequency - RMS position error [mm], orientation error [deg]

Simulation II

The second simulation evaluated the ability of the controllers to handle unmodeled end-effector loads. As seen in Table 5.2 and Figures 4.2 and 4.3, due to the unknown mass of the end-effector, imperfect linearization and decoupling cause a decrease in tracking performance of the CT controller. Both the LWPR and ACT controller are able to outperform the CT controller after 30s of additional training. Although the ACT controller has a-priori knowledge of the structure of the dynamic equation of the manipulator, it does not perform any better than the LWPR controller in position tracking after the same length of adaptation time. This is due to the slow convergence of estimated masses to their actual values, which can in turn be explained by the relatively low adaptive gain and the lack of a PE trajectory. However, as seen in the orientation results in Table 5.2, when tracking sinusoidal angular velocities on each joint of the wrist, the ACT controller yields better orientation tracking as compared to the LWPR controller, illustrating the importance of PE trajectories in the performance of the ACT controller. The LWPR controller was able to learn the inverse dynamics for both the 0.5kg and 1kg payload. However, masses greater than 1kg were sufficient to push the system to operate in a region outside its training, causing the controller to rely entirely on the feedback component of the control law. In this case, the performance of the LWPR controller was unable to converge due to the fact that the system was now operating so far from the desired trajectory that the receptive fields corresponding to the desired trajectory were never updated with new data.

Simulation III

Simulation three introduces parameter estimate errors into all the link masses, centre of mass locations and the moments of inertia of each link. Table 4.3 and Figures 4.4 and 4.5 illustrate that inaccurate knowledge of the dynamic parameters causes significant

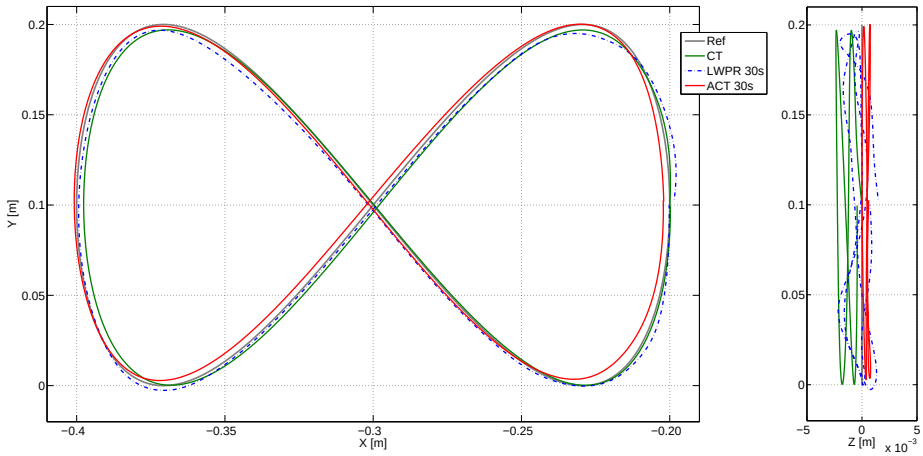

Figure 4.2: Position Tracking with 0.5kg Load

Payload	**+0.5 kg**		**+1kg**	
	[mm]	[deg]	[mm]	[deg]
PD	7.34	0.92	19.07	3.39
CT	4.95	0.60	16.45	2.15
ACT, 30s	4.44	0.44	2.70	1.26
LWPR, +30s	4.16	0.65	7.01	1.29

Table 4.2: Payload - RMS position error [mm], orientation error [deg]

degradation in performance for the CT controller, due to the imperfect linearization of the system dynamics. As seen in Table 4.3, the performance of the ACT controller was particularly poor in comparison to the LWPR controller, and even the CT controller. This can be explained by the fact that the perturbation of the inertia parameters was applied to all the joints of the manipulator, unlike the case of end-effector loading where only the parameters of one link were perturbed. Hence, it is expected that the adaptive controller would require both a persistently exciting trajectory, which excites all the dynamic modes of the structure, [14] and a longer adaptation time than 30s to yield better tracking results in this scenario. The importance of PE trajectories will be illustrated in the next simulation. For the LWPR controller, similar findings to that of simulations one and two were observed in that inertia parameter perturbations greater than 10% were sufficient to prevent LWPR from predicting accurate joint torques, causing the system to rely entirely on the feedback portion of the control law.

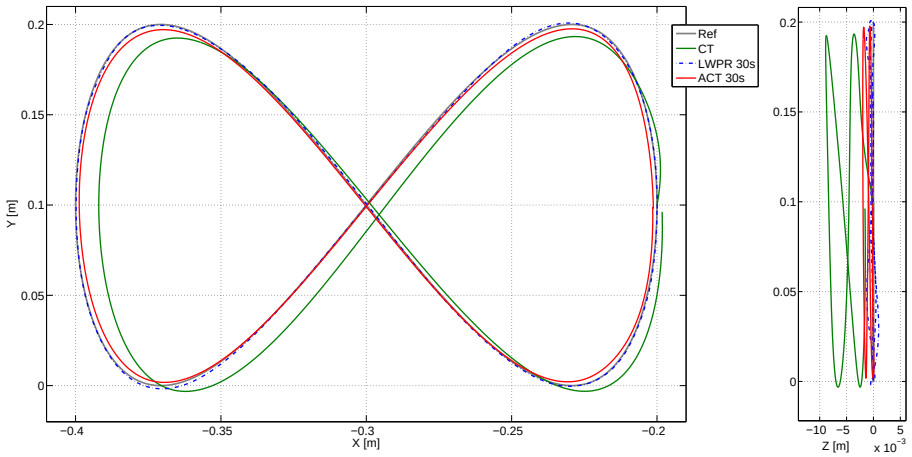

Figure 4.3: Position Tracking with 1kg Load

Parameter Error	**+1%**		**+5%**	
	[mm]	[deg]	[mm]	[deg]
PD	4.56	1.32	4.56	1.32
CT	2.15	0.25	2.65	0.30
ACT, 40s	2.20	0.12	2.75	0.15
LWPR, +40s	1.55	0.75	1.66	0.86

Table 4.3: Parameter Error - RMS position error [mm], orientation error [deg]

Simulation IV

Simulation four introduces the PE trajectory and Coulomb and viscous friction in addition to inertia parameter error. As seen in Table 4.3, the CT controller performed the poorest due to the error in its model parameters. When the ACT is implemented with the full friction model, the resulting tracking performance is significantly better than the CT controller, as also shown in Table 4.3.

However, when only partial knowledge of the model is available (in this case only viscous friction), the performance gain is no longer observed. The ACT controller outperforms the LWPR controller only if the friction model is fully known. This illustrates the importance of: 1) a persistently exciting trajectory and 2) accurate knowledge of structure of the dynamic model when using adaptive control.

The PE trajectory was also chosen to be significantly different from the figure 8 in order

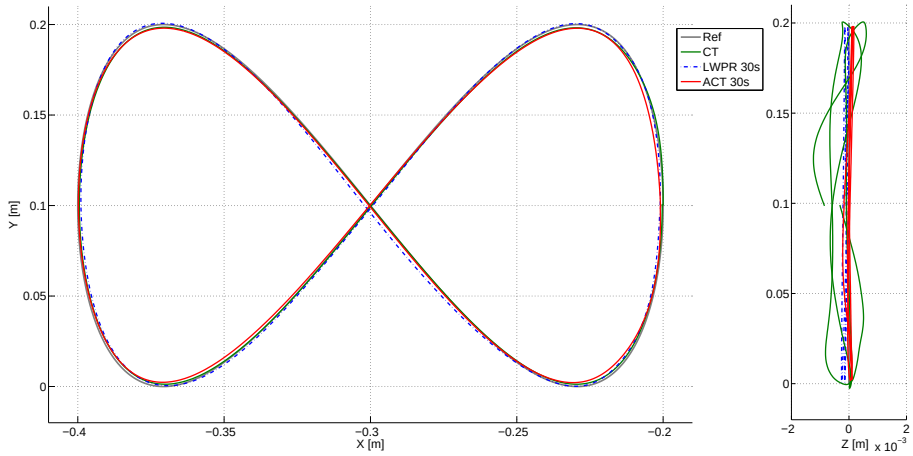

Figure 4.4: Position Tracking with 1% Parameter Error

to test the generalization of the LWPR model. The first test involved using the model that was learned in simulation three to attempt to track the PE trajectory. This model had seen roughly 10,000 training points, all of which were localized to the figure 8 trajectory. As expected, this LWPR model was unable to predict torques in the operating range of the PE trajectory. Hence, the LWPR model had to be re-trained on the PE trajectory, taking roughly 240 seconds to achieve good performance in the presence of friction.
This simulation was then repeated with a model that was initialized through the use of motor babbling [39]. Here, a joint space trajectory was made by randomly selecting a point in the robot's expected operating range about which small sinusoidal trajectories were executed by the joints. This sequence was repeated at different points until sufficient coverage of the operating range was seen. Roughly 10,000 training points were generated from motor babbling. A PD controller was then used to track this trajectory and the resulting data was used to train the LWPR model. As seen in Table 4.3, by initializing the LWPR model this way, good performance on the PE trajectory could be learned in half the time compared to the case without motor babbling.

Even without any a-priori knowledge of the structure or parameters of the dynamics, LWPR is able to learn the inverse dynamics function of a manipulator accurately enough to yield near-optimal control results within minutes of training. Unlike the adaptive controller which relies on persistence of excitation for tracking performance, the LWPR approach can be trained on an arbitrary trajectory provided that it is given sufficient time to learn.

When tracking a PE trajectory with a known dynamic model, the ACT clearly outperforms the LWPR controller in terms of tracking accuracy and adaptation time, which is

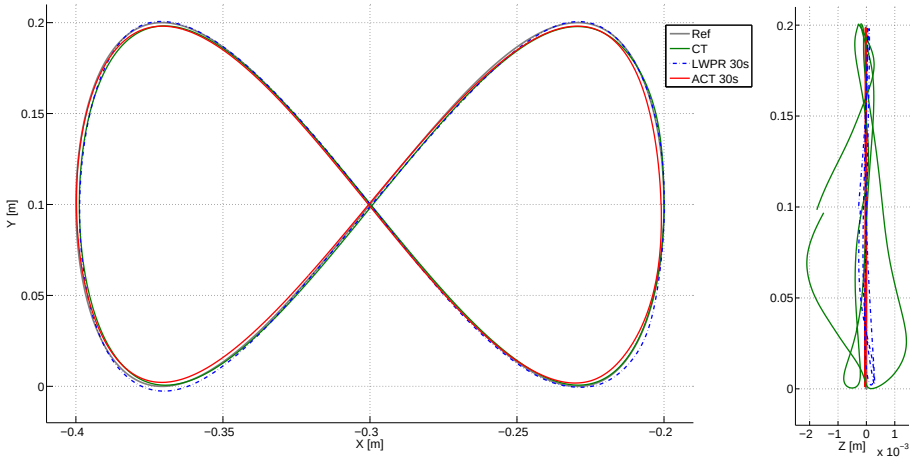

Figure 4.5: Position Tracking with 5% Parameter Error

Parameter Error	**+1%**		**+5%**	
	[mm]	[deg]	[mm]	[deg]
PD	4.12	0.95	4.12	0.95
CT	2.45	0.55	3.10	0.80
ACT, partial friction, 40s	2.0	0.40	2.1	0.45
ACT, full friction, 40s	1.65	0.30	1.70	0.32
LWPR, 240s	1.75	0.45	1.80	0.50
LWPR, motor babbling, 120s	1.82	0.50	1.85	0.55

Table 4.4: Parameter Error and Friction - RMS position error [mm], orientation error [deg]

expected due to its incorporation of a-priori knowledge. However, if LWPR is presented with sufficient time to learn, its performance will closely approach that of ACT, but will not surpass it due to the use of local linear approximations of the system dynamics. However, the ACT controller is at a disadvantage since not all trajectories meet the PE requirement. For this reason, the identification of system parameters is often done offline on a predetermined trajectory which is optimized to yield the best parameter estimates [24]. While this may yield results better than the LWPR controller, the benefits of online, incremental learning are lost, where LWPR excels.

The performance of LWPR outside of areas in which it has trained is poor. This was clearly illustrated when a large perturbation to the inertia parameters caused the manipulator to move outside its trained region, rendering the system completely reliant

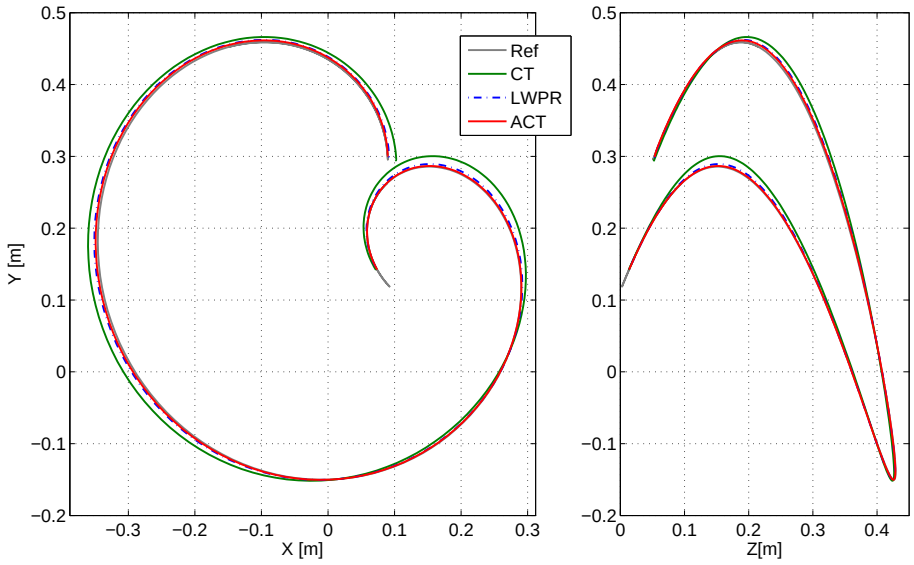

Figure 4.6: PE Trajectory - 5% inertia and friction error

upon the feedback portion of the control law. This problem can be overcome through proper initialization of the model through motor babbling. However, for applications in which the manipulator workspace is large, significant amounts of training covering the robot's entire workspace must be carried out in order to achieve good results. This is not an issue for the adaptive controller, as it incorporates a-priori knowledge of the system dynamics, which is applicable to the entire workspace of the robot.

