1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Examinez votre reproduction

Consultez votre référence

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.

Canada

Sampled-Data Robot Adaptive Control

by

GABRIEL D. WARSHAW, B.Sc., M.Eng.

A thesis submitted to
the Faculty of Graduate Studies and Research
in partial fulfilment of
the requirements for the degree of
Doctor of Philosophy

Department of Systems and Computer Engineering

**Carleton University
Ottawa, Ontario
January 14, 1994**

**© Copyright
1994, Gabriel D. Warshaw**

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your title / Votre référence

Your title / Votre référence

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.

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.

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 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 0-315-92972-3

Canada

AUTHOR NAME: GABRIEL WARSHAW

(TITLE) Name SAMPLED - DATA ROBOT ADAPTIVE CONTROL

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

ELECTRONICS AND ELECTRICAL ENGINEERING

0544

U·M·I

SUBJECT TERM

SUBJECT CODE

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language 0679
General 0289
Ancient 0290
Linguistics 0291
Modern
Literature 0401
General 0294
Classical 0295
Comparative 0297
Medieval 0298
Modern 0316
African 0591
American 0305
Asian 0352
Canadian (English) 0355
Canadian (French) 0593
English 0311
Germanic 0312
Latin American 0315
Middle Eastern 0313
Romance 0314
Slavic and East European

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion 0318
General 0321
Biblical Studies 0319
Clergy 0320
History of 0322
Philosophy of 0469
Theology

SOCIAL SCIENCES

American Studies 0323
Anthropology 0324
Archaeology 0326
Cultural 0327
Physical
Business Administration 0310
General 0272
Accounting 0770
Banking 0454
Management 0338
Marketing 0385
Canadian Studies
Economics 0501
General 0503
Agricultural 0505
Commerce-Business 0508
Finance 0509
History 0510
Labor 0511
Theory 0358
Folklore 0366
Geography 0351
Gerontology
History 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science 0615
General
International Law and Relations 0616
Public Administration 0617
Recreation 0814
Social Work 0452
Sociology 0626
General
Criminology and Penology 0627
Demography 0938
Ethnic and Racial Studies 0631
Individual and Family Studies 0628
Industrial and Labor Relations 0629
Public and Social Welfare 0630
Social Structure and Development 0700
Theory and Methods 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture 0473
General 0285
Agronomy
Animal Culture and Nutrition 0475
Animal Pathology 0476
Food Science and Technology 0359
Forestry and Wildlife 0478
Plant Culture 0479
Plant Pathology 0480
Plant Physiology 0817
Range Management 0777
Wood Technology 0746
Biology 0306
General 0287
Anatomy 0308
Biostatistics 0309
Botany 0379
Cell 0329
Ecology 0353
Entomology 0369
Genetics 0793
Limnology 0410
Microbiology 0307
Molecular 0317
Neuroscience 0416
Oceanography 0433
Physiology 0821
Radiation 0778
Veterinary Science 0472
Zoology
Biophysics 0786
General 0760
Medical

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
General 0566
Audiology 0300
Chemotherapy 0992
Dentistry 0567
Education 0350
Hospital Management 0769
Human Development 0758
Immunology 0982
Medicine and Surgery 0564
Mental Health 0347
Nursing 0569
Nutrition 0570
Obstetrics and Gynecology 0380
Occupational Health and Therapy 0354
Ophthalmology 0381
Pathology 0571
Pharmacology 0419
Pharmacy 0572
Physical Therapy 0382
Public Health 0573
Radiology 0574
Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
Chemistry 0485
General 0749
Agricultural 0486
Analytical 0487
Biochemistry 0488
Inorganic 0738
Nuclear 0490
Organic 0491
Pharmaceutical 0494
Physical 0495
Polymer 0754
Radiation 0405
Mathematics
Physics 0605
General 0986
Acoustics
Astronomy and Astrophysics 0606
Atmospheric Science 0608
Atomic 0748
Electronics and Electricity 0607
Elementary Particles and High Energy 0798
Fluid and Plasma 0759
Molecular 0609
Nuclear 0610
Optics 0752
Radiation 0756
Solid State 0611
Statistics 0463
Applied Sciences
Applied Mechanics 0346
Computer Science 0984

Engineering 0537
General
Aerospace 0538
Agricultural 0539
Automotive 0540
Biomedical 0541
Chemical 0542
Civil 0543
Electronics and Electrical 0544
Heat and Thermodynamics 0348
Hydraulic 0545
Industrial 0546
Marine 0547
Materials Science 0794
Mechanical 0548
Metallurgy 0743
Mining 0551
Nuclear 0552
Packaging 0549
Petroleum 0765
Sanitary and Municipal 0554
System Science 0790
Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

The undersigned hereby recommend to
The Faculty of Graduate Studies and Research
acceptance of the thesis,
Sampled-Data Robot Adaptive Control
submitted by
Gabriel D. Warshaw, B.Sc., M.Eng.
in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

Chairman, Department of Systems and Computer Engineering

Thesis Supervisor

External Examiner

Carleton University

Abstract

This thesis addresses the stability and performance of discretized adaptive control algorithms for robotic manipulator control, and the compensation of these algorithms for improved stability and tracking performance.

While many contemporary robot adaptive controllers are designed and analyzed in continuous-time, the use of a digital computer as the controller results in a sampled data system. By formulating the entire sampled-data system in continuous-time, Lyapunov's second method is used to examine the stability and error bounds.

