1

ALCANTARA

**National Library
of Canada**

**Bibliothèque nationale
du Canada**

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

**Investigations of Adaptive Control
for a
Direct Drive Robotic Manipulator**

by

GABRIEL D. WARSHAW, B.Sc.

**A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Master of Engineering**

Department of Systems and Computer Engineering

**Carleton University
Ottawa, Ontario
January 19, 1990**

**© Copyright
1990, Gabriel D. Warshaw**

National Library
of Canada

Bibliothèque nationale
du Canada

Canadian Theses Service Service des thèses canadiennes

Ottawa, Canada
K1A 0N4

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-59030-0

**The undersigned recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis
Investigations of Adaptive Control
for a
Direct Drive Robotic Manipulator
submitted by
Gabriel D. Warshaw, B.Sc.
in partial fulfilment of the requirements for
the degree of Master of Engineering in Electrical Engineering**

Thesis Supervisor

Chairman, Department of Systems and Computer Engineering

Carleton University

April 18, 1990

Abstract

High speed robotic manipulators, such as direct drive manipulators, require effective controller designs. A number of adaptive control techniques have been proposed which offer stable and accurate performance. The direct and indirect methods of model-based adaptive control were selected for investigation.

Simulation of a two degree-of-freedom direct drive manipulator is used to demonstrate how adaptive control laws can lead to inferior performance when the richness condition is not satisfied. A non-identifiable desired trajectory is chosen and a realistic level of gaussian white noise is added to the velocity and acceleration measurements. Without persistent excitation, the control algorithms perform worse than a PD controller alone. Errors in parameter estimation, even after convergence, cause the manipulator to deviate from its desired path. Prior to convergence of the parameter estimates, adaptive control of the manipulator produces unacceptably large errors.

The integration of a two degree-of-freedom, direct drive manipulator system is also described. Control algorithms for single joint motion were tested, and simulation results were validated.

Acknowledgements

This thesis document represents the culmination of a great deal of research work. I am indebted to my thesis supervisor, Professor Howard M. Schwartz, for his inspiration, guidance and support throughout the process.

I would also like to thank Danny Lemay and Dave Sword for their assistance in integrating and testing the robot system. My appreciation extends to many others in the Systems and Computer Engineering Department for helping me along the way, as well as to John Batky and others in the Faculty of Engineering's Mechanical Technical Center.

My fellow students and research colleagues also deserve a vote of thanks. Particularly deserving of this thanks are: Richard Gourdeau, Shoeb Rajguru and Talib Janabi.

Finally, I gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada (NSERC), who awarded me with a Post-Graduate Scholarship.

Contents

Acceptance Sheet	ii
Abstract	iii
Acknowledgements	iv
Table of Contents	v
List of Tables	viii
List of Figures	ix
List of Plates	xiii
List of Acronyms and Symbols	xiv
1 Introduction	1
1.1 Background	1
1.2 Thesis Summary	4
2 Adaptive Control Theory	5
2.1 Identification Techniques	5
2.2 Identifiability	10
2.3 Manipulator Equations of Motion	13

2.4	Computed-Torque Control	14
2.5	Adaptation Laws	18
3	Simulations	26
3.1	Simulation Goals	26
3.2	Manipulator Model	26
3.3	Simulation Software	28
3.4	Simulation Runs	31
3.5	Simulation Results	34
4	Experiments	54
4.1	Robot Hardware Description	54
4.2	Control Software Description	59
4.3	Robot Integration and Functional Testing	60
4.4	System Identification for Motor #2	64
4.5	Test Run Description	71
4.6	Test Results	76
5	Conclusions	84
5.1	Discussion and Conclusions	84
5.2	Recommendations	86
	Bibliography	87
	APPENDICES	90
A	Derivation of Recursive Least Squares	90
B	Manipulator Equations of Motion	95
C	Simulation Software — Indirect Adaptive Control	101

D Control Software — Direct Adaptive Control	120
E Robot System Components Specifications	132
E.1 Servo Amplifier Specifications	133
E.2 A/D Interface Card Specifications	136
E.3 D/D Interface Card Specifications	143
E.4 Servo Motor Specifications	145
E.5 Resolver Specifications	145
E.6 Tachometer Specifications	145

List of Tables

3.1	Parameter Identification Comparison	39
4.1	Test Run Matrix	78

List of Figures

2.1	One Degree-of-Freedom Spring/Mass System	16
2.2	Inverse Dynamics Controller	16
2.3	Computed-Torque Controller	17
2.4	Feedforward Controller	17
2.5	Manipulator Computed-Torque Controller	21
2.6	Direct Adaptive Controller	21
2.7	Indirect Adaptive Controller	25
3.1	Serial Link Manipulator Geometry	33
3.2	Noise on Velocity Measurements, Joint 1	33
3.3	Noise on Acceleration Measurements, Joint 1	38
3.4	Loopy Trajectories with PD Control	39
3.5	Straight-Line Trajectories with PD Control	40
3.6	PD Control — Tracking Error for Loopy Path without Noise	40
3.7	PD Control — Tracking Error for Straight Path without Noise	41
3.8	PD Control — Tracking Error for Loopy Path with Noise	41
3.9	PD Control — Tracking Error for Straight Path with Noise	42
3.10	Indirect Adaptive Control — Tracking Error for Loopy Path without Noise	42
3.11	Indirect Adaptive Control — Tracking Error for Loopy Path with Noise	43

3.12 Indirect Adaptive Control — Parameter Estimates for Loopy Path without Noise	43
3.13 Indirect Adaptive Control — Parameter Estimates for Loopy Path with Noise	44
3.14 Direct Adaptive Control — Tracking Error for Loopy Path without Noise	44
3.15 Direct Adaptive Control — Parameter Estimates for Loopy Path without Noise	45
3.16 Direct Adaptive Control — Trend of Parameter Estimates for Loopy Path without Noise, over 50 seconds	45
3.17 Direct Adaptive Control — Parameter Estimates for Loopy Path without Noise, after 50 seconds	46
3.18 Indirect Adaptive Control — Tracking Error for Straight Path with- out Noise	46
3.19 Indirect Adaptive Control — Parameter Estimates for Straight Path without Noise	47
3.20 Direct Adaptive Control — Tracking Error for Straight Path without Noise	47
3.21 Direct Adaptive Control — Parameter Estimates for Straight Path without Noise	48
3.22 Direct Adaptive Control — Tracking Error for Straight Path without Noise, after 50 seconds	48
3.23 Direct Adaptive Control — Trend of Parameter Estimates for Straight Path without Noise, over 50 seconds	49
3.24 Indirect Adaptive Control — Tracking Error for Straight Path with Noise	49
3.25 Direct Adaptive Control — Tracking Error for Straight Path with Noise	50

3.26 Indirect Adaptive Control — Parameter Estimates for Straight Path with Noise	50
3.27 Direct Adaptive Control — Parameter Estimates for Straight Path with Noise	51
3.28 Indirect Adaptive Control — Tracking Error for Straight Path with Noise, after 50 seconds	51
3.29 Direct Adaptive Control — Tracking Error for Straight Path with Noise, after 50 seconds	52
3.30 Indirect Adaptive Control — Parameter Estimation for Straight Path with Noise, after 50 seconds	52
3.31 Direct Adaptive Control — Parameter Estimation for Straight Path with Noise, after 50 seconds	53
4.1 Direct Drive Manipulator Geometry	56
4.2 Robot System Block Diagram	56
4.3 Noise Effects	63
4.4 Noise on Measured Velocity	63
4.5 Noise on Measured Position	67
4.6 Closed Loop System Block Diagram	67
4.7 Root Locus for $k_v = 0$	69
4.8 Root Locus for $k_v = 1$	69
4.9 Closed Loop Step Response	75
4.10 Motor #2 Square Wave Response	79
4.11 Motor #2 Step Response with Pre-filtered Input	79
4.12 Motor #2 Sine Response with Pre-filtered Input	80
4.13 Tracking Error for Pre-filtered Step Input	80
4.14 Tracking Error for Pre-filtered Sine Input	81
4.15 Motor #2 Sine Response with Direct Adaptive Controller	81
4.16 Parameter Estimates from Direct Adaptive Controller	82

4.17 Sine Response with Direct Adaptive Controller, longer run	82
4.18 Parameter Estimates from Adaptive Controller, longer run	83
B.1 Manipulator Geometry and Coordinates	96

List of Plates

4.1	Direct Drive Manipulator.....	57
4.2	Robot System Laboratory Setup.....	58

List of Acronyms and Symbols

a	first order (single joint) model parameter
a_i	i^{th} model parameter
A	block diagonal constant matrix, or filter constant matrix
A_d	discrete filter constant matrix
ARX	Auto-Regressive eXogenous (finite difference model)
A/D	Analog to Digital (convertor)
b	viscous damping for a single joint
b_i	i^{th} model parameter (for input terms)
B	block diagonal constant matrix, or filter constant matrix
B_d	discrete filter constant matrix
C	scalar constant, or block diagonal constant matrix
D/D	digital to digital convertor (parallel interface)
e	vector of joint tracking errors
e_1	vector of filtered joint tracking errors
EMI	Electro-Magnetic Interference
$f()$	a function
F	force input
$G(s)$	arbitrary system transfer function
$G_c(s)$	PD controller transfer function

$H_{ol}(s)$	open loop transfer function for a single joint
$H_d(s)$	closed loop transfer function for a single joint
i	joint index, or effective inertia for a single joint
I_r	identity matrix of dimension r
k_a	amplifier constant
k_m	motor torque constant
k_r	effective robot single joint gain constant
k_1, k_2	spring constants
K	filter constant
K_p	matrix of position feedback gains
K_v	matrix of velocity feedback gains
KHz	Kilo-Hertz
l	link length
m	link mass
M	scalar mass, or mass matrix
$M(y)$	manipulator mass matrix (function of y)
$\hat{M}$	estimated M matrix
$\tilde{M}$	error in estimation of M matrix
MCS	Motion Control Systems, Inc.
MHz	Mega-Hertz
MKS	Meter-Kilogram-Second metric system of units
n	number of manipulator joints
N	number of samples
N/A	Not Applicable
$P(t)$	covariance matrix
$\mathcal{P}$	symmetric positive definite matrix
PD	Proportional Derivative (feedback controller)
PID	Proportional Integral Derivative

PWM	Pulse Width Modulated
$Q(y)$	vector of (non-input) manipulator torque terms
$\hat{Q}$	estimated Q vector
$\tilde{Q}$	error in estimation of Q vector
Q	symmetric positive definite matrix
r	number of parameters to be estimated
$\hat{R}(t)$	matrix equal to $P^{-1}(t)$
RLS	Recursive Least Squares
SCARA	Selective Compliance Assembly Robot Arm
t	time
t_0	arbitrary moment in time
T	vector of joint torques, or discrete time sampling interval
T_c	vector of joint control torques
TTL	Transistor-Transistor Logic
$u(s)$	servo amplifier input voltage (single joint)
$u(t)$	input signal
UDU	Upper-Diagonal-Upper matrix factorization
$v(t)$	noise process
$V_N(\theta)$	least squares cost function
$V(X, \tilde{\theta})$	Lyapunov function
W	matrix of functions of joint positions and derivatives
W_d	W evaluated along desired trajectory
x	position (translational)
$x(t)$	system state, as a function of time
$\hat{x}(t)$	estimate of system state
X	vector of system states
y	vector of joint (angular) position
y_d	vector of desired joint position

y_r	reference joint position
$\tilde{y}^*$	vector of desired joint acceleration plus feedback
α	positive scalar constant, also filter constant
α_i	sample weighting factor
β	filter constant
Γ	diagonal matrix of adaptation gains
Δt	discrete time step
ζ	damping ratio
θ	vector of model parameters
$\hat{\theta}$	estimate of θ vector
$\tilde{\theta}$	error in estimation of θ vector
θ_1, θ_2	joint 1 and 2 (angular) positions
λ	forgetting factor, or dummy variable
ρ	positive scalar constant
τ	time constant
τ_i	torque for i^{th} manipulator joint
τ_{ol}	open loop time constant for single joint
ϕ	regression vector
$\phi_m(t)$	matrix formed of ϕ at successive t
Ψ	diagonal matrix for scaling e
ω	angular frequency (in radians per second)
ω_n	natural frequency

Chapter 1

Introduction

1.1 Background

High speed robotic manipulators, such as direct drive manipulators, require effective controller designs. Direct drive manipulators are currently the subject of extensive research, due to their potential for high speed and accurate motion. The advantages of direct drive manipulators derive from their low friction, backlash and drive-train inertia, and high stiffness.

A problem which is particularly of interest with direct drive manipulators is that they must be treated as highly non-linear and coupled systems. For the more conventional indirect (gear-driven) robots, there is not as large a degree of coupling between the joints of the manipulator compared to the direct drive case; nor is there as great an effect due to changes in the loading conditions. This is because the inertia "seen" by the motor of a joint is not significantly affected by the inertias connected to the other end of the gear box (due to the large speed reduction ratio of the gear box). Hence, a conventional robot can often be successfully treated as a collection of uncoupled systems for the purposes of control¹. In contrast, for

¹Other characteristics of gear-driven manipulators do, however, result in significant nonlinearities, notably backlash and stiction.

direct drive manipulators, motion of other joints and changes in the manipulator load result in significant changes in the dynamics seen by the motor of a given joint. Thus, for direct drive manipulators one is concerned with the control of non-linear and highly coupled systems.

When the inertial dynamics of the manipulator are known a-priori, one of a number of non-linear compensation techniques can be used to drive the manipulator along the desired trajectory. "Feedforward compensation" and "computed torque" techniques are well known model-based methods for performing this type of control. Control of a direct drive manipulator using feedforward compensation was investigated by Asada et al. [14], and their experiments showed improvement in dynamic performance. However, if the manipulator dynamics are not known precisely or if the manipulator handles unknown loads, then the performance of the manipulator will be significantly impaired. In fact, the system may even become unstable. More recently, Khosla and Kanade [17] experimentally compared three types of model-based controllers (computed torque, feedforward, reduced- model feedforward). Their results underscore the need for accurate estimates of the inertial parameters to achieve the benefits of model-based control.

To overcome these problems inherent in model-based control of manipulators, several adaptive control techniques have been proposed. These methods will achieve stable and accurate controller performance even though the manipulator dynamics are either not completely known or they are slowly changing.

Early research on adaptive control of robotic manipulators was performed by Dubowsky and Desforges [20]. A "model-reference" adaptive control scheme was proposed. This "self-tuning gain" scheme adaptively modified the gains for the separate controller for each joint, according to a linear second- order performance model.

A model-reference adaptive scheme was also investigated by Koivo and Guo [15]. This work employed linear finite difference equation in a "black-box" fashion to model both single-variable (independent joint) and multi-variable (coupled joint) manipulator systems. Recursive least squares was used to estimate the parameters of the Auto-Regressive eXogenous (ARX) model, and gain tuning using optimal

control laws was performed.

Horowitz and Tomizuka [21] proposed and simulated a model-based adaptive controller which used a (non-linear) inverse model of the manipulator dynamics and the computed torque method. The parameter adaptation law was derived using the Hyperstability approach and used tracking errors to drive the adaptation.

This thesis focuses on two categories of adaptive control techniques. The first method is indirect adaptive control proposed by Middleton and Goodwin [1], and Hsu et al. [2]. In this method the parameter adaptation algorithm is formulated based on a linear regression model of the inertial dynamics. The linear regression model of manipulator dynamics was proposed by Atkenson et al. [3]. Given a linear regression model one can use least squares, or recursive least squares techniques to identify the unknown parameters, even though the system model is non-linear. An excellent review of least squares identification techniques, as applied to robotic manipulators, can be found in Li and Slotine [4]. The identified model is used in a computed torque control structure. The adaptation mechanism for the indirect adaptive control technique is driven by the prediction errors. The prediction errors are given as the difference between the measured (and filtered) torque and the predicted torques.

In the direct adaptive control methods, proposed by Craig et al. [7], and Slotine and Li [8], the parameter adaptation algorithms are driven by the trajectory tracking errors. The adaptation algorithms are derived based on the Lyapunov stability criteria, which guarantees the global convergence of the error dynamics. The guarantee of global convergence is still valid even if the desired trajectory does not satisfy the richness condition. As in the indirect adaptive control methods, the inertial dynamics are formulated as a linear regression. The advantage of the direct method is that the tracking error is directly minimized by the adaptation algorithm. Furthermore, direct measurements of the joint accelerations are not needed if the algorithm in [8] is used.

This thesis investigates the effect on the performance of the adaptive control algorithms when the desired trajectory does not satisfy the richness condition. The simulation results indicate that when no noise is added to the measurements the

adaptive algorithms can exhibit excellent performance. This is because even small deviations from the desired trajectory will result in a persistently excited path, and hence will satisfy the richness condition. However, once noise is added to the measurement signals the performance of these algorithms is severely impaired.

1.2 Thesis Summary

Chapter 2 of this document contains a treatment of basic adaptive control theory, as it relates to control of robotic manipulators. System identification concepts and the issue of identifiability are discussed.

Chapter 3 is a description of the simulations that were carried out to investigate the adaptive control techniques for an example of a 2 degree of freedom direct drive robotic manipulator.

Chapter 4 describes the direct drive manipulator that was built as a research tool, which was intended to provide the experimental means to verify the results of the analyses and simulations. The design, integration and test of this robot, as well as initial test results for various control algorithms for single joint motion, are described.

A discussion of the results and conclusions drawn from the analyses, simulation and testing is contained in Chapter 5. Recommendations for further research are also discussed.

Following the list of references, a number of appendices can be found which contain various types of detailed information including software source listings and mathematical derivations.

Chapter 2

Adaptive Control Theory

2.1 Identification Techniques

The following is a general introduction to system identification, and it draws freely from Ljung and Söderström [10].

When dealing with dynamic systems, it is necessary to have mathematical models which are representative of the system behaviour. These mathematical models are needed for the purposes of control, signal processing and prediction. There are two main methods of deriving a mathematical model of a system:

1. direct modelling of the system by analysis;
2. use of observed system data (i.e. signals or states produced by the system, along with the corresponding inputs and initial conditions) to perform “system identification”.

System identification to produce a mathematical model is desirable when direct modelling is not practical. This may result from a number of reasons, including:

1. incomplete knowledge of the system’s internal workings;
2. unpredictable changes in the system properties;
3. excessive effort required to model the system;
4. complex analysis resulting in an overly complex model.

The process of system identification can be divided into two steps. The first step is to select the “class” or “family” for the candidate mathematical model, i.e., select the model structure. The second step is to select the particular member of the family which best reconstructs the observed system data. This is done by choosing parameters to assign to the model structure; this is termed “parameter estimation”.

To further explain the system identification process, one can discuss a simple example, such as a lumped parameter spring/mass/damper system. This system is typically modelled by a second order linear, constant coefficient differential equation — this is the selection of the model structure. Selection of the appropriate mass, damping and stiffness terms to result in the observed system performance (for example: natural frequency, damping ratio and steady-state gain) is the parameter estimation step.

System identification techniques can be classified into two broad types:

1. off-line identification;
2. on-line identification.

When a batch of system data is collected and subsequently is used to construct a mathematical model, this is termed “off-line”. “On-line” identification is used when it is necessary to construct and update the model at the same time the data is being collected. The updating for on-line identification is performed by a recursive algorithm, hence “recursive identification” is the term used for this type of method.

A convenient framework commonly employed for the structure of a chosen model is the “linear regression”. A linear regression model expresses the system state as a linear combination of the components of the observed system data plus noise. The linear regression model can be written as follows:

$$x(t) = \theta^T \phi(t) + v(t), \quad (2.1)$$

where

$x(t)$ = system state, as a function of time

θ = parameter vector

$\phi(t)$ = observed data vector

$v(t)$ = noise.

The parameter vector, θ , contains the parameters which define the linear combination of the components of the observed data vector, $\phi(t)$. $\phi(t)$ is called the regression vector, and its terms are the regression variables. For this model, the noise $v(t)$ is expressed as a disturbance on the state $x(t)$.

To illustrate the formulation of a model as a linear regression, consider the example of a linear difference equation, which has the general form:

$$\begin{aligned} x(t) = & a_1x(t-1) + a_2x(t-2) + \cdots + a_nx(t-n) + \\ & b_0u(t) + b_1u(t-1) + \cdots + b_mu(t-m), \end{aligned} \quad (2.2)$$

where

t = discrete time

$x(t)$ = output signal

$u(t)$ = input signal.

In this case, the parameter vector and regression vector are as follows:

$$\theta^T = [a_1, a_2, \dots, a_n, b_0, b_1, \dots, b_m]$$

$$\phi^T(t) = [x(t-1), x(t-2), \dots, x(t-n), u(t), u(t-1), \dots, u(t-m)].$$

In the absence of complete knowledge of the characteristics of the noise, $v(t)$, it is reasonable to formulate a prediction or estimate of the system state, $x(t)$, as the following:

$$\hat{x}(t) = \theta^T \phi(t). \quad (2.3)$$

This expression is simply equation 2.1 with $v(t)$ set to zero. If $v(t)$ is a noise process consisting of a sequence of zero mean and independent random variables (i.e. a white noise process), then equation 2.3 is a prediction in the exact statistical sense. It will be seen that this fact is key to the use of the linear regression model in adaptive control.

Returning to the identification problem, when the model is formulated as a linear regression, the objective is to estimate the parameter vector θ from measurements of $x(t)$ and $\phi(t)$ (taken over a range of t).

Equation 2.1 can be rewritten as follows:

$$v(t) = x(t) - \hat{\theta}^T \phi(t). \quad (2.4)$$

The notation $\hat{\theta}$ denotes the estimate of θ . $v(t)$ can now be interpreted as the “equation error” or “prediction error”, since $v(t)$ is what is not explained or predicted by the linear regression model. This is another way of expressing the important conclusions conveyed in the previous paragraph. With the goal of identification being to choose the model which best explains the observed system data, it is natural to do this by minimizing $v(t)$, with respect to variations in the parameters, θ .

By choosing to minimize the square of the prediction error, a cost function for $v(t)$ can be defined as follows:

$$V_N(\theta) = \frac{1}{N} \sum_{t=1}^N \alpha_t [x(t) - \theta^T \phi(t)]^2, \quad (2.5)$$

where

N = number of samples (observations)

α_t = variable weighting factor (positive scalar) for each observation.

Since $V_N(\theta)$ is quadratic in θ , it can be minimized analytically, which results in the following expression (provided the inverse exists):

$$\hat{\theta}(N) = \left[\sum_{t=1}^N \alpha_t \phi(t) \phi^T(t) \right]^{-1} \sum_{t=1}^N \alpha_t \phi(t) x(t). \quad (2.6)$$

The notation $\hat{\theta}(N)$ denotes the estimate of θ (as a function of N). This expression represents the well-known “Least Squares” estimate.

It should be highlighted that the formulation of the linear regression model described by equation 2.1 embodies the noise term as a disturbance on the state. In this case, the estimate of θ is unbiased (assuming $v(t)$ is a zero mean noise process). However, if $v(t)$ is instead added as noise on the measurements (which compose

$\phi(t)$), the estimate of θ will be biased to a small degree due to the filtering effect of the system on the contribution of the measurement noise. It was shown by the results of simulations (to be discussed in Chapter 3) that no significant bias effect due to measurement noise occurred.