4.4 Summary

These simulations indicate that the performance of the CT control scheme is highly dependent upon accurate knowledge of the dynamic parameters of the system. A slight perturbation from the actual inertia and friction parameters of the system caused a decrease in performance, while the LWPR and ACT performance remained more consistent under the same conditions.

The ACT approach was able to outperform the CT controller in most scenarios due to its ability to generate online estimates of the actual inertia parameters of the system. It was shown that when the trajectory is persistently exciting and the structure of the dynamics is

well known, the ACT controller also outperforms the LWPR. However, in practice, not all trajectories are PE and the structure of the friction model may not be known well. Hence, it is expected that when dealing with a physical robot, the simplified models of friction used in this paper will further degrade the performance of the ACT controller while the LWPR will be able to learn the additional nonlinearities present in the physical robot.

Lastly, although the LWPR controller was able to handle parametric uncertainty without any a-priori knowledge of the system, its greatest limitation is its local learning, which dictates that successful performance requires adequate initial training of the system. Significant perturbations caused the system to operate outside of its trained region, resulting in the system relying solely on the feedback component of the control law for tracking performance. Initializing the model through motor babbling can partially mitigate this issue, but the larger the workspace of the robot, the larger the training data set must be.

Chapter 5

Incorporating Prior Knowledge

In the previous chapter, the ability of learning controllers to cope with varying levels and sources of parametric uncertainty was illustrated. However, this performance is contingent upon the availability of large amounts of relevant training data for the algorithm, especially for the case of local learning techniques such as LWPR. This chapter [1] presents two types of online, incremental learning algorithms which incorporate prior knowledge to improve the generalization performance of the machine learning algorithms. An approach is developed for incorporating prior knowledge for both local (LWPR) and global (GPR) learning. Prior knowledge is incorporated into the LWPR framework by initializing the local linear models with a first order approximation of the available prior information. Prior knowledge is incorporated into the mean function of Sparse Pseudo Input Gaussian Processes (SPGP) and Sparse Online Gaussian Process (SOGP) regression, and a modified version of the algorithms is proposed to allow for online, incremental updates. It is shown that the proposed approaches allow the system to operate well even without any initial training data, and further improve performance with additional online training.

5.1 Prior Knowledge

The RBD equation (2.23) provides a globally valid model representing the dynamics of a robot manipulator. Hence, providing the RBD equation as prior knowledge to the learning

[1] A version of this chapter has been published. J. Sun de la Cruz, D. Kulić and W. Owen, Online Incremental Learning of Inverse Dynamics Incorporating Prior Knowledge, International Conference of Autonomous and Intelligent Systems, pp. 167-176, 2011.

controllers would greatly improve their generalization performance. However, due to the complexity of robotic systems, obtaining a complete and accurate RBD model can be difficult and tedious. Therefore, even if partial knowledge of the RBD model is available, this information should be incorporated into the learning algorithms to potentially boost performance. The simplest term of the RBD model in (2.23) is the gravity loading vector, $G(q)$ (2.22), as it depends upon the least number of inertial parameters of the system compared to the inertia and centripetal/Coriolis terms of the model [45]. In this chapter, techniques for incorporating partial prior knowledge to reduce training requirements for learning controllers are developed.

5.2 LWPR

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, as demonstrated in Chapter 4. In order to improve the generalization performance of LWPR, an algorithm for incorporating a-priori knowledge from the RBD model (2.23) into the LWPR algorithm is proposed. This is done by initializing the receptive fields in the LWPR model with a first order approximation of the available RBD model:

$$\boldsymbol{\beta} \leftarrow \frac{\partial \boldsymbol{\tau}}{\partial \mathbf{x}}\Big|_{x=\mathbf{x}^*} \tag{5.1}$$

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 5.1 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 $\mathbf{x}^*$:

$$\beta_{ik} = \frac{\tau_i(\mathbf{x}^*_{k-tol}) - \tau_i(\mathbf{x}^*)}{tol} \tag{5.2}$$

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 4 for initializing the LWPR model is added to the standard prediction procedure

of LWPR:

Algorithm 4 LWPR with Prior Knowledge

```
1: if there is no existing RF centered near x* then
2:         Compute (β) according to (5.1)
3:         Initialize a new RF centered at x* with hyperparameters β
4: else
5:         Compute prediction with standard LWPR procedure (3.2.1)
6: 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 3.2.1, i.e., if there is no existing RF that produces a weight (3.6) greater than w_{gen} given $\mathbf{x}^*$, the initialization procedure is applied.

By evaluating the partial derivative of the RBD equation (2.23) 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 as outlined in Chapter 3.

5.3 GPR Approaches

With Gaussian process regression techniques, including the SPGP and SOGP algorithms described in Sections 3.2.4 and 3.2.5, typically a zero mean function for the Gaussian process is assumed, and hence predictions in areas of the workspace where few training points have been observed are inaccurate. A simple method of incorporating a-priori knowledge is to set the mean function in Equation (3.10) equal to the available model of the system, thus biasing the system towards the specified a-priori knowledge [32].

While the mean function is biased towards the prior knowledge, the covariance function k (3.10) still requires adaptation to the data that is observed. Specifically, the hyperparameters of the covariance function, as described in Section 3.2.3 must be optimized to the incoming training data. Nonlinear conjugate gradient descent (NCG) [46] is used to minimize the negative log marginal likelihood with respect to the hyperparameters, $\boldsymbol{\Theta}$. As an iterative gradient-based optimization technique, NCG requires the specification of an initial guess of $\boldsymbol{\Theta}$, which is determined by performing the optimization on a batch of training data collected from the system during motor babbling in Chapter 4.

With standard gradient descent optimization, the location, $\boldsymbol{\chi}$, of the extremum of a function, f, is found by iteratively taking steps in the direction of the gradient of the function at the current point. That is:

$$\boldsymbol{\chi}_{i+1} = \boldsymbol{\chi}_i + a_i \mathbf{d}_i \tag{5.3}$$

where $\boldsymbol{\chi}_i$ is the current solution (i^{th} iteration) to the optimization problem, a_i represents the length of the step to be taken, and $\mathbf{d}_i$ is the step direction. This iterative procedure often results in steps being taken in the same direction as in earlier steps resulting in a slower convergence. By employing NCG, this repetition is avoided by ensuring that all search directions are conjugate, or A-orthogonal, and that the necessary step lengths are taken such that only one step must be taken in each search direction to arrive at the extremum for that direction. Thus, the convergence of NCG is better than or at least the same as standard gradient descent. A comparison between the convergence of NCG and standard gradient descent is illustrated in Figure 5.1, where a 2-dimensional example is given, where the green lines represent gradient descent convergence and the red lines represent NCG.

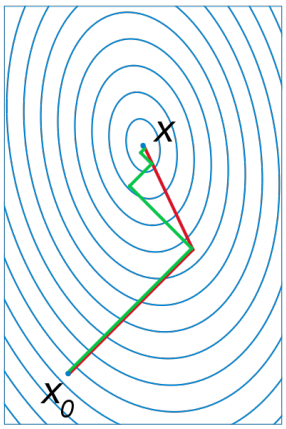

Figure 5.1: Convergence of Gradient Descent vs NCG

Two vectors, $\mathbf{d}_i$ and $\mathbf{d}_j$, are conjugate, or A-orthogonal, if they have the following property:

$$\mathbf{d}_i^T \mathbf{A} \mathbf{d}_j = 0 \tag{5.4}$$

where $\mathbf{A}$ is a positive semi-definite matrix. In order to find the set of conjugate search

directions, $\mathbf{d}$, the Polak-Ribiére formula is used to generate coefficients $\boldsymbol{\xi}$:

$$\boldsymbol{\xi}_{i+1} = \frac{\mathbf{r}_{i+1}^T(\mathbf{r}_{i+1} - \mathbf{r}_i)}{\mathbf{r}_i^T\mathbf{r}_i} \tag{5.5}$$

where i represents the current iteration, and the residual $\mathbf{r}$ is set to the negation of the gradient at the current location $\boldsymbol{\chi}_i$, i.e. $\mathbf{r} = -f'(\boldsymbol{\chi}_i)$. Thus, the conjugate search directions, $\mathbf{d}$, can be determined:

$$\mathbf{d}_{i+1} = \mathbf{r}_{i+1} + \boldsymbol{\xi}_{i+1}\mathbf{d}_i \tag{5.6}$$

With the selection of the step direction, $\mathbf{d}_i$, the problem is reduced to a one-dimensional optimization case which involves finding the best step length, a, to take in direction $\mathbf{d}_i$. This is done by through the use of a line search procedure which finds the a_i that minimizes $f(\boldsymbol{\chi}_i + a_i\mathbf{d}_i)$. The interpolation method of line searching achieves this by firstly approximating the function $f(\boldsymbol{\chi} + a\mathbf{d})$ as a polynomial through a Taylor series expansion. Here we drop the iteration index for simplicity:

$$\begin{aligned} f(\boldsymbol{\chi} + a\mathbf{d}) \approx & f(\boldsymbol{\chi}) + a[\frac{d}{da}f(\boldsymbol{\chi} + a\mathbf{d})]_{a=0} + \frac{a^2}{2}[\frac{d^2}{da^2}f(\boldsymbol{\chi} + a\mathbf{d})]_{a=0} \\ = & f(\boldsymbol{\chi}) + a[f'(\boldsymbol{\chi})]^T\mathbf{d} + \frac{a^2}{2}\mathbf{d}^T f''(\boldsymbol{\chi})\mathbf{d} \end{aligned} \tag{5.7}$$

The secant method is then used to approximate the second derivative, f'', as a finite difference equation of the first derivative f' evaluated at two points, $a = 0$ and $a = \sigma$:

$$\frac{d^2}{da^2} \approx \frac{1}{\sigma}[f'(\boldsymbol{\chi} + \sigma\mathbf{d})]^T\mathbf{d} - [f'(\boldsymbol{\chi})]^T\mathbf{d} \tag{5.8}$$

where σ is a non-zero number such that the closer σ is to 0, the better the approximation of the second derivative. Substituting this expression into (5.7) and differentiating:

$$\frac{d}{da}f(\boldsymbol{\chi} + a\mathbf{d}) \approx [f'(\boldsymbol{\chi})]^T\mathbf{d} + \frac{a}{\sigma}\left\{[f'(\boldsymbol{\chi} + \sigma\mathbf{d})]^T\mathbf{d} - [f'(\boldsymbol{\chi})]^T\mathbf{d}\right\} \tag{5.9}$$

Minimization of $f(\boldsymbol{\chi} + a\mathbf{d})$ can then be achieved by equating its derivative to zero and rearranging for a:

$$a = -\sigma\frac{\boldsymbol{\zeta}}{\boldsymbol{\zeta}_{prev} - \boldsymbol{\zeta}} \tag{5.10}$$

where $\boldsymbol{\zeta}_{prev} = [f'(\boldsymbol{\chi} + \sigma\mathbf{d})]^T\mathbf{d}$ and $\boldsymbol{\zeta} = [f'(\boldsymbol{\chi})]^T\mathbf{d}$. This process is then repeated, with the current $\boldsymbol{\chi}$ equated to the value of $\boldsymbol{\chi} + a\mathbf{d}$ calculated in the previous iteration. The stop

condition for the line search is based on the gradient approaching orthogonality with the current search direction, i.e. $f'^{T}\mathbf{d} \approx 0$. High tolerances for this term have been found to be too computationally inefficient for the relatively small gain in accuracy [46] that is achieved.

Algorithm 5 Nonlinear Conjugate Gradient Optimization

given: initial guess $\boldsymbol{\chi}$, maximum allowable CG and line search iterations, $\epsilon_{CG}, \epsilon_{LS}$
initialize: $\mathbf{r} \leftarrow -f'(\boldsymbol{\chi})$, $d \leftarrow \mathbf{r}$, $i \leftarrow 0$
while $i < i_{max}$ and $\|\mathbf{r}_i\| \leq \epsilon_{CG}\|\mathbf{r}_0\|$ **do**
    initialize: $j \leftarrow 0$
    $\boldsymbol{\zeta}_{prev} \leftarrow [f'(\boldsymbol{\chi} + \sigma\mathbf{d})]^T\mathbf{d}$
    do
        $\boldsymbol{\zeta} \leftarrow [f'(\boldsymbol{\chi})]^T\mathbf{d}$
        $a \leftarrow -\sigma\frac{\boldsymbol{\zeta}}{\boldsymbol{\zeta}_{prev}-\boldsymbol{\zeta}}$
        $\boldsymbol{\chi} \leftarrow \boldsymbol{\chi} + a\mathbf{d}_i$
        $\boldsymbol{\zeta}_{prev} \leftarrow \boldsymbol{\zeta}$
        $j \leftarrow j + 1$
    while $j < j_{max}$ and $\|a\mathbf{d}\| \leq \epsilon_{LS}$
    $\mathbf{r}_{prev} \leftarrow \mathbf{r}$
    $\mathbf{r} \leftarrow -f'(\boldsymbol{\chi})$
    $\boldsymbol{\xi} \leftarrow \frac{\mathbf{r}^T(\mathbf{r}-\mathbf{r}_{prev})}{\mathbf{r}_{prev}^T\mathbf{r}_{prev}}$
    $\mathbf{d} \leftarrow \mathbf{r} + \boldsymbol{\xi}\mathbf{d}$
    $i \leftarrow i + 1$
end while

In order to test the effects of limiting the maximum number of line search iterations on the accuracy of the SPGP algorithm, the system was trained on data sets obtained from the simulations in Chapter 4 and the MSE of prediction was monitored. In the first case, illustrated at the top of Figure 5.2, up to 50 line search iterations were allowed (i.e. $j_{max} = 50$), and below in the second graph, 25 iterations were allowed. Despite this difference, the final MSE of both cases is nearly identical. The only noticeable discrepancy is the initially slower rate of convergence of the joints in the case of early line search termination. Joint 1 also exhibits non-monotonic convergence initially. However, due to the lower number of allowable line search iterations, the computational burden is reduced and thus the frequency at which the overall hyperparameter updates are performed can be increased. Hence, for the simulations, less accurate, but more frequent line searches were used by terminating the search after 25 iterations.

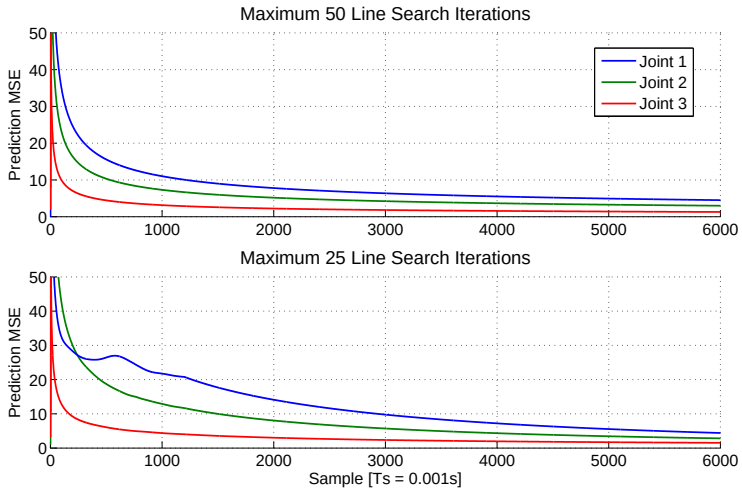

Figure 5.2: Prediction MSE