Simulation of Slotine and Li's direct adaptive control algorithm is used to demonstrate the effect on performance and stability when a continuous-time adaptive algorithm is converted to discrete-time to be implemented in a sampled-data system. It is shown that the bounds on stability are a function of sampling time, adaptation gains, controller gains, and the desired trajectory. The discretization of Slotine and Li's algorithm results in a sampled-data system for which stability has not been guaranteed. Furthermore, simulation results indicate that under certain conditions, the system appears to be unstable.

The stability condition derived using Lyapunov's second method is used to formulate a compensator to guarantee the stability of the sampled-data implementation, and improve the tracking performance. This stability condition also gives a prediction of the bound on the filtered tracking error.

The proposed non-linear discrete-time compensating term is added to Slotine and Li's adaptive algorithm, to stabilize the sampled-data system. For sufficiently high gain, globally stable performance and a known bound on the norm of the filtered error is proven. A multi-rate discrete-time controller implementation of

the compensated adaptive controller is investigated. The stabilizing effect of the compensator and validity of the error bound predictions are demonstrated through simulation and implementation of two degree-of-freedom manipulator control.

Acknowledgements

I am indebted to my Ph.D. supervisor, Professor Howard M. Schwartz; he has helped and directed me in many ways towards the successful completion of this dissertation. His inspiration, guidance and support throughout my graduate studies were invaluable, and greatly appreciated.

I would also like to thank Danny Lemay and Dave Sword for their assistance with the care and feeding of the robot hardware and other lab equipment and computers. My appreciation extends to many others in the Systems and Computer Engineering Department for helping me along the way, and especially for the friendly help of the department office staff and Naren Mehta.

My fellow students and research colleagues also deserve a vote of thanks. Particularly deserving of this thanks are Victor Aitken and Erik Dravnieks.

Finally, I gratefully acknowledge the financial support of the Natural Sciences and Engineering Research Council of Canada, and the Ontario Graduate Scholarship program.

Dedication

to my parents, and to Wendy and Haley

Contents

Acceptance Sheet	ii
Abstract	iii
Acknowledgements	v
Dedication	vi
Table of Contents	vii
List of Tables	xi
List of Figures	xii
List of Plates	xvii
List of Acronyms and Symbols	xviii
1 Introduction	1
1.1 Adaptive Control Background	1
1.2 Robot Adaptive Control Background	2
1.3 The Sampled-Data Robot Adaptive Control Problem	5
1.4 Thesis Outline	5

2	Theoretical Background	7
2.1	Lyapunov Stability Theory	7
2.1.1	Introduction	7
2.1.2	Definitions	9
2.1.3	Lyapunov's Second Method	11
2.1.4	Lyapunov's Second Method for Discrete-Time Systems	13
2.1.5	Lyapunov's Second Method for Adaptive Systems	14
2.2	Craig's Adaptive Control Algorithm	15
2.2.1	Formulation of the Manipulator Dynamics	15
2.2.2	Formulation of the Adaptive Controller	16
2.3	Slotine and Li's Adaptive Control Algorithm	21
2.3.1	Formulation of the Adaptive Controller	22
2.3.2	Theoretical Stability Properties	24
2.4	Discussion of the Implementation of Robot Adaptive Control	25
3	Sampled-Data Adaptive Control for Robots	27
3.1	Background and Literature Review	27
3.1.1	The Analysis of Discrete-Time and Sampled-Data Adaptive Controllers	27
3.1.2	Robust Control and Compensation of Sampled-Data Adaptive Control	30
3.2	Problem Statement	32
3.3	Analysis of the State of the Art	32
3.4	Demonstration of the Problem by Simulation	34
3.4.1	Continuous-Time System	35
3.4.2	Sampled-Data System	37
4	Stability Analysis	41
4.1	Lyapunov's Second Method for Non-Linear Sampled-Data Systems	42

4.2	Analysis of the Sampled-Data Robot Adaptive System using Lyapunov's Second Method	44
4.2.1	Formulating the Stability Condition	44
4.2.2	A Steady-State Bound on the Error	48
4.2.3	Exact Evaluation of the Stability Condition for the One DOF Case	54
4.2.4	An Adaptive Controller with Variable Feedback and Adaptation Gains	60
4.3	Attempts at Robust Compensation for the Discretized Controller . .	79
4.3.1	Robust Compensation of Craig's Adaptive Controller	79
4.3.2	Robust Compensation of Slotine and Li's Adaptive Controller . .	84
4.4	A Stability Condition using Bounds	85
5	Compensation of the Adaptive Controller	89
5.1	Addition of a Discrete-Time Compensator for Guaranteed Stability .	90
5.1.1	The Stability Condition for the Compensated System	94
5.1.2	The Effect of Torque Saturation	100
5.1.3	Two DOF Robot Manipulator Example	101
5.2	A Multi-Rate Discrete-Time Adaptive Controller with Compensation	109
5.2.1	The Stability Condition	112
5.2.2	Two DOF Robot Manipulator Example	116
5.3	Derivation of an Error Bound using a Local Approach	120
5.3.1	The Error Bound Condition	120
5.3.2	Two DOF Manipulator Example	124
6	Two DOF Manipulator Implementation	130
6.1	Description of the Manipulator	130
6.2	Kinematics and Dynamics of the Manipulator	135
6.3	The Control Software	138