The above Least Squares estimate can be used for off-line identification. For use in on-line identification, the least squares algorithm can be rewritten in recursive form. The derivation is detailed in Appendix A, and the resulting Recursive Least Squares (RLS) formulation for discrete time is as follows:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + P(t)\phi(t)\alpha_t[x(t) - \hat{\theta}^T(t-1)\phi(t)], \quad (2.7)$$

$$P(t) = P(t-1) - \frac{P(t-1)\phi(t)\phi^T(t)P(t-1)}{1/\alpha_t + \phi^T(t)P(t-1)\phi(t)}, \quad (2.8)$$

where $P(t)$ = recursively calculated covariance matrix.

The above RLS algorithm is satisfactory for performing the on-line parameter estimation task; however, it must be emphasized that this method uses the observations for all t for the estimates. In order to track changes in the system parameters that may occur over time, it is necessary to have a means of discounting old observations. The "exponentially forgetting" form of the RLS algorithm provides the means for discounting the old measurements with a decaying exponential profile. The Exponentially Forgetting RLS algorithm employs the same parameter update law (Equation 2.7), however the update law for the covariance matrix, $P(t)$, is changed to the following:

$$P(t) = \frac{1}{\lambda} \left[P(t-1) - \frac{P(t-1)\phi(t)\phi^T(t)P(t-1)}{[\lambda/\alpha_t] + \phi^T(t)P(t-1)\phi(t)} \right], \quad (2.9)$$

where λ = forgetting factor.

In practice, λ is chosen to be close to 1. In this case, the contribution of an observation to the parameter estimate will decay in time, with a time constant, τ , as follows:

$$\tau = \frac{1}{1 - \lambda}. \quad (2.10)$$

Hence, adjustment of the value of λ can be thought of as a way of adjusting the smoothing effect of the RLS algorithm. According to this algorithm, observations older than τ time steps would have a weighting of less than 36% relative to new observations. For values of λ very close to one, τ is large and the algorithm will produce highly smoothed parameter estimates which react slowly to changes in the system. For values of λ much less than one, the behaviour is the opposite: quickly reacting estimates, which are therefore more sensitive to both system changes and noise.

There are other means to perform the parameter estimation task besides recursive least squares. These include the Maximum Likelihood, Kalman Filter and Gradient methods (see Ljung and Söderström [10]). One technique which is useful in adaptive control is a discrete-time implementation of a differential equation based on Lyapunov stability. This is a gradient method which employs system tracking errors (errors between the actual and desired system response when the system is being controlled) to converge to the best estimates of the system parameters. Because the derivation of this method is dependant on the formulation of the system dynamics, further description of this method is delayed until the discussion of adaptation laws in Section 2.5.

2.2 Identifiability

Up until this point, the discussion and derivations in Chapter 2 on the subject of adaptive control theory have been general and thus apply to any dynamic system. However, to simplify the explanation which follows, the discussion will be restricted to the class of dynamic systems which can be called "robotic manipulators". In this context, the system input is the torque provided by the motors at the manipulator joints, and the output is the manipulator joint positions in time, which will be referred to as the manipulator "trajectory". A key measure of manipulator performance is the deviation of the actual trajectory followed by the manipulator from the desired trajectory; this is called the "trajectory tracking error", or simply the

“tracking error”¹.

If parameters can be uniquely determined from the available data (torque inputs and resulting manipulator trajectory), then these parameters are “identifiable”. The identifiability of the system depends on whether the data set is sufficiently informative. More specifically, the identification relies upon appropriate excitation of the system to produce a data set which is sufficiently informative to permit identification of all the parameters (i.e. the dynamic response of the system must be sufficiently excited by the trajectory so that the effect of each system parameter can be uniquely determined). This is variously called the “richness condition” or “persistent excitation”.

The following is a simple example of the parameter identification problems that occur when the trajectory is not identifiable. Consider a one degree-of-freedom lumped parameter spring/mass system as shown in Figure 2.1. The equation of motion for a linear model of the system is:

$$F = M\ddot{x} + k_1x + k_2x, \quad (2.11)$$

where

x = position of mass

F = force input

M = mass

k_1, k_2 = spring constants.

In observing the system input and output (F and x), it can be seen by inspection that it is possible to estimate the value of the mass and the sum of the spring constants, but not the spring constants separately (their effect on the system cannot be separately distinguished, assuming a truly one-dimensional problem). Thus, any scheme to estimate the parameters of the system based on the model structure with three parameters would fail. However, if the number of parameters was reduced to

¹When actually using manipulators to perform tasks, the trajectory of interest is that of the tool at the end point of the arm; however, by assuming rigid links, the results based on the joint trajectories are entirely applicable.

two, a new parameter equal to the sum of the spring constants k_1 and k_2 , and the mass parameter M would be identifiable.

It will be shown by the simulations described later in this document that the parameter identification can be successful even if the trajectory is (theoretically) non-identifiable. This is due to the information provided by the finite tracking error that inevitably occurs when the manipulator arm attempts to exactly follow the desired trajectory. This result is misleading, however, since a more realistic simulation that includes noise in the measurements used for the regression vector shows the failure of the identification to converge. This indicates that the non-identifiable trajectory is not capable of supplying sufficient information in the presence of realistic levels of noise.

It will be demonstrated that the issue of identifiability is important when considering adaptive control of robotic manipulators using parameter estimation. One can envision a scenario where a robot is made to follow a non-identifiable trajectory at some point in its operation while adaptive control is active, resulting in divergence of the parameter estimates and resulting degradation of performance (increased trajectory tracking error).

It is worthwhile to point out that this condition could be avoided if one is always aware of the desired trajectory a-priori; then it can be tested for identifiability beforehand and appropriate measures taken (namely, selection of an alternative trajectory) if it poses a problem in this respect. However, one can also envision circumstances where this is not possible; for example, if the trajectory planning is done on-line (near real-time), as would be the case for advanced robots receiving high-level task-oriented instructions and operating in unstructured and changing environments. This is as opposed to "assembly-line" type robots, performing repetitive tasks in rigidly structured environments.

A non-identifiable trajectory can arise from either of two conditions:

1. linear dependence between motion of two or more joints of the robot;
2. a joint which experiences no motion at all.

These conditions are equivalent in terms of the effect on the identification of the

parameters.

An example of a non-identifiable trajectory (i.e. one which does not produce a sufficiently informative data set) for the case of a two-link serial manipulator arm (see Figure 3.1) is a straight line path of the end point through the origin. For the arm to follow a straight line in cartesian space, in joint space the velocity of the second joint will be linearly dependant on the velocity of the first joint. If the two links are equal in length, the second joint velocity will be related to the first by a ratio of exactly two.

Specific tests for the identifiability of a trajectory are discussed below when the adaptive control laws are described, in Section 2.5.

2.3 Manipulator Equations of Motion

The dynamic model of a manipulator can be expressed as a set of non-linear differential equations. Thus, equations of motion can be written as a general differential equation in vector form:

$$T = f(\ddot{y}, \dot{y}, y) \quad (2.12)$$

where

T = vector of joint torques

y = vector of joint (angular) positions.

Both the T and y vectors have dimension n , where n is the number of joints of the manipulator.

For a serial link manipulator (rigid bodies connected by n joints), the equations of motion can be specified as a matrix equation according to the above formulation as follows, according to Craig et al. [7]:

$$T = M(y)\ddot{y} + Q(\dot{y}, y) \quad (2.13)$$

where

M = manipulator mass matrix

Q = vector of other torques.

As indicated, the M matrix is a function of the joint angles. The Q vector contains the centrifugal, coriolis, gravity and frictional torque terms, and is thus a function of joint angles and velocities.

It was shown by Atkeson et al. [3] that the equations of dynamics of a manipulator can also be formulated as a linear regression, as follows:

$$\tau_i = \phi_i^T(t)\theta_i(t) \quad (2.14)$$

where

τ_i = torque for i^{th} joint

$\theta(t)$ = vector of model parameters

$\phi(t)$ = regression vector.

With this formulation, the terms of the regression vector appear linearly in the equations of dynamics, but each term can itself be a non-linear function of joint angles and their derivatives. The subscript i refers to the linear regression for the i^{th} joint of the manipulator. The linear regression can be expressed equivalently using a matrix formulation as in Hsu et al. [2], as follows:

$$T = W(\tilde{y}, \dot{y}, y)\theta \quad (2.15)$$

where W = matrix of functions of the joint positions and their derivatives.

For an example of the above formulation of the equations of motion for a specific two degree-of-freedom serial link manipulator, refer to Section 3.2 ("Manipulator Model") which follows.

2.4 Computed-Torque Control

The concept of model-based control is an important one in the field of robot control. When the dynamic model of a robot is used to construct the inverse model, the inverse dynamics can be used in an attempt to achieve perfect compensation of the

manipulator dynamics (i.e. pole-zero cancellation) which would thus result in a system with infinite bandwidth, and hence high performance. Of course, this result is not possible in practice, because it is necessarily an idealization.

One can think of constructing an inverse dynamics controller as a “pre-filter”, which filters the input to the system being controlled (the desired state) and produces the appropriate control input to the system to achieve the desired state as the output. This pre-filter must in fact have a transfer function which is the exact inverse of the controlled system transfer function for perfect compensation to occur. For the case of a typical manipulator, which is highly non-linear and has significant dynamic coupling between the various joints, the inverse dynamics can be used to linearize and decouple the system. Figure 2.2 is a block diagram representation of the inverse dynamics controller.

The well-known “computed-torque” technique is a type of inverse dynamics controller which employs the inverse model of the manipulator dynamics inside the controller feedback loop. This is in contrast to the also well-known feedforward compensation controller, which uses the inverse model outside the feedback loop. Although the feedforward technique linearizes the manipulator system, it does not in addition completely decouple the dynamics, as does the computed-torque scheme. This is because the computed-torque method includes the inverse dynamics in the feedback loop and hence the error dynamics are not dependant on the arm position. In contrast, the error dynamics for the feedforward case do change as the arm position changes (see Khosla and Kanade [17], Craig [6]). Figures 2.3 and 2.4 show the structures of the computed-torque and feedforward controllers respectively.

The computed-torque control law for a manipulator can be expressed in the following matrix form, derived from the matrix form of the dynamics equation (Equation 2.13), after Craig et. al. [7]:

$$T_c = M(y)\ddot{y}^* + Q(\dot{y}, y) \quad (2.16)$$

where

T_c = vector of joint control torques

$\ddot{y}^* = \ddot{y}_d + K_v\dot{e} + K_p e$ = vector of desired acceleration plus feedback

Figure 2.1: One Degree-of-Freedom Spring/Mass System

Figure 2.2: Inverse Dynamics Controller

Figure 2.3: Computed-Torque Controller

Figure 2.4: Feedforward Controller

$e = y_d - y$ = vector of joint tracking errors

y_d = vector of desired joint positions

K_v = matrix of velocity feedback gains

K_p = matrix of position feedback gains.

Note that to use this method, the dynamics of the manipulator (M matrix and Q vector) must be known a-priori, so that the inverse dynamic model can be employed in the control law. Figure 2.5 shows this control law in block diagram form.

Because it is not possible in practice to accurately know the equations of dynamics of a manipulator, the ideal performance of a model-based controller is not achievable. This issue was discussed in the context of system identification in Section 2.1 above. A major difficulty that was mentioned was the unpredictable changes that occur in a system over time. It will be seen that adaptive control can address these problems which are inherent in model-based control.

2.5 Adaptation Laws

As we have seen in the previous discussion on computed torque control, the identification of the system model is a key issue in the use of this control strategy. Thus we arrive at the motivation for adaptive control: computed torque control for the real world i.e. for the case where the system model is not accurately known a-priori, or it changes over time.

Consider a typical robotic manipulator system. This robot would have a position controller (such as a PID controller) whose design is based on a combination of analysis of the robot's dynamics and experimental tuning. The controller parameters are adjusted until the desired performance is achieved. The dynamics properties of a manipulator may drift over time (eg. increasing friction) or may change suddenly, as when the effective mass is changed when an object is picked up. Therefore, retuning of the controller is required to maintain the desirable performance. Having a controller which is capable of retuning itself is thus very attractive - hence, the motivation for adaptive control.

By combining the computed-torque control technique embodied in Equation 2.16 with a means of on-line parameter estimation, it is possible to derive various forms of adaptive control. The adaptation laws are the algorithms used to perform the parameter estimation.

The parameter estimation algorithms can be classified into two types according to the method of adaptation: the direct method, which uses the tracking errors; the indirect method, which uses the prediction errors.

Tracking errors are defined as the deviation between the desired and actual trajectory of the manipulator. Prediction (or estimation) errors are the deviation between the predicted torque (calculated from the system model including the identified parameters) and the actual (or measured and filtered) torque.

By using the tracking errors to drive the adaptation of the parameters, the direct method is thus oriented towards reducing the system tracking errors through the adaptive process, and is less directly concerned with extracting accurate estimates of the system parameters. Because the indirect method uses the prediction errors to drive the adaptation, its orientation is exactly the opposite.

The following observations can be made about the expected performance of the direct method versus the indirect method. Since the direct method is not concerned with prediction errors, it is possible that the model identified is not a good estimate of the real system; even though the tracking performance may be good for the current trajectory, switching to a different trajectory could invalidate the parameter estimates and result in poor controller performance. The indirect method does not suffer from this problem, however it will be seen that the algorithms to implement it are more complex than the direct method algorithms, and hence consume more computation time.

The indirect adaptive control method was proposed by Middleton and Goodwin [1], and Hsu et al. [2]. In this method the parameter adaptation algorithm is formulated based on a linear regression model of the inertial dynamics. As we have seen in Section 2.1, given a linear regression model one can use recursive least squares techniques to identify the unknown parameters. An excellent review of least squares identification techniques, as applied to robotic manipulators, can be found

in Li and Slotine [4]. The identified model is used in a computed-torque control structure.

The indirect method of parameter adaptation can be implemented using the well known exponentially-forgetting least-square estimation, as in Li and Slotine [4]. For the indirect method using Recursive Least Squares, the richness condition on the trajectory is that the covariance matrix $P(t)$ must exist. Therefore, the matrix defined by $R(t) = P^{-1}(t)$ as the summation $\sum_{i=1}^N \phi(t)\phi^T(t)$ must not be singular (see Appendix A). Equivalently, $\phi_m(t)\phi_m^T(t)$ must not be singular, where $\phi_m(t)$ is a matrix formed from the vector $\phi(t)$ at successive $t = 1$ to N . This implies that the terms of the regression vector are linearly independent, i.e., the parameters being identified are independent.

The direct adaptive control methods were proposed by Craig et al. [7], and Slotine and Li [8]. The adaptation algorithms are derived based on the Lyapunov stability criteria, which guarantees the global convergence of the tracking error dynamics. The guarantee of global convergence is still valid even if the desired trajectory does not satisfy the richness condition. As in the indirect adaptive control methods, the inertial dynamics are formulated as a linear regression. The advantage of the direct method is that the tracking error is directly minimized by the adaptation algorithm. Furthermore, direct measurements of the joint accelerations are not needed if the algorithm in [8] is used.

Figures 2.6 and 2.7 show block diagrams of adaptive control systems, using the direct and indirect method of identification respectively.

As described by Craig et al. [7], a gradient update algorithm based on the direct method can be employed to identify the system parameters. The derivation of the direct adaptation law employed by Craig et al. is summarized below. The second method of Lyapunov is used to guarantee the global convergence of the error dynamics.

The equations of motion for a manipulator were stated as a matrix equation in Section 2.3 (Equation 2.12). This equation is repeated here for convenience:

$$T = M(y)\ddot{y} + Q(\dot{y}, y). \quad (2.17)$$

Figure 2.5: Manipulator Computed-Torque Controller

Figure 2.6: Direct Adaptive Controller

For an adaptive controller, the computed-torque control law for a manipulator derived from Equation 2.17 can be written as (see Equation 2.16):

$$T_c = \hat{M}(y)\ddot{y}^* + \hat{Q}(\dot{y}, y), \quad (2.18)$$

where

T_c = vector of joint control torques

$\hat{M}$ = estimate of M

$\hat{Q}$ = estimate of Q .

By equating Equations 2.17 and 2.18, and substituting in the expression for $\ddot{y}^*$, we get the following error equation:

$$\hat{M}(y)\ddot{y}_d + \hat{M}(y)K_v\dot{e} + \hat{M}(y)K_p e + \hat{Q}(\dot{y}, y) = M(y)\ddot{y} + Q(\dot{y}, y). \quad (2.19)$$

We can drop the arguments of M and Q for convenience, and define the errors in the dynamic model used in the controller (arising from the errors in the parameter estimates):

$$\tilde{M} = M - \hat{M} \quad (2.20)$$

$$\tilde{Q} = Q - \hat{Q}. \quad (2.21)$$

Combining Equation 2.19 and $\tilde{e} = \ddot{y}_d - \ddot{y}$, we get:

$$\hat{M}(\tilde{e} + \ddot{y}) + \hat{M}K_v\dot{e} + \hat{M}K_p e = M\ddot{y} + Q - \hat{Q}. \quad (2.22)$$

This can be rewritten using Equation 2.21 to get:

$$\hat{M}\tilde{e} + \hat{M}K_v\dot{e} + \hat{M}K_p e = (\tilde{M}\ddot{y} + \tilde{Q}). \quad (2.23)$$

Rewriting once again and substituting in Equation 2.20, we get the (tracking) error equation:

$$\tilde{e} + K_v\dot{e} + K_p e = \hat{M}^{-1}(\tilde{M}\ddot{y} + \tilde{Q}). \quad (2.24)$$

If the alternative form of the matrix equations of motion given previously by Equation 2.15 is employed, the following terms are used:

$W(y, \dot{y}, \ddot{y})$ = matrix of functions of joint positions and derivatives

θ = parameter vector (containing all parameters appearing in M and Q).

The parameter error vector, $\tilde{\theta}$, can now be defined:

$$\tilde{\theta} = \theta - \hat{\theta} \quad (2.25)$$

The error equation 2.24 can now be rewritten:

$$\ddot{e} + K_v \dot{e} + K_p e = \hat{M}^{-1} W \tilde{\theta}. \quad (2.26)$$

Define the filtered tracking error vector

$$e_1 = \dot{e} + \Psi e \quad (2.27)$$

where Ψ is a diagonal matrix of positive constants. The error equation 2.26 can be written in state space form as follows:

$$\dot{X} = AX + B\hat{M}^{-1}W\tilde{\theta} \quad (2.28)$$

$$e_1 = CX \quad (2.29)$$

where A , B and C are block diagonal constant matrices and X is a vector of the states (consisting of the terms of e and their first derivatives).

Assuming appropriate selection of the terms of Ψ , and by employing the positive real lemma, it can be written that

$$A^T \mathcal{P} + \mathcal{P} A = -Q \quad (2.30)$$

$$\mathcal{P} B = C^T \quad (2.31)$$

where $\mathcal{P}$ and Q are symmetric and positive definite matrices.

The second method of Lyapunov can now be used to derive an adaptation law for the parameters (θ). A Lyapunov function candidate can be selected which is a function of $\tilde{\theta}$ and will make the system asymptotically stable:

$$V(X, \tilde{\theta}) = X^T \mathcal{P} X + \tilde{\theta}^T \Gamma^{-1} \tilde{\theta} \quad (2.32)$$

where Γ is a diagonal matrix of positive constants. The above function must therefore be positive for any non-zero X and $\tilde{\theta}$.

By differentiating Equation 2.32 with respect to time and making use of Equations 2.30 and 2.31, we get

$$\dot{V}(X, \tilde{\theta}) = -X^T Q X + 2\tilde{\theta}^T (W^T \hat{M}^{-T} e_1 + \Gamma^{-1} \dot{\tilde{\theta}}). \quad (2.33)$$

By choosing

$$\dot{\tilde{\theta}} = -\Gamma W^T \hat{M}^{-T} e_1 \quad (2.34)$$

we get

$$\dot{V}(X, \tilde{\theta}) = -X^T Q X \leq 0.$$

Thus, the stability is guaranteed by the second method of Lyapunov.

By making use of Equation 2.25, a parameter adaptation law can be written from Equation 2.34 as follows:

$$\dot{\hat{\theta}} = \Gamma W^T \hat{M}^{-1} e_1, \quad (2.35)$$

where $\hat{\theta}$ is the vector of estimated model parameters and Γ can now be interpreted as the matrix of adaptation gains.

To satisfy the richness condition, the desired trajectory must satisfy the following:

$$\alpha I_r \leq \int_{t_0}^{t_0+\rho} W_d^T W_d dt, \quad \text{for all } t_0, \quad (2.36)$$

where

$\alpha, \rho =$ positive scalars

$I_r =$ identity matrix, of dimension r

$r =$ number of parameters to be estimated

$W_d = W$ evaluated along desired trajectory.

Figure 2.7: Indirect Adaptive Controller

Chapter 3

Simulations

3.1 Simulation Goals

The main goal of the simulations was to investigate the effect of a non-identifiable trajectory on adaptive control schemes, in the presence of measurement noise. The effect was primarily evaluated on the basis of trajectory tracking performance.

Before the above objective could be reached, it was necessary to establish some baselines for performance, i.e., standards against which the performance could be judged. Thus, the performance of a standard PD control scheme, and the performance of the adaptive controllers for identifiable trajectories without and with measurement noise, and the performance of adaptive controllers for non-identifiable trajectories without noise were all simulated and evaluated. This spectrum of simulations also provided the necessary opportunity to verify that the simulation model and software were providing reasonable and consistent results.

Another goal was to compare the performance of two types of adaptive controllers (direct and indirect) under the range of conditions described above.