Despite its improved convergence over standard gradient descent, each call to NCG optimization requires a significant number of time steps of the control loop to compute. Thus, the code is broken down into a sequence of smaller, 'atomic' code segments, so that the computation of NCG occurs over multiple controller time steps. This was done so that each code segment would have at most one call to evaluate the function derivative f'. Based on Algorithm 5, the code segments are separated into lines $1:2$, $3:5$, $6:12$ and $13:18$, with each code segment being executed over a single time step of the controller. The optimized location of the hyperparameters, represented by $\boldsymbol{\chi}$, is not used in the control loop until the entire algorithm has been executed.

5.3.1 Simulation Setup

The proposed approaches are evaluated in simulation on a 6-DOF Puma 560 robot using the Robotics Toolbox (RTB) [13]. The open-source LWPR [55], SPGP [50] and SOGP [16] code were modified to incorporate a-priori knowledge and incremental updating as described above. In order to simulate the nonlinearities present in a physical robot, the effects of Coloumb and viscous friction were simulated according to Equation 2.26, with the friction constants obtained from the defaults for the Puma 560 in the RTB. Furthermore, to simulate the effects of imprecise knowledge of the inertial parameters of the robot, a 10% percent error in the inertial parameters of the a-priori knowledge was introduced. For LWPR, the tunable parameters were set to the same values as determined in Chapter 4.

For SPGP and SOGP, the maximum size of the sparse subset of datapoints, M, was set to 35. This number was determined by trial-and-error to be the smallest possible value before a noticeable degradation in prediction performance was introduced.

A-priori knowledge from the full RBD model in (2.23), or partial knowledge from the gravity loading vector (2.22) are used to initialize each algorithm. Standard computed torque control (2.41) is also implemented for comparison.

Tracking performance of the controllers is evaluated on a 'star-like' asterisk pattern [31]. The asterisk trajectory is a more challenging trajectory than the figure-8 due to the high components of velocity and acceleration, and thus requires model-based control for good tracking accuracy. Illustrated in Figure 5.3, this trajectory is formed in the horizontal XY plane, by first moving in a straight line outwards from a centre point, then retracing back inwards to the centre, repeating this pattern in eight different directions in a sequential manner. The inverse kinematics algorithm presented in Chapter 2 is used to convert this trajectory from task to joint space.

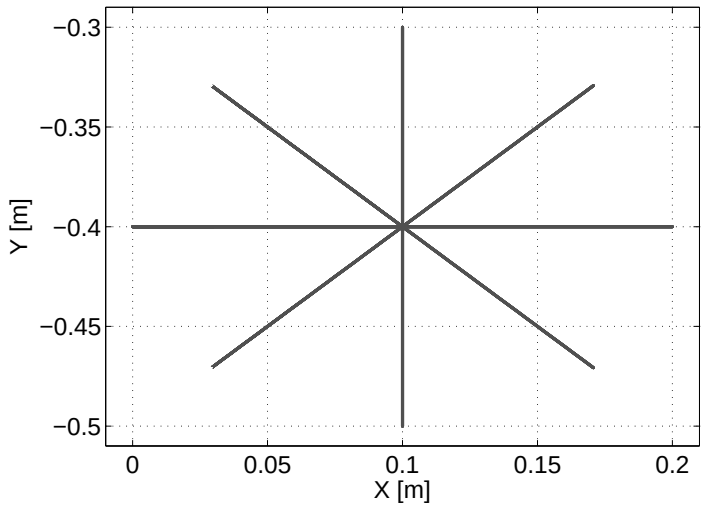

Figure 5.3: 'Asterisk' trajectory

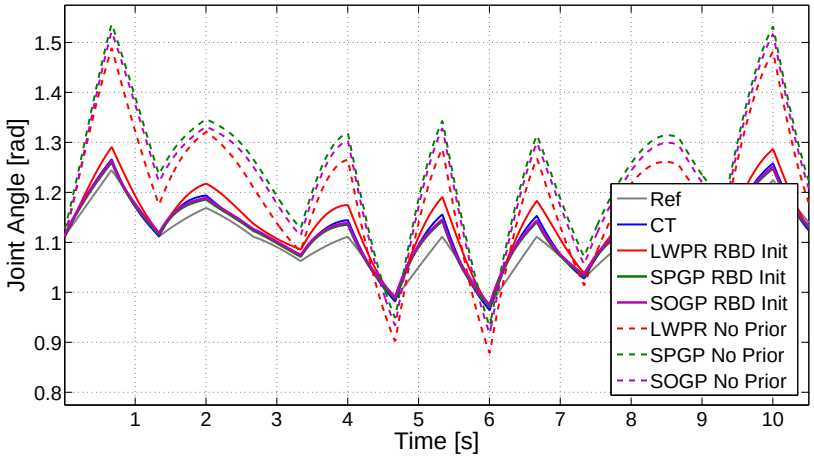

Figure 5.4: Initial Tracking Performance for Joint 2 with full RBD Prior and No Prior

5.3.2 Results

Full Knowledge of RBD

Figure 5.4 depicts the joint space tracking performance of the computed torque (CT) controller using a model with 10% error in the inertial and friction parameters of the system. As seen from the figure, the parameter error causes poor tracking results. The same RBD model is used to initialize both GP and LWPR models, and the resulting controllers are trained online while tracking the asterisk trajectory. Figure 5.4 and Table 5.1 show the joint space tracking performance after one cycle of the trajectory. The performance of the GP controllers is very similar to that of the computed torque controller, as they are initialized with the same RBD equation. On the other hand, as the LWPR model is initialized with a set of linear approximations of RBD equation, the initial performance of LWPR is not as good as the GP or the CT methods.

Figure 5.5 and Table 5.1 show the results after 90 seconds of additional training. Due to the high computational efficiency of LWPR, incremental updates are made at a rate of 400 Hz, while the more computationally taxing GP algorithms limit updates to a rate of 100 Hz. Hence, as time progresses, LWPR is able to accumulate more training data than the GP controllers, eventually performing nearly as well as SPGP and SOGP. Initially, SPGP performs better than SOGP due to the fact that SPGP uses the full $M = 35$ sparse inputs from the start, whereas SOGP is initialized with $M = 1$ and adds sparse inputs as data is incrementally processed until the limit $M = 35$ is reached. Thus, after seeing

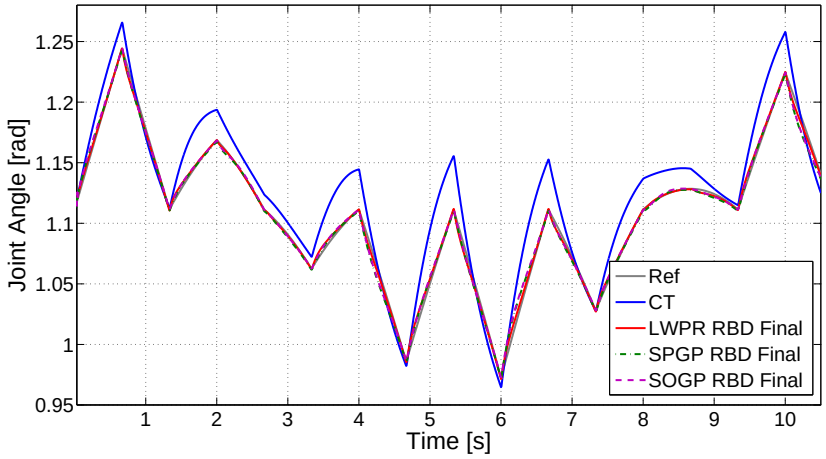

Figure 5.5: Final Tracking Performance for Joints 2 with full RBD Prior

significant amounts of training data, both SOGP and SPGP perform very similarly. Lastly, given sufficient data, all three learning controllers are able to outperform the CT method by learning the nonlinear behaviour of Coloumb and viscous friction and compensating for the initial inaccurate knowledge of the RBD equation.

Joint	1	2	3	4	5	6	Avg
LWPR, Initial	1.25	2.26	1.74	0.65	0.75	0.80	1.24
LWPR, 90s	0.75	0.94	0.82	0.25	0.38	0.45	0.60
SPGP, Initial	1.10	1.75	1.55	0.60	0.68	0.75	1.07
SPGP, 90s	0.70	0.85	0.78	0.22	0.32	0.45	0.55
SOGP, Initial	1.21	1.88	1.60	0.63	0.70	0.82	1.14
SOGP, 90s	0.72	0.84	0.80	0.22	0.32	0.47	0.56

Table 5.1: RMS tracking error with full knowledge (deg)

Partial Knowledge of RBD

Figure 5.6 and Table 5.2 illustrate the joint space tracking performance of the SPGP, SOGP and LWPR models when initialized with only the gravity loading vector (2.22) of the RBD equation. In this case, LWPR outperforms SPGP during the first cycle. This can be attributed to the higher update rate of LWPR, allowing the model to adapt more

quickly to the data it receives. After multiple cycles through the trajectory, this advantage of LWPR vanishes, as the GP models have then observed enough training data across the entire asterisk trajectory to learn a more accurate model. Figure 5.7 and Table 5.2 illustrate the performance after 150 seconds of training. Similarly to the case of full knowledge of the RBD, the learning controllers have all compensated for friction and clearly outperform the CT controller. However, since the system was initialized with only partial knowledge of the RBD equation, it has taken longer for both models to achieve the same tracking performance in the case of full RBD knowledge. Similarly to the case of full knowledge of RBD, SPGP initially performs better than SOGP, but after training this discrepancy is minimized.

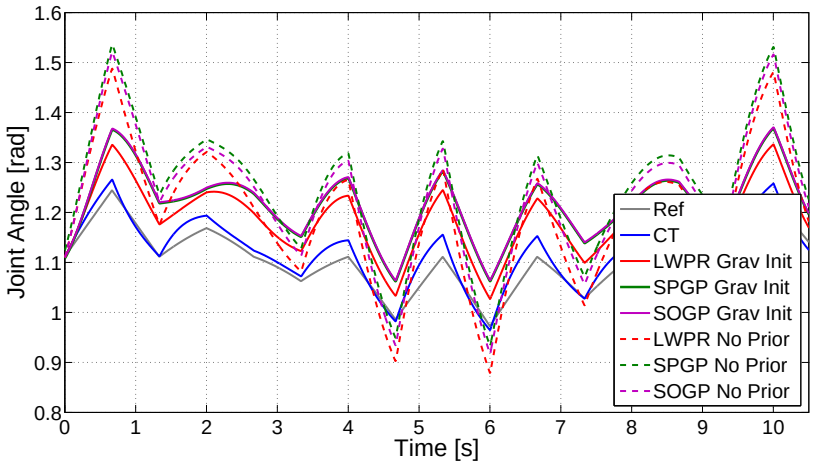

Figure 5.6: Initial Tracking Performance for Joint 2 with Gravity Prior and No Prior

Joint	1	2	3	4	5	6	Avg
LWPR, Initial	2.57	4.78	3.62	0.95	0.80	0.75	2.25
LWPR, 150s	0.80	0.95	0.80	0.22	0.41	0.44	0.60
SPGP, Initial	0.48	6.66	5.29	0.05	0.12	0.27	2.15
SPGP, 150s	0.75	0.84	0.75	0.25	0.30	0.40	0.55
SOGP, Initial	0.50	6.75	5.43	0.10	0.14	0.30	2.20
SOGP, 150s	0.77	0.83	0.76	0.25	0.32	0.39	0.55

Table 5.2: RMS tracking error with partial knowledge (deg)

Without the use of a-priori knowledge, learning algorithms were typically initialized with large training data sets obtained through motor babbling [55], [39], [33] in order to

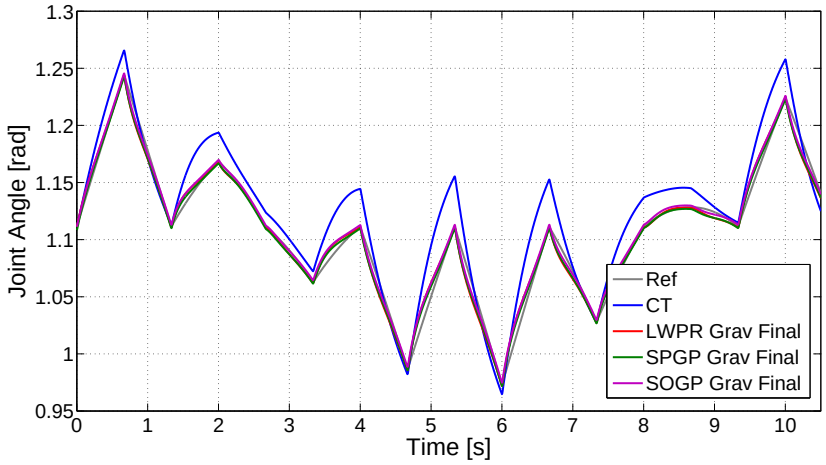

Figure 5.7: Final Tracking Performance for Joints 2 with Gravity Prior

achieve decent tracking performance after initialization. By incorporating a-priori knowledge of the RBD equation, whether partial or full, it is shown in these results that the proposed systems are able to perform reasonably well from the start, even without undergoing such an initialization procedure.

Algorithm Computation Time

In order to evaluate the computational efficiency of LWPR, SPGP and SOGP, each algorithm was trained on data sets of varying size (2,4,6,8,10,12 and 14 thousand training points) obtained from the simulation work in this chapter. Figure 5.8 illustrates the computation time required to compute a single torque prediction given an input $\mathbf{x}^*$ as a function of the number of training data points that the algorithm has processed. The results were obtained on a PC running Windows XP with a CPU clock speed of 2.66 GHz and 4 GB of RAM. For SOGP and SPGP, the maximum size of the sparse representation set was limited to 35.

As seen in Figure 5.8, LWPR has the lowest computational cost which remains relatively constant with an increasing data set size. This is due to the fact that LWPR does not explicitly store the entire training set, but rather incorporates them into local linear models. For the sparse GP techniques, SOGP clearly outperforms SPGP. This can be attributed to the online procedure of SOGP which allows data points to be processed incrementally. Although SPGP also uses a sparse representation, data is still processed in batches, and

no provision for incremental updates is provided. Because of this, the computation time for a single prediction from SPGP increases much more rapidly than SOGP as the number of training points increases.

Although the simulation results presented in this chapter show that the SPGP controller marginally outperforms the SOGP controller in terms of tracking performance (see Tables 5.1 and 5.2), the performance gains from SPGP are not significant enough to offset the higher computational costs which increase significantly as the training data set increases. Although the simulation platform was able to handle the computational requirements for SPGP while maintaining near-real time performance, it should be noted that this was only for 150 seconds of operation (corresponding to 15,000 training points). For long-term incremental learning, it is expected that the computational requirements of SPGP will prohibit real-time control. Thus, for the following chapters, SOGP is chosen over SPGP.