6.3.1	The Implementation of Slotine and Li's Adaptive Controller . . .	141
6.4	Experimental Results	142
6.4.1	Single-Rate Experiments	142
6.4.2	Multi-Rate Experiments	148
7	Conclusions	153
7.1	Discussion and Conclusions	153
7.2	Future Research	154
7.3	Summary of Contributions	155
	Bibliography	156
	APPENDICES	163
A	Simulation Software Listing	163
A.1	Main Program Listing	164
A.2	Program Unit for General Purpose Simulation Routines	178
A.3	Program "Include File" for Runge-Kutta Routine	180
B	Experimental Control Software Listing	182
B.1	Main Program Listing	183
B.2	Program Unit for Interface Card Routines	197
B.3	Program Unit for Filtering Routines	202
B.4	Program Unit for Matrix Math Routines	207

List of Tables

5.1	Evaluation of Stability Condition for 2 DOF Serial Link Manipulator	108
5.2	Stability Condition for 2 DOF Serial Link Manipulator with Multi-Rate Controller	116
5.3	Error Bounds for 2 DOF Serial Link Manipulator	121
6.1	Link Parameters for the Carleton Direct Drive Manipulator	137

List of Figures

2.1	Concepts of Stability and Asymptotic Stability	10
3.1	Desired Trajectories (— joint 1, - - - joint 2)	35
3.2	Norm of Filtered Tracking Errors (— continuous-time, - - - sampled- data)	36
3.3	Parameter Estimation Errors, Continuous-Time System	36
3.4	Norm of Filtered Tracking Errors, Continuous-Time System	37
3.5	Parameter Estimation Errors, Continuous-Time System	37
3.6	Parameter Estimation Errors, Sampled-Data System	38
3.7	Norm of Filtered Tracking Errors (— continuous-time, - - - sampled- data)	39
3.8	Parameter Estimation Errors, Sampled-Data System	39
4.1	Norm of Filtered Tracking Errors, $\ s[k]\ $, for Sampled-Data System .	53
4.2	Tracking Error for Unmodified Adaptive Algorithm, 1 DOF Sampled- Data System	62
4.3	Parameter Estimate for Unmodified Adaptive Algorithm, 1 DOF Sampled-Data	62
4.4	ΔV_k for Unmodified Adaptive Algorithm, 1 DOF Sampled-Data System	63
4.5	Tracking Error for Variable k_D Adaptive Algorithm, 1 DOF Sampled- Data System	63

4.6	Parameter Estimate for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System	64
4.7	Time-Varying Gain, k_D , for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System	64
4.8	ΔV_k for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System	65
4.9	Tracking Error for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled- Data System	65
4.10	Parameter Estimate for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System	66
4.11	Time-Varying Gain, γ^{-1} , for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System	66
4.12	ΔV_k for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data Sys- tem	67
4.13	ΔV_k Surface for Variable k_D	69
4.14	ΔV_k Surface for Variable γ^{-1}	70
4.15	ΔV_k Surface for Variable k_D and γ^{-1}	72
4.16	Selection of k_D and γ^{-1} using Extrema	73
4.17	Tracking Error for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System	74
4.18	Parameter Estimate for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System	74
4.19	Time-Varying Gain, k_D , for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System	75
4.20	Time-Varying Gain, γ^{-1} , for Variable k_D and γ^{-1} Adaptive Algo- rithm, 1 DOF Sampled-Data System	75
4.21	ΔV_k for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled- Data System	76
4.22	Tracking Error for Undersampled Case	76

4.23	Parameter Estimate for Undersampled Case	77
4.24	Time-Varying Gain, k_D , for Undersampled Case	77
4.25	Time-Varying Gain, γ^{-1} , for Undersampled Case	77
4.26	Norm of Position Tracking Errors, $\ e\ $ (— Craig, - - - Slotine and Li)	82
4.27	Parameter Estimation Errors, Craig's Adaptive Controller	82
4.28	Norm of Position Tracking Errors, $\ e\ $, Craig's Controller (- - - small initial param. error, — robust compensator added, large initial param. error)	83
5.1	Vector Representation of Filtered Error, s	91
5.2	Cubic Function for the Stability Condition ($\circ$ = roots, $\bullet$ = $\ s[k]\ _{bound}$)	99
5.3	Geometry of Serial Link Manipulator	103
5.4	Multi-Rate Adaptive Control	112
5.5	Parameter Estimates, Uncompensated Controller	118
5.6	Norm of Filtered Tracking Errors (— compensated, - - - uncompensated)	118
5.7	Control Torques, Uncompensated Controller (— joint 1, - - - joint 2)	119
5.8	Control Torques, Compensated Controller (— joint 1, - - - joint 2)	119
5.9	Effective Compensator Gain	119
5.10	Norm of Desired Torque, $\ \tau_d\ $	124
5.11	Norm of Filtered Tracking Errors, $\ s\ $ (— actual (simulation), - - - predicted bound)	125
5.12	Norm of Maximum Input Torque, $\ \tau\ _{bound}[k]$	126
5.13	Norm of Maximum Acceleration, $\ \ddot{y}\ _{max}[k]$	126
5.14	Bound on Filtered Tracking Errors, $\ s\ _{bound}[k]$, baseline adaptive case	126
5.15	Roots for Filtered Tracking Error Bound, $\ s\ _{bound}[k]$, for multi-rate adaptive control	127
5.16	Norm of Filtered Tracking Errors, $\ s\ $, for Simulation without Compensator	127