3.2 Manipulator Model

A two degree of freedom manipulator was chosen for the simulations described in the sections of this document which follow. This manipulator is assumed to operate in the horizontal plane (i.e. gravity terms are neglected), and its geometry is shown

in Figure 3.1. The link masses are modelled as being evenly distributed along their length, and friction is neglected. The equations of motion have been derived by Lagrangian mechanics by Schwartz and Youcef-Toumi [12]. The derivation is repeated in Appendix B. The equations of motion are as follows:

Given

$$T = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} ; y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\theta = \begin{bmatrix} a_1 = 1/3m_1l_1^2 + m_2l_1^2 + 1/3m_2l_2^2 \\ a_2 = -m_2l_1l_2 \\ a_3 = 1/3m_2l_2^2 \\ a_4 = -1/2m_2l_1l_2 \\ a_5 = m_2l_1l_2 \\ a_6 = 1/2m_2l_1l_2 \\ a_7 = 1/3m_2l_2^2 \\ a_8 = -1/2m_2l_1l_2 \\ a_9 = 1/3m_2l_2^2 \\ a_{10} = -1/2m_2l_1l_2 \end{bmatrix} \quad (3.1)$$

and where

m_1, m_2 = link masses = 2.0 kg

l_1, l_2 = link lengths = 0.3 m,

then the scalar equations of motion are:

$$\tau_1 = a_1\ddot{y}_1 + a_2\cos(y_2)\ddot{y}_1 + a_3\ddot{y}_2 + a_4\cos(y_2)\ddot{y}_2 + a_5\sin(y_2)\dot{y}_1\dot{y}_2 + a_6\sin(y_2)\dot{y}_2^2 \quad (3.2)$$

$$\tau_2 = a_7\ddot{y}_1 + a_8\cos(y_2)\ddot{y}_1 + a_9\ddot{y}_2 + a_{10}\sin(y_2)\dot{y}_1^2. \quad (3.3)$$

For the matrix form of the equations (see Equation 2.13), the resulting M matrix and Q vector are defined as follows:

$$M(y) = \begin{bmatrix} a_1 + a_2\cos(y_2) & a_3 + a_4\cos(y_2) \\ a_7 + a_8\cos(y_2) & a_9 \end{bmatrix} \quad (3.4)$$

$$Q(\dot{y}, y) = \begin{bmatrix} a_5 \sin(y_2) \dot{y}_1 \dot{y}_2 + a_6 \sin(y_2) \dot{y}_2^2 \\ a_{10} \sin(y_2) \dot{y}_1^2 \end{bmatrix}. \quad (3.5)$$

Equivalently, the W matrix for the alternative form of the matrix equations of motion (given by Equation 2.15) can be defined as follows:

$$W^T = \begin{bmatrix} \ddot{y}_1 & 0 \\ \cos(y_2) \ddot{y}_1 & 0 \\ \ddot{y}_2 & 0 \\ \cos(y_2) \ddot{y}_2 & 0 \\ \sin(y_2) \dot{y}_1 \dot{y}_2 & 0 \\ \sin(y_2) \dot{y}_2^2 & 0 \\ 0 & \ddot{y}_1 \\ 0 & \cos(y_2) \ddot{y}_1 \\ 0 & \ddot{y}_2 \\ 0 & \sin(y_2) \dot{y}_1^2 \end{bmatrix}. \quad (3.6)$$

Note that the above formulations of the equations of motion are only valid for the angular position vector, y , expressed in radians. The other quantities (m_1, m_2, l_1, l_2) must be expressed in consistent units (the MKS system was used in this case).

The dynamics of the motors and their servo amplifiers are neglected in this model. This implies that the transfer function of an amplifier/motor combination is identically one, i.e., a torque is produced by the motor at the manipulator joint which is identical to the torque command sent to the motor by the controller. This simplification is a reasonable one for direct-drive motors coupled with amplifiers which are linear and have a bandwidth much higher than that of the inertial dynamics of the manipulator.

3.3 Simulation Software

The simulations of the manipulator control were implemented in the Turbo Pascal programming language (a product of Borland International, Inc.), version 3.0, using the 8087 coprocessor support option. These simulations were run on a PC-AT

compatible computer equipped with a 80287 numeric coprocessor, and running at a clock speed of 12 MHz.

Three versions of the simulation program were used:

1. PD control;
2. Direct adaptive control;
3. Indirect adaptive control.

Appendix C contains the source listing for the indirect adaptive control case.

The simulation software consists of four main components:

1. program control and input/output;
2. manipulator dynamics simulation;
3. inertial parameter identification;
4. control algorithm.

The manipulator dynamics are simulated using a standard 4th order Runge-Kutta algorithm to numerically solve the non-linear differential equations. For the PD control simulation, the parameter identification code was not included.

The adaptive control law uses the discretized version of the control law given previously as Equation 2.16, and is as follows:

$$T_c(t) = \hat{M}(t)\ddot{y}^*(t) + \hat{Q}(t) \quad (3.7)$$

where

$T_c(t)$ = vector of joint control torques at time t

$\ddot{y}^*(t) = \ddot{y}_d(t) + K_v\dot{e}(t) + K_p e(t)$

$e(t) = y_d(t-1) - y(t)$

$\dot{e}(t) = \dot{y}_d(t-1) - \dot{y}(t)$

$\hat{M}(t)$ = estimate of $M(y)$ at time t

$\hat{Q}(t)$ = estimate of $Q(\dot{y}, y)$ at time t .

For the Direct adaptive control, $\hat{M}$ and $\hat{Q}$ are derived using the discrete time form of the adaptation law given in Equation 2.35. The discrete adaptation algorithm is:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + \Delta t(\dot{\hat{\theta}}(t)) \quad (3.8)$$

where

$$\dot{\hat{\theta}} = \Gamma W^T(t) \hat{M}^{-1}(t-1) e_1(t) \quad (3.9)$$

and

$$e_1(t) = \dot{e}(t) + \Psi e(t)$$

Δt = discrete time step.

The elements of $\hat{M}$ and $\hat{Q}$ can thus be calculated directly from the parameter estimates found in $\hat{\theta}(t)$ and the measurements of the manipulator states $(y, \dot{y})$, as shown in Section 3.2.

Similarly, for the Indirect adaptive control, the exponentially forgetting recursive least squares algorithm given by Equations 2.7 and 2.9, is used to identify the parameters from which $\hat{M}$ and $\hat{Q}$ can be calculated. This algorithm was modified for the purposes of the simulation to use the UDU factorization of the covariance matrix, P . The UDU factorized version of P is used to improve the conditioning of the matrix, and hence the stability of the indirect adaptation law (see Ljung [11], Ljung and Söderström [10] and Bierman [16]). The variable weighting factor for each observation, α_i , is set to 1. Thus, the weighting is governed solely by the forgetting factor, λ .

To add gaussian white noise to measurements of the manipulator state, a procedure was written to generate random numbers with a standard normal distribution. This procedure uses uniformly distributed random numbers generated by a Turbo-Pascal function call, and an algorithm based on a method devised by Box and Mueller which was described by Forsythe et al. [18]. The normally distributed random numbers which are generated are then scaled by the desired standard deviation (the mean remains zero) and added to the manipulator states (velocity and acceleration).

3.4 Simulation Runs

To accomplish the goals mentioned above, a wide range of simulations were run. The simulation runs were selected with various combinations of the following alternatives:

1. standard PD control;
2. two types of adaptive computed-torque control (direct and indirect);
3. identifiable and non-identifiable trajectories;
4. without and with measurement noise.

Two different trajectories were used; one which was identifiable and one which was not. These trajectories were defined in joint space, i.e., joint angles which are functions of time. The identifiable trajectory that was selected consists of a combination of sines and cosines of three different frequencies, which produces a "loopy" path of the manipulator arm end point in cartesian space. The joint positions for this loopy path are as follows:

$$y_1 = 1.5 + \cos(2\omega t) - \cos(4\omega t) \quad (3.10)$$

$$y_2 = (\pi/2) - 1.5 + \sin(\omega t) + \sin(2\omega t) \quad (3.11)$$

where

y_1, y_2 = positions of joints 1 and 2 (in radians)

t = time (in seconds)

ω = angular frequency (in radians/second).

The non-identifiable trajectory that was selected produces a straight-line path of the arm end point in cartesian space. This is produced by giving joint 2 twice the velocity of joint 1. The trajectory was chosen to extend the arm and return it to the starting point smoothly. The joint positions for this straight path are defined as follows:

$$y_1 = \cos^{-1}(0.45 - 0.45 \sin(\omega t + \pi/2)) \quad (3.12)$$

$$y_2 = \pi - 2y_1. \quad (3.13)$$

The loopy and straight paths were simulated with ω equal to 1.257 and 5.027 rad/sec respectively, to give an equal frequency for the highest frequency component for the two different paths. Care was taken to roughly equate the frequency content to ensure that the comparison between the tracking performance (i.e. dynamic response of the system) for the two paths was meaningful. At the above-mentioned frequencies, the time to complete a single cycle of the path was 5.0 and 1.25 seconds for the loopy and straight paths respectively.

The measurement noise simulated was a gaussian white noise process added to the measurements of joint 1 and 2 velocities and accelerations. The magnitude of this additive noise was chosen from typical levels of noise determined from experimental data taken from a direct drive manipulator tested by Schwartz and Youcef-Toumi at MIT. The standard deviation of the noise on the velocity and acceleration measurements is 0.2 rad/sec and 2.0 rad/sec² respectively (with zero mean). The appearance of this level of noise on typical velocity and acceleration measurements can be seen in Figures 3.2 and 3.3.

It is generally true that the position measurements are free of measurement noise (aside from the quantization noise which is unavoidable in digital systems), so no noise was added to the angular position measurement. It will be seen that the levels of noise selected for the simulations are indeed representative of the levels observed for the tests of the direct drive arm described later on in this document. However, the assumption that position measurements are free of noise will be shown to be faulty.

The time step used for the simulations was 0.005 seconds, which therefore simulated a controller with a bandwidth of 200 Hz. In all the simulation runs, the position and velocity error feedback gains were set at $K_p = 80I$ and $K_v = 8I$ (where I is the identity matrix). For simulation runs using the Indirect Adaptive control, the forgetting factor for the exponentially forgetting recursive least squares algorithm, λ , was always set equal to 0.999. This gives a time constant for the parameter estimation of 1000 samples (per Equation 2.10), which is equivalent to 5 seconds for the control algorithm running at 200 Hz.

Figure 3.1: Serial Link Manipulator Geometry

Figure 3.2: Noise on Velocity Measurements, Joint 1 (— velocity with noise, - - - added noise)

For all the adaptive control simulation runs, the starting values of the parameters to be estimated were set arbitrarily to either 0.0 or 0.1. Non-zero starting values are necessary in the Direct adaptive control case to prevent singularity of the estimated M matrix, $\hat{M}$, which would prevent $\hat{M}^{-1}$ from being evaluated. It is commonly accepted that the starting value of the covariance matrix for recursive least squares parameter estimation should be set to $C \cdot I$, where C is a constant which is large relative to the uncertainties in the parameters (see Ljung and Söderström [10]). Hence, for the simulations using the Indirect adaptive control, the covariance matrix was set to $10I$.

3.5 Simulation Results

The tracking performance for the standard PD controller is shown in Figures 3.4 to 3.9. Examples of the desired and actual trajectories for the manipulator are shown in Figures 3.4 and 3.5. Figures 3.6 to 3.9 show the tracking error for both joints of the manipulator, for the loopy and straight paths both without and with measurement noise. The maximum tracking error is approximately 6.5 degrees for the loopy path and 2.8 degrees for the straight path. It can be seen that the presence of measurement noise has negligible effect on the maximum values of the tracking error, although small perturbations due to the noise are evident.

Figures 3.10 and 3.11 show the tracking error for the indirect adaptive controller, for the loopy path without and with noise respectively. Figures 3.12 and 3.13 show the corresponding parameter estimates over time (during the 5 second cycle of the loopy trajectory). Without noise, the parameter estimates converge quickly and smoothly to their true values, and the tracking errors are much lower than for the PD controller throughout the run (less than 0.9 degrees). However, when noise is present the parameter estimates undergo initial oscillations and take longer to converge to their true values. This results in poor tracking performance during the first second of the trajectory. Thereafter, the tracking errors are limited to the maximum errors closer to those seen for the case without noise; approximately 1.0 degrees. Thus, even in the presence of noise, the indirect adaptive controller

tracking performance is much superior to the PD controller after allowing sufficient time for parameter convergence — assuming a sufficiently exciting trajectory such as the loopy path.

Figure 3.14 shows the tracking error for the direct adaptive controller, for one cycle of the loopy path without noise. Figure 3.15 shows the corresponding parameter estimates. Since the parameter estimates have not yet converged, the tracking errors are large (maximum of about 13 degrees). After running this controller for 50 seconds (10 cycles of the trajectory), the parameter estimates appear to be converging. Figure 3.16 shows the estimates over the 10 cycles, and Figure 3.17 shows the parameter estimates during the last cycle. During this last cycle, the maximum tracking errors have been reduced to 2.7 degrees. This is worse than the indirect adaptive controller without noise, but better than the PD controller.

Figure 3.18 shows the tracking error using the indirect adaptive control algorithm for the straight path when no noise is present. The performance is excellent and the parameters quickly converge to their true values (as shown in Figure 3.19). Although the desired trajectory is not persistently exciting, the actual trajectory does contain sufficient information such that the parameters can be identified (refer to the discussion on identifiability in Section 2.2). Figures 3.20 and 3.21 illustrate the corresponding performance of the direct adaptive control algorithm. The tracking performance is poor, because convergence of the parameter estimates has not yet been achieved. Apparent convergence can be observed after 50 seconds when no noise is present, and the resulting tracking performance is excellent (error less than 0.8 degrees). This performance is shown in Figures 3.22 and 3.23. However, the converged values of the parameters remain significantly different from the true values. This can be seen by referring to Table 3.1, which shows a comparison of the true and estimated inertial parameters for both the direct and indirect adaptation methods, for the straight path.

Figures 3.24 and 3.25 illustrate the performance of the indirect and direct adaptive control algorithms respectively, when noise is added to the velocity and acceleration measurements, for the straight path. Under these conditions the initial performance for both the indirect and direct algorithms has such a large error that

the manipulator could not be used. The tracking error for both types of controller reaches a maximum of approximately 20 degrees. Figures 3.26 and 3.27 illustrate the parameter estimates for the indirect and direct cases respectively. The initial parameter estimates vary widely until there is sufficient information in the actual trajectory such that the parameters can be identified.

The performance of the adaptive controllers after allowing sufficient time for parameter convergence to occur is considered next. Figures 3.28 and 3.29 illustrate the performance of the indirect and direct adaptive controllers for the straight path after 50 seconds, when the velocity and acceleration measurements are corrupted by noise (the performance of a simple PD controller under the same conditions was shown in Figure 3.9). The performance of the adaptive controllers is approximately the same as for the PD controller and is perhaps even worse. Figures 3.30 and 3.31 illustrate the indirect and direct parameter estimation after 50 seconds. The reason for the poor performance of the indirect algorithm can be seen by referring to Figure 3.30. The lack of convergence, as evidenced by the continued variations in the indirect parameter estimates, result in relatively large tracking errors as seen in Figure 3.28.

Returning to inspection of Table 3.1, it can be concluded that the large tracking errors for a controller using the direct adaptation algorithm (in the presence of measurement noise) result from large errors in the final values of the identified parameters. It is well known that even small errors in the parameter estimates can result in large tracking errors.

It is useful to compare the parameter estimation performance of the direct and indirect methods. To prevent unstable divergent behaviour of the direct algorithm, very low gain values (Γ) must be used in the adaptation law, unless explicit bounds are used to define the range of acceptable values of the parameters as in Craig et al. [7]. There is no clear way to select these gains, and the selection is dependant on the level of noise present, so trial and error must be used. The effect of this problem is that the direct method results in slower convergence compared to the indirect (least squares) method. This is evident from inspection of the estimated parameter values after 1.25 seconds, in Table 3.1.

Another issue to consider is the means of convergence for the direct method; because the tracking errors are used to drive the adaptation (and not the prediction errors, as in the case of the indirect method), apparent convergence of the parameter estimates to values which are significantly different from the true values may occur. This can be seen in Table 3.1, for the estimates after 50 seconds without noise. As a consequence of this, after convergence of the parameter values using the direct algorithm, switching to a different trajectory may invalidate the parameter estimates and result in large tracking errors.

Figure 3.3: Noise on Acceleration Measurements, Joint 1 (— acceleration with noise, - - - added noise)

Table 3.1: Parameter Identification Comparison

Parameter	True Value	Estimates				
		1.25 sec, no noise		50 sec, no noise	50 sec, with noise	
		Direct	Indirect	Direct	Direct	Indirect
a_1	+0.30	+0.25	+0.29	+0.27	+0.28	+0.14
a_2	-0.18	+0.091	-0.17	+0.11	+0.11	-0.10
a_3	+0.060	+0.0022	+0.054	+0.041	+0.046	-0.019
a_4	-0.090	+0.11	-0.084	+0.058	+0.049	-0.047
a_5	+0.18	+0.23	+0.17	+0.26	+0.14	+0.10
a_6	+0.090	+0.21	+0.085	+0.14	+0.091	+0.049
a_7	+0.060	+0.50	+0.060	+0.43	+0.11	+0.032
a_8	-0.090	+0.26	-0.090	-0.073	-0.062	-0.089
a_9	+0.060	+0.34	+0.060	+0.25	+0.096	+0.046
a_{10}	-0.090	+0.079	-0.090	-0.058	-0.057	-0.087

Figure 3.4: Loopy Trajectories with PD Control (— desired, - - - actual)

Figure 3.5: Straight-Line Trajectories with PD Control (— desired, - - - actual)

Figure 3.6: PD Control — Tracking Error for Loopy Path without Noise (— joint 1, - - - joint 2)

Figure 3.7: PD Control — Tracking Error for Straight Path without Noise (— joint 1, - - - joint 2)

Figure 3.8: PD Control — Tracking Error for Loopy Path with Noise (— joint 1, - - - joint 2)

Figure 3.9: PD Control — Tracking Error for Straight Path with Noise (— joint 1, - - - joint 2)

Figure 3.10: Indirect Adaptive Control — Tracking Error for Loopy Path without Noise (— joint 1, - - - joint 2)

Figure 3.11: Indirect Adaptive Control — Tracking Error for Loopy Path with Noise (— joint 1, - - - joint 2)

Figure 3.12: Indirect Adaptive Control — Parameter Estimates for Loopy Path without Noise

Figure 3.13: Indirect Adaptive Control — Parameter Estimates for Loopy Path with Noise

Figure 3.14: Direct Adaptive Control — Tracking Error for Loopy Path without Noise (— joint 1, - - joint 2)

Figure 3.15: Direct Adaptive Control — Parameter Estimates for Loopy Path without Noise

Figure 3.16: Direct Adaptive Control — Trend of Parameter Estimates for Loopy Path without Noise, over 50 seconds

Figure 3.17: Direct Adaptive Control — Parameter Estimates for Loopy Path without Noise, after 50 seconds

Figure 3.18: Indirect Adaptive Control — Tracking Error for Straight Path without Noise (— joint 1, - - - joint 2)

Figure 3.19: Indirect Adaptive Control — Parameter Estimates for Straight Path without Noise

Figure 3.20: Direct Adaptive Control — Tracking Error for Straight Path without Noise (— joint 1, - - - joint 2)

Figure 3.21: Direct Adaptive Control — Parameter Estimates for Straight Path without Noise

Figure 3.22: Direct Adaptive Control — Tracking Error for Straight Path without Noise, after 50 seconds (— joint 1, - - - joint 2)

Figure 3.23: Direct Adaptive Control — Trend of Parameter Estimates for Straight Path without Noise, over 50 seconds

Figure 3.24: Indirect Adaptive Control — Tracking Error for Straight Path with Noise (— joint 1, - - - joint 2)

Figure 3.25: Direct Adaptive Control — Tracking Error for Straight Path with Noise
(— joint 1, - - - joint 2)

Figure 3.26: Indirect Adaptive Control — Parameter Estimates for Straight Path with Noise

Figure 3.27: Direct Adaptive Control — Parameter Estimates for Straight Path with Noise

Figure 3.28: Indirect Adaptive Control — Tracking Error for Straight Path with Noise, after 50 seconds (— joint 1, - - - joint 2)

Figure 3.29: Direct Adaptive Control — Tracking Error for Straight Path with Noise, after 50 seconds (— joint 1, - - - joint 2)

Figure 3.30: Indirect Adaptive Control — Parameter Estimation for Straight Path with Noise, after 50 seconds

Figure 3.31: Direct Adaptive Control — Parameter Estimation for Straight Path with Noise, after 50 seconds

Chapter 4

Experiments

4.1 Robot Hardware Description

This section is a description of the robotic manipulator that was built and tested in the Robotics Lab of the Systems and Computer Engineering department at Carleton University. This robot was intended to provide the means for experimental validation of the analytical and simulation results discussed previously. The robot is designed as a high-speed direct drive manipulator, to serve as a flexible and state-of-the-art research tool. The mechanical design and construction of the robot was described by Rajguru [19]. The author of this thesis was responsible for design and implementation of the wiring, interfacing, software and ultimately the integration of the entire robot system.

The robot is a direct drive, two degree-of-freedom, manipulator which operates in the horizontal plane. The manipulator is a *Selective Compliance Assembly Robot Arm* (SCARA) type; it consists of a parallelogram linkage, which is capable of inertial decoupling of the two joints by tuning the masses of the links. Figure 4.1 is a diagram of the manipulator configuration, and Plate 4.1 is a photograph of the manipulator. The motors are Motion Control Systems, Inc. (MCS) direct drive brushless DC servo motors, equipped with tachometers for velocity sensing and resolvers for position sensing.

The robot motors are supplied with current by two independent MCS servo amplifiers, and they are powered by two MCS high voltage power supplies. The

commutation of the motors is performed electronically by the servo amplifiers using the resolver position signals.

An IBM AT-compatible personal computer equipped with a 80287 coprocessor and running at 12 Mhz is used to perform all the control and data acquisition functions for the robot. The computer is interfaced to the robot motors and servo amplifiers via two interface cards manufactured by Data Translation, Inc. An analog/digital and digital/analog conversion card acquires the analog tachometer signals and outputs the analog control voltages to the servo amplifiers. A digital/digital (parallel) interface card acquires the digital position signals and various status flags from the servo amplifiers. A patch panel with terminal blocks is used to interconnect the amplifiers, motors and computer interfaces. Figure 4.2 is a block diagram of the entire robot system, and Plate 4.2 shows the system as set up in the lab.

The specifications for the motors, amplifiers and interface cards can be found in Appendix E.

Figure 4.1: Direct Drive Manipulator Geometry

Figure 4.2: Robot System Block Diagram

Plate 4.1: Direct Drive Manipulator

Plate 4.2: Robot System Laboratory Setup

4.2 Control Software Description

All control and data acquisition software was written in the Turbo Pascal (Version 3.0) programming language, using the "Turbo-87" (numeric coprocessor) compiling option. The control software uses a simple sequential programming structure, employing the A/D card's built-in timer to synchronize the control loop timing to the desired frequency (to achieve a specific controller bandwidth). The A/D card is polled by the software until a converted analog measurement (the tachometer voltage signal) is ready. This programming strategy results in straightforward and easily modified code, but it is obviously not optimized for speed.

A simple program was written to acquire and display the current angular position of both motors of the robot, as a test of the position measurement function.

Numerous versions of a program to control Motor #2 of the robot (the top motor) were written and tested. These control programs fall under three broad categories:

1. standard PD control;
2. computed torque control using a pre-defined ARX model;
3. adaptive computed torque control by the Direct method.

Various combinations of options for each of these three types of control programs were implemented including:

1. user interface and data storage alternatives;
2. pre-filtering of the desired trajectory;
3. rejection of spurious position measurement points;
4. Kalman filtering of the position and velocity measurements;
5. capture of a "taught" trajectory to a data file, and its replay.

The control software design (for all versions of the software) consists of six main components:

1. program control and input/output;
2. desired trajectory generation;
3. acquisition of motor measurements (position,velocity);
4. Kalman filtering (if present);
5. dynamic parameter estimation (if present);
6. control algorithm.

Appendix D contains the software listing for the direct adaptive version of the Motor #2 control program.