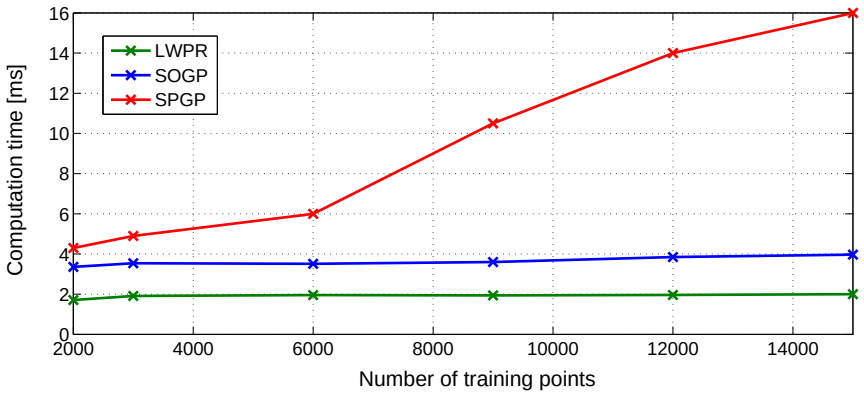

Figure 5.8: Computation time required for a single prediction

Chapter 6

Experimental Work

The results presented in Chapters 4 and 5 are all carried out in simulation using MATLAB/Simulink. In order to closely simulate the nonlinear dynamics present in a physical system, the effects of Coulomb and viscous friction were introduced in simulation. In this chapter, the results of experimental work on a physical robot, in addition to simulations, are reported to fully evaluate the performance of the proposed methods. Experimental work was conducted at the Collaborative Advanced Robotics and Intelligent Systems (CARIS) Laboratory at the University of British Columbia.

6.1 Experimental Platform

All experimental work was carried out on a CRS A460 robotic manipulator with a custom designed open-architecture controller [21] 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 open-architecture controller is illustrated in Figure 6.1 and consists of a custom motor amplifier unit and a PC with a processor speed of 2.4GHz and 2 GB of RAM running Windows XP. The Ardence RTX real-time kernel [1] 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 software [2] allows Simulink-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.

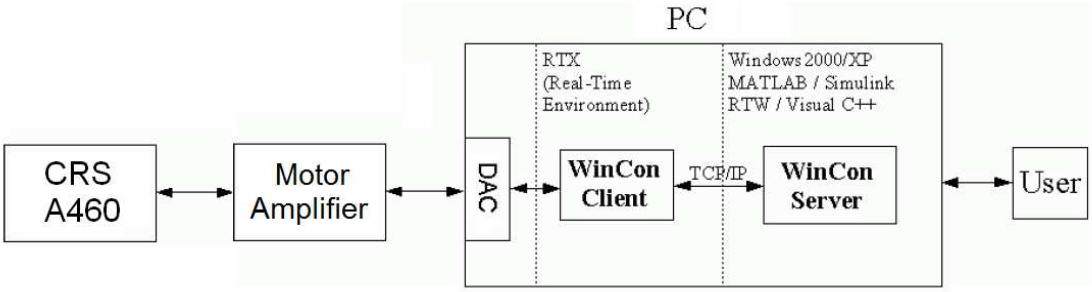

Figure 6.1: CRS Open-Control Architecture, *adapted from [2]*

Due to the requirements of the Wincon system, the algorithms written in C/C++ code (LWPR, SOGP, and RNE) were wrapped in C-coded S-functions and called as Simulink function blocks. The multi-rate block sampling function of the Simulink engine allowed the computationally-expensive functions to be executed at a slower sampling rate. 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. Due to the limited availability of the CRS arm at the CARIS lab, experimental work for the SOGP controller could not be completed, and simulation results (indicated by *SIM*) are presented for the SOGP controller instead. For these simulations, the same global time step of 1ms was used, the updates are processed at a frequency of 10 Hz, and hyperparameter updates are completed at a rate of 1 Hz. The number of basis vectors for SOGP was set to 35.

6.2 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 [40] which were determined offline through the procedure of dynamic parameter identification as described briefly in Chapter 3. 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 [40] 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 (2.23) and the gravity vector (2.22) 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.

For the SOGP simulations, the CRS A460 arm was modeled using the kinematic parameters supplied in the user's manual. The same dynamic parameters obtained above for the LWPR experiments are used for the SOGP controller. For the plant model used in the SOGP simulations, the available dynamic parameters of the CRS arm were perturbed until the performance of the experimentally implemented FF controller matched that of the simulated FF controller as closely as possible. This was done so that the results in simulation would be comparable to the experimental results obtained for the LWPR controller.

6.3 Experiments

The following test cases were carried out for the Feedforward (FF), LWPR, and SOGP controllers in order to evaluate their ability to deal with uncertainty in the dynamic model:

1. Tracking a figure-8 trajectory by initializing the system through Motor Babbling (MB), rigid body dynamics (RBD) and gravity loading information
2. Tracking a figure-8 trajectory at different frequencies and varied end-effector masses

The FF and LWPR controllers were tested experimentally on the CRS A460 arm while the tests for the SOGP controller were carried out in simulation as described above. As seen in Figure 6.2, a figure-8 trajectory in the horizontal XY plane, covering a length and width of approximately 0.2m x 0.4m, is used to test the trajectory-following capabilities of the controllers. 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 - corresponding to position-only control.

The task space equations for 'the figure 8' trajectory are as follows:

$$\begin{aligned} x_d &= 0.1\sin(\frac{2\pi}{T_c}t) - 0.47 \\ y_d &= 0.2\cos(\frac{4\pi}{T_c}t) \\ z_d &= 0.29 \end{aligned} \tag{6.1}$$

where $\mathbf{p_d} = [x_d\ y_d\ z_d]^T$ is the vector of the desired end-effector task space position, and T_c is the desired duration of one cycle of the figure-8 in seconds. Desired velocity and acceleration terms are generated through numerical differentiation as in Chapter 4.

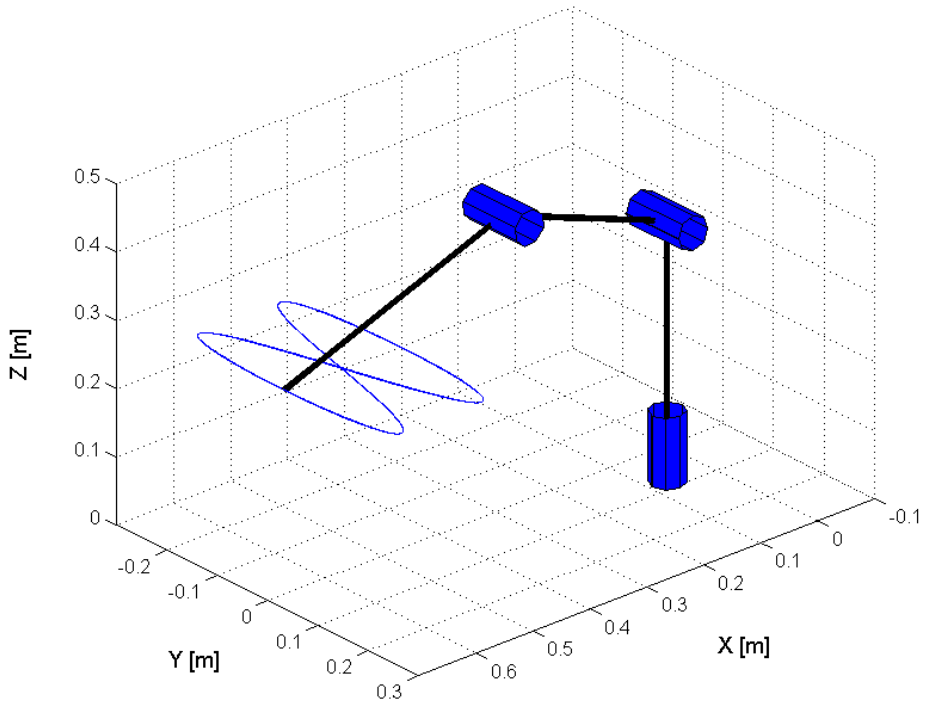

Figure 6.2: Figure-8 trajectory

The simulation work carried out in Chapter 4 assumed 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 and simulated SOGP controllers were used to learn both the dynamics of the arm (2.23) as well as the motor dynamics (2.27). The mapping to be learned by LWPR and SOGP is given by:

$$(\mathbf{q}, \dot{\mathbf{q}}, \ddot{\mathbf{q}}_\mathbf{d}) \mapsto \mathbf{v_m} \tag{6.2}$$

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

6.3.1 Feedforward vs Computed Torque

Both the feedforward (FF) and Computed Torque (CT) controllers presented in Chapter 2 were implemented experimentally to determine which 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, some

form of state estimation [37], [57] 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. The filter was designed using the Filter design toolbox in MATLAB and was implemented at 1KHz, which is the smallest allowable sampling time in Wincon. Both the FF and CT controller were used to track the figure-8 trajectory in Equation 6.1. As seen in Figure 6.3 and Table 6.1, 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 Butterworth filter at 1KHz, 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 [37], 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.

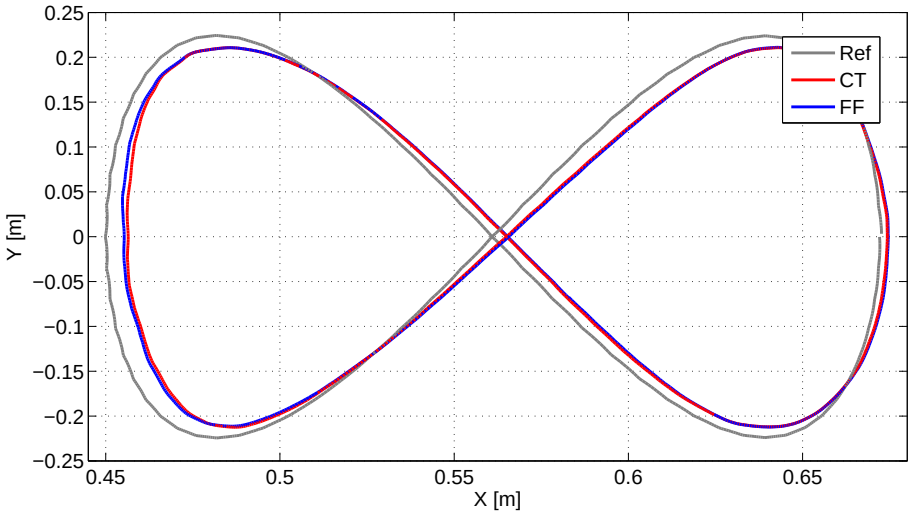

Figure 6.3: FF vs CT Tracking Performance

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

Table 6.1: FF vs CT RMS Joint Space Error (deg)

6.3.2 Initialization Technique

The effects of different initialization strategies on LWPR tracking performance are shown in Figure 6.6. For this experiment, LWPR was initialized either through motor babbling, or through full or partial knowledge of the RBD equation as outlined in Chapter 5. The same motor babbling strategy as in Chapter 4 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 2.23. However, after repeating the trajectory for 50 cycles, the performance of all three initialization cases converges.

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

Table 6.2: LWPR Initialization - Task Space Error

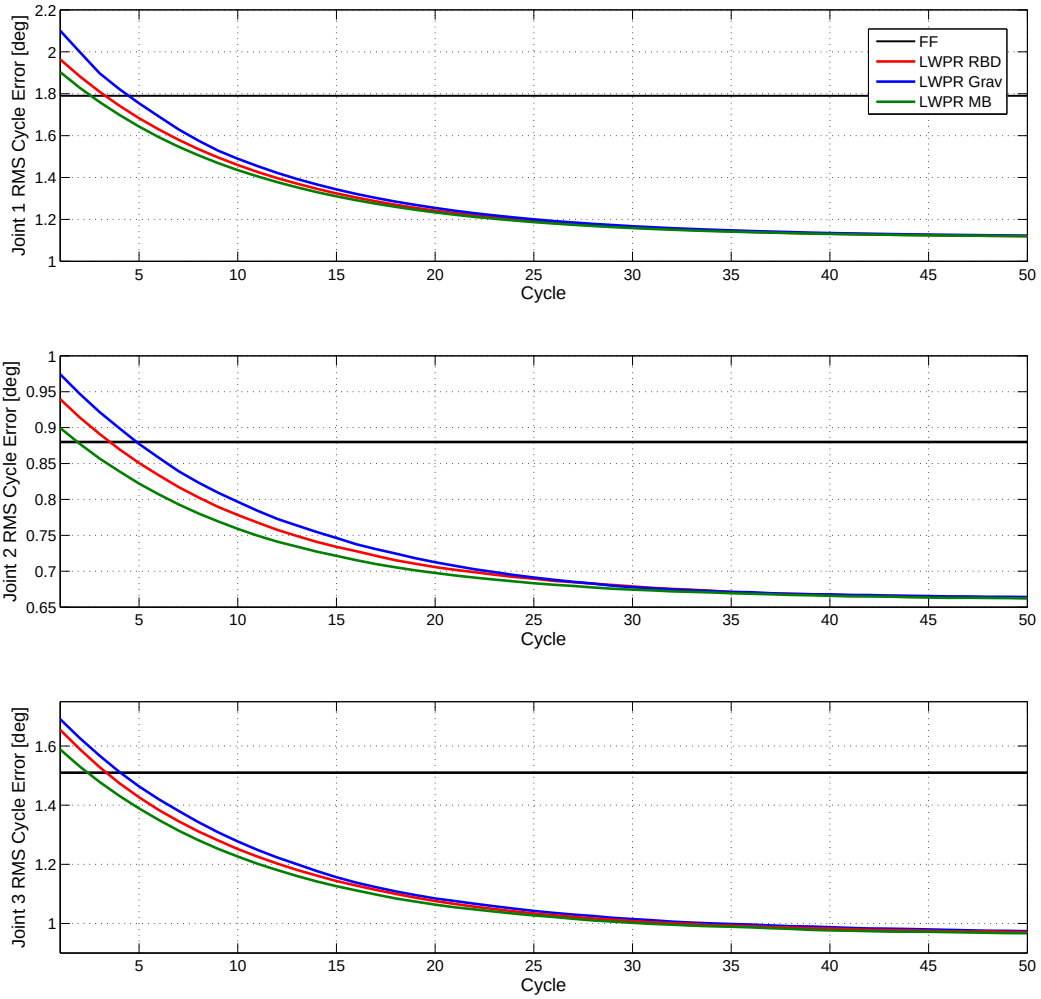

Figure 6.4: LWPR Initialization - Joint Space Per Cycle Average Error

RMS Error (mm)	x	y	z	Avg
RBD Initial	5.17	19.33	0.49	8.33
RBD Final	4.05	11.57	0.31	5.31
Grav Initial	8.54	21.34	1.52	10.47
Grav Final	4.01	12.01	0.32	5.45
MB Initial	4.51	15.50	0.39	6.80
MB Final	4.00	11.78	0.31	5.36

Table 6.3: (*SIM*) SOGP Initialization - Task Space Error

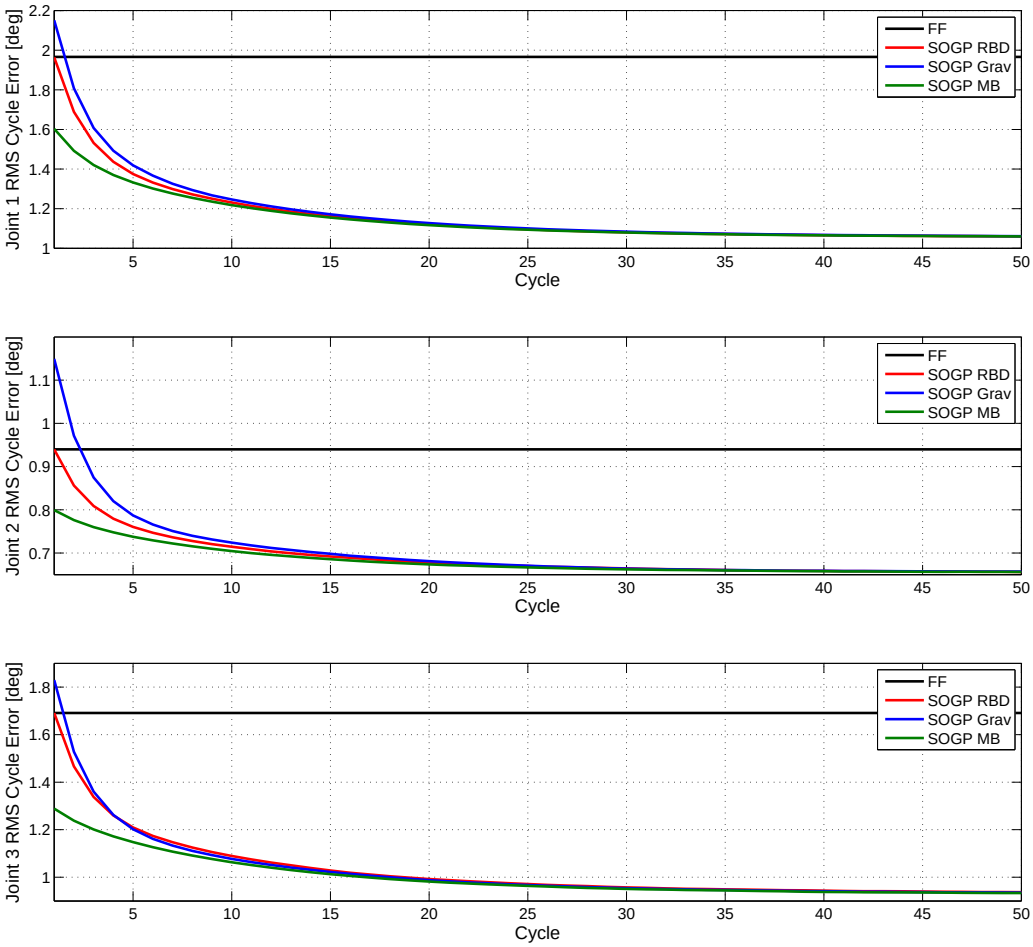

Figure 6.5: (*SIM*) SOGP Initialization - Joint Space Per Cycle Average Error

Although the same RBD model is used for the FF controller and for initializing LWPR, the FF controller initially outperforms LWPR with RBD initialization. As with the simulation work in Chapter 5, 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.

The SOGP controller was evaluated in simulation using the same initialization strategies as with LWPR. For the simulated SOGP controller with motor babbling initialization, it was found that only 5,000 points were necessary to initially outperform the RBD or gravity vector initialized cases. This can be seen in Table 6.3 and Figure 6.7 where the root-mean-squared (RMS) error of each cycle through the trajectory per joint is illustrated. The performance of the model-based FF controller is also included for comparison. The 5,000 points for SOGP were selected by uniformly sampling every 10^{th} point of the full motor babbling training set. With SOGP, because the mean function of the Gaussian Process is equated to the available RBD model, the initial performance of SOGP is equivalent to the FF controller. Through additional training, the performance of all three initialization strategies begins to converge and exceed that of the FF controller. For SOGP, the initial performance of the system trained through motor babbling is significantly better than the FF controller, as the motor babbling data is obtained from the actual system.

Tables 6.2 and 6.3 give 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 examining the Cartesian equation for the reference trajectory (6.1) and 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.

For both learning controllers, 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 controllers 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 through motor babbling, it could also outperform the FF controller from the start.