5.17	Bound on Filtered Tracking Errors, $\ s\ _{bound}[k]$, for high adaptation gain and compensation	128
5.18	Norm of Torque (— predicted bound $\ \tau\ _{bound}[k]$, - - - actual (simulation))	129
6.1	Geometry of Carleton Direct Drive Manipulator (top view)	131
6.2	System Block Diagram of Carleton Direct Drive Manipulator	132
6.3	Coordinates and Link Lengths for the Manipulator (top view)	135
6.4	Desired Position (— joint 1, - - - joint 2)	142
6.5	Norm of Filtered Tracking Errors, $\ s\ $ (· · · PD, - - - adaptive, — compensated adaptive)	143
6.6	Norm of Desired Torque, $\ \tau_d\ $	144
6.7	Norm of Filtered Tracking Errors, $\ s\ $, PD Control (— actual, - - - predicted bound)	144
6.8	Position Tracking Errors, Adaptive Control (— joint 1, - - - joint 2)	145
6.9	Parameter Estimates, Adaptive Control (a_4 , a_5 values divided by 10)	145
6.10	Norm of Maximum Input Torque, $\ \tau\ _{bound}[k]$, Adaptive Control	146
6.11	Norm of Maximum Acceleration, $\ \ddot{y}\ _{max}[k]$, Adaptive Control	146
6.12	Norm of Filtered Tracking Errors, $\ s\ $, Adaptive Control (— actual, - - - predicted bound)	147
6.13	Norm of Input Torque, Adaptive Control (— actual, - - - predicted bound, · · · · desired)	147
6.14	Norm of Filtered Tracking Errors, $\ s\ $, Adaptive Control with Compensator (— actual, - - - predicted bound)	148
6.15	Effective Compensator Gain	149
6.16	Parameter Estimates, Multi-Rate Adaptive Control without Compensator	150
6.17	Parameter Estimates, Multi-Rate Adaptive with Compensator	150
6.18	Input Torque, Multi-Rate Adaptive Control	151

6.19	Input Torque, Multi-Rate Adaptive with Compensator	151
6.20	Effective Compensator Gain, Multi-Rate Controller	151
6.21	Norm of Filtered Tracking Errors, $\ s\ $, Multi-Rate (- - - adaptive, compensated adaptive)	152
6.22	Norm of Filtered Tracking Errors, $\ s\ $, Multi-Rate Adaptive Control with Compensator (— actual, - - - predicted bound)	152

List of Plates

6.1	Direct Drive Manipulator	133
6.2	Robot System Laboratory Setup	134

List of Acronyms and Symbols

a	constant in a polynomial or positive scalar compensator gain
a_1, a_2, a_3	constants in a polynomial or inertial parameters
a_4, a_5	torque bias parameters
A	block diagonal constant matrix
$\bar{A}$	stable system matrix
A^*	square system matrix for error dynamics
ARX	Auto-Regressive eXogenous (finite difference model)
A/D	Analog to Digital (convertor)
b, b_1, b_2	constants in a polynomial
B	block diagonal constant matrix
B^*	input matrix for error dynamics
$\mathcal{B}$	positive diagonal constant matrix
c	constant in a polynomial
c_{ij}	ij^{th} component of the C matrix
C	block diagonal constant matrix
$C(y, \dot{y})$	matrix of coriolis, centrifugal and frictional terms
$\hat{C}(y, \dot{y})$	estimate of $C(y, \dot{y})$
$\hat{C}^*[k]$	estimate of C using current state and previous parameter estimates
$\tilde{C}$	error in estimation of $C(y, \dot{y})$

C-of-G	Center of Gravity
CPU	computer's Central Processing Unit
d	constant in a polynomial
DOF	degrees of freedom
DSP	Digital Signal Processor
D/A	Digital to Analog (convertor)
D/D	digital to digital convertor (parallel interface)
e	vector of joint position tracking errors
e_1	vector of filtered joint tracking errors
$\bar{e}$	vector composed of e and $\dot{e}$
E^*	term related to discretization error for M
$f(\cdot)$	a non-linear scalar or vector function
$f[k]$	discrete-time non-linear compensation term
$f_{adaptation}$	discrete adaptation frequency
$f_s, f_{sampling}$	sampling frequency
f^*	positive scalar effective compensator gain
g	scalar effective non-linear compensator gain
$G(y)$	vector of manipulator gravity torques
$\hat{G}(y)$	estimate of $G(y)$
$\hat{G}^*[k]$	estimate of G using current state and previous parameter estimates
$\tilde{G}$	error in estimation of $G(y)$
$H\{\cdot\}$	the Zero Order Hold operator
i_j	j^{th} scalar inertia parameter
I	the identity matrix, or effective inertia of a 1 DOF system
k	discrete time index
$\bar{k}$	discrete time index for parameter adaptation
k_D	scalar derivative feedback gain
K	matrix of position and velocity feedback gains