4.3 Robot Integration and Functional Testing

After devising a design for interfacing all the components of the robot system and after completing the wiring, an extended period of time was spent in integrating the robot and testing its basic functionality. During this stage, a number of equipment “breaking-in” problems were discovered and corrected, such as:

- mis-labelling by the manufacturer of the connections on the power terminal block on one of the servo amplifier power supplies;
- failure of the logic power supplied to one of the servo amplifiers;
- failure of the digital output of the motor position output by the servo amplifier;

The direct drive, brushless DC servo motors used for the manipulator employ a three phase winding design. To obtain the correct torque from the motor as it rotates, the current to each of the three windings must be varied sinusoidally, and with a phase difference between the windings of exactly 120 degrees. It is the servo amplifier that provides the commutation of the currents for correct operation of the motor, using the position signal from the motor resolver. Adjustment of the resolvers is thus one of the tasks required to initially set up the motors.

The purpose of the resolver adjustment process is to align the resolver position such that correct commutation of the currents provided to the motor windings is obtained. This adjustment is done by carefully rotating the resolver stator within the motor housing, to obtain the correct angular alignment between the resolver and motor rotors and stators. Once the adjustment is complete, the resolver stator is clamped securely in place.

The resolver adjustment process proved to be a difficult job, and in fact at times did not result in correct operation of the motors. During testing of the motors, their performance would change over time (eg. system gain). It was discovered by disassembly and inspection of the motors that the motor rotors were not securely clamped on their shafts. This permitted relative motion of the motor and resolver parts, thus disturbing the commutation adjustments and affecting the motor operation. This problem was corrected by modifying the motor shaft assembly for a more secure and positive locking of the motor rotors in position.

Before integration of the robot, it was expected that electromagnetic noise would be a problem for the measurements, especially for the analog tachometer signals. In fact, as discussed previously, the noise problem is an essential element of the research described in this document. A brief discussion of the noise sources and effects follows.

The motor servo amplifiers are a Pulse Width Modulated (PWM) type, which operate at 20 KHz. This type of amplifier is known to be problematic in terms of generated Electro-Magnetic Interference (EMI) which occurs due to the 20 KHz switching of the current outputs. The sharp pulses of current produce high-frequency ringing in the power circuits; this ringing transmits EMI throughout the system wiring, which in turn generates noise voltages which are seen on the measurements acquired by the computer. The noise source and effects for the system are shown schematically in Figure 4.3.

The noise produced by the amplifiers can be considered as additive to both the measured position and velocity signals. Because the measurements are digitized, the effect of the noise is different from the analog case. For the analog velocity measurements which come from the tachometer, the noise produces an error in the

velocity reading due to an instantaneous noise voltage which adds to the "true" voltage;

The effect of this error is to produce random noise on the measured velocity signals which can be modeled as additive gaussian white noise. Measured velocity data taken from the robot shows typical values of the noise standard deviation of 10 to 15 degrees/second (with a zero mean). The appearance of this noise is shown for an example of velocity data, in Figure 4.4 (for a 0.33 Hz sine trajectory).

The effect of the noise on the position signals is different, since the measurement is a digital word provided by the resolver and its associated electronics. A spurious position measurement is produced when the instantaneous noise voltage on one or more bits of the TTL parallel interface between the amplifier and computer is sufficient to cross a switching threshold for the TTL circuit. This toggles bits of the position word and gives a faulty position value. This type of noise-induced error results in occasional spurious points in the measured position, which appear as obviously invalid spikes on the position record, as shown in Figure 4.5.

A further effect of the noise on the velocity and position measurements occurs due to the feedback employed by the control algorithms. The motor control algorithms implemented in the computer (such as a PD controller) use the position and velocity error signals (tracking errors) to generate a torque command to the motor. Since the velocity and position measurements are corrupted by noise, so in turn are the tracking errors. Hence, the generated torque command signal becomes noisy, and the noise propagates around the system again. This is evidenced by vibration of the motor. Thus, the inherent feedback in the controllers aggravates the noise problem.

With the noise problem in mind, care was taken in properly shielding all cables, providing good grounds for the signals and shields, and keeping digital, analog and power cables separate. A true differential mode of input for the A/D conversion of the tachometer signals was selected to minimize the effects of noise. It was observed that these measures succeeded in reducing the noise to the point where control of the motors can be achieved, while sufficient noise remains on the measurements to pose a challenge to the operation of the control algorithms of interest.

Figure 4.3: Noise Effects

Figure 4.4: Noise on Measured Velocity

While the tachometer constant (voltage output for angular velocity input, eg. volts/rpm) is a specification which is supplied by the manufacturer of the tachometer, it is well known that this is only a nominal figure which can have much variability between tachometers, and can also change as a function of motor assembly tolerances, temperature, etc. It is also understood that the quoted tachometer constant is only valid for a linearized working range of the tachometer. Therefore, it is necessary to calibrate the tachometers before they can be used. This was done for the tachometer for Motor #2 by acquiring velocity and position data from the motor in response to a range of motor inputs. This data was run through a Kalman filter by Richard Gourdeau (a Phd student in Mechanical Engineering), to estimate the tachometer constant. The resulting estimate of the tachometer constant for Motor #2 is 244 (deg/sec)/volt. This can be compared with the specification of 300 (deg/sec)/volt.

4.4 System Identification for Motor #2

A necessary first step that must be taken before the control of a system can be attempted is to obtain an accurate model of the system. The discussion which follows is a summary of the model structure selection and parameter estimation activities and results for Motor #2.

The system under consideration is the combination of Motor #2 and the links of the arm which are attached to it (i.e. the links which are seen as an effective inertia by the motor). To isolate possible effects of Motor #1 dynamics, the single link directed attached to Motor #1 was clamped securely so that no rotation of it was possible. The Motor #2 plus attached links system can be modelled in the continuous domain as follows:

$$T = i\ddot{y} + b\dot{y}, \quad (4.1)$$

where

T = motor torque (N-m)

y = motor angular position (radians)

i = effective inertia (N-m-sec²)

b = viscous damping (N-m/(rad/sec)).

Equation 4.1 can be rewritten using Laplace transforms as

$$T(s) = (s^2 i + sb)y(s), \quad (4.2)$$

which by rearranging can be expressed as

$$H(s) = \frac{y(s)}{T(s)} = \frac{1/i}{s(s + b/i)}, \quad (4.3)$$

where $H(s)$ is the system transfer function for motor torque as the input and motor position as the output. Thus, the open loop system is a linear first order system with an added pure integrator.

The motor torque can be expressed as

$$T(s) = k_a k_m u(s) \quad (4.4)$$

where

k_a = amplifier constant (amp/volt)

k_m = motor torque constant (N-m/amp)

$u(s)$ = servo amplifier input voltage (volts).

Therefore, the open loop system transfer function becomes

$$H_{ol}(s) = y(s)/u(s) = \frac{1}{s} \frac{k_a k_m / i}{(s + b/i)}. \quad (4.5)$$

By combining constants, the transfer function can be simplified as follows:

$$H_{ol}(s) = \frac{1}{s} \frac{k_r}{(s + a)}, \quad (4.6)$$

where

$k_r = k_a k_m / i$ = robot gain (1/(volt-sec²))

$a = b/i = 1/\tau_{ol}$ (sec⁻¹)

and τ_{ol} = open loop time constant (sec).

Thus, there are two parameters to estimate: k_r and a . It is assumed that the dynamics of the servo amplifier and motor can be completely characterized by constant gains (for the frequency range of interest).

The dynamic parameters were estimated from analysis of open loop response data. Step and sine wave input voltages were applied to the amplifier control input, and the response (angular position and velocity) was recorded. From the data, average values of the parameters were selected. The values initially selected from the gathered data for k_r and a were 3.0 and 1.1 respectively. Using the constants from the motor and servo amplifier specification sheets, the effective inertia, damping and open loop time constant can be derived as follows:

$$k_a = 1.0 \text{ amp/volt}$$

$$k_m = 0.442 \text{ N-m/amp}$$

$$i = k_a k_m / k_r = 0.15 \text{ N-m-sec}^2$$

$$b = ai = 0.16 \text{ N-m/(rad/sec)}$$

$$\tau_{ol} = 1/a = 0.9 \text{ sec.}$$

Figure 4.6 shows the block diagram for the model of the manipulator system with PD feedback added. The closed loop transfer function for this system can be derived by making use of Equation 4.6 and the following property derived from system block diagram algebra:

$$H_d(s) = \frac{G(s)}{1 + G(s)}, \quad (4.7)$$

where $G(s)$, the open loop transfer function of the system including the feedback controller is

$$G(s) = G_c(s)H_{ol}(s), \quad (4.8)$$

and where

$$G_c(s) = k_p + k_v s = \text{PD controller transfer function}$$

$$H_{ol}(s) = \text{transfer function of plant being controlled}$$

$$k_p = \text{position feedback gain}$$

$$k_v = \text{velocity feedback gain.}$$

Figure 4.5: Noise on Measured Position

Figure 4.6: Closed Loop System Block Diagram

By substituting Equation 4.6 into Equation 4.8, and then into Equation 4.7, we get

$$H_d(s) = \frac{(k_p + k_v s)(\frac{1}{s} \frac{k_r}{s+a})}{1 + (k_p + k_v s)(\frac{1}{s} \frac{k_r}{s+a})}. \quad (4.9)$$

Rearranging, we end up with the following second order transfer function for the closed loop system with PD feedback:

$$\begin{aligned} H_d(s) &= \frac{y(s)}{y_d(s)} \\ &= \frac{k_r(k_p + k_v s)}{s^2 + (k_r k_v + a)s + k_r k_p} \end{aligned} \quad (4.10)$$

where $y_d(s)$ is the desired motor position.

Using this transfer function and the above parameter estimates, a family of root locus plots for closed loop PD control were constructed. The root-locus plots for two values of velocity feedback gain are shown in Figures 4.7 and 4.8.

Based on the root-locus plots, various values of position and velocity feedback gains were selected for testing the PD control of the motor #2 system. These tests gave confidence that the derived system model was valid, so gains could then be picked to produce the desired closed loop performance, employing parameters and performance measures for the standard form of a second order system.

A damping ratio value, ζ , of 0.7 was chosen to give good control (quick response with small overshoot). k_p was selected by experiment to give a good balance between quick response and amplification of noise which is fed back. The selected value of k_p was 5.7 (0.1 for position in degrees). This gives a value for the natural frequency of the closed loop system, ω_n , of 4.1 rad/sec (0.65 Hz). Using the chosen values of ζ and ω_n , the required value for k_v becomes 1.5 (0.026 for degrees). The step response for the resulting closed loop system is shown in Figure 4.9.

As discussed in the previous section, shifts in the relative positions of parts within the motor assemblies resulted in changes in the motor performance. This necessitated re-estimation of the open loop system parameters. The final parameter estimation was performed once again using Richard Gourdeau's Kalman filter code, as was done for the tachometer calibration. Measured data from various step and

Figure 4.7: Root Locus for $k_v = 0$ Figure 4.8: Root Locus for $k_v = 1$

sine response runs with the motor under PD control was used. The final estimated values are $k_v = 3.4$ and $a = 0.35$.

Aside from the use of the derived system model in selecting the feedback gains for the PD controller, the identified model of the system was used for four other purposes:

1. construction of a digital pre-filter for filtering the user-defined desired trajectory to ensure that it remains within the bandwidth of the closed-loop system — this is particularly important when dealing with model-based control approaches, to maintain stability;
2. construction of a Kalman observer for use in filtering out the noise from the motor velocity and position measurements;
3. derivation of an ARX model to be used for a model-based (computed torque) control algorithm;
4. selection of initial values of the parameter estimates for an adaptive computed torque control algorithm (using the direct method).

At this point it is useful to consider the accuracy of the derived system model. There are a number of potential sources of error between the (idealized) model and the real Motor #2 plus links system. Most of these potential errors are a result of employing a strictly linear model structure for a real system which must necessarily include significant non-linearities. There are at least two major non-linearities:

1. there are noticeable non-linear friction and damping effects such as stiction (as evidenced by the motor dead-zone), as opposed to the linear damping model;
2. the system gain is not identical in the two rotational directions i.e. the effective torque constant of the motor is higher in one direction than the other.

Another potential source of error is the extreme sensitivity of the system performance to the resolver adjustment. Any drift in the adjustment can result in a large change in the system model.

Without employing a more complex (non-linear) model, the above-mentioned errors must be accepted. In fact, the results show that the simple linear model for Motor #2 is an adequate representation of reality for the purposes of the research described here. Care in adjustment of the motor assembly to prevent performance drift and periodic rechecking of the parameter estimates minimizes the chance of problems from the error due to resolver adjustment.

4.5 Test Run Description

Table 4.1 shows the tests that were run on Motor #2, and their main characteristics.

The control software was run at the highest possible sampling frequency (i.e. controller bandwidth), limited by the time required within each cycle for computation, A/D conversion and other input/output. When the maximum usable sampling frequency was exceeded, an error flag was raised for the A/D conversion (the interface card detected that the last sample had not been completely acquired before the timer triggered the next sample conversion). The sampling frequency ranged from 120 Hz to 200 Hz depending on the control algorithm. Even at the lowest sampling frequency, the controller bandwidth is well above that of the closed loop system (eg. 0.65 Hz), so no noticeable detrimental effect of the digital implementation of the controller was expected.

For the control programs using pre-filtering of the desired trajectory, a digital filter was constructed using a state-space formulation of a second order filter. In addition to filtering the desired trajectory, the form of the pre-filter provides the means to handle the initial conditions of the motor; specifically, if the actual initial position of the joint is far from the desired initial position, a smooth trajectory between the two is generated. The dynamics of the pre-filter were chosen to be critically damped (two real equal poles), with a natural frequency equal to that of the closed-loop motor system being controlled. Thus, the natural frequency ranged between 4 and 13 rad/sec.

The following is a brief description of the derivation of the digital pre-filter. First, let us formulate the continuous domain model of a constant coefficient, linear,

second order filter in state-space form:

$$\dot{\tilde{y}}_d = \begin{bmatrix} \dot{y}_d \\ \tilde{y}_d \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -\alpha & -\beta \end{bmatrix} \begin{bmatrix} y_d \\ \dot{y}_d \end{bmatrix} + \begin{bmatrix} 0 \\ K \end{bmatrix} y_r, \quad (4.11)$$

where

α, β, K = filter constants

y_r = "reference" position to be pre-filtered

y_d = pre-filtered, desired position

The continuous domain transfer function for the prefilter is

$$\frac{y_d}{y_r} = \frac{K}{s^2 + \beta s + \alpha}. \quad (4.12)$$

From Equation 4.12, the dynamics of the prefilter can be selected: from the characteristic equation, $s^2 + \beta s + \alpha = 0$, the poles can be set by picking appropriate α and β ; the steady state gain can be set to 1 by setting $K/\alpha = 1$. For example, for critical damping, the natural frequency equal to 13 radians/second, and steady state gain equal to 1, the constants become:

$$\alpha = 169$$

$$\beta = 26$$

$$K = 169$$

The discrete form of the state-space equations for the pre-filter given by Equation 4.11 is

$$\begin{bmatrix} y_d(t+1) \\ \dot{y}_d(t+1) \end{bmatrix} = \begin{bmatrix} a_1 & a_2 \\ a_3 & a_4 \end{bmatrix} \begin{bmatrix} y_d(t) \\ \dot{y}_d(t) \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix} y_r(t), \quad (4.13)$$

where the a 's and b 's are the discrete filter coefficients (which form the constant matrices A and B respectively). Equivalently, the discrete pre-filter can be written as a pair of difference equations:

$$y_d(t+1) = a_1 y_d(t) + a_2 \dot{y}_d(t) + b_1 y_r(t) \quad (4.14)$$

$$\dot{y}_d(t+1) = a_3 y_d(t) + a_4 \dot{y}_d(t) + b_2 y_r(t) \quad (4.15)$$

There are a number of well-known methods of deriving the discrete filter coefficients from the linear, constant coefficient, continuous domain filter description. One straightforward way is to transform the continuous filter constant matrices A and B to the discrete domain equivalents, A_d and B_d , using the following equations:

$$A_d = e^{AT} \quad (4.16)$$

$$B_d = \left(\int_0^T e^{A\lambda} d\lambda \right) B \quad (4.17)$$

where T is the sampling interval. For the example mentioned above ($\omega_n = 13$ rad/sec), and with sampling at 150 Hz ($T = 0.0067$ sec) the discrete filter coefficients become

$$a_1 = 0.9965$$

$$a_2 = 0.0061$$

$$a_3 = -1.0331$$

$$a_4 = 0.8375$$

$$b_1 = 0.0035$$

$$b_2 = 1.0331.$$

The control programs employing the Kalman filtering of the measurements were constructed using a formulation of a Kalman observer supplied by Richard Gourdeau. No attempt is made here to describe the derivation or implementation of the Kalman filter, as it was used simply as a “black box” in the control programs. See Appendix D for a listing of the control software for Motor #2 which uses a Kalman filter.

The computed torque control program employs an ARX model with two autoregressive terms and one input term, corresponding to a first order system with an added pure integrator. The model was derived as a discrete-time version of the identified continuous time model (using state-space methods), as is as follows:

$$y(t) = a_1 y(t-1) + a_2 y(t-2) + b_1 u(t-1) \quad (4.18)$$

where

$$a_1 = 1.9983$$

$$a_2 = -0.9983$$

$$b_1 = 0.0170$$

$$u(t) = \text{input (volts)}$$

$$y(t) = \text{output position (degrees)}$$

When the ARX model is reformulated to serve as a computed torque control law, the resulting expression for the control torque input is as follows (refer to Section 2.4):

$$u_c(t) = \frac{y^*(t+1) - a_1 y_d(t) - a_2 y_d(t-1)}{b_1}, \quad (4.19)$$

where

$$y^*(t+1) = y_d(t+1) + k_p e(t) + k_v \dot{e}(t).$$

refer to program listing in Appendix

The direct adaptive control program uses the algorithm that was described in Section 2.5. For the case of motor #2, the matrix equation of motion (equation 2.13) simplifies to the linear, scalar case given by Equation 4.1. The initial estimates of the system parameters used for the adaptive control runs were calculated from the system identification work for Motor #2 described above. Both parameter #1 and #2 (the two elements of the θ vector) were initialized to 0.006. Appendix D contains a listing of the control software for Motor #2 which uses direct adaptive control.

Figure 4.9: Closed Loop Step Response

4.6 Test Results

A comparison of step responses with low and high gains is shown in Figure 4.10. The response for the low gain case is for $k_p=0.1$ and $k_v=0.026$. The high gain response in Figure 4.10 is for $k_p=1.0$ and $k_v=0.12$. The natural frequency of the closed loop system is calculated to be approximately 2.1 Hz for this case, versus approximately 0.65 Hz for the previous (lower) set of gains (while maintaining $\zeta = 0.7$). As expected, the high gain system response shows the quicker rise time and smaller steady state errors.

The use of a pre-filter to generate the trajectory to be followed by the controller is illustrated in Figure 4.11. A simple step trajectory becomes the smooth trajectory when pre-filtered, as shown. A typical response of the closed-loop system to the pre-filtered trajectory when under PD control is also shown. The pre-filtering and response for a 0.25 Hz sine trajectory is shown in Figure 4.12.

A comparison of the tracking error with and without computed torque control for the pre-filtered step and sine responses are shown in Figures 4.13 and 4.14 respectively. It can be seen that the computed torque controller achieves some reduction in tracking errors for the step input, but not for the sine input case. The conclusion that can be drawn is that the fixed model used in the computed torque algorithm (derived from the off-line identification described previously in Sections 4.4 and 4.5) does provide a useful feedforward signal for the step case, but not for the sine case. In other words, the identified model is valid to some degree for the step but not for the sine. This is reasonable, because the ARX model used in the computed torque controller was derived from response data for step inputs¹. This result contributes to the justification for using *adaptive* computed torque control.

The performance of the Direct adaptive controller as tested was very poor. Figure 4.15 shows the desired and actual trajectories for a 0.7 Hz sine, and Figure 4.16 shows the corresponding parameter estimates. Several straightforward observations can be made:

¹Response data from the Kalman filtered PD controller with square wave inputs (i.e. repeated steps) at 0.25 and 0.5 Hz and with a 40 degree amplitude was used to identify the ARX model.

1. tracking error is very large;
2. convergence of the parameter estimates is not apparent;
3. the dynamics of the parameter adaptation are oscillatory.

Figures 4.17 and 4.18 show similar results for a longer duration test run under the same conditions. In both cases the adaptation gain, Γ , was set to a very low value (1×10^{-10}) to prevent unstable behaviour of the controller.

The test results for the direct adaptive controller are largely consistent with results (for the identifiable trajectory) from the simulations. The simulations predicted slow convergence for the direct method because of the need to limit the adaptation gain to small values to prevent unstable behaviour. One significant difference between the tests and simulations are the strict physical limits imposed upon the motion of the joints of the real robot, versus the simulation; i.e., the simulation is more forgiving since very large tracking errors are not prohibited, whereas for the real hardware very large tracking errors must necessarily produce motion which is out-of-limits. Another difference is the non-linear behaviour which exists in the real robot. This may explain the large offset between the desired and actual trajectories (approximately 45 degrees) seen in Figure 4.15 — higher gain of the motor in the negative rotational direction than the positive direction results in a bias when applying the (linear and hence symmetric) computed torque model. This bias is only counteracted when the proportional feedback becomes large when the tracking error becomes large.

Table 4.1: Test Run Matrix

Run number	1	2	3	4	5	6	7	8	9
Control type:									
-PD	X								
-PD w/Pre-filter				X		X			
-PD w/Kalman filter		X	X						
-Computed Torque					X		X		
-Direct Adaptive								X	X
Gains:									
-Position Feedback	0.1	0.1	1.0	0.1	0.1	0.1	0.1	0.1	0.1
-Velocity Feedback	0.026	0.026	0.12	0.026	0.026	0.026	0.026	0.026	0.026
-Adapt.Gain(Γ)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	1×10^{-10}	1×10^{-10}
Path:									
-Single Step				X	X				
-Square Wave	0.25Hz	0.25Hz	0.25Hz						
-Sine						0.25Hz	0.25Hz	0.7Hz	0.7Hz
Sampling Freq.(Hz)	200	200	200	200	200	200	200	120	120
Length of Run (sec)	6	6	6	5	5	9	9	30	60

N/A = Not Applicable

2

OF/DE

2

MICRO

Figure 4.10: Motor #2 Square Wave Response ($\cdots$ desired, $—$ low gain, $- - -$ high gain)

Figure 4.11: Motor #2 Step Response with Pre-filtered Input ($\cdots$ desired, $- -$ pre-filtered desired, $—$ actual)

Figure 4.12: Motor #2 Sine Response with Pre-filtered Input ($\cdots$ desired, $--$ pre-filtered desired, $—$ actual)

Figure 4.13: Tracking Error for Pre-filtered Step Input

Figure 4.14: Tracking Error for Pre-filtered Sine Input

Figure 4.15: Motor #2 Sine Response with Direct Adaptive Controller (- - - desired, — actual)

Figure 4.16: Parameter Estimates from Direct Adaptive Controller (— parameter #1 = i , - - - parameter #2 = b)

Figure 4.17: Sine Response with Direct Adaptive Controller, longer run (- - - desired, — actual)

Figure 4.18: Parameter Estimates from Adaptive Controller, longer run (— parameter #1 = i , - - - parameter #2 = b)

Chapter 5

Conclusions

5.1 Discussion and Conclusions

The research described in this thesis demonstrated by simulation how adaptive control laws can lead to inferior performance when the richness condition is not satisfied. A non-identifiable (straight line) desired trajectory was chosen and a realistic level of gaussian white noise was added to the velocity and acceleration measurements. Under such conditions the adaptive control algorithms performed worse than the PD controller alone. Errors in parameter estimation, even after convergence, caused the manipulator to deviate from its desired path. For a non-identifiable path, controllers using either PD alone, direct adaptation or indirect adaptation all have approximately the same performance after parameter convergence. Prior to convergence of the parameter estimates, adaptive control of the manipulator produced unacceptably large errors. In fact, operating a manipulator under such conditions could be dangerous.