6.3.3 Comparison with PD Control

Figure 6.6 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.

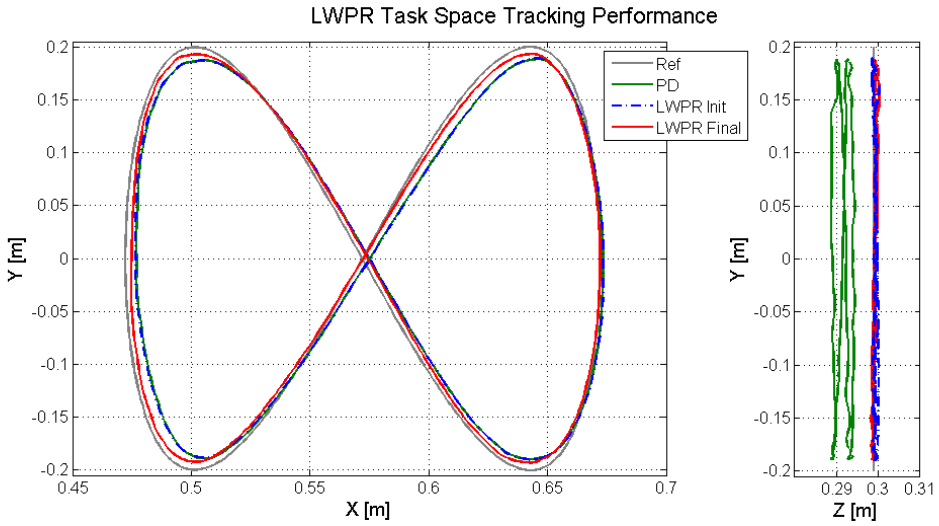

Figure 6.6: PD vs LWPR with RBD Init. - Task Space

The same test case was applied to SOGP and a PD controller in simulation. As seen in Figure (6.7), the performance of SOGP initialized with the RBD model results in an initial performance in the XY plane that is better than the PD controller. Similar to LWPR, the issue of inaccurately known parameters is compensated for by further training of the SOGP system for an additional 50 cycles. This allows the simulated SOGP system to obtain 4,000 training points, thus improving the tracking performance.

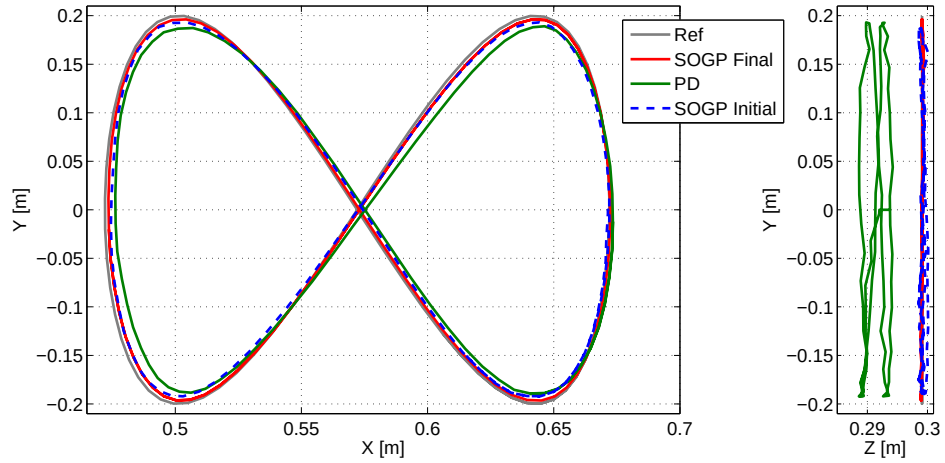

Figure 6.7: (*SIM*) PD vs SOGP with RBD Init. - Task Space

6.3.4 Unknown Payload

In order to simulate the case of unknown payloads on the end-effector, the learning controllers were initialized with the RBD (2.23) model corresponding to an unloaded end-effector. Two different masses - 1.1kg and 1.6kg, or equivalently 55% and 80% of the rated end-effector loading (2kg) were attached to the end-effector and the figure-8 trajectory was tracked as before using LWPR and the FF controller, also based on a model with an unloaded end-effector. The same scenario was simulated with the SOGP controller. Figure 6.8 and Table 6.4 show the effect of the unknown masses on the performance of the base joint 1 (Figure 6.2) for the LWPR controller. From these results, it is clear that the end-effector mass does not have a significant effect on the tracking performance for joint 1, as indicated by the relatively small change in performance of both the FF and learning controllers. This can be explained by the fact that the additional masses on the end-effector only change the dynamics of the inertia matrix $\mathbf{M}(\mathbf{q})$, whereas the gravity loading vector, $\mathbf{G}(\mathbf{q})$ and Coriolis/centripetal terms $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ corresponding to joint 1 are not significant due to the orientation of joint 1. Furthermore, the additional mass of 1.6kg is not significant when considering the entire mass of the arm which must be accelerated by joint 1. A similar trend is observed for the simulated SOGP controller, as shown in Figure 6.10 and Table 6.5.

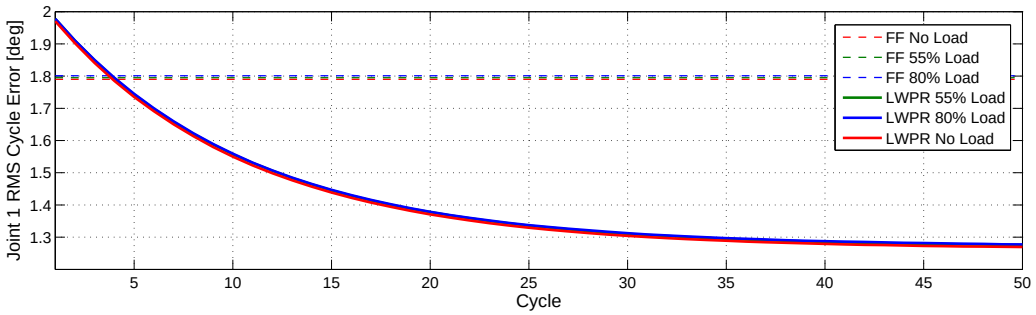

Figure 6.8: LWPR Payload - Joint 1 Per Cycle Average Error

RMS Error (mm)	x	y	z	Avg
No Load Initial	5.12	19.33	0.49	8.31
No Load Final	4.01	13.21	0.34	5.85
55% Initial	5.29	19.47	0.89	8.55
55% Final	4.08	13.74	0.44	6.08
80% Initial	5.62	20.05	0.99	8.89
80% Final	4.2	14.06	0.45	6.23

Table 6.4: LWPR End-Effector Loading - Task Space Error

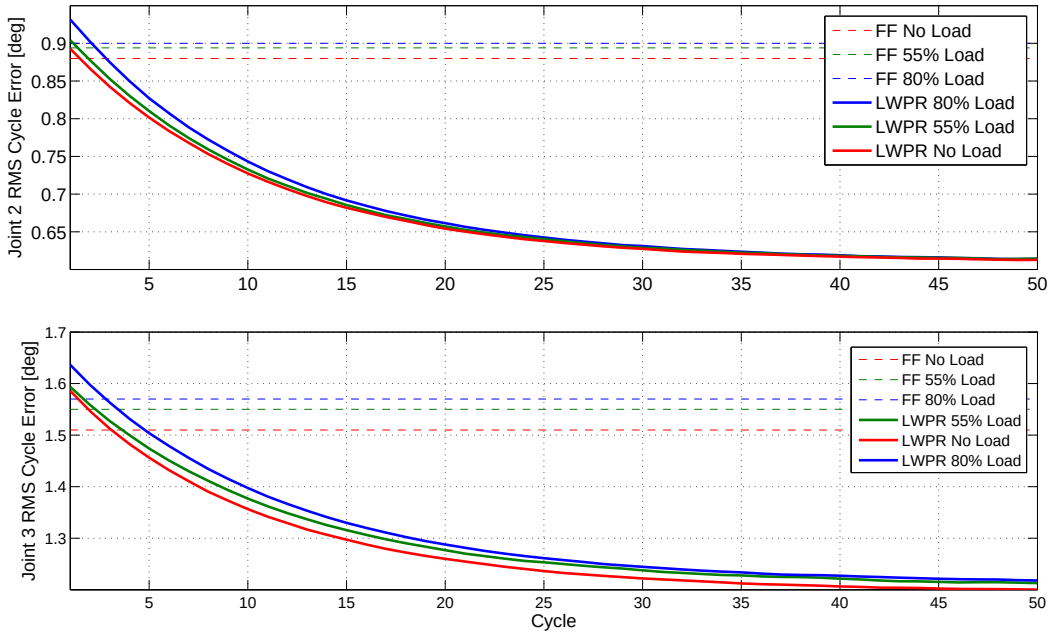

Figure 6.9: LWPR Payload - Joints 2,3 Per Cycle Average Error

RMS Error (mm)	x	y	z	Avg
No Load Initial	5.31	19.42	0.53	8.42
No Load Final	3.65	11.03	0.32	5.00
55% Initial	5.19	18.47	0.89	8.55
55% Final	3.81	11.14	0.34	5.09
80% Initial	5.95	21.05	1.02	9.34
80% Final	3.61	10.95	0.30	4.95

Table 6.5: (*SIM*) SOGP End-Effector Loading - Task Space Error

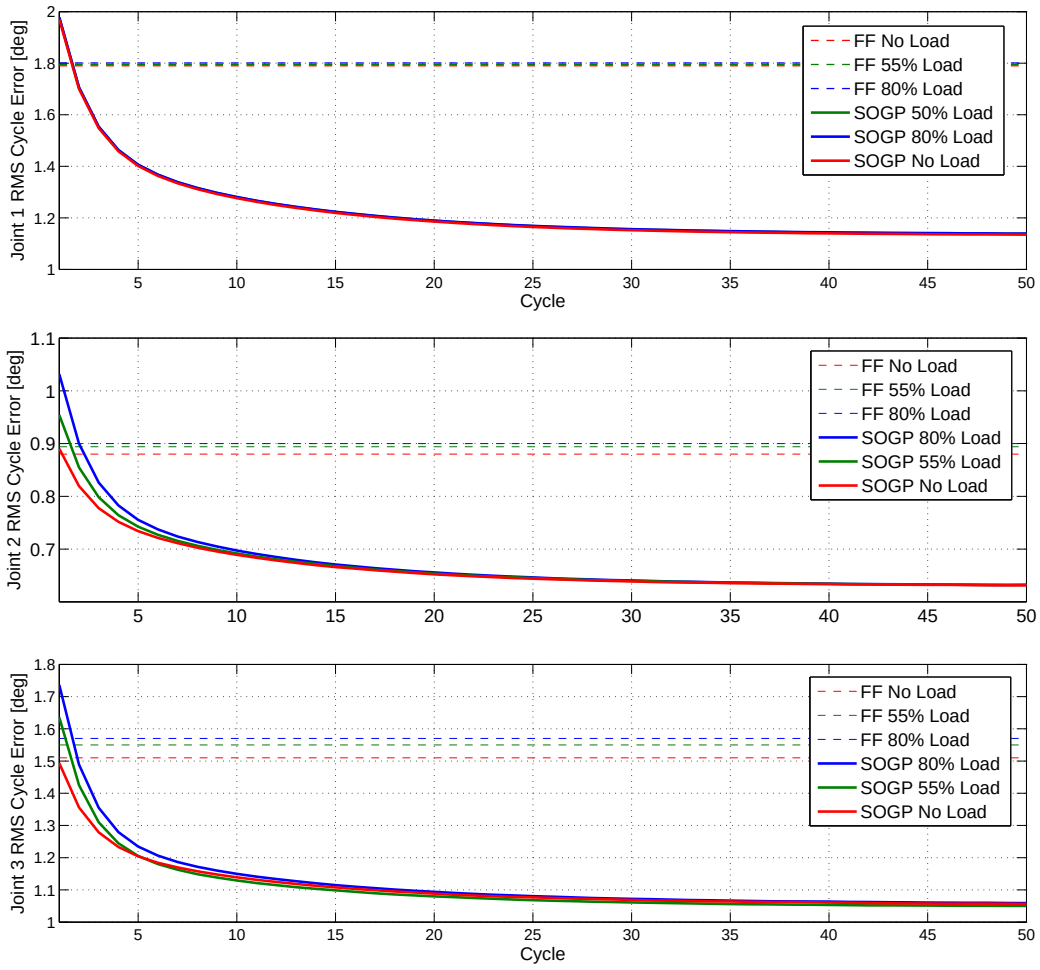

Figure 6.10: (*SIM*) SOGP Payload - Per Cycle Average Error

Contrary to joint 1, joint 2 and particularly joint 3 are more significantly impacted by the addition of end-effector masses, as seen in Figure 6.9 and Tables 6.4 and 6.5. For joint 2, the initial performance of the learning controllers is noticeably impacted by the additional masses at the end-effector. However, after 50 cycles of training, the error has converged close to that of the no-load scenario. In the case of joint 3, whose dynamic parameters are effectively altered through the additional mass, the tracking performance is most affected out of all three joints. Although LWPR quickly outperforms FF through additional training, the convergence of the tracking error when compared to the no-load case is much slower, and even after 50 cycles of training, there still exists a significant

discrepancy in tracking performance. However, the per cycle error of both the 55% and 80% loading cases appear to be decreasing steadily even at 50 cycles, whereas the no-load scenario seems to have converged to a steady-state. Thus, this discrepancy in performance can likely be solved through further training. Similar behaviour for joints 1,2 and 3 is observed in the simulation results for SOGP, as indicated in Figure 6.10 and Table 6.5.

In order to further test the ability of the learning controllers to cope with uncertainty due to changing payloads, the payload experiment was repeated for LWPR using the model learned from the 1.6kg load case, but with the mass removed from the end-effector. The same test was performed in simulation for the SOGP controller. As seen in Figure 6.11 the initial performance of the LWPR controller is no longer as good as FF control, as the model corresponding to an end-effector mass of 1.6kg was learned previously. However, in this case, the convergence rate of LWPR is much better than the first payload experiment. This is because LWPR automatically places a higher weight on the most recently observed data while updating its individual receptive fields. Furthermore, because learning with LWPR is highly localized to a specific region of state space, there is a decreased risk of negative inference when observing data that is contrary to what has already been learned. For the case of simulated SOGP, convergence of the tracking error takes just as long as the original case where the no-load model was used to learn the model for the 1.6kg load, as seen in Figure 6.12. This is due to the global learning over the input space, where a change in the system dynamics reflected in the observed data may conflict with the previously constructed model. Thus, SOGP must re-learn the dynamics by re-optimizing both the location of the basis vectors as well as the hyperparameters (3.11) of the covariance function.

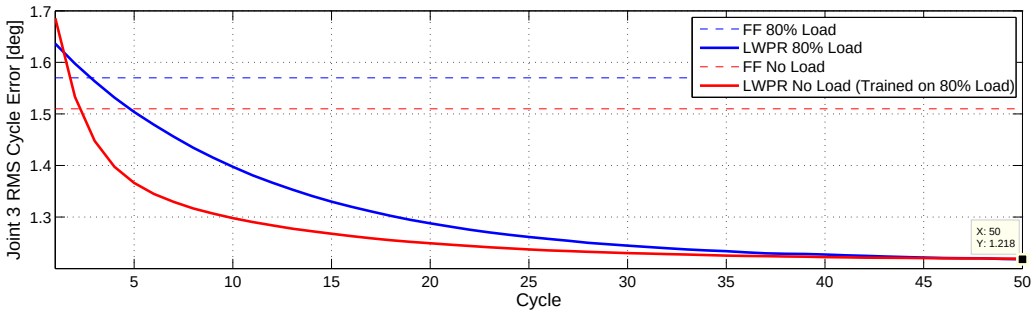

Figure 6.11: LWPR Payload Change - Joint 3 Per Cycle Average Error

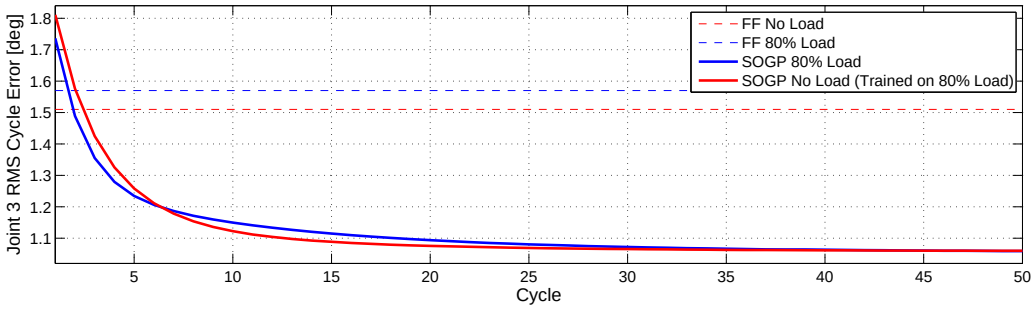

Figure 6.12: (*SIM*) SOGP Payload Change - Joint 3 Per Cycle Average Error

6.3.5 Trajectory Speed

The final experiment involves increasing the speed of the figure-8 trajectory by decreasing the period, T_c from 8 seconds to 5 seconds. The learned models for LWPR with RBD and gravity only initialization from the no-load experiment were used in tracking the fast figure-8. The same experiment was repeated in simulation for the case of the SOGP controller. The results are shown in Figure 6.13 and Table 6.6. The performance of LWPR with gravity initialization is consistently worse than that of the full RBD initialization, even though the system was previously trained and had achieved performance similar to the case of RBD initialization on the slower trajectory. This can be explained by the fact that during initialization, only the gravity loading vector was used, neglecting the Coriolis/centripetal $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ term. When the system was trained online on the slow trajectory, the effects of the Coriolis/centripetal term was learned, however it was localized about the specific $\mathbf{q}, \dot{\mathbf{q}}$ state associated with the slow trajectory. When the same system was asked to track the fast trajectory, $\dot{\mathbf{q}}$ was increased, and the dynamic behaviour learned for the compensation of the Coriolis/centripetal terms on a smaller $\dot{\mathbf{q}}$ was no longer covered by the same local model. Hence, LWPR with gravity initialization required more training to improve its performance again.

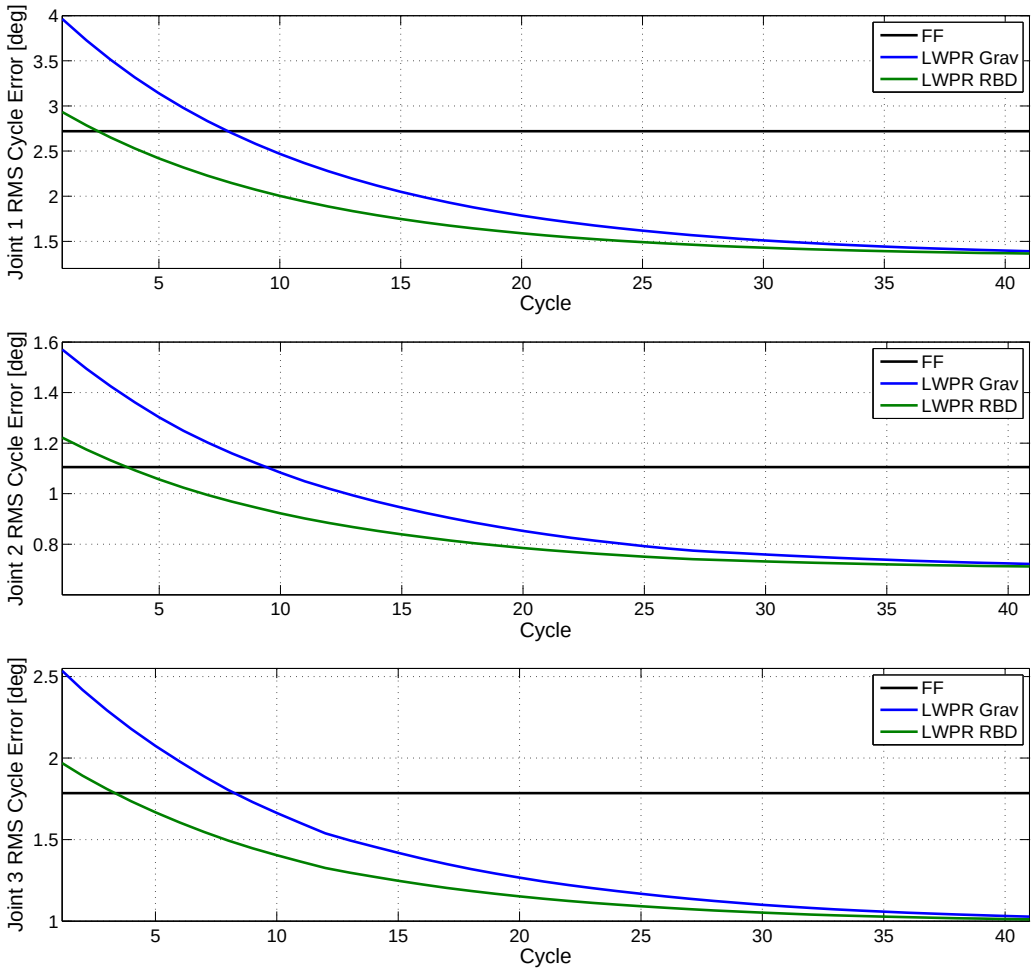

Figure 6.13: LWPR Fast Trajectory - Joint Space Per Cycle Average Error

For LWPR with RBD initialization, the dynamic effect of the Coriolis/centripetal term is already compensated for, but this is done with an inaccurate dynamic model. Similarly to the gravity initialized case, the RBD initialized LWPR controller had already compensated for inaccurate dynamic parameters, but this learning was localized around the slow trajectory, and hence the system had to retrain to compensate for inaccurate dynamic parameters. For the simulated SOGP controller, similar trends to that of the LWPR controller are observed, as indicated in Figure 6.15 and Table 6.15, and the same reasoning can be applied to explain these trends.

RMS Error (mm)	x	y	z	Avg
RBD Initial	6.72	20.05	0.65	9.14
RBD Final	4.12	18.45	0.34	7.63
Grav Initial	8.11	29.95	0.89	12.98
Grav Final	4.23	19.06	0.44	7.91

Table 6.6: LWPR Fast Trajectory - Task Space Error

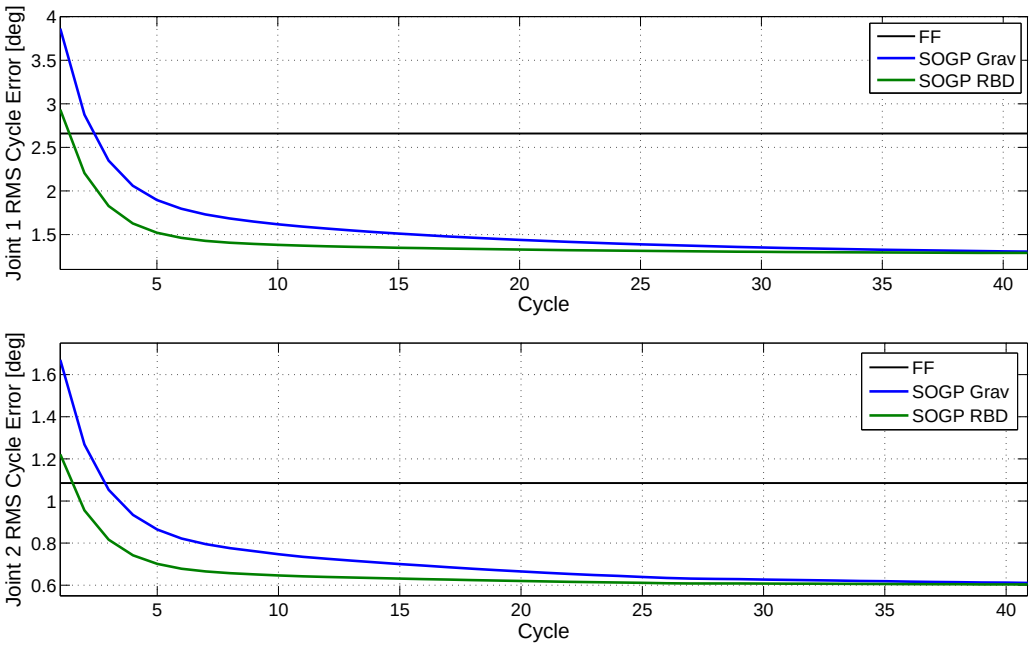

Figure 6.14: (*SIM*) SOGP Fast Trajectory - Joints 1,2 Per Cycle Average Error