K_D	positive definite derivative feedback gain matrix
K_p	matrix of position feedback gains
K_v	matrix of velocity feedback gains
l	link length
LTI	Linear Time-Invariant
m	link mass, or positive integer multiplying Δt to get Δt
m_{ij}	ij^{th} component of the mass matrix M
m^*	$\sum_{j=1}^{m-1} j$
$M(y)$	manipulator mass matrix
$\hat{M}(y)$	estimate of $M(y)$
$\hat{M}^*[k]$	estimate of M using current state and previous parameter estimates
$\tilde{M}$	error in estimation of $M(y)$
$\bar{M}$	bound on norm of M^{-1}
MCS	Motion Control Systems, Inc.
MHz	Mega-Hertz
MIMO	Multiple-Input/Multiple-Output
MKS	Meter-Kilogram-Second metric system of units
n	number of degrees of freedom
p_x, p_y	manipulator end point cartesian coordinates
PC	Personal Computer
PD	Proportional Derivative (feedback controller)
$\mathcal{P}$	symmetric positive definite matrix
$Q(y, \dot{y})$	vector of centrifugal, coriolis, gravity and frictional torques
$\hat{Q}(y, \dot{y})$	estimate of $Q(y, \dot{y})$
$\tilde{Q}$	error in estimation of $Q(y, \dot{y})$
Q_1	bound on $\ \tilde{y}_d\ $
$\mathcal{Q}$	symmetric positive definite matrix
RC	Resistor/Capacitor (filter)

RLS	Recursive Least Squares
$\mathbb{R}^n$	set of vector valued real numbers of dimension n
$\mathbb{R}^{n \times n}$	set of matrix valued real numbers of dimension n by n
s	vector of filtered joint tracking errors
$s'[k]$	a row vector component of σ_1
S_k	summation used for multi-rate adaptive law
SCARA	Selective Compliance Assembly Robot Arm
SISO	Single-Input/Single-Output
STR	Self-Tuning Regulator
t	time
t_0	arbitrary or initial moment in time
t_k	time at the sampling instant k
t_k^-	infinitesimally small time before t_k
$\bar{t}$	time since the last sampling instant t_k
$u_{feedback}$	control torque due to feedback
$V(x, t)$	Lyapunov function candidate
$V_1, \dots, V_9$	terms composing ΔV
VSC	Variable Structure Control
$W(y, \dot{y}, \ddot{y})$	regression matrix for the manipulator model
W_d	W evaluated along the desired trajectory
$W_r(y, \dot{y}, \dot{y}_r, \ddot{y}_r)$	regression matrix using reference velocities and accelerations
x	vector of system states
$x(t; x_0, t_0)$	solution path for the states of a system, starting from initial time t_0 and initial state x_0
x_e	vector of system equilibrium states
X	vector of system states
$\mathcal{X}$	function bounding $\ \Delta v\ $
y	vector of joint (actual) positions

y_d	vector of desired joint positions
y_{lim}	vector of operational limits on the joint positions
y_r	vector of reference joint positions
$\dot{y}_{lim}$	vector of operational limits on the joint velocities
$\tilde{y}^*$	vector of desired joint accelerations plus feedback
$z(t)$	vector of system states
ZOH	Zero Order Hold
α	positive scalar constant
$\alpha(\ x\)$	time-invariant non-decreasing positive definite scalar function
α_1, α_2	positive scalar constants
β	positive scalar constant
$\beta(\ x\)$	a continuous, non-decreasing scalar function
γ^{-1}	scalar adaptation gain
Γ	diagonal matrix of positive constant adaptation gains, for Craig's adaptive law
Γ^{-1}	positive definite matrix of adaptation gains, for Slotine and Li's adaptive law
δ	a real scalar
$\delta(t)$	unit impulse function
δ_i	maximum travel distance for the i^{th} joint
δv	robust compensation term
ΔQ	discretization error for Q
$\Delta s[i, k]$	$s[i] - s[k]$
Δt	discrete sampling time
$\Delta \bar{t}$	discrete sampling time for (slower) parameter adaptation
Δv	discretization error for δv
$\Delta V(x)$	change in the Lyapunov function between time steps k
$\Delta \tilde{y}_r[k]$	$\tilde{y}_r[k+1] - \tilde{y}_r[k]$

$\Delta' \ddot{y}_r[k]$	$-\Delta \ddot{y}_r[k]$
ϵ	a real scalar, boundary layer thickness parameter
θ	vector of model inertial parameters
$\hat{\theta}$	estimate of θ vector
$\hat{\theta}_{bound}$	limit set on estimate of θ vector
$\tilde{\theta}$	error in estimation of θ vector
λ	positive scalar constant
Λ	positive definite constant matrix
ρ	positive scalar constant
$\rho(\tilde{e}, t)$	uncertainty bound
$\sigma_{max}(\cdot)$	maximum eigenvalue of a matrix
σ_1	integral of s^T over a sampling interval
τ	vector of joint torques
τ_{bias}	vector of bias torques
τ_c	vector of joint control torques
τ_d	vector of desired joint torques
τ_h	vector of held joint control torques (in continuous-time)
τ_i	joint torques for the i^{th} joint
τ_{sat}	vector of saturation torques
Ψ	diagonal matrix of positive constants
Ω	a region of the state space
$\nabla f(x)$	the gradient $\frac{\partial f(x)}{\partial x}$
$\ \cdot\ $	Euclidean norm (of a vector)
$\ \cdot\ _2$	matrix 2-norm
$\ \cdot\ _{max}$	maximum possible value of the norm
$\ \cdot\ _{bound}$	predicted maximum bound on the norm
$\ \Delta s_{max}\ $	maximum possible change in $\ s[k]\ $ between discrete time steps

Chapter 1

Introduction

1.1 Adaptive Control Background

It is difficult to succinctly define adaptive control. One definition of an adaptive controller is a system that monitors its own performance and modifies its parameters in the direction of better performance. However, this definition can make it difficult to distinguish between a conventional feedback system and a system commonly described as adaptive (see Narendra and Annaswamy [34]). An alternative definition of an adaptive controller is a special type of nonlinear feedback controller in which the states of the process can be separated into two categories, which change at different rates (Astrom and Wittenmark [5]). A fast time scale can be attributed to the ordinary feedback and a slower one for updating the adaptive parameters.