When comparing the direct and indirect algorithms for parameter identification, it is evident that the direct method that was investigated needs improvement. Specifically, the choice of gain in the direct algorithm and the resulting rates of parameter convergence require further study. It has been found that, in particular, for non-exciting trajectories with noise on the velocity and acceleration measurements, the tracking performance after apparent convergence of the parameters was still no better than a simple PD controller. For the case of the indirect algorithm, use of the

well-known and robust least squares algorithm results in more confidence in the convergence qualities, and the convergence is expected to be quicker. However, it has been shown that the non-identifiable trajectory with noise results in performance degradation for the indirect adaptive controller as well.

A further comparison of the direct and indirect algorithms regarding their means of convergence was also discussed previously (see Section 3.5). Because the tracking errors and not prediction errors are used to drive the parameter adaptation for the direct method, after convergence of the parameter values, switching to a different trajectory may invalidate the parameter estimates and result in large tracking errors.

Another accomplishment of the research described herein was the successful integration of a two degree-of-freedom, direct drive manipulator system. A range of algorithms for control of the motion of a single joint were tested. These experiments confirmed the previously discussed short-comings of the direct adaptive control method which were predicted from the simulation results. To prevent instability, the parameter adaptation gains had to be kept low, so the parameter convergence performance was poor.

An important point which becomes obvious when running an experimental implementation (versus a simulation) of an adaptive controller is that the large tracking errors experienced prior to parameter convergence are not acceptable. This is because very large tracking errors mean that the manipulator joints reach their physical limits. This has serious implications for the practical use of direct adaptive controllers. The parameter adaptation of the indirect adaptive algorithm can initially be run off-line (i.e. while the manipulator is being controlled along some path by a simple non-adaptive controller, such as a PD controller) until the parameters have converged and it is desirable to enable the adaptive control. In contrast, the direct adaptive algorithm cannot be run off-line since it is driven by tracking error and not prediction error. This is an inherent disadvantage of the direct method.

5.2 Recommendations

Further experimental work is suggested to validate the simulation results discussed in this thesis. The following research steps are called for:

1. implementation of simultaneous two joint control for the Carleton direct drive manipulator;
2. implementation of direct and indirect adaptive control for two joints;
3. experiments using identifiable and non- identifiable trajectories.

It is also recommended that improved direct algorithms be investigated; the algorithm published by Slotine and Li [9] is a prime candidate.

Assuming that the conclusions of this thesis are confirmed by the further investigations and experimentation, it is recommended that research be undertaken to investigate the means of avoiding the problems caused for adaptive control algorithms when the richness condition is not satisfied. Reduction of the inertial model used by the adaptive control laws by eliminating the problematic terms is a promising direction for this research.

Bibliography

- [1] Middleton, R.H., and Goodwin, G.C., "Adaptive Computed Torque Control for Rigid Link Manipulators", *Proceedings of 25th IEEE Conference on Decision and Control*, Athens, Greece, December 1986.
- [2] Hsu, P., Bodson, M., Sastry, S., and Paden, B., "Adaptive Identification and Control for Manipulators without Using Joint Accelerations", *Proceedings of the IEEE Conference on Robotics and Automation*, 1987.
- [3] Atkeson, C.G., An, C.H., and Hollerbach, J.M., "Estimation of Inertial Parameters of Manipulator Loads and Links", *Proceedings of the Third International Symposium on Robotics Research*, October 1986.
- [4] Li, W., and Slotine, J-J.E., "Parameter Estimation Strategies for Robotic Applications", *Proceedings of the ASME Winter Annual Meeting*, 1987.
- [5] Li, W., and Slotine, J-J.E., "Indirect Adaptive Robot Control", *Proceedings of the IEEE Conference on Robotics and Automation*, 1988.
- [6] Craig, J.J., *Introduction to Robotics — Mechanics and Control*, Addison-Wesley, Reading, Massachusetts, 1986.
- [7] Craig, J.J., Hsu, P., and Sastry, S.S., "Adaptive Control of Mechanical Manipulators", *International Journal of Robotics Research*, Vol.6, No.2, 1987, pp.16-28.
- [8] Slotine, J-J.E., and Li, W., "On the Adaptive Control of Robot Manipulators", *International Journal of Robotics Research*, Vol.6, No.3, Fall 1987, pp.49-59.

- [9] Slotine, J.-J.E., and Li, W., "Adaptive Manipulator Control: A Case Study", *IEEE Transactions on Automatic Control*, Vol.33, No.11, November 1988, pp.995-1003.
- [10] Ljung, L., and Söderström, T., *Theory and Practice of Recursive Identification*, MIT Press, Cambridge, Massachusetts, 1983.
- [11] Ljung, L., *System Identification: Theory for the User*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.
- [12] Schwartz, H.M., and Youcef-Toumi, K., "A Recursive Identification Controller for Manipulator Trajectory Tracking", *Proceedings of the ASME Winter Annual Meeting*, December 1987.
- [13] Schwartz, H.M., and Youcef-Toumi, K., "A Recursive Identification Controller for Trajectory Tracking", *Proceedings of the American Control Conference*, 1987.
- [14] Asada, H., Kanade, T., and Takeyama, I., "Control of a Direct-Drive Arm", *Transactions of the ASME, Journal of Dynamic Systems, Measurement and Control*, Vol.105, September 1983, pp.136-142.
- [15] Koivo, A.J., and Guo, T.-H., "Adaptive Linear Controller for Robotic Manipulators", *IEEE Transactions on Automatic Control*, Vol.AC-28, No.2, February 1983, pp.162-171.
- [16] Bierman, G.J., *Factorization Methods for Discrete Sequential Estimation*, Academic Press, Orlando, Florida, 1977.
- [17] Khosla, P.K., and Kanade, T., "Experimental Evaluation of Nonlinear Feedback and Feedforward Control Schemes for Manipulators", *The International Journal of Robotics Research*, Vol.7, No.1, February 1988.
- [18] Forsythe, G.E., Malcolm, M.A., and Moler, C.B., *Computer Methods for Mathematical Computations*, Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

- [19] Rajguru, S., *Analysis and Design of a Direct Drive Robot*, Master's Thesis, Carleton University, Ottawa, Ontario, 1989.
- [20] Dubowsky, S., and DesForges, D.T., "The Application of Model-Referenced Adaptive Control to Robotic Manipulators", *Journal of Dynamic Systems, Measurement and Control*, Vol.101, September 1979.
- [21] Horowitz, R., and Tomizuka, M., "An Adaptive Control Scheme for Mechanical Manipulators — Compensation of Nonlinearity and Decoupling Control", *Journal of Dynamic Systems, Measurement and Control*, Vol.108, June 1986.
- [22] Schwartz, H.M., and Warshaw, G., "On the Richness Condition for Robot Adaptive Control", *Proceedings of the ASME Winter Annual Meeting*, DSC-Vol.14, December 1989, pp.43-49.

Appendix A

Derivation of Recursive Least Squares

The following derivation of the Recursive Least Squares (RLS) algorithm is drawn from Ljung and Söderström [10].

The starting point for the derivation of a recursive algorithm is the formula for the standard least squares estimate which was presented in Section 2.1, as Equation 2.6. For convenience, this is repeated here:

$$\hat{\theta}(N) = \left[\sum_{t=1}^N \alpha_t \phi(t) \phi^T(t) \right]^{-1} \sum_{t=1}^N \alpha_t \phi(t) x(t). \quad (\text{A.1})$$

Define the matrix

$$\bar{R}(t) = \sum_{k=1}^t \alpha_k \phi(k) \phi^T(k). \quad (\text{A.2})$$

Combining Equations A.1 and A.2 and setting $t = k$ and $N = t - 1$ gives the following:

$$\sum_{k=1}^{t-1} \alpha_k \phi(k) x(k) = \bar{R}(t-1) \hat{\theta}(t-1). \quad (\text{A.3})$$

From Equation A.2 it also follows that

$$\bar{R}(t-1) = \bar{R}(t) - \alpha_t \phi(t) \phi^T(t). \quad (\text{A.4})$$

and similarly

$$\bar{R}(t) = \bar{R}(t-1) + \alpha_t \phi(t) \phi^T(t). \quad (\text{A.5})$$

Rewriting Equation A.1 by setting $N = t$ and $t = k$:

$$\hat{\theta}(t) = \left[\sum_{k=1}^t \alpha_k \phi(k) \phi^T(k) \right]^{-1} \sum_{k=1}^t \alpha_k \phi(k) x(k). \quad (\text{A.6})$$

Substituting in $\bar{R}(t)$ using Equation A.2 and splitting the right summation we get

$$\hat{\theta}(t) = \bar{R}^{-1}(t) \left[\sum_{k=1}^{t-1} \alpha_k \phi(k)x(k) + \alpha_t \phi(t)x(t) \right], \quad (\text{A.7})$$

which can be written, by substituting into Equation A.3, as

$$\hat{\theta}(t) = \bar{R}^{-1}(t) [\bar{R}(t-1)\hat{\theta}(t-1) + \alpha_t \phi(t)x(t)]. \quad (\text{A.8})$$

Employing Equation A.4 we get

$$\hat{\theta}(t) = \bar{R}^{-1}(t) (\bar{R}(t)\hat{\theta}(t-1) + \alpha_t \phi(t) [-\phi^T(t)\hat{\theta}(t-1) + x(t)]). \quad (\text{A.9})$$

This result can be reduced to the following expression by rearranging terms and using the vector property that $\hat{\theta}^T \phi = \phi^T \hat{\theta}$:

$$\hat{\theta}(t) = \hat{\theta}(t-1) + \bar{R}^{-1}(t) \phi(t) \alpha_t [x(t) - \hat{\theta}^T(t-1)\phi(t)]. \quad (\text{A.10})$$

If we define a matrix $R(t) = \frac{1}{t} \bar{R}(t)$, it can be shown that the following relationship holds:

$$R(t) = R(t-1) + \frac{1}{t} [\alpha_t \phi(t) \phi^T(t) - R(t-1)]. \quad (\text{A.11})$$

The pair of equations A.10 and A.11 define the desired recursive algorithm for the estimation of θ . The current estimate (at discrete time instant t), $\hat{\theta}(t)$, is calculated from the last value, $\hat{\theta}(t-1)$ and current values $R(t)$, $\phi(t)$ and $x(t)$. $R(t)$ is in turn calculated from its last value $R(t-1)$ and the current value $\phi(t)$.

Since the calculation of $\hat{\theta}(t)$ using $R(t)$ requires a matrix inversion ($\bar{R}^{-1}(t)$), it is preferable from the computational point of view to introduce a new quantity which can be used instead without requiring the inversion. Hence, the derivation of Equation A.11 will not be described here. Instead, let us define the covariance matrix, $P(t)$, as follows:

$$P(t) = \bar{R}^{-1}(t). \quad (\text{A.12})$$

$P(t)$ must be updated directly to avoid the matrix inversion of Equation A.10. To derive the update law for $P(t)$, the following matrix inversion lemma is used (without further proof), from Ljung and Söderström [10]:

Lemma A.1 (Matrix Inversion) *Let A , B , C and D be matrices of compatible dimensions, so that the product BCD and the sum $A + BCD$ exist. Then,*

$$[A + BCD]^{-1} = A^{-1} - A^{-1}B[DA^{-1}B + C^{-1}]^{-1}DA^{-1}. \quad (\text{A.13})$$

First, combining Equations A.5 and A.12 we get

$$\begin{aligned} P(t) &= [\hat{R}(t-1) + \alpha_t \phi(t) \phi^T(t)]^{-1} \\ &= [P^{-1}(t-1) + \phi(t) \alpha_t \phi^T(t)]^{-1}. \end{aligned} \quad (\text{A.14})$$

Applying Lemma A.1 with $A = P^{-1}(t-1)$, $B = \phi(t)$, $C = \alpha_t$ and $D = \phi^T(t)$ gives

$$P(t) = P(t-1) - P(t-1)\phi(t) \left[\phi^T(t)P(t-1)\phi(t) + \frac{1}{\alpha_t} \right]^{-1} \phi^T(t)P(t-1). \quad (\text{A.15})$$

Since the term in the square brackets is a scalar, Equation A.15 can be rearranged to form the following:

$$P(t) = P(t-1) - \frac{P(t-1)\phi(t)\phi^T(t)P(t-1)}{1/\alpha_t + \phi^T(t)P(t-1)\phi(t)}. \quad (\text{A.16})$$

Equation A.10 can be restated in terms of $P(t)$ as

$$\hat{\theta}(t) = \hat{\theta}(t-1) + P(t)\phi(t)\alpha_t[x(t) - \hat{\theta}^T(t-1)\phi(t)]. \quad (\text{A.17})$$

Thus, Equations A.16 and A.17 define the parameter adaptation law required for the RLS algorithm.

The preceding derivation can be extended to arrive at the Exponentially Forgetting RLS.

Consider a modification of the cost function defined previously by Equation 2.5, so that older measurements are discounted, instead of using the average behaviour over the entire time interval $1 \leq t \leq N$ as before. Thus, define a new cost function which is representative of the current character of the system (i.e., at time N) as follows:

$$\tilde{V}_N(\theta) = \sum_{t=1}^N \tilde{\beta}(N, t) [x(t) - \theta^T \phi(t)]^2, \quad (\text{A.18})$$

where $\tilde{\beta}(N, t)$ is a function which increases with t for a given N , i.e., discounts earlier measurements. $\tilde{V}_N(\theta)$ remains quadratic in θ , as for $V_N(\theta)$, so that the estimate of

θ which minimizes $\tilde{V}_N(\theta)$ can be analytically derived as for Equation 2.6, to arrive at:

$$\hat{\theta}(N) = \left[\sum_{t=1}^N \tilde{\beta}(N, t) \phi(t) \phi^T(t) \right]^{-1} \sum_{t=1}^N \tilde{\beta}(N, t) \phi(t) x(t). \quad (\text{A.19})$$

Equation A.19 represents the off-line estimate of θ . To permit recursive computation of $\hat{\theta}$, the following structure for $\tilde{\beta}(N, t)$ must be assumed:

$$\tilde{\beta}(N, t) = \lambda(t) \tilde{\beta}(t-1, k), \quad 1 \leq k \leq t-1 \quad (\text{A.20})$$

Equivalently,

$$\tilde{\beta}(t, k) = \left[\prod_{j=k+1}^t \lambda(j) \right] \alpha_t, \quad (\text{A.21})$$

where $\tilde{\beta}(k, k) = \alpha_k$.

$\lambda(t)$ can be assumed to be a constant, i.e., $\lambda(k) = \lambda$ for all k (typically $\lambda \leq 1$). In this case, Equation A.21 becomes

$$\tilde{\beta}(t, k) = \lambda^{t-k} \alpha_k. \quad (\text{A.22})$$

This form of $\tilde{\beta}(t, k)$ gives the *exponentially forgetting* profile when employed in the cost function of Equation A.18. λ is termed the *forgetting factor* in this case.

Define a new quantity $\tilde{R}(t)$, which is Equation A.2 with $\tilde{\beta}(t, k)$ substituted for α_k :

$$\tilde{R}(t) = \sum_{k=1}^t \tilde{\beta}(t, k) \phi(k) \phi^T(k). \quad (\text{A.23})$$

Using a derivation similar to that for Equation A.5, we get,

$$\tilde{R}(t) = \lambda \tilde{R}(t-1) + \alpha_t \phi(t) \phi^T(t). \quad (\text{A.24})$$

It can then be shown by following the steps for the derivation of Equations A.6 to A.10 that the update law for $\hat{\theta}(t)$ is identical to the previous result, Equation A.10.

Once again, by introducing $P(t) = \tilde{R}^{-1}(t)$ and applying Lemma A.1, we get

$$P(t) = \frac{1}{\lambda} P(t-1) - \frac{1}{\lambda} P(t-1) \phi(t) \left[\phi^T(t) \frac{1}{\lambda} P(t-1) \phi(t) + \frac{1}{\alpha_t} \right]^{-1} \phi^T(t) P(t-1) \frac{1}{\lambda}. \quad (\text{A.25})$$

Rearranging, we get

$$P(t) = \frac{1}{\lambda} \left[P(t-1) - \frac{P(t-1)\phi(t)\phi^T(t)P(t-1)}{[\lambda/\alpha_t] + \phi^T(t)P(t-1)\phi(t)} \right]. \quad (\text{A.26})$$

The pair of equations A.17 and A.26 define the parameter adaptation law required for the exponentially forgetting RLS algorithm. It can be seen that setting $\lambda = 1$ reduces Equation A.26 to Equation A.16, i.e., with $\lambda = 1$ the exponentially forgetting RLS reduces to the standard RLS.

Appendix B

Manipulator Equations of Motion

The equations of motion for the two degree-of-freedom serial link manipulator used in the simulations can be found in Schwartz and Youcef-Toumi [12]. The derivation of these equations of motion using Lagrange's Equations has been repeated and is detailed below.

The key assumptions used to derive the dynamics are as follows:

1. link masses are evenly distributed, so that the link C of G is at the geometric center;
2. the link inertias can be modeled as equivalent to that of thin rods;
3. the links can be considered as rigid bodies;
4. there is no effect of gravity, i.e., the manipulator moves in the horizontal plane;
5. there is no friction or backlash.

Lagrange's Equations for a rotational system can be stated as,

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_i} \right) - \frac{\partial \mathcal{L}}{\partial \theta_i} = \tau_i, \quad (\text{B.1})$$

where

$i = 1$ to number of rotational joints

θ_i = angle of i^{th} joint

τ_i = torque for i^{th} joint.

The Lagrangian is defined as,

$$\mathcal{L} = T^* - \mathcal{V} \quad (\text{B.2})$$

where

T^* = kinetic co-energy

$\mathcal{V}$ = potential energy.

For horizontal motion, the $\mathcal{V}$ term can be neglected (no change in the potential energy of the system occurs due to its motion). Hence, Equation B.2 reduces to,

$$\mathcal{L} = T^*. \quad (\text{B.3})$$

Figure B.1 defines the geometry and coordinate systems for the derivation which follows.

Figure B.1: Manipulator Geometry and Coordinates

The first step is to calculate the kinetic co-energy term, T^* . T^* consists of the kinetic co-energy for:

- translation of the link C of G's;
- rotation of the links about their C of G's.

Therefore, T^* for both links is defined as follows:

$$\begin{aligned} T^* &= T_{\text{translation}} + T_{\text{rotation}} \\ &= T_{t_1} + T_{t_2} + T_{r_1} + T_{r_2} \end{aligned} \quad (\text{B.4})$$

The kinetic co-energy for translation of a mass (in this case, of the link C of G) is

$$T_t = \frac{1}{2}mv^2, \quad (\text{B.5})$$

where m is the mass and v is the translational velocity.

Considering first Joint 1,

$$\begin{aligned} v_1 &= r_1 \dot{\theta}_1 \\ &= \frac{l_1}{2} \dot{\theta}_1 \end{aligned} \quad (\text{B.6})$$

where r is the radius from the center of rotation to the C of G, and $\dot{\theta}$ is the angular velocity.