Comparing the FF and final LWPR and SOGP performance on the fast trajectory in Tables 6.6 and 6.7 to the same controllers on the slower trajectory in Tables 6.2 and 6.3, it is clear that there is a significant degradation in tracking performance when the fast trajectory is tracked, even though the dynamics of the arm are being compensated for by the controllers. This discrepancy in error between the fast and the slow trajectories is primarily seen in the y-direction, which suggests that the desired velocity of the trajectory in the y-direction exceeds that of the physical velocity limits of the CRS arm. This reasoning is further strengthened by the fact that the discrepancy in the performance between fast and slow trajectories is not as large when considering the simulated SOGP case compared to

the experimental results of LWPR, as no limitations on the performance of the arm were imposed in simulation.

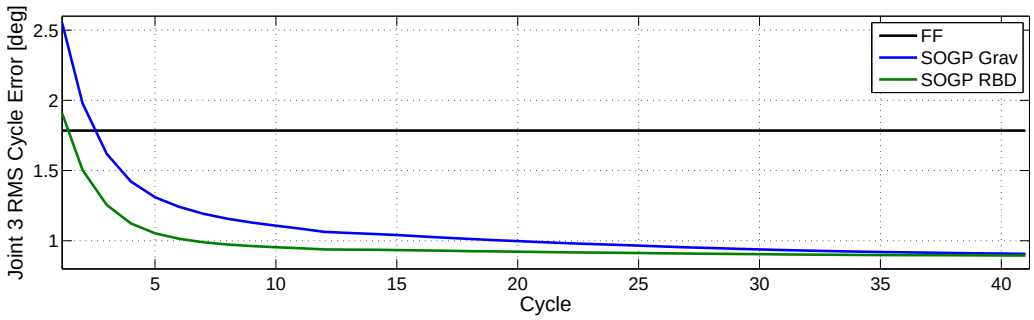

Figure 6.15: (*SIM*) SOGP Fast Trajectory - Joint 3 Per Cycle Average Error

RMS Error (mm)	x	y	z	Avg
RBD Initial	6.61	19.25	0.52	8.79
RBD Final	4.12	12.94	0.40	5.82
Grav Initial	7.83	28.17	0.82	12.27
Grav Final	4.20	13.06	0.41	5.89

Table 6.7: (*SIM*) SOGP Fast Trajectory - Task Space Error

Chapter 7

Conclusions and Future Work

This thesis analyzed the performance of model-based controllers for multi-DOF manipulators, and presented strategies for control and initialization when the model parameters and structure were not fully known. A comparison between a fixed model-based control strategy, an adaptive controller and the LWPR learning controller was presented to characterize the performance of the various controllers in the presence of modeling uncertainty and error. Simulations were carried out in order to evaluate the position and orientation tracking performance of each controller under varied end effector loading, trajectory velocities and inaccuracies in the known dynamic parameters. Both the adaptive controller and LWPR controller were shown to have comparable performance in the presence of parametric uncertainty. However, the LWPR controller was unable to generalize well outside of the regions in which it had been trained. To address the issue of poor generalization performance, two new methods which learn the inverse dynamics function in an online, incremental manner while incorporating a-priori knowledge of the system were presented. First, the method of LWPR was initialized with first-order approximations of the available rigid body dynamics (RBD) equation. Second, a-priori knowledge was included in the GPR framework and its variants by biasing the mean function towards the RBD equations. These approaches were firstly validated in simulation using MATLAB/Simulink. Experimental validation for a fixed model-based controller and the LWPR algorithm was then conducted on the CRS A460 robot arm at the University of British Columbia (UBC), supplemented by simulation work for the SOGP controller. Details of the findings of this thesis are presented below.

7.1 Summary of Findings

The performance of the standard model-based feedforward (FF) control scheme was found to be dependent upon accurate knowledge of the dynamic parameters of the system. A slight perturbation from the actual inertia and friction parameters of the system caused a decrease in performance, while the learning controllers and the simulated adaptive controller were able to compensate for the parameter errors, thus yielding better tracking performance. It was shown that when the trajectory is persistently exciting (PE) and the structure of the dynamics is well known, the ACT controller can outperform the LWPR learning controller. However, in practice, not all trajectories are PE and the structure of the dynamics may not be known well. Hence, it is expected that when dealing with a physical robot, the simplified models of friction used in this paper will further degrade the performance of the ACT controller while the LWPR will be able to learn the additional nonlinearities present in the physical robot.

Although the LWPR controller was able to handle parametric uncertainty without any a-priori knowledge of the system, its greatest limitation is its local learning, which dictates that successful performance requires adequate initial training of the system. It was found that significant perturbations to the system from unmodeled dynamics or changes in the trajectory speed were able to push the system to operate far enough outside of its trained region. Initializing the model through motor babbling was shown to partially mitigate this issue, but the larger the workspace of the robot, the larger the training data set must be.

Two incremental, online learning methods were proposed for approximating the inverse dynamics equation while incorporating full or partial knowledge of the RBD equation. First, prior knowledge was incorporated into the LWPR framework by initializing the local models with a first-order approximation of the RBD equation. Second, prior knowledge was incorporated into the SOGP framework by setting the mean function equal to the RBD equation. The incorporation of prior knowledge into these two algorithms improved their generalization performance, as the models are able to outperform independent joint PD control without having seen any relevant training data beforehand.

Online learning with SOGP was achieved by spreading out the optimization phase of the algorithm over several timesteps, allowing the system to train itself at a rate of 10 Hz. Although the greater computational efficiency of LWPR allows updates to occur at a rate of 100 Hz, after a short period of training, the performance of LWPR and SOGP was nearly identical in simulation, even though LWPR had accumulated more training data. This is due to the global nature of GPR techniques, which have been found to yield better accuracy compared to the local learning of LWPR under the same conditions [33]. Both

approaches are able to compensate for the nonlinear effects of friction, as well as the initial inaccuracy in the known inertial parameters.

Experimental validation of the FF and LWPR controllers was carried out on the CRS A460 robot arm, and due to limited access to the equipment, further simulation work was presented for the case of SOGP. Both LWPR and SOGP were allowed to learn the motor dynamics in addition to the RBD dynamics of the arm. For the LWPR experiments, the initial performance with motor babbling was consistently better than initialization with the RBD equation or the gravity loading vector, due to the inaccurate knowledge of the model parameters. However, given sufficient training time, the performance of all three initialization methods converged to the same value, which was significantly better than the FF controller, illustrating the advantage of learning control in dealing with parameter uncertainty. LWPR was also shown to be capable of handling parametric uncertainty arising from unknown end-effector loads and an increased trajectory velocity. The same test cases were carried out in simulation for the SOGP controller, and similar trends were found. For both cases, incorporation of a-priori knowledge allowed the system to perform well without having seen any relevant training data beforehand. The simulation results suggest that the convergence of tracking error with SOGP is faster than LWPR due to the global learning nature of GPR as opposed to the highly localized learning of LWPR. However, this must be verified experimentally.

7.2 Future Work

The directions for future work are primarily focused in two areas. First, further validation of the proposed algorithms presented in this thesis is required through more extensive simulation and experimental work. Second, investigation of other supervised learning algorithms is recommended.

7.2.1 Simulations

More extensive test cases are required to draw stronger conclusions about the ability of the learning controllers to deal with various trajectories and model uncertainty. For example, to test the effect of parameter uncertaintly, random samples from the space of parameters in a predefined 'error set' could be taken and the resulting mean tracking error and variance could be reported. To generalize the trajectory-following capabilities, random samples of trajectories could be taken and the results could be compared through a Monte Carlo

simulation. In addition to testing the controllers under various scenarios, the long-term behaviour of the learning algorithms must also be verified by allowing the system to train for longer periods of time.

7.2.2 Experiments

More experimental work is required to fully evaluate the proposed learning control methods. Specifically, the SOGP algorithm must be tested experimentally so that a direct comparison to LWPR can be made. The simulation work for SOGP in Chapters 5 and 6 suggests that SOGP will yield tracking results that are very similar to LWPR, but the rate of convergence of the error will be much faster for SOGP. In addition to this, it would be beneficial to implement the ACT controller presented in Chapter 4 experimentally, as the performance of this controller was found in simulation to be very sensitive to the accuracy of the model structure. It is expected that a large discrepancy between the simple model of friction used in simulation and the actual effects of friction in real life would cause degradation in the performance of the ACT controller. However, this must also be verified experimentally.

7.2.3 Other Supervised Learning Algorithms