Although research in adaptive control dates back to the 1950's, it is only within the last decade that the use of adaptive controllers in industry has become relatively commonplace. The decreasing cost and increasing performance of microcomputers have facilitated the commercial introduction of many types of adaptive controllers. The techniques of automatic tuning, gain scheduling¹, and continuous adaptation

¹It is debatable whether gain scheduling should be considered as an adaptive technique, since the parameters are changed in open loop (using an auxiliary variable); however, the gain schedule can be derived using automatic tuning controllers.

of feedforward and feedback have been applied to improve the performance of flight control, ship autopilots and process control (Astrom and Hagglund [4]). However, most adaptive controllers in use so far are for single-loop process control. The practical application of adaptive control techniques to multivariable and non-linear systems such as robotic manipulators is still in its infancy.

1.2 Robot Adaptive Control 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 control². In contrast, for 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"

²Other characteristics of gear-driven manipulators do, however, result in significant nonlinearities, notably backlash and stiction.

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. [3], 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 [25] 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 theoretically 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 [16]. 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 [24]. This work employed linear finite difference equations 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 [20] 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.

There are currently two main categories of inertial model based robot adaptive control techniques:

- indirect methods, which use the torque prediction error to update the parameters of the system dynamic model, which is in turn used to modify the control law (see for example Middleton and Goodwin [31], Hsu et al [21] or Li and Slotine [28]);
- direct methods, which use the tracking error to directly modify the controller parameters (see for example Craig et al [12], Slotine and Li [51] or Sadegh and Horowitz [43]).

For these methods, 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 Atkeson et al. [6].

For the indirect adaptive algorithms, the linear regression model formulation permits the use of least squares, or recursive least squares techniques to identify the unknown parameters, even though the system model is non-linear. The identified model is used in a computed torque control structure. Thus 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. An excellent review of least squares identification techniques, as applied to robotic manipulators, can be found in Li and Slotine [27].

For the direct adaptive control methods, the parameter adaptation algorithms are driven by the trajectory tracking errors. The adaptation algorithms are usually derived based on the Lyapunov stability criteria, which guarantees the global convergence of the error dynamics. 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 [51] is used.

1.3 The Sampled-Data Robot Adaptive Control Problem

Adaptive control laws for robotic manipulator control are typically formulated in continuous-time, and then they are discretized for the implementation. The resulting manipulator system is a sampled-data one. Because the formulation of the adaptive controllers is in continuous-time, the theoretical predictions of performance and stability based on this formulation do not apply to the sampled-data implementation.

A method for predicting the stability and bounds on the filtered tracking error for a sampled-data adaptive manipulator system is presented in this thesis. Lyapunov's direct method is used to examine the stability and error bounds, by formulating the entire sampled-data manipulator system in continuous-time. The derived stability condition is used to formulate a non-linear discrete-time compensator to guarantee the stability of the sampled-data implementation of Slotine and Li's direct adaptive controller, and improve the tracking performance. The validity of the stability analysis and compensation approach is demonstrated by a combination of simulation and implementation results for two DOF manipulators.

1.4 Thesis Outline

Chapter 2 of this thesis provides background theory for Lyapunov stability and direct adaptive robot control algorithms.

Chapter 3 is a statement of the previously unsolved sampled-data robot adaptive control problem addressed in this thesis, and includes a critical review of the literature. The sampled-data adaptive control problem is illustrated by simulation results for a two DOF manipulator.

Chapters 4 through 6 describe how the sampled-data robot adaptive control problem has been investigated. Chapter 4 presents the approach for analysis of the sampled-data adaptive robot system using Lyapunov's second method. The compensation of the discretized robot adaptive controller to guarantee stability is described in Chapter 5. A multi-rate version of the compensated adaptive controller, and a

local approach for improving the predictions of stability and tracking performance, are also proposed in Chapter 5. The implementation of the compensated adaptive controller for a two DOF direct drive manipulator is described in Chapter 6.

The conclusions are found in Chapter 7. Recommendations for future research and a summary of contributions embodied in this thesis are also included in this chapter.

Appendices containing software source listings for the simulations and implemented controllers follow the list of references at the end this thesis document.

Chapter 2

Theoretical Background

2.1 Lyapunov Stability Theory

2.1.1 Introduction

The analysis of stability is an important element when characterizing the behaviour of any dynamic system. For the design of practical control systems, such as robotic manipulator controllers, it is essential. While many proven techniques for stability analysis of linear time-invariant (LTI) systems exist, the analysis of time-varying and non-linear systems is more problematic. Robotic manipulators are generally non-linear and coupled Multiple-Input/Multiple-Output (MIMO) systems. Manipulators using motors with high gear ratios ("indirect" drive) have traditionally been analyzed as a collection of independent and linear joint dynamics. As manipulator speeds and accuracy requirements have increased, this idealization has become increasingly inadequate, especially for direct drive manipulators¹. Thus, manipulator dynamics must be considered as inherently non-linear.