Hence, using Equations B.5 and B.6 we get

$$T_{t_1} = \frac{1}{8}m_1 l_1^2 \dot{\theta}_1^2. \quad (\text{B.7})$$

The derivation of T_{t_2} is more complex. To do the derivation, the use of vector quantities is introduced. Define

$$\vec{r}_2 = a\vec{i} + b\vec{k}, \quad (\text{B.8})$$

where $\vec{i}$ and $\vec{k}$ are the unit vectors in the x and z directions respectively. The following trigonometric identity will be used for the derivation:

$$\cos(\alpha - \beta) = \cos \alpha \cos \beta + \sin \alpha \sin \beta. \quad (\text{B.9})$$

The magnitudes of the two vector components of $\vec{r}_2$, a and b , can be shown by simple geometry and using the trigonometric identity above to be:

$$a = l_1 \cos \theta_1 - \frac{l_2}{2} \cos(\theta_1 + \theta_2) \quad (\text{B.10})$$

$$b = l_1 \sin \theta_1 - \frac{l_2}{2} \sin(\theta_1 + \theta_2). \quad (\text{B.11})$$

Taking the time derivative of $\vec{r}_2$, we get:

$$\begin{aligned} \frac{d\vec{r}_2}{dt} = & [-l_1\dot{\theta}_1 \sin \theta_1 + \frac{l_2}{2}(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2)]\vec{i} + \\ & [l_1\dot{\theta}_1 \cos \theta_1 - \frac{l_2}{2}(\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2)]\vec{k}. \end{aligned} \quad (B.12)$$

Using Equation B.12, v_2 can now be derived:

$$\begin{aligned} v_2^2 = & \left| \frac{d\vec{r}_2}{dt} \right|^2 \\ = & [-l_1\dot{\theta}_1 \sin \theta_1 + \frac{l_2}{2}(\dot{\theta}_1 + \dot{\theta}_2) \sin(\theta_1 + \theta_2)]^2 + \\ & [l_1\dot{\theta}_1 \cos \theta_1 - \frac{l_2}{2}(\dot{\theta}_1 + \dot{\theta}_2) \cos(\theta_1 + \theta_2)]^2. \end{aligned} \quad (B.13)$$

Collecting terms and once again making use of Equation B.9 and the additional trigonometric identity

$$\sin^2 \alpha + \cos^2 \alpha = 1, \quad (B.14)$$

we end up with

$$v_2^2 = l_1^2 \dot{\theta}_1^2 + \frac{l_2^2}{4}(\dot{\theta}_1 + \dot{\theta}_2)^2 - l_1 l_2 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \cos \theta_2. \quad (B.15)$$

Returning to Equation B.5, we now have

$$T_{t_2} = \frac{1}{2} m_2 l_1^2 \dot{\theta}_1^2 + \frac{1}{8} m_2 l_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 - \frac{1}{2} m_2 l_1 l_2 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \cos \theta_2. \quad (B.16)$$

The kinetic co-energy for rotation of a mass about its C of G is

$$T_r = \frac{1}{2} I \dot{\theta}^2, \quad (B.17)$$

where I is the moment of inertia and $\dot{\theta}$ is the rotational velocity. For a thin rod of length l and mass m , its inertia for "end-over-end" rotation is

$$I = \frac{1}{12} m l^2. \quad (B.18)$$

Considering first Link 1, by combining Equations B.17 and B.18 we get

$$T_{r_1} = \frac{1}{24} m_1 l_1^2 \dot{\theta}_1^2. \quad (B.19)$$

For Link 2, the angular velocity to be considered is the sum of $\dot{\theta}_1$ and $\dot{\theta}_2$. Hence, by the same method used for Link 1, we get

$$\mathcal{T}_2 = \frac{1}{24} m_2 l_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2. \quad (\text{B.20})$$

By combining the separate components (Equations B.7, B.16, B.19 and B.20) and collecting terms, the total kinetic co-energy for the manipulator becomes:

$$\begin{aligned} \mathcal{T}^* = & \frac{1}{6} m_1 l_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 l_1^2 \dot{\theta}_1^2 + \frac{1}{6} m_2 l_2^2 (\dot{\theta}_1 + \dot{\theta}_2)^2 - \\ & \frac{1}{2} m_2 l_1 l_2 \dot{\theta}_1 (\dot{\theta}_1 + \dot{\theta}_2) \cos \theta_2 \end{aligned} \quad (\text{B.21})$$

The next step is to evaluate Lagrange's Equations. Since we know that $\mathcal{L} = \mathcal{T}^*$, by using Equation B.21 and expanding terms we get

$$\begin{aligned} \mathcal{L} = & \frac{1}{6} m_1 l_1^2 \dot{\theta}_1^2 + \frac{1}{2} m_2 l_1^2 \dot{\theta}_1^2 + \frac{1}{6} m_2 l_2^2 \dot{\theta}_1^2 + \frac{1}{3} m_2 l_2^2 \dot{\theta}_1 \dot{\theta}_2 + \\ & \frac{1}{6} m_2 l_2^2 \dot{\theta}_2^2 - \frac{1}{2} m_2 l_1 l_2 \cos(\theta_2) \dot{\theta}_1^2 - \frac{1}{2} m_2 l_1 l_2 \cos(\theta_2) \dot{\theta}_1 \dot{\theta}_2. \end{aligned} \quad (\text{B.22})$$

Rewriting Lagrange's Equations for Joint 1 (from Equation B.1):

$$\frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} \right) - \frac{\partial \mathcal{L}}{\partial \theta_1} = \tau_1. \quad (\text{B.23})$$

Evaluating the derivative terms using Equation B.22, we get

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} = & \frac{1}{3} m_1 l_1^2 \dot{\theta}_1 + m_2 l_1^2 \dot{\theta}_1 + \frac{1}{3} m_2 l_2^2 \dot{\theta}_1 + \frac{1}{3} m_2 l_2^2 \dot{\theta}_2 - \\ & m_2 l_1 l_2 \cos(\theta_2) \dot{\theta}_1 - \frac{1}{2} m_2 l_1 l_2 \cos(\theta_2) \dot{\theta}_2 \\ \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_1} \right) = & \frac{1}{3} m_1 l_1^2 \ddot{\theta}_1 + m_2 l_1^2 \ddot{\theta}_1 + \frac{1}{3} m_2 l_2^2 \ddot{\theta}_1 + \frac{1}{3} m_2 l_2^2 \ddot{\theta}_2 - \\ & m_2 l_1 l_2 \cos(\theta_2) \ddot{\theta}_1 + m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1 \dot{\theta}_2 - \frac{1}{2} m_2 l_1 l_2 \cos(\theta_2) \ddot{\theta}_2 \\ & + \frac{1}{2} m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_2^2 \\ \frac{\partial \mathcal{L}}{\partial \theta_1} = & 0. \end{aligned}$$

Thus, by Equation B.23, the equation of motion for Joint 1 is:

$$\begin{aligned} \tau_1 = & \left(\frac{1}{3} m_1 l_1^2 + m_2 l_1^2 + \frac{1}{3} m_2 l_2^2 + m_2 l_1 l_2 \cos \theta_2 \right) \ddot{\theta}_1 + \\ & \left(\frac{1}{3} m_2 l_2^2 - \frac{1}{2} m_2 l_1 l_2 \cos \theta_2 \right) \ddot{\theta}_2 + m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1 \dot{\theta}_2 + \\ & \frac{1}{2} m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_2^2. \end{aligned} \quad (\text{B.24})$$

The above derivation can be repeated in a similar fashion to derive the equations of motion for Joint 2. Evaluating the derivative terms, we get

$$\begin{aligned}
 \frac{\partial \mathcal{L}}{\partial \dot{\theta}_2} &= \frac{1}{3}m_2 l_2^2 \dot{\theta}_1 + \frac{1}{3}m_2 l_2^2 \dot{\theta}_2 - \frac{1}{2}m_2 l_1 l_2 \cos(\theta_2) \dot{\theta}_1 \\
 \frac{d}{dt} \left(\frac{\partial \mathcal{L}}{\partial \dot{\theta}_2} \right) &= \frac{1}{3}m_2 l_2^2 \ddot{\theta}_1 + \frac{1}{3}m_2 l_2^2 \ddot{\theta}_2 - \frac{1}{2}m_2 l_1 l_2 \cos(\theta_2) \ddot{\theta}_1 + \\
 &\quad + \frac{1}{2}m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1 \dot{\theta}_2 \\
 \frac{\partial \mathcal{L}}{\partial \theta_2} &= \frac{1}{2}m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1^2 + \frac{1}{2}m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1 \dot{\theta}_2.
 \end{aligned}$$

Thus, the equation of motion for Joint 2 becomes

$$\begin{aligned}
 \tau_2 = & \left(\frac{1}{3}m_2 l_2^2 - \frac{1}{2}m_2 l_1 l_2 \cos \theta_2 \right) \ddot{\theta}_1 + \frac{1}{3}m_2 l_2^2 \ddot{\theta}_2 - \\
 & \frac{1}{2}m_2 l_1 l_2 \sin(\theta_2) \dot{\theta}_1^2.
 \end{aligned} \tag{B.25}$$

Appendix C

Simulation Software — Indirect Adaptive Control

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 102

```
program robot_simulation_10;
```

```
{Revised Feb.14 '89, by Gabriel Warshaw}
```

```
(* This program simulates the motion of a robotic  
manipulator under indirect adaptive control.  
The controller uses a law based on the  
computed torque method. A fourth order Runge-Kutta method  
is used to solve the nonlinear differential equations  
which model the manipulator's behaviour.
```

```
The program does Recursive Identification of  
the system parameters. These parameters are based  
on inertial terms. The identified  
parameters are used in the computed torque control law.*)>
```

```
type
```

```
vector4 = array[1..4] of real;  
vector11 = array[1..11] of real;  
matrix20 = array[1..20,1..20] of real;  
ivect11 = array[1..11] of integer;
```

```
var
```

```
y,      (* Robot joint angles and angular velocities *)  
f,      (* Derivatives of joint angles and angular velocities *)  
yd,     (* Desired robot joint angles and angular velocities one  
        time step into the future *)  
yd0,    (* Desired robot joint angles and angular velocities at  
        the present time *)  
yd1 :   (* Desired robot joint angles and angular velocities one  
        time step into the past *)  
        vector4;
```

```
c1,c2,  (* Criteria vector *)  
phi1,   (* Linear regression vector for motor 1 *)  
phi2,   (* Linear regression vector for motor 2 *)  
phis1,  (* Selected linear regression vector for motor 1 *)  
phis2,  (* Selected linear regression vector for motor 2 *)  
phid1,  (* Desired linear regression vector for motor 1 *)  
phid2,  (* Desired linear regression vector for motor 2 *)  
para1,  (* Parameter vector for motor 1 *)  
para2,  (* Parameter vector for motor 2 *)  
spar1,  (* Temporary storage for motor 1 parameter vector *)  
spar2:  (* Temporary storage for motor 2 parameter vector *)  
        vector11;
```

```
selpar1, (* Vector which contains the selected parameters *)  
selpar2 : (* Vector which contains the motor 2, parameters *)  
        ivect11;
```

```
p1,     (* cov. matrix for motor 1 params., UDU factorized*)  
p2,     (* cov. matrix for motor 2 params., UDU factorized*)  
cov1,   (* covariance matrix for motor 1 parameters *)  
cov2:   (* covariance matrix for motor 2 parameters *)  
        matrix20;
```

```
debug,  (* Matrix of data to be plotted for debugging *)  
data :  (* Matrix which contains the data to be plotted *)  
        array[1..300,1..8] of real;
```

```
t,      (* Time in simulation *)  
dt,     (* Time interval for control action and simulation *)  
ftime,  (* Final time of the simulation *)
```

```

w,      (* Input frequency to manipulator *)
y11,    (* Position of motor 1, one time step into the past *)
y12,    (* Position of motor 1, two time steps into the past *)
y21,    (* Position of motor 2, one time step into the past *)
y22,    (* Position of motor 2, two time steps into the past *)
yv1,    (* Velocity of motor 1, one time step into the past *)
yv2,    (* velocity of motor 2, one time step into the past *)
error1, (* Model prediction error for motor 1 *)
error2, (* Model prediction error for motor 2 *)
err1,   (* robot position error motor 1 *)
err2,   (* robot position error motor 2 *)
derri,  (* error in the velocity of motor 1 *)
derr2,  (* error in the velocity of motor 2 *)
deg_err1, (* Robot position error in degrees, motor 1 *)
deg_err2, (* Robot position error in degrees motor 2 *)
landa,  (* Recursive least squares forgetting factor *)
kp,kv,  (* Position and velocity feedback gains *)
u1nom,  (* Feedforward component of control input for motor 1 *)
u2nom,  (* Feedforward component of control input for motor 2 *)
u1in,   (* Control input for motor 1 *)
u2in,   (* Control input for motor 2 *)
a1,a2,a3,a4,a5,a6,a7,a8,a9,a10, (* constants representing moments
                                of inertia *)
yn1,yn2,yn3,yn4, {measurements with added noise}
cosyn3,sinyn3, {cos and sin functions of yn3}
y1star,y2star, {desired accel.of motors, feedback added}
yda1,yda2, {desired angular velocities for motor 1 & 2}
yd0a1,yd0a2, { " " " " ,last time step}
m11,m12,m21,m22, {elements of est. Mass matrix, M}
q1,q2, {elements of est.other torques vector, Q}
npoints: {number of output data points desired}
real;

rerun,mode,npar,counter,i,j,jj,select,nsel1,nsel2,
skip,skipcount: {number of time steps to skip for
                 data output,current skip count}
integer;

indata,partrend,outpmtx,
outdata,outpara,outcov,xdata: {file assignment names}
text;

noise: boolean;
choice,yes: char;

(*****)
procedure error(errno,erraddr:integer);

(*this procedure does handling of run-time errors --
it saves all data and closes files*)

begin
  writeln('***Run-Time Error --> Saving Data...');
  writeln('      Error type is ',hi(errno));
  writeln('      Error number is ',lo(errno));

  counter := counter + 1;
  if counter <= 300
  then

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 104

```

begin
  data[counter,1] := yd0[1]*180.0/pi; (*desired pos.1*)
  data[counter,2] := y[1]*180.0/pi;  (*actual pos.1*)
  data[counter,3] := err1*180.0/pi;  (*tracking error1*)
  data[counter,4] := error1;          (*prediction error1*)
  data[counter,5] := yd0[3]*180.0/pi; (*desired pos.2*)
  data[counter,6] := y[3]*180.0/pi;  (*actual pos.2*)
  data[counter,7] := err2*180.0/pi;  (*tracking error2*)
  data[counter,8] := error2;          (*prediction error2*)

  debug[counter,1] := para1[1]; (*parameter values*)
  debug[counter,2] := para1[2];
  debug[counter,3] := para1[3];
  debug[counter,4] := para1[4];
  debug[counter,5] := para1[5];
  debug[counter,6] := para1[6];
  debug[counter,7] := para2[1];
  debug[counter,8] := para2[2];

end;

(* write out the data needed for plotting the results *)
for i := 1 to counter do
  begin
    for j := 1 to 8 do begin
      write(outdata,data[i,j]);
      write(xdata,debug[i,j]);
    end;
    writeln(outdata);
    writeln(xdata);
  end;

  close(outdata);
  close(outpara);
  close(outcov);
  close(indata);
  close(outpmtr);
  close(partrend);
  close(xdata);

  halt;
end;
(*****)
function inverse_cos(value: real): real;

(* This function computes the inverse cosine of a number
   between -1.0 and 1.0. *)

begin
  inverse_cos := arctan(sqrt(1.0 - sqr(value))/value);
end; (* inverse_cos *)
(*****)

procedure initializea;
(* This procedure initializes all the parameters in
   the system such that the robot will start in the
   desired position *)

var

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 105

```

l1,l2,m1,m2 : real;
i,j : integer;

begin
  writeln(' Do you wish to accept default values for');
  writeln(' sampling time, gains, link parameters');
  writeln(' .005,80 8,.3 .3 2 2 (y/n)?');
  readln(yes);
  if (yes='y') or (yes='Y') then begin
    dt:=0.005;kp:=80.0;kv:=8.0;
    l1:=0.3;l2:=0.3;m1:=2.0;m2:=2.0;
  end
  else begin
    writeln(' Enter the sampling time ');
    readln(dt);
    writeln(' Enter feedback gains, position and velocity ');
    readln(kp, kv);
    writeln(' Enter the link lengths and masses (l1,l2,m1,m2) ');
    readln(l1, l2, m1, m2);
  end; {if}

  (* compute the inertial constants *)
  a1 := (1.0/3.0)*m1*l1*l1 + m2*l1*l1 + (1.0/3.0)*m2*l2*l2;
  a2 := l1*l2*m2;
  a3 := (1.0/3.0)*l2*l2*m2;
  a4 := 0.5*l1*l2*m2;
  a5 := l1*l2*m2;
  a6 := 0.5*l1*l2*m2;
  a7 := (1.0/3.0)*l2*l2*m2;
  a8 := 0.5*l1*l2*m2;
  a9 := (1.0/3.0)*l2*l2*m2;
  a10:= a6;

  nsel1 := 6;
  nsel2 := 4;
  npar := 11;
  yes := 'y';

  for i:= 1 to npar do
    begin
      selpar1[i] := i;
      selpar2[i] := i;
    end;

  (* Read in the initial parameter vector and covariance matrix *)
  for i := 1 to nsel1 do
    read(indata, para1[i]);
  end;
  for i := 1 to nsel1 do
    begin
      for j := 1 to i do
        read(indata, p1[i,j]);
      end;
    end;
  for i := 1 to nsel2 do
    read(indata, para2[i]);
  end;
  for i := 1 to nsel2 do
    begin
      for j := 1 to i do
        read(indata, p2[i,j]);
      end;
    end;

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 106

```

end;
(*****
procedure initializeb;
begin
    writeln(' Enter the forgetting factor ');
    readln(lamda);
    writeln(' Enter if noise to be added to signals (y/n)');
    readln(choice);
    if (choice='y') or (choice='Y') then noise:=true
    else noise:=false;
    writeln(' Enter the final time of the simulation ');
    readln(ftime);
    writeln(' Enter number of data points to be output');
    readln(npoints);
    skip:=trunc((ftime/dt)/npoints);
    writeln(' --> Data will be output every <',skip,
            '> time steps');
    writeln(' Enter the desired frequency ');
    readln(w);
    writeln(' Enter number of times to do simulation ');
    readln(rerun);
end;
(*****
procedure initializec;
var
    i: integer;
begin
    (* initialize the inputs *)
    u1in := 0.0;
    u2in := 0.0;
    f[1] := 0.0;
    f[2] := 0.0;
    f[3] := 0.0;
    f[4] := 0.0;
    (* initialize the manipulator state, i.e. position and velocity *)
    {for "loopy path"}
    (* y[1] := 1.5;
    y[2] := 0.0;
    y[3] := 3.141592/2.0 - 1.5;
    y[4] := 3.0*w; *)
    {for straight line path}
    y[1] := 3.141592/2.0;
    y[2] := 0.0;
    y[3] := 0.0;
    y[4] := 0.0;
    t := 0.0; (* initial time is zero *)
    counter := 0; (* initialize data counter to zero *)
    skipcount:=skip; (* initialize output skip counter *)
    (* initialize the desired state to the initial state *)
    for i := 1 to 4 do

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 107

```

begin
  yd0[i] := y[i];
  yd[i] := y[i];
end;
{initialize desired accelerations}
(* yda1:=12.0*w*w; {loopy path}
   yda2:=0.0; *)

      {straight line path}
yda1:=0.45*(-w*w);
yda2:=-2.0*yda1;

end;

(*****)

procedure compute_desired_state;
(* This procedure computes the desired position
   and velocity of the manipulator.*)

const
  pi=3.14159265;
var
  theta1,theta2: real; {intermediate values used for
                        computing efficiency}
  i: integer;

begin
  {store values of desired state for last time step}
  for i:=1 to 4 do
    begin
      yd0[i] := yd[i];
    end;
  yd0a1:=yda1;
  yd0a2:=yda2;

  {Compute desired state for current time step}

  {This trajectory follows a "loopy" path in the x-y plane}
  (* theta1:=2.0*w*t;
   yd[1] := 1.5 + cos(theta1) - cos(2.0*theta1);
   yd[2] := -2.0*w*sin(theta1) + 4.0*w*sin(2.0*theta1);
   yd[3] := pi/2.0 - 1.5 + sin(w*t) + sin(theta1);
   yd[4] := w*cos(w*t) + 2.0*w*cos(theta1);

   yda1:=-4.0*w*w*cos(theta1)+16.0*w*w*cos(2.0*theta1);
   yda2:=-w*w*sin(w*t)-4.0*w*w*sin(theta1); *)

  (* The following trajectory would cause the end effector of the
     manipulator to go along the x - axis. *)
  theta2:=w*t+pi/2.0;
  yd[1] := inverse_cos(0.45 - 0.45*sin(theta2));
  yd[2] := 0.45*w*cos(theta2)/(sin(yd[1]));
  yd[3] := pi - 2.0*yd[1];
  yd[4] := -2.0*yd[2];

  yda1:= 0.45*((sin(yd[1]))*(-w*w*sin(theta2)))-(w*cos(theta2)*
    (yd[2]*cos(yd[1])))/(sqr(sin(yd[1])));
  yda2:= -2.0*yda1;

end;

```


APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 108

```

(*****)
procedure compute_f;

(* This procedure computes the derivative vector
   f(y(t)) = F(t,y(t)) + b(t,y(t))*u(t). This
   procedure is called four times by the procedure
   runge_kutta. The equations in this procedure
   represent the dynamics of a two degree of freedom
   manipulator, which is operating in the horizontal plane. *)

var
  x1,x2,x3,x4,sny3,csy3 : real;

begin
  sny3 := sin(y[3]);
  csy3 := cos(y[3]);
  x1 := a1 - a2*csy3;
  x2 := a3 - a4*csy3;
  x3 := a7 - a8*csy3;
  x4 := a9*x1 - x2*x3;

  (* compute the derivative vector f(t,y(t)) *)
  f[1] := y[2];
  f[2] := (1.0/x4)*(a9*u1in - a10*x2*sny3*y[2]*y[2] -
    a5*a9*sny3*y[2]*y[4] - a6*a9*sny3*y[4]*y[4] -
    x2*u2in);
  f[3] := y[4];
  f[4] := (1.0/x4)*(a5*x3*sny3*y[2]*y[4] +
    a6*x3*sny3*y[4]*y[4] + a10*x1*sny3*y[2]*y[2] -
    x3*u1in + x1*u2in);

end;

(*****)
procedure runge_kutta;

(* This is the numerical solution to the nonlinear dynamics
   of the robot manipulator. This procedure calls another
   procedure, which is compute_f. Compute_f computes the
   derivatives of the states in vector y. These derivatives
   are stored in vector f(y(t)). The state of the
   manipulator is calculated based on the following recursion;

   y(t + dt) = y(t) + 1/6(k0 + 2k1 + 2k2 + k3)

   where
       k0 = f(y(t))*dt
       k1 = f(y(t) + 1/2k0)*dt
       k2 = f(y(t) + 1/2k1)*dt
       k3 = f(y(t) + k2)*dt

   From the above formula it can be seen that to calculate the
   new state at the next time step, the function f(y(t)) has to be
   calculate four times. *)

var
  i,j : integer;
  k : array[1..4,1..4] of real; (* k0,k1,k2,k3 for all four states *)
  y1 : array[1..4] of real; (* manipulator state at present *)

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 109

```

begin
  for i := 1 to 4 do
    begin
      compute_f;  (* Compute the derivative vector f[j] *)

      (* If this is the first time through the loop we will
         store the present state of the Manipulator
         and compute k0 and y(t) + (1/2)k0 which will be used
         as the argument for f(y(t) + (1/2)k0) *)

      if i = 1
      then
        begin
          for j := 1 to 4 do
            begin
              y1[j] := y[j];          (* store present robot state *)
              k[i,j] := f[j]*dt;      (* compute k0 = f(y(t))*dt *)
              y[j] := y1[j] + 0.5*k[i,j]; (* compute y(t) + (1/2)k0 *)
            end;
          end;
        end;

      if i = 2
      then
        begin
          for j := 1 to 4 do
            begin
              k[i,j] := f[j]*dt;      (* compute k1 = f(x(t))*dt *)
              y[j] := y1[j] + 0.5*k[i,j]; (* compute y(t) + (1/2)k1 *)
            end;
          end;
        end;

      if i = 3
      then
        begin
          for j := 1 to 4 do
            begin
              k[i,j] := f[j]*dt;      (* compute k2 = f(x(t))*dt *)
              y[j] := y1[j] + k[i,j]; (* compute y(t) + k2 *)
            end;
          end;
        end;

      if i = 4
      then
        begin
          for j := 1 to 4 do
            begin
              k[i,j] := f[j]*dt;      (* compute k3 = f(x(t))*dt *)

              (* compute updated state of the robot *)
              y[j] := y1[j] + (1.0/6.0)*(k[i,j] + 2.0*k[2,j] +
                2.0*k[3,j] + k[4,j]);

              (* The robot state has been updated *)
            end;
          end;
        end;
      end;
    end;
  end;

  (* end of the for loop i=1,4 *)
  (* end of the procedure *)

  (*****)
  procedure compute_input;

```

```

(* This procedure computes the actual input to the
  robot's motors, using the computed torque
  technique. The actual input is made up from
  the desired acceleration plus the feedback due to the
  errors in robot position and velocity. *)

begin
  err1 := yd0[1] - yn1; {errors including measurement noise}
  derr1 := yd0[2] - yn2;
  err2 := yd0[3] - yn3;
  derr2 := yd0[4] - yn4;

  {compute desired acceleration with error feedback included}
  y1star:=yda1+kv*derr1+kp*err1;
  y2star:=yda2+kv*derr2+kp*err2;

  {calculate elements of estimated Mass Matrix and Other Torques
   vector, M and Q}
  m11:=para1[1]+para1[2]*cosyn3;
  m12:=para1[3]+para1[4]*cosyn3;
  m21:=para2[1]+para2[2]*cosyn3;
  m22:=para2[3];
  q1:=para1[5]*phi1[5]+para1[6]*phi1[6];
  q2:=para2[4]*phi2[4];