Aside from LWPR and GPR, regression with Support Vector Machines (SVM) has been of recent interest to the robotics community, and some work has already been carried out using SVM to learn inverse dynamics [35]. Further work in exploring the advantages and disadvantages of this method as well as methods of incorporating a-priori knowledge into the SVM framework would be beneficial.

References

[1] Ardence RTX, http://www.intervalzero.com/. 60

[2] Wincon 5.0 user's guide, quanser consulting. xi, 60, 61

[3] C. Abdallah, D.M. Dawson, P. Dorato, and M. Jamshidi. Survey of robust control for rigid robots. *Control Systems Magazine, IEEE*, 11(2):24–30, Feb 1991. 22, 24

[4] C.H. An, C.G. Atkeson, J.D. Griffiths, and J.M. Hollerbach. Experimental evaluation of feedforward and computed torque control. *Robotics and Automation, IEEE Transactions on*, 5(3):368 –373, jun 1989. 16, 25

[5] Suguru Arimoto, Sadao Kawamura, and Fumio Miyazaki. Bettering operation of robots by learning. *Journal of Robotic Systems*, 1(2):123–140, 1984. 24

[6] Brian Armstrong-Hélouvry, Pierre Dupont, and Carlos Canudas de Wit. A survey of models, analysis tools and compensation methods for the control of machines with friction. *Automatica*, 30(7):1083–1138, 1994. 19, 24

[7] C.G. Atkeson. Using locally weighted regression for robot learning. 2:958–963, Apr 1991. 25

[8] Ko Ayusawa, G. Venture, and Y. Nakamura. Identification of humanoid robots dynamics using floating-base motion dynamics. In *IEEE/RSJ Int Conf on Intelligent Robots and Systems*, pages 2854–2859, Sept. 2008. 1, 19, 21

[9] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2nd edition, 2007. 24

[10] D. Bristow, M. Tharayil, and A. Alleyne. A survey of iterative learning control. *IEEE Control Systems Magazine*, 26(3):96–114, 2006. 24

[11] E. Burdet and A. Codourey. Evaluation of parametric and nonparametric nonlinear adaptive controllers. *Robotica*, 16(1):59–73, 1998. 23, 24, 35

[12] C. Canudas, K. Astrom, and K. Braun. Adaptive friction compensation in dc motor drives. 3:1556 – 1561, Apr 1986. 24

[13] P.I. Corke. A robotics toolbox for MATLAB. *IEEE Robotics and Automation Magazine*, 3(1):24–32, March 1996. 35, 53

[14] J. Craig, Ping Hsu, and S. Sastry. Adaptive control of mechanical manipulators. *Int Journal of Robotics Research*, 6(2):16–28, June 1987. 1, 17, 22, 23, 36, 41

[15] Lehel Csato. *Gaussian Processes - Iterative Sparse Approximations*. PhD thesis, Aston University, 2002. 32

[16] Lehel Csato and Manfred Opper. Sparse on-line gaussian processes. *Neural Computation*, 14(3):641–668, 2002. 2, 30, 32, 33, 53

[17] J. Denavit and R. S. Hartenberg. A kinematic notation for lower-pair mechanisms based on matrices. *J. Appl. Mechanics*, 22:215–221, 1955. 5

[18] Yaakov Engel, Shie Mannor, and Ron Meir. Sparse online greedy support vector regression. In *13th European Conference on Machine Learning*, pages 84–96, 2002. 29

[19] M.S. Fadali, M. Zohdy, and B. Adamczyk. Robust pole assignment for computed torque robotic manipulators control. In *American Control Conf, 1989*, pages 37–41, June 1989. 22

[20] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 1999. 28, 29

[21] J. Ho. Open architecture controller for the crs a465 robot arm. Technical report, 2003. 60

[22] P. Ioannou and B. Fidan. *Adaptive Control Tutorial*. SIAM, 2006. 24

[23] M. Kalakrishnan, S. Chitta, E. Theodorou, P. Pastor, and S. Schaal. Stomp: Stochastic trajectory optimization for motion planning. In *IEEE Int Conf on Robotics and Automation, 2011*, pages 4569 – 4574, 2011. 24

[24] P.K. Khosla. Categorization of parameters in the dynamic robot model. *IEEE Trans on Robotics and Automation*, 5(3):261 –268, Jun 1989. 1, 21, 44

[25] Jens Kober and Jan Peters. Learning motor primitives for robotics. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2112 –2118, may 2009. 24

[26] I.D. Landau and R. Horowitz. Synthesis of adaptive controllers for robot manipulators using a passive feedback systems approach. In *IEEE Int Conf on Robotics and Automation*, pages 1028 –1033, Apr 1988. 23

[27] Frank L. Lewis. Neural network control of robot manipulators. *IEEE Expert: Intelligent Systems and Their Applications*, 11:64–75, June 1996. 28, 29

[28] R. H. Middleton and G. C. Goodwin. Adaptive computed torque control for rigid link manipulations. *Systems and Control Letters*, 10(1):9 – 16, 1988. 23

[29] Kian Ming, A. Chai, Christopher K. I. Williams, Stefan Klanke, and Sethu Vijayakumar. Multi-task gaussian process learning of robot inverse dynamics. In *Advances in Neural Information Processing Systems*, 2009. 30

[30] Kohl N. and Stone P. Machine learning for fast quadrupedal locomotion. In *AAAI'04*, pages 611–616, 2004. 24

[31] Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational Space Control: A Theoretical and Empirical Comparison. *Int Journal of Robotics Research*, 27(6):737–757, 2008. 36, 54

[32] D. Nguyen-Tuong and J. Peters. Using model knowledge for learning inverse dynamics. In *IEEE Int Conf on Robotics and Automation*, pages 2677–2682, 2010. 2, 34, 49

[33] D. Nguyen-Tuong, J. Peters, M. Seeger, B. Schlkopf, and M. Verleysen. Learning inverse dynamics: A comparison. Technical report, 2008. 30, 57, 80

[34] Duy Nguyen-Tuong and Jan Peters. Learning robot dynamics for computed torque control using local gaussian processes regression. In *Proceedings of the 2008 ECSIS Symposium on Learning and Adaptive Behaviors for Robotic Systems*, pages 59–64, 2008. 30

[35] Duy Nguyen-Tuong, Bernhard Scholkopf, and Jan Peters. Sparse online model learning for robot control with support vector regression. In *IEEE/RSJ Int Conf on Intelligent Robots and Systems*, pages 3121–3126, Oct. 2009. 1, 16, 19, 25, 29, 82

[36] Duy Nguyen-tuong, Matthias Seeger, and Jan Peters. Computed torque control with nonparametric regression models. In *Proceedings of the 2008 American Control Conference*, pages 212–217, 2008. 2, 17, 30

[37] R. Ortega and M.W. Spong. Adaptive motion control of rigid robots: a tutorial. In *IEEE Conf on Decision and Control*, pages 1575–1584, Dec 1988. 1, 23, 64

[38] H.D. Patino, R. Carelli, and B.R. Kuchen. Neural networks for advanced control of robot manipulators. *Neural Networks, IEEE Transactions on*, 13(2):343 –354, mar 2002. 24, 29

[39] Jan Peters and Stefan Schaal. Learning to Control in Operational Space. *Int Journal of Robotics Research*, 27(2):197–212, 2008. 2, 39, 43, 57

[40] K. Radkhah, D. Kulic, and E. Croft. Dynamic parameter identification for the crs a460 robot. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3842 –3847, Nov 2007. 21, 61

[41] C.E. Rasmussen and C.K. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2006. 25, 29, 30

[42] J.S. Reed and P.A. Ioannou. Instability analysis and robust adaptive control of robotic manipulators. *Robotics and Automation, IEEE Transactions on*, 5(3):381 –386, Jun 1989. 24

[43] S. Schaal, C.G. Atkeson, and S. Vijayakumar. Scalable techniques from nonparametric statistics for real time robot learning. *Appl. Intelligence*, 16:49–60, 2002. 2, 25, 26, 27, 28

[44] Stefan Schaal and Christopher G. Atkeson. Constructive incremental learning from only local information. *Neural Computation*, 10(8):2047–2084, 1998. 25

[45] L. Sciavicco and B. Scicliano. *Modelling and Control of Robot Manipulators*. Springer, 2nd edition, 2000. 1, 9, 11, 13, 14, 16, 17, 18, 21, 48

[46] Jonathan R. Shewchuk. An Introduction to the Conjugate Gradient Method Without the Agonizing Pain. Technical report, Pittsburgh, PA, USA, 1994. 49, 52

[47] J.-J. E. Slotine and W. Li. On the adaptive control of robot manipulators. *Int Journal of Robotics Research*, 6(3):49–59, 1987. 23

[48] Jean-Jacques E. Slotine. The robust control of robot manipulators. *Int Journal of Robotics Research*, 4(2):49–64, 1985. 22

[49] Alex J. Smola and Bernhard Schlkopf. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, August 2004. 29

[50] E. Snelson and Z. Ghahramani. Sparse gaussian processes using pseudo-inputs. In *Advances in Neural Information Processing Systems*, pages 1257–1264. MIT press, 2006. 2, 30, 31, 53

[51] M. Spong and M. Vidyasagar. Robust linear compensator design for nonlinear robotic control. *Robotics and Automation, IEEE Journal of*, 3(4):345–351, August 1987. 22, 23

[52] M. Spong and M. Vidyasagar. Robust linear compensator design for nonlinear robotic control. *Robotics and Automation, IEEE Journal of*, 3(4):345–351, August 1987.

[53] Richard S. Sutton and Andrew G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. 24

[54] M. Tedrakem, R. Zhang, and S. Seung. Learning to walk in 20 minutes. In *Fourteenth Yale Workshop on Adaptive and Learning Systems*, 2005. 24

[55] Sethu Vijayakumar, Aaron D'souza, and Stefan Schaal. Incremental online learning in high dimensions. *Neural Computation*, 17(12):2602–2634, 2005. 1, 2, 24, 25, 26, 35, 38, 53, 57

[56] L.L. Whitcomb, A.A. Rizzi, and D.E. Koditschek. Comparative experiments with a new adaptive controller for robot arms. *IEEE Trans on Robotics and Automation*, 9(1):59–70, Feb 1993. 19

[57] H. Yu and S. Lloyd. Adaptive control of robot manipulators including motor dynamics. In *Proceedings of the American Control Conf*, pages 3803 –3807, Jun 1995. 23, 24, 64