Robotic manipulator systems are generally implemented as sampled-data systems, which are a form of hybrid continuous/discrete-time systems. The manipulator is the continuous-time plant, and the controller is implemented in discrete-time using a digital computer and associated analog-to-digital (A/D), sampling and digital-to-

¹The high gear ratios of indirect drive manipulators makes the inertia "seen" by a motor relatively insensitive to changes such as those due to motion of the other manipulator joints or due to a change in the payload mass.

analog (D/A) conversion hardware. When adaptive control laws are implemented in discrete-time, an additional non-linear element is added to the system due to the adaptation dynamics. The problem becomes the stability analysis of MIMO coupled non-linear hybrid system — a formidable analytical challenge.

Lyapunov's second (or direct) method is a general method for determining the stability of non-linear and time-varying dynamic systems. For a dynamic system described by ordinary differential equations, Lyapunov's direct method can be used to determine the stability without solving the differential equations. This is a very important feature, because some non-linear and time-varying differential equations can be extremely difficult to solve. Lyapunov's first method for determining stability requires the solution of the differential equations; this method will not be discussed further. While the application of Lyapunov's direct method to the stability analysis of a non-linear system is often difficult, it can provide useful stability results which are not obtainable by other methods.

Lyapunov's direct method makes use of an "energy" function for the system being analyzed, which is a function of all of the states of the system. By showing that this energy function has certain properties (namely, that it cannot grow in an unbounded fashion), the stability properties of the system can be proven. In order to make use of Lyapunov's direct method, an energy function (the Lyapunov function candidate) must first be chosen. Selecting an appropriate Lyapunov function candidate can be a difficult step. Once this is accomplished, examination of its time-derivative indicates the stability properties of the system.

The sections which follow describe Lyapunov's direct method. This material was drawn from a number of texts, most prominently Kalman and Bertram [23], Narendra and Annaswamy [34], Lindorff [29] and Ogata [37]. The system under consideration is defined, as are various types of stability. Selection of a Lyapunov function candidate is discussed next. The direct method is then presented for the continuous-time case. Finally, extension of the method to the discrete-time case and application of the method for adaptive systems are discussed.

2.1.2 Definitions

Define the general form of the non-linear non-autonomous system as a system of differential equations:

$$\dot{x} = f(x, t) \quad (2.1)$$

where $x \in \mathbb{R}^n$ is the vector of system states and $f(x, t) \in \mathbb{R}^n$ is the vector of functions of x and t . For a given initial condition $x = x_0$ at $t = t_0$, it is assumed that a unique solution to the system (2.1) exists, and is denoted by $x(t; x_0, t_0)$. An equilibrium state of the system, x_e , is a state which corresponds to a constant solution of the system, and is given by:

$$f(x_e, t) = 0 \quad \text{for all } t. \quad (2.2)$$

The stability of the equilibrium state of the system can be transformed into the equivalent problem of the stability of the origin of the states, by a translation of the coordinates of the states. Thus, in the development which follows, only the stability of the equilibrium states at the origin will be examined, i.e., for $f(0, t) = 0$.

Stability (in the sense of Lyapunov)

The equilibrium state $x = 0$ of (2.1) is *stable* (in the sense of Lyapunov) if for every $\varepsilon > 0$ and $t_0 \geq 0$, there exists a $\delta(\varepsilon, t_0) > 0$ such that $\|x_0\| < \delta$ implies that $\|x(t; x_0, t_0)\| < \varepsilon \quad \forall \quad t \geq t_0$. In other words, stability (in the sense of Lyapunov) means that the solution trajectory stays close to the origin if the initial condition is sufficiently close to it. This concept is graphically represented in Figure 2.1. If δ is independent of the initial time t_0 , then the equilibrium state is uniformly stable.

Asymptotic Stability

The equilibrium state $x = 0$ of (2.1) is *asymptotically stable* if it is stable (in the sense of Lyapunov) and $\|x_0\| < \delta$ implies that $\|x(t; x_0, t_0)\| < \varepsilon \quad \forall \quad t \geq t_0$ and this solution trajectory converges to the origin as $t \rightarrow \infty$. In other words, asymptotic stability means that all solution trajectories converge to the origin if the initial condition is sufficiently close to it. This concept is also graphically represented in Figure 2.1.

Figure 2.1: Concepts of Stability and Asymptotic Stability

Global Stability

If stability holds for all initial conditions $x = x_0$ in the state space, then the origin has *global stability* (stability in the large).

Positive Definiteness of Scalar Functions

Define Ω as a region of the state space which includes the origin. A scalar function $V(x)$ is *positive definite* in the region Ω if $V(x) > 0$ for all states $x \neq 0$, and $V(0) = 0$.

A time-varying scalar function $V(x, t)$ is *positive definite* in the region Ω if it is bounded from below by a time-invariant non-decreasing positive definite scalar function $\alpha(\|x\|)$ such that:

$$V(x, t) \geq \alpha(\|x\|) \quad \forall \quad t \geq t_0, \quad x \neq 0 \quad (2.3)$$

$$V(0, t) = 0 \quad \forall \quad t \geq t_0. \quad (2.4)$$

Negative Definiteness of Scalar Functions

A scalar function $V(x)$ is *negative definite* if $-V(x)$ is positive definite.