  {calculate control torque inputs for both motors}
  u1in:=m11*y1star+m12*y2star+q1;
  u2in:=m21*y1star+m22*y2star+q2;

end;
(*****)

procedure identification;

(* This procedure identifies the parameters
  which describe the dynamics of the manipulator *)

var
  motor,i,j : integer;
  sum,u : real;
  p : array[1..20,1..20] of real;
  f1,g,phi,v,l : array[1..20] of real;
  b : array[1..21] of real;

begin
  b[1] := lamda;
  for motor := 1 to 2 do
    begin
      if motor = 1
      then
        begin
          npar := nsel1;
          for i:= 1 to npar do
            begin
              for j:= 1 to i do
                p[i,j] := p1[i,j];
                phi[i] := phis1[i];
              end;
            end;
          end;
        end;
      if motor = 2
      then
        begin
          npar := nsel2;
          for i:= 1 to npar do
            begin
              for j:= 1 to i do
                p[i,j] := p2[i,j];
                phi[i] := phis2[i];
              end;
            end;
          end;
        end;
    end;
  end;

```

```

else
begin
  npar := nsel2;
  for i := 1 to npar do
    begin
      for j := 1 to i do
        p[i,j] := p2[i,j];
      phi[i] := phis2[i];
    end;
  end;

sum := 0.0;
if motor = 1
then
begin
  for i := 1 to npar do
    begin
      sum := sum + para1[i]*phis1[i];
    end;
  error1 := u1in - sum;
end
else
begin
  for i := 1 to npar do
    begin
      sum := sum + para2[i]*phis2[i];
    end;
  error2 := u2in - sum;
end;

(* Do the UDU update of the parameters and covariance matrix *)
f1[1] := phi[1];
for i := 2 to npar do
begin
  sum := 0.0;
  for j := 1 to i-1 do
    sum := sum + p[i,j]*phi[j];
  f1[i] := phi[i] + sum;
end;

for i := 1 to npar do
  g[i] := p[i,i]*f1[i];

(* Update the p matrix *)
for j:= 1 to npar do
begin
  b[j+1] := b[j] + f1[j]*g[j];
  p[j,j] := b[j]*p[j,j]/(b[j+1]*b[1]);
  v[j] := g[j];
  if j > 1
  then
begin
  for i := 1 to j-1 do
begin
  u := p[j,i];
  p[j,i] := p[j,i] - v[i]*f1[j]/b[j];
  v[i] := v[i] + u*v[j];
end;
end;
end;

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 112

```

    end;
(* Compute the parameter update gains *)
    for i := 1 to npar do
        l[i] := v[i]/b[npar+1];
(* update the parameter estimates *)
    if motor = 1
    then
        begin
            for i := 1 to npar do
                para1[i] := para1[i] + l[i]*error1;
            for i := 1 to npar do
                begin
                    for j := 1 to i do
                        p1[i,j] := p[i,j];
                    end;
                end;
            end;
        else
            begin
                for i := 1 to npar do
                    para2[i] := para2[i] + l[i]*error2;
                for i := 1 to npar do
                    begin
                        for j := 1 to i do
                            p2[i,j] := p[i,j];
                        end;
                    end;
                end;
            end;
        end;
    end;
end;

(*****)
procedure covariance;

(* This procedure takes the P (UDU factorized) matrix and
   computes the covariance matrix. Recall the update algorithm
   uses a UDU factorization. The diagonal of P stores
   the elements of D. The Triagonal elements of P store the
   triangular elements of U. The procedure calculates the
   covariance matrix for motor 1 and motor 2 in turn.*/)

var
    d,ud,u: Array[1..10,1..10] of real;
    i,j,k : integer;

begin
    {compute covariance matrix for motor 1}
    for i := 1 to nsel1 do
        begin
            d[i,i] := p1[i,i];
            u[i,i] := 1.0;
            if i > 1 then
                begin
                    for j := 1 to i-1 do
                        begin

```

```

                u[j,i] := p1[i,j];
                u[i,j] := 0.0;
                d[i,j] := 0.0;
                d[j,i] := 0.0;
            end;
        end;
    end;
    (* Initialize intermediate values *)
    for i := 1 to nsel1 do
        begin
            for j := 1 to nsel1 do
                begin
                    ud[i,j] := 0.0;
                    cov1[i,j] := 0.0;
                end;
            end;
        end;
    (* Multiply out the UDU matrix *)
    for i := 1 to nsel1 do
        begin
            for j := 1 to nsel1 do
                begin
                    for k := 1 to nsel1 do
                        uc[i,j] := ud[i,j] + u[i,k]*d[k,j];
                    end;
                end;
            end;
        end;
    (* Now compute UDU *)
    for i := 1 to nsel1 do
        begin
            for j := 1 to nsel1 do
                begin
                    for k := 1 to nsel1 do
                        cov1[i,j] := cov1[i,j] + ud[i,k]*u[j,k];
                    end;
                end;
            end;
        end;
    {compute covariance matrix for motor 2}
    for i := 1 to nsel2 do
        begin
            d[i,i] := p2[i,i];
            u[i,i] := 1.0;
            if i > 1 then
                begin
                    for j := 1 to i-1 do
                        begin
                            u[j,i] := p2[i,j];
                            u[i,j] := 0.0;
                            d[i,j] := 0.0;
                            d[j,i] := 0.0;
                        end;
                    end;
                end;
            end;
        end;
    (* Initialize intermediate values *)
    for i := 1 to nsel2 do
        begin

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 114

```

    for j := 1 to nsel2 do
        begin
            ud[i,j] := 0.0;
            cov2[i,j] := 0.0;
        end;
    end;

(* Multiply out the UDU matrix *)
for i := 1 to nsel2 do
    begin
        for j := 1 to nsel2 do
            begin
                for k := 1 to nsel2 do
                    ud[i,j] := ud[i,j] + u[i,k]*d[k,j];
                end;
            end;
        end;
    end;

(* Now compute UDU *)
for i := 1 to nsel2 do
    begin
        for j := 1 to nsel2 do
            begin
                for k := 1 to nsel2 do
                    cov2[i,j] := cov2[i,j] + ud[i,k]*u[j,k];
                end;
            end;
        end;
    end;

end;

(*****)
procedure nrand(var x1,x2:real);

{this procedure generates two normally distributed random
numbers, with standard normal distribution (i.e. mean=0,
std.dev.=1)}

var v1,v2,u1,u2,s,sroot:real;{local variables}

begin
    repeat
        u1:=random;
        u2:=random;
        v1:=(2.*u1)-1.;
        v2:=(2.*u2)-1.;
        s:=(v1*v1)+(v2*v2);
    until s<=1.;
    sroot:=sqrt((-2.*ln(s))/s);
    x1:=v1*sroot;
    x2:=v2*sroot;
end;

(*****)
procedure regression;

(* This procedure defines the regression vector to be used
by the system identification algorithm *)
const
    signap = 0.0; {noise on position, rad}
    sigmav = 0.2; {noise on velocity, rad/sec}
    sigmaa = 2.0; {noise on acceleration, rad/sec^2}

var
    v1,v2,v3,v4,v5,v6,fn2,fn4 : real;

```

```

begin
  compute_f;

  if noise then begin
    nrand(v1,v2);
    nrand(v3,v4);
    nrand(v5,v6);
    v1:=sigmaa*v1;
    v2:=sigmaa*v2;
    v3:=signav*v3;
    v4:=signav*v4;
    v5:=signav*v5;
    v6:=signav*v6;
  end
  else begin
    v1 := 0.;
    v2 := 0.;
    v3 := 0.;
    v4 := 0.;
    v5 := 0.;
    v6 := 0.;
  end;{if}

  {calculate regression vector with noise added}
  fn2:=v1+f[2]; {y1''}
  fn4:=v2+f[4]; {y2''}
  yn1:=v6+y[1]; {y1}
  yn2:=v3+y[2]; {y1'}
  yn3:=v4+y[3]; {y2}
  yn4:=v5+y[4]; {y2'}

  {calculate repeated terms}
  cosyn3:=cos(yn3);
  sinyn3:=sin(yn3);

  {calculate regression vectors}
  phi1[1] := fn2;
  phi1[2] := fn2*cosyn3;
  phi1[3] := fn4;
  phi1[4] := fn4*cosyn3;
  phi1[5] := yn2*yn4*sinyn3;
  phi1[6] := yn4*yn4*sinyn3;

  phi2[1] := fn2;
  phi2[2] := fn2*cosyn3;
  phi2[3] := fn4;
  phi2[4] := yn2*yn2*sinyn3;

  end;

  (*****)

  procedure paraselection;
  var
    i: integer;

  (* This procedure selects the regression vector which will
     be identified -- has no effect for this simulation*)
  begin

```


APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 116

```

for i := 1 to nsel1 do
begin
j := selpar1[i];
phis1[i] := phi1[j];
end;

for i := 1 to nsel2 do
begin
j := selpar2[i];
phis2[i] := phi2[j];
end;
end;

(*****)
procedure collect_data;

(* This procedure collects data which will be plotted at
a later time, as well as data used for debugging *)

begin
counter := counter + 1;
if counter <= 300
then
begin
data[counter,1] := yd0[1]*180.0/pi; (*desired pos.1*)
data[counter,2] := y[1]*180.0/pi; (*actual pos.1*)
data[counter,3] := err1*180.0/pi; (*tracking error1*)
data[counter,4] := error1; (*prediction error1*)
data[counter,5] := yd0[3]*180.0/pi; (*desired pos.2*)
data[counter,6] := y[3]*180.0/pi; (*actual pos.2*)
data[counter,7] := err2*180.0/pi; (*tracking error2*)
data[counter,8] := error2; (*prediction error2*)

debug[counter,1] := para1[1]; (*parameter values*)
debug[counter,2] := para1[2];
debug[counter,3] := para1[3];
debug[counter,4] := para1[4];
debug[counter,5] := para1[5];
debug[counter,6] := para1[6];
debug[counter,7] := para2[1];
debug[counter,8] := para2[2];
end;
end;

(*****)

procedure simulate;

begin
(* The simulation loop starts here. *)

while t <= ftime do
begin
t := t + dt;

(* Compute the desired state one time step into the future *)

compute_desired_state;
(* Define the regression vectors *)

```

APPENDIX C. SIMULATION SOFTWARE—INDIRECT ADAPTIVE CONTROL 117

```

    regression;
(* Select the data which will be used as the regression vector *)
    paraselection;
(* Compute the identification algorithm *)
    identification;
    (* Compute the input into the robot motors *)
    compute_input;
    (* Collect data which will be plotted at a later time,
       every "skip" number of timesteps *)
    if skipcount = skip then begin
        collect_data;
        skipcount:=0;
    end;
    skipcount:=skipcount+1;
    (* simulate the robot motion over one time interval *)
    runge_kutta;
end;
end;
{*****}
{*****}
begin
    errorptr:=ofs(error); {pointer to error handler procedure}
    assign(outdata,'outdat.dat');
    rewrite(outdata);
    assign(xdata,'debug.dat');
    rewrite(xdata);
    assign(outpara,'outpar.dat');
    rewrite(outpara);
    assign(outcov,'outcov.dat');
    rewrite(outcov);
    assign(outpmtr,'outpmtr.dat');
    rewrite(outpmtr);
    assign(partrend,'partrend.dat');
    rewrite(partrend);
    assign(indata,'in.dat');
    reset(indata);
    (* Initialize all the data in the system *)
    initializea;
    (* Simulate the robot as many times as you like *)
    while (yes = 'y') or (yes='Y') do
        begin
            (* Initialize the robot for the next simulation *)
            initializeb;
            (* Simulate the robot *)
            for i := 1 to rerun do

```

APPENDIX C. SIMULATION SOFTWARE — INDIRECT ADAPTIVE CONTROL 118

```

begin
  initializec;
  writeln('--> Running simulation #',i);
  simulate;
  {write values of parameter vector for this run}
  for jj:=1 to nsel1 do
    write(partrend,para1[jj]);
  for jj:=1 to nsel2 do
    write(partrend,para2[jj]);
  writeln(partrend);
end;

(* Do you want another simulation ? *)

  for i := 1 to nsel1 do
    write(para1[i]);
  writeln(' ');
  for i := 1 to nsel2 do
    write(para2[i]);
  writeln(' ');

  writeln(' do you want another simulation (y/n)?');
  readln(yes)
end;

{output values of parameters to a file}
  writeln(outpara);

  for i := 1 to nsel1 do
    write(outpara,para1[i]);
  writeln(outpara);
  for i:=1 to nsel2 do
    write(outpara,para2[i]);

{calculate covariance matrices from UDU factorized version, and
output to a file}

  covariance;

writeln(outcov);
for i := 1 to nsel1 do
  begin
    for j := 1 to nsel1 do
      write(outcov,cov1[i,j]);
    writeln(outcov);
  end;

writeln(outcov);
for i := 1 to nsel2 do
  begin
    for j := 1 to nsel2 do
      write(outcov,cov2[i,j]);
    writeln(outcov);
  end;

(* Output parameter vectors and (UDU factorized) covariance
matrices to a file -- for use as inputs in subsequent runs*)

  for i := 1 to nsel1 do
    write(outpmtx, para1[i]);
  writeln(outpmtx);

  for i := 1 to nsel1 do
    begin

```

```

    for j := 1 to i do
        write(outpmtx, p1[i,j]);
        writeln(outpmtx);
    end;

    for i := 1 to nsel2 do
        write(outpmtx, para2[i]);
        writeln(outpmtx);
    end;

    for i := 1 to nsel2 do
        begin
            for j := 1 to i do
                write(outpmtx, p2[i,j]);
                writeln(outpmtx);
            end;
        end;

    (* write out the data needed for plotting the results *)
    for i := 1 to counter do
        begin
            for j := 1 to 8 do begin
                write(outdata, data[i,j]);
                write(xdata, debug[i,j]);
            end;
            writeln(outdata);
            writeln(xdata);
        end;

    close(outdata);
    close(xdata);
    close(outpara);
    close(outcov);
    close(indata);
    close(outpmtx);
    close(partrend);

    writeln(' The program is finished, press any key ');
    repeat until keypressed;
end.

```

Appendix D

Control Software — Direct Adaptive Control

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 121

```
program robot_control10;
```

```
{-----}
{Written by: Gabriel D. Warshaw}
{Revised:   Sept. 22, 1989}
{
{ This program performs adaptive computed-torque control
{ of 1 degree-of-freedom of a robotic manipulator, using the Direct
{ method of adaptation. The manipulator is made to follow a
{ pre-defined path in joint space (motor 2). Position and velocity
{ measurements are Kalman filtered.
{-----}
```

```
const
```

```
{i/f card register addresses}
ad_status_addr = $218;
ad_chan_addr = $219;
da0_loadaddr = $21A; {same as a/d data low byte }
da0_hiaddr = $21B; { " " " hi " }
da1_loadaddr = $21C;
da1_hiaddr = $21D;
timer_addr = $21F;
dd_cntrlreg_addr = $228;
dd_port0_addr = $229;
dd_port1_addr = $22A;
dd_port2_addr = $22B;
dd_port3_addr = $22C;

maxpoints = 800; {maximum number of data points to be saved}
zerovolts_bin = $800; {12 bit binary value for 0 volts, based
                      on -5 to +5 range}
ktach1=244.; {tach.1&2 constant =244 (deg/sec)/volt}
ktach2=244.; { (no voltage divider in use)}
torque_limit=4.0;{control volt. amplitude limit i.e. s/w defined}
{ saturation point.}
njoint=1; {number of joints under control}

reject=5.5; {value of position change used to detect
            faulty data, in degrees}

para2_init1=0.006; {initial estimates for parameter}
para2_init2=0.006; { identification}

freq_code=$2B; {hex code for sampling frequency}
freqs=120; {sampling frequency, Hz}

a = 0.35; {viscous friction term for kalman observer}
b = 194.0; {input term for kalman observer}
```

```
type
```

```
state = array[1..2] of real;
matrix = array[1..2] of state;
```

```
var
```

```
dummy_input, {dummy value of a/d input (binary)}
nsteps, {number of steps during run}
counter, {current step}
skip, {number of time steps to skip for data output}
skipcount, {current skip count}
hr,min,sec,frac, {arguments from procedure which gets time}
pos1_bin,pos2_bin, {12 bit binary values of motor positions}
```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 122

```

    vel1_bin,vel2_bin  { " " " " " " velocities}
    :integer;

    pos1,pos2,         {joint angular positions in degrees}
    vel1,vel2,         {joint angular velocities in deg/sec}
    acc2,              {joint 2 angular accel. in deg/sec-squared}
    pos2last,          {last value of motor 2 position}
    vel2last,          { " " " " " " velocity}
    pos1d,pos2d,vel1d,vel2d, {desired angles and velocities}
    acc2d,             {desired joint 2 acceleration}
    pos2_init,         {actual value of motor 2 position at start of run}
    acc2star,          {desired value of motor 2 accel. including feedback}
    t,                 (* Time from start of run *)
    dt,dt2,            (* Time interval for a step, twice the time interval *)
    ftime,             (* Final time of the run *)
    err1,              (* robot position error motor 1 *)
    err2,              (* robot position error motor 2 *)
    derr1,             (* error in the velocity of motor 1 *)
    derr2,             (* error in the velocity of motor 2 *)
    kp,kv,             (* Position and velocity feedback gains *)
    control1,          (* Control input for motor 1, in volts *)
    control2,          (* " " " " motor 2 *)
    gamma {value of adaptation gain for parameter identification}
    :real;

    para2,             {vector of joint 2 parameters}
    phi2               {joint 2 regression vector}
    :array[1..2] of real;

    data: array[1..10,1..maxpoints] of real; {array of data for output to
    a file (state of robot)}

    outdata,outinfo: text; {output file assignment names}

    filename: string[20];
    comments: string[160];

    enabled,           {flag indicating if motors are enabled}
    ad_error           { " " " " occurrence of a/d conversion error}
    : boolean;

    yes_comment,       {variable for user to select input of comments}
    choice: char;       {variable for input of user selection character}

    x,xk:state;        {filtered and predicted outputs}
    kal:state;         {Kalman gain}

    (*****)
    procedure predict_next_states;
    begin
        xk[1] := x[1]+(dt-dt*dt*a/2)*x[2]+dt*dt/2*b*control2;
        xk[2] := (1-dt*a)*x[2]+b*dt*control2;
    end;

    (*****)
    procedure kalman_gains;
    var
    i: integer;
    hphv,vv,vw:real;
    pk,p:matrix;
    begin

```

APPENDIX D. CONTROL SOFTWARE— DIRECT ADAPTIVE CONTROL 123

```

vv := 0.1;
vw := 0.04;
p[1][1] := 0.0;
p[1][2] := 0.0;
p[2][2] := 0.0;

for i := 1 to 100 do
begin
  pk[1][1] := dt*dt*b/2*vw*dt*dt*b/2+p[1][1]+(dt*(1-dt*a/2))*
    p[1][2]+(p[1][2]+(dt*(1-dt*a/2))*p[2][2])*
    (dt*(1-dt*a/2));
  pk[1][2] := dt*dt*b/2*vw*dt*dt*b+(p[1][2]+(dt*(1-dt*a/2))*
    p[2][2])*(1-dt*a);
  pk[2][2] := dt*b*vw*dt*b+((1-dt*a)*p[2][2])*(1-dt*a);
  hphv := 1.0/(pk[1][1]+vv);
  p[1][1] := pk[1][1]-pk[1][1]*hphv*pk[1][1];
  p[1][2] := pk[1][2]-pk[1][1]*hphv*pk[1][2];
  p[2][2] := pk[2][2]-pk[1][2]*hphv*pk[1][2];
end;
kal[1] := p[1][1]/vv;
kal[2] := p[1][2]/vv;
end;

(*****)
procedure filter_states;

var
  perr:real; {prediction error}

begin
  if t < 2*dt then xk[1] := pos2;
  perr := pos2-xk[1];
  x[1] := xk[1]+kal[1]*perr;
  x[2] := xk[2]+kal[2]*perr;
end;

(*****)
procedure send_da(lobyte_addr,hibyte_addr,out_value:integer);

begin
  port[lobyte_addr]:=lo(out_value);
  port[hibyte_addr]:=hi(out_value);
end;

{*****}
procedure check_status(first_addr,second_addr:integer;
  var enable_status:boolean);

const bit6 = $40;

begin
  if(port[second_addr] AND bit6) = bit6 then
    enable_status:=true
  else enable_status:=false;
end;

(*****)
procedure get_position_word(lobyte_addr,hibyte_addr:integer;
  var pos_word: integer);

const lo4bits = $0F;

```


APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 124

```

var lo_data,hi_data: byte;

begin
  lo_data:=port[lobyte_addr];
  hi_data:=port[hibyte_addr] AND 104bits;
  pos_word:=lo_data + hi_data shl 8;
end;

{*****}
procedure get_initial_position2;
  {gets initial value of motor 2 position}

begin
  get_position_word(dd_port2_addr,dd_port3_addr,pos2_bin);
  {correct binary position by reversing direction
                                   (neg.angle ->pos.angle)}

  pos2_bin:=-pos2_bin+4096;
  {convert binary position to degrees}
  pos2_init:=(pos2_bin/4096.0)*360.0;
  pos2:=pos2_init;
  pos2last:=pos2;
end;

{*****}
procedure initialize_interfaces;

begin
  {initialize a/d}
  port[dd_cntrlreg_addr]:=0; {set all d/d ports for input}
  port[ad_status_addr]:=$10; {set a/d to mode 0, clear error bit}
  delay(50);
  dummy_input:= port[da0_loaddr]+(port[da0_hiaddr])shl 8;

  port[ad_status_addr]:=$11; {set a/d to mode 1, clear error bit}
  port[timer_addr]:=freq_code; {set timer for chosen sample rate}

  {set d/a outputs to 0 Volts}
  send_da(da0_loaddr,da0_hiaddr,zerovolts_bin);
  send_da(da1_loaddr,da1_hiaddr,zerovolts_bin);

  {set a/d conversion error flag to false}
  ad_error:=false;

  {check to see if motor (servos) are enabled}
  check_status(dd_port1_addr,dd_port3_addr,enabled);
  if enabled=false then begin
    writeln('**Motor is disabled --> enable it now. ');
    writeln('      (press any key when ready)');
    repeat until keypressed;
  end; {if}
  enabled:=true;
end;

{*****}
procedure create_info_file;
  (* This procedure creates an information file with the same
     name as the data file but with a ".inf" extension. The
     file contains numeric and text information which describes
     the run which produced the data.*)

begin
  writeln(outinfo,'filename: ',filename);

```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 125

```

writeln(outinfo,' data: sampling freq(Hz),runtime(sec)',
        ',skip interval,$ data pts,$ joints');
writeln(outinfo,' successive lines list: Kp,Kv');
writeln(outinfo);
writeln(outinfo,freqs,' ',ftime,' ',skip,' ',counter,' ',njoint);
writeln(outinfo,kp,kv);
writeln(outinfo);

write('Do you wish to enter comments for the information file (y/n)? ');
readln(yes_comment);
if ((yes_comment = 'y') OR (yes_comment = 'Y')) then begin
    writeln(' Enter comments, terminate with <CR> (max 160 characters):');
    readln(comments);
    writeln(outinfo,'Comments:');
    writeln(outinfo,comments);
end;

close(outinfo);
end;

(*****)
procedure initialize_parameters;
(* This procedure initializes all the parameters in
   the system, including those input by the user. *)
begin
{set up output files}
    writeln('Enter path and name for data output file ');
    writeln(' (extensions ".dat" for the data file and ".inf" for the');
    writeln(' matching information file will be appended automatically):');
    readln(filename);
    assign(outdata,filename+'.dat');
    rewrite(outdata);
    assign(outinfo,filename+'.inf');
    rewrite(outinfo);

    write('Enter feedback gains (position and velocity): ');
    readln(kp,kv);

    write('Enter value of adaptation gain, Gamma: ');
    readln(gamma);

    write('Enter number of seconds for run: ');
    readln(ftime);
    write('Enter number of data points to be stored during the run: ');
    readln(nsteps);
    skip:=trunc((ftime*freqs)/nsteps);
    writeln('--> At ',freqs,' Hz sampling, data will be stored'
        ', every <',skip,'> time steps');

{initialize variables}
    skipcount:=skip;
    t:=0.0;
    dt:=1.0/freqs;dt2:=dt*2.0;
    counter:=0;
    vel2last:=0.0;
    err2:=0.0; derr2:=0.0;
    para2[1]:=para2_init1; para2[2]:=para2_init2;

    kalman_gains;      {compute Kalman gains}
    zk[2] := 0.0;      {set observer velocity to zero}

```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 126

```

end;

(*****)
procedure compute_desired_state;

(* This procedure computes the desired position, velocity
   and acceleration of the manipulator -- this trajectory must
   be continuous and start from zero velocity.*)

const twopi=6.283185;
      f=0.7; {freq.for sine input (Hz)}
      amplitude=3.0; {amplitude of sine input (deg)}

var w {freq. for sine input (radians/sec)}
    :real;

begin
{sinusoidal input of position for motor 2}
  w:=twopi*f;
  pos2d:=pos2_init+amplitude - amplitude*cos(w*t);
  vel2d:=amplitude*w*sin(w*t);
  acc2d:=amplitude*w*w*cos(w*t);

end;

(*****)
procedure get_ad(var inval:integer; var error:boolean; status_addr,
                 lobyte_addr,hibyte_addr: integer);

const bit6 = $40;
      bit7 = $80;

var status:byte;

begin
  repeat status:=port[status_addr]
    until (status AND bit7) = bit7;
  if(status and bit6) = bit6 then error:=true;
  inval:=port[lobyte_addr]+(port[hibyte_addr])shl 8;
end;
{*****}
procedure save_data_to_file;

var i,j:integer; {loop indices}

begin
  writeln('--> data being saved to file....');
  if counter > maxpoints then counter:=maxpoints;
  for i := 1 to counter do
    begin
      for j := 1 to 10 do write(outdata,data[j,i]);
      writeln(outdata);
    end;
  close(outdata);
end;

(*****)
procedure get_measurements;

begin
{get velocity of motor 2}

```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 127

```

get_ad(vel2_bin,ad_error,ad_status_addr,da0_loaddr,da0_hiaddr);
if ad_error then begin
  writeln('A/D conversion error has occurred -- PROGRAM ABORTED');
  save_data_to_file;
  halt;
end;

{convert velocity (binary) voltages to deg/sec}
vel2:=(((vel2_bin/4096.0)*10.0)-5.0)*ktach2;

{get positions of both motors}
get_position_word(dd_port2_addr,dd_port3_addr,pos2_bin);
{correct binary position by reversing direction (neg.angle ->pos.angle)}
pos2_bin:=-pos2_bin+4096;
{convert binary position to degrees}
{including check and correct for spurious position data}
pos2:=(pos2_bin/4096.0)*360.0;
if (abs(pos2-pos2last) > reject) and (counter > 1)
  then pos2:=pos2last;
  pos2last:=pos2;
end;

(*****
procedure identify_parameters;

{This procedure identifies the parameters which describe the
dynamics of the manipulator. A direct adaptation law based
on Lyapunov stability and the tracking errors is used}

const
  psi2=1.0; {scaling of pos.err. in "filtered error" for joint 2}

var
  e2f, {filtered error for joint 2}
  pmult2: {intermediate calculated value}
    real;

  k: integer; {loop index}

begin
  e2f:=derr2+psi2*err2;      {calculate filtered tracking error}
  acc2:=(x[2]-vel2last)*freqs; {estimate acceleration}
  vel2last:=x[2];

  {define regression vector}
  phi2[1]:=acc2;
  phi2[2]:=x[2];

  {calculate updated values of parameters}
  pmult2:=dt*gamma*e2f/para2[1];
  for k:=1 to 2 do
    para2[k]:=para2[k]+pmult2*phi2[k];
  end;

  (*****
procedure compute_control_input;

var control1_bin,control2_bin {control voltages converted to}
  :integer;                  {12-bit binary numbers}

  abs_val:real;      {absolute value of control input voltage}

begin

```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 128

```

{calculate tracking errors}
err2 := pos2d - x[1];
derr2 := vel2d - x[2];

{compute desired accel. with error feedback included, and calculate
control torque voltage command}
acc2star:=acc2d+kv*err2+kv*derr2;
control2:=para2[1]*acc2star+para2[2]*phi2[2];

{check control voltages for saturation (of d/a)}
abs_val:=abs(control2);
if abs_val > torque_limit then begin
    control2:=(abs_val/control2)*torque_limit;
end;

{convert control voltages to 12 bit binary, and send to d/a}
control2_bin:=round(((control2+5.0)/10.0)*4096.0);
send_da(dal_loadaddr,dal_hiaddr,control2_bin);
end;

(*****
procedure store_data;

(* This procedure collects data which will be plotted at
a later time, as well as data used for debugging *)

begin
    if counter <= maxpoints then
        begin
            data[1,counter] := pos2;
            data[2,counter] := vel2;
            data[3,counter] := pos2d;
            data[4,counter] := control2;
            data[5,counter] := x[1];
            data[6,counter] := x[2];
            data[7,counter] := para2[1];
            data[8,counter] := para2[2];
            data[9,counter] := acc2;
            data[10,counter] := acc2star;
        end;
end;

(*****
procedure display_state;
{this procedure displays the state of the robot on the screen}

begin
    gotoXY(1,2);
    writeln('pos1d:',pos1d,' pos1:',pos1,' err1:',err1);
    writeln('          vel1:',vel1,' control1:',control1);
    writeln('pos2d:',pos2d,' pos2:',pos2,' err2:',err2);
    writeln('          vel2:',vel2,' control2:',control2);
end;

(*****
procedure timer(var hour,min,sec,frac:integer);
{this procedure gets time values from DOS}

type regpack = record
    ax,bx,cx,dx,bp,si,di,ds,es,flags:integer;
end;

```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 129

```

var regs: regpack;

begin
  with regs do
    begin
      ax:= $2C00;
      mados(regs);
      hour:=hi(cx);
      min:=lo(cx);
      sec:=hi(dx);
      frac:=lo(dx);
    end;
end; {procedure timer}

(*****)
procedure error(errno,erraddr:integer);
(*this procedure does handling of run-time errors --
  it saves all data and closes files*)

begin
  writeln('****Run-Time Error:');
  writeln('          Error type is ',hi(errno));
  writeln('          Error number is ',lo(errno));

  {set d/a back to 0 volts}
  send_da(da0_loaddr,da0_hiaddr,zervolts_bin);
  send_da(da1_loaddr,da1_hiaddr,zervolts_bin);

  save_data_to_file;

  create_info_file;

  writeln('** Please Disable Motor 2 servo Now **');

  halt;
end;

(*****)
(*****)
begin {main}

{pointer to error handler procedure}
  errorptr:=ofs(error);

{start-up message to user}
  writeln;
  writeln('++++++ ROBOT PD CONTROL PROGRAM +++++');
  writeln;

{enter loop to do repeated runs}
  repeat
    initialize_interfaces;

    initialize_parameters;

    get_initial_position2;
    writeln;
    writeln('Current position of Motor 2 is ',pos2,' degrees. ');
    writeln;

    writeln('Hit any key to start operation (after 5 second delay). ');
    repeat until keypressed;
    clrscr;
  
```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 130

```

    get_initial_position2; {in case operator repositioned motor 2}
{5 second delay to give operator time}
    delay(5000);

{get the time}
    gotoXY(1,1);
    timer(hr,min,sec,frac);
    if frac<10 then
        writeln('--> Operation starting now, at time ',hr,':',min,':',
            sec, '.',0,frac)
        else writeln('--> Operation starting now, at time ',hr,':',min,':',
            sec, '.',',frac);

{start a/d conversion for channel 1 (motor 2), gain=1}
    port[ad_chan_addr]:=1;

{enter loop for repeated steps of robot during a run}
    repeat
        t:=t+dt;

        compute_desired_state;

        get_measurements;

        filter_states;

        identify_parameters;

        compute_control_input;

        predict_next_states;

        if skipcount = skip then begin
            counter:=counter+1;
            store_data;
            skipcount:=0;
        end;
        skipcount:=skipcount+1;

    (*      display_state; *)

    (*      check_status(dd_port1_addr,dd_port3_addr,enabled); *)

    until (enabled=false) OR (t>=ftime);

{stop operation if motor(s) disabled}
    if enabled=false then
        writeln('***Operation halted:  motor is disabled.');
```

```

{set d/a back to 0 volts}
    send_da(da0_loaddr,da0_hiaddr,zervolts_bin);
    send_da(da1_loaddr,da1_hiaddr,zervolts_bin);

{get the time}
    timer(hr,min,sec,frac);
    gotoXY(1,10);
    if frac<10 then
        writeln('--> Operation complete, at time ',hr,':',min,':',
            sec, '.',0,frac)
        else writeln('--> Operation complete, at time ',hr,':',min,':',
            sec, '.',',frac);

{prompt user for another run}
    write('Do you wish to quit -- saving data to file (y/n)? ');
```

APPENDIX D. CONTROL SOFTWARE — DIRECT ADAPTIVE CONTROL 131

```
    readln(choice);  
until ((choice='Y') OR (choice='y'));  
save_data_to_file;  
create_info_file;  
writeln;  
writeln('** Please Disable Motor 2 servo Now **');  
end.
```


Appendix E

Robot System Components Specifications

E.1 Servo Amplifier Specifications

Customer:	Carleton University	Motor:	
M C S Job:	1199	Distribution #:	
System type:	SOF-300	Interface #:	
Com #:	1677	EFM BD #:	
Gate/base:	AT97612	PWR STG #:	
Control #:	1657	EFM #:	
Interlock #:		PSM #:	10748 8826
Freq Lock #:		SCM #:	92468 8826
Divider #:			

Maximum Current: 10 Amps
Maximum DC Bus Voltage: 340 Volts
Operating Velocity: Min -0- Nominal Max 7200
Command Signal: Voltage X Freq Binary
Scaling 10 volts = 10 amps full scale
Interlock (U19) #: 1700060 V3.0
Commutation (U5, U6) #: 1700070-12, 1700080-12 V2.0
R/D Converter: 158F376 721588
Required Forced Air Cooling: Fans on Rack

OPTIONS: (option checked if installed)

1. Analog Tachometer Output
Type
Proportionality Constant (V/RPM)
- [X 2. Latching Fault Detect
Reset Method Open circuit enable pins (SCI-7,8)
to reset, short pins together to re-enable
3. External Tachometer
Model
Proportionality Constant (V/RPM)
4. Programmable Current Limit
Max Rated Current Amps
KT (torque sensitivity) lb in/Amp
Control Method:
 A: Controlled from pins 3 & 4 of MC1 control connector
 B: Controlled from front panel potentiometer on Extra Function Module
 1. turns full scale
 C: Other

- _____ 5. Simulated Incremental Encoder Output
 A. Frequency of Least Significant Bit (LSB)
 _____ cycles/rev
 B. Frequency of Most Significant Bit (MSB)
 _____ cycles /rev
- _____ 6. Velocity Command Ramp
 A. Fixed Rate _____ Volts/second
 B. Variable Rate:
 1. Control Method _____

 2. Rates:
 a. min _____ Volts/sec
 b. max _____ Volts/sec
- ^x _____ 7. Rotor Position Output Word
 A. Resolution 12 bits

SCM-150/300

	Min	Typical	Max.
--	-----	---------	------

Power Requirements:**DC Bus**

SCM150	140V	165V	180V
SCM300	260V	311V	350 V

Logic Supply

+5V@1.0A	+4.75VDC	+5.00VDC	+5.25 VDC
+15V@1.0A	+14.25VDC	+15.00VDC	+15.75 VDC
-15V@1.0A	-14.25VDC	+15.00VDC	-15.75 VDC

OUTPUT POWER RATINGS**SCM-150**

Continuous Current	10 Amps
Continuous Power	1650 W

SCM-300

Continuous Current	10 Amps
Continuous Power	3000 W

GENERAL SPECIFICATIONS

Operating Temperature	0 C	-	40 C
PWM Frequency	20 KHZ	23 KHZ	25 KHZ
Bandwidth	-	2.5 KHZ	-
Internal Tach Linearity	-	.1%	-

PSM-150 and 300**Input Voltage**

PSM-150	100VAC	115VAC	130VAC
PSM-300 (1 OR 3 PHASE)	185 VAC	220 VAC	242VAC

Output Voltage

PSM-150	140 VDC	165 VDC	180 VDC
PSM-300	260 VDC	311 VDC	342 VDC

DC Bus Current

10 A

Logic Supplies (Output)

+5V,1A	+4.75VDC	+5.00VDC	+5.25VDC
+15V,1A	+15.25VDC	+15.00VDC	+15.75VDC
-15V,0.5A	-15.25VDC	-15.00VDC	-15.75VDC

E.2 A/D Interface Card Specifications

2.2 A/D SUBSYSTEM

ANALOG INPUTS

Number of Analog Inputs	16SE/8DI
Input Ranges (Jumper-selectable)	0 to +5V (unipolar) $\pm 5V$, $\pm 2.5V$, (bipolar)
Output Data Codes	Straight binary (unipolar) or offset binary (bipolar)
Gain * DT2811-PGH DT2811-PGL	1, 2, 4, 8 1, 10, 100, 500
Input Impedance Off Channel	100M Ω , 10pF in parallel;
On Channel	100M Ω , 100pF in parallel
Bias Current	$\pm 50nA$
Common Mode Input Voltage, Maximum	$\pm 11V$
Common Mode Rejection Ratio Gain = 1	70dB at 60Hz, 1k Ω unbalance;
Gain = 500	100dB at 60Hz, 1k Ω unbalance
Maximum Input Voltage Without Damage, Power On	$\pm 32V$
Maximum Input Voltage Without Damage, Power Off	$\pm 20V$
Amplifier Input Noise	2.0 μV RMS
ACCURACY	
Resolution	12 bits
Nonlinearity	Less than $\pm 3/4$ LSB
Differential Nonlinearity	Less than $\pm 1/2$ LSB
Inherent Quantizing Error	$\pm 1/2$ LSB

System Accuracy	
DT2811-PGH, Gain = 1	Within $\pm .03\%$ FSR
Gain = 2	Within $\pm .035\%$ FSR
Gain = 4	Within $\pm .04\%$ FSR
Gain = 8	Within $\pm .05\%$ FSR
DT2811-PGL, Gain = 1	Within $\pm .03\%$ FSR
Gain = 10	Within $\pm .05\%$ FSR
Gain = 100	Within $\pm .07\%$ FSR
Gain = 500	Within $\pm .15\%$ FSR
Channel Crosstalk	-80dB at 1kHz
Gain Error	Adjustable to 0
Zero Error	Adjustable to 0

DYNAMIC PERFORMANCE

Channel Acquisition Time	
Gains of 1, 2, 4, 8, 10	20 μ s (average)
Gains of 100, 500	370 μ s (average)
A/D Conversion Time	30 μ s
A/D Converter Throughput	
Gains of 1, 2, 4, 8, 10	20kHz
Gains of 100, 500	2.5kHz
Dynamic Error	$\pm .75$ LSB for a 10V change (max.)

THERMAL CHARACTERISTICS

A/D Zero Drift	± 10 ppm of FSR/ $^{\circ}$ C
Gain Drift	± 30 ppm of FSR/ $^{\circ}$ C
Differential Linearity Drift	+/-3ppm of FSR/ $^{\circ}$ C
Monotonicity	0 to 70 $^{\circ}$ C

2.3 D/A SUBSYSTEM

ANALOG OUTPUTS

Number of Channels	2 (DAC0, DAC1)
Output Ranges	0 to +5V (unipolar) $\pm 2.5V$ (bipolar) $\pm 5V$ (bipolar)
Input Data Codes	Straight binary (unipolar); Offset binary (bipolar)
Output Drive Current	$\pm 5mA$ max.
Source Impedance (dc)	0.1Ω max.
Capacitive Drive Capability	$0.004\mu F$ typical
Protection Against	Short circuit indefinitely to Analog Common

ACCURACY

Resolution	12 bits
Nonlinearity	To within 0.01% FSR
Differential Nonlinearity	Less than $\pm 3/4$ LSB
Inherent Quantizing Error	Less than $\pm 1/2$ LSB
Gain Error	Adjustable to 0
Zero Error	Adjustable to 0
System Accuracy	$\pm 0.025\%$ FSR

DYNAMIC PERFORMANCE

Settling Time to 0.01%	
10V Step	$7\mu s$ max.
100mV Step	$1\mu s$ max.
Slew Rate	$10V/\mu s$ (up) min. $10V/\mu s$ (down) min.
DAC Throughput	50kHz max.

	(single channel)
Output Noise	0.2 LSB's max.
THERMAL CHARACTERISTICS	
Offset Drift (Bipolar)	$\pm 10\text{ppm}/^{\circ}\text{C}$ max.
Gain Drift (Bipolar)	$\pm 30\text{ppm}/^{\circ}\text{C}$ max.
Differential Linearity Drift	3ppm of FSR/ $^{\circ}\text{C}$
Monotonicity	0 to 70°C

2.4 DIGITAL I/O SUBSYSTEM

Number of Ports	Two 8-line ports
Logic Family	Positive true (ALSTTL)

DIGITAL INPUTS

Number of Lines	8
Logic Family	LSTTL
Logic Load	1 ALSTTL
Logic High Input Voltage	2.0V min.
Logic Low Input Voltage	0.8V max.
Logic High Input Current	20.0 μ A
Logic Low Input Current	-0.1mA

DIGITAL OUTPUTS

Number of Lines	8
Fanout	20 LSTTL loads
Logic High Output Voltage	2.4V min.
Logic Low Output Voltage	0.4V max.
Logic High Output Current	-15mA max.
Logic Low Output Current	24.0mA min.

2.5 INTERNAL CLOCK

Base Frequency	600kHz $\pm$ 100ppm
Divisor Values	1 to 120,000,000
Usable Frequency Range	.005Hz - 20kHz

2.6 EXTERNAL TRIGGER

Logic	Asserted Low LSTTL
Trigger Pulse Width (Low)	500ns min.
High Level Input Voltage	2.0V min.
Low Level Input Voltage	.8V max.
High Level Input Current	20 μ A max.
Low Level Input Current	-.2mA max.

2.7 EXTERNAL OSCILLATOR

Logic	High to low transition LSTTL
Pulse Width (Low)	500ns min.
Frequency	DC to 1MHz
High Level Input Voltage	2.0V min.
Low Level Input Voltage	.8V max.
High Level Input Current	20 μ A max.
Low Level Input Current	-.2mA max.

2.8 INTERFACE CHARACTERISTICS

Connector J1	3M-3596-5002; mating connector: 3M-3425
Compatible Bus	IBM PC/XT/AT and compatibles
Interface Type	I/O mapped with 10-bit addressing
Number of Locations Occupied	8 consecutive I/O locations
Number of I/O Registers	8
Address Range	200 to 3F8 (hex)
Factory Assigned Base Address	218 (hex)

Data Path	Eight bits
Interrupt Options	Selectable to level 2,3,5, or 7
Interrupt Source	A/D Done bit or Error bit in A/D Control/Status Register
Analog Data Format	12-bit, right justified

2.9 POWER REQUIREMENTS

Power Supply Range	+5V, $\pm 5V$, @ 290mA +12V, $\pm 5V$, @ 50mA -12V, $\pm 5V$, @ 100mA
--------------------	--

2.10 PHYSICAL ENVIRONMENTAL

Dimensions	4.5"HX7.0"WX0.75"D (11.4X17.8X1.9cm)
Weight	6 oz (168g)
Altitude	To 7500ft (2286m)
Operating Temperature Range	0 to +50°C (32 to 122°F)
Storage Temperature Range	-25 to +70°C (13 to 158°F)
Relative Humidity	To 95%, non-condensing

E.3 D/D Interface Card Specifications

2.2 DIO LINES

Number of Lines	32, configurable in groups of eight
Logic Family	LSTTL

2.3 DIGITAL INPUTS

Logic Family	LSTTL
Logic Load	1/8 LSTTL load
Logic High Input Voltage	2.0V min
Logic Low Input Voltage	0.8V max
Logic High Input Current	20.0 μ A
Logic Low Input Current	-0.2mA

2.4 DIGITAL OUTPUTS

Fanout	15 LSTTL loads
Logic High Output Voltage	2.4V min
Logic Low Output Voltage	0.4V max
Logic High Output Current	-15mA max
Logic Low Output Current	24.0mA min

2.5 INTERFACE CHARACTERISTICS

Compatible Bus	IBM PC/XT/AT and compatibles
Interface Type	I/O mapped with 10-bit addressing
Number of Locations Occupied	5 consecutive I/O locations
Address Range	200 to 3F8 (hex)
Factory Assigned Base Address	228 (hex)

Data Path	Eight bits
Number of I/O Registers	5

2.6 POWER REQUIREMENTS

Pover Supply Range	+5V +/- .5V
Current Drain	300mA typical 400mA max

2.7 PHYSICAL ENVIRONMENTAL

Dimensions	4.3"HX4.3"WX1.0"D (10.8X10.8X2.5cm)
Weight	3.5 oz (98g)
Altitude	To 7500ft (2286m)
Operating Temperature Range	0 to +70°C
Storage Temperature Range	-25 to +85°C
Relative Humidity	To 90%, non-condensing

E.4 Servo Motor Specifications

Motor Type:	DC, frameless
Windings:	3 phase
Torque Rating:	Peak 210 in-lb (24 N-m) Stall 36 in-lb (4 N-m)
Motor Constant:	4.16 in-lb/ $\sqrt{W}$ (0.470 N-m/ $\sqrt{W}$)

E.5 Resolver Specifications

Input Voltage:	7.0 Vrms, 2500 Hz
Output Voltage:	3.5 Vrms
Sensitivity:	61 μ V/degree
Electrical Error:	± 10 minutes

E.6 Tachometer Specifications

Constant:	300 deg/sec per volt (nominal)
Output Voltage:	90 V (max)
Ripple Voltage:	2.0 % (average to peak)
Speed Range:	0 to 450 rad/sec
Linearity:	1.0 % maximum

END

1 9/0 2/9 1

FIN