Positive Semidefiniteness of Scalar Functions

A scalar function $V(x)$ is *positive semidefinite* if $V(x) > 0$ for all states x in the region Ω except for $x = 0$ and for certain other states x , where $V(x) = 0$.

Negative Semidefiniteness of Scalar Functions

A scalar function $V(x)$ is *negative semidefinite* if $-V(x)$ is positive semidefinite.

Indefiniteness of Scalar Functions

A scalar function $V(x)$ is *indefinite* in the region Ω if it can be either positive or negative, no matter how small the region Ω is.

2.1.3 Lyapunov's Second Method

Lyapunov's main stability theorem gives a sufficient condition for the asymptotic stability of a general non-linear continuous-time system, which may be non-autonomous.

Theorem 1 *The equilibrium state $x = 0$ of (2.1) is uniformly asymptotically stable if there exists a scalar function $V(x, t)$ which has continuous first partial derivatives with respect to x and t and satisfies the following conditions:*

1. $V(x, t)$ is *positive definite*
2. $\dot{V}(x, t)$ is *negative definite*
3. $V(x, t)$ is *decreasing*, i.e. there exists a continuous non-decreasing scalar function $\beta(\|x\|)$ such that $\beta(0) = 0$ and $V(x, t) \leq \beta(\|x\|)$ for all t .

If in addition, $V(x, t)$ is *radially unbounded*, such that $\alpha(\|x\|) \rightarrow \infty$ as $\|x\| \rightarrow \infty$, then the equilibrium is uniformly *globally* asymptotically stable. If condition 2 above is replaced with the condition that $\dot{V}(x, t) \leq 0$ for all x and t , then the equilibrium is no longer uniformly asymptotically stable, but it is uniformly stable (in sense of Lyapunov).

Note that $\dot{V}(x, t)$ is the total derivative of $V(x, t)$:

$$\dot{V}(x, t) = \frac{dV(x, t)}{dt} = \frac{\partial V}{\partial t} + (\nabla V)^T f(x, t) \quad (2.5)$$

where ∇V indicates the gradient of $V(x, t)$, i.e. $\frac{\partial V}{\partial x}$.

If the function f in (2.1) is not explicitly dependant on time, then the system is *autonomous*. The autonomous system can be defined as:

$$\dot{x} = f(x). \quad (2.6)$$

Assume that the equilibrium is at the origin, such that $f(0) = 0$. Lyapunov's stability theorem for this autonomous system can be stated in a more simplified form than for the general (non-autonomous) case.

Theorem 2 *The equilibrium state $x = 0$ of (2.6) is uniformly globally asymptotically stable if there exists a scalar function $V(x)$ which has continuous first partial derivatives with respect to x and satisfies the following conditions:*

1. $V(x)$ is positive definite
2. $\dot{V}(x) < 0$ for all $x \neq 0$
3. $V(x) \rightarrow \infty$ as $\|x\| \rightarrow \infty$

To use these stability theorems to prove the stability properties of a system, the Lyapunov function candidate, $V(x, t)$, must be selected. $V(x, t)$ must be a function of all the states of the system contained in the vector x ; otherwise $V(x, t)$ could be equal to zero at some points in the state space other than the origin. Although any scalar function which satisfies the conditions stated for $V(x, t)$ in Theorem 1 can be used as the Lyapunov function, it can be difficult in practice to find a function which does satisfy all the conditions (except for some of the most simple systems, where an educated guess may be sufficient). No explicit Lyapunov function selection method exists except for the case of linear time-invariant (LTI) systems (see Narendra and Annaswamy [34] or Ogata [37]).

Proving stability using Theorem 1 requires that $\dot{V}(x, t)$ is negative definite; this can be very difficult to satisfy. In practice, it can be much easier to select a Lyapunov function $V(x, t)$ for which $\dot{V}(x, t)$ is negative *semidefinite*. If the additional restriction is placed on $\dot{V}(x, t)$ that it does not vanish identically along any solution trajectory except at the origin, then negative semidefiniteness of $\dot{V}(x, t)$ is sufficient to prove asymptotic stability. The resulting stability theorem for the autonomous system case, originally attributed to J.P. LaSalle, is as follows (see [23], [34]):

Theorem 3 *The equilibrium state $x = 0$ of (2.6) is globally asymptotically stable if there exists a scalar function $V(x)$ which has continuous first partial derivatives with respect to x and satisfies the following conditions:*

1. $V(x)$ is positive definite
2. $\dot{V}(x) \leq 0$ for all x
3. $V(x) \rightarrow \infty$ as $\|x\| \rightarrow \infty$
4. $\dot{V}(x(t; x_0, t_0))$ does not vanish identically in $t \geq t_0$ for any t_0 and any $x_0 \neq 0$.

There is also a theorem corresponding to Theorem 3 for the non-autonomous case (see [34, page 58], [37]). For the non-autonomous case, if there exists a positive definite scalar function $V(x, t)$ for which $\dot{V}(x, t)$ is negative *semidefinite* and $\dot{V}(x, t)$ can be identically zero for a trajectory in state space outside the origin, then the system can remain in a limit cycle, and the origin is stable in the sense of Lyapunov, and not asymptotically stable.

2.1.4 Lyapunov's Second Method for Discrete-Time Systems

Lyapunov's second method can also be used to investigate the stability of discrete-time non-linear systems (see Kalman and Bertram [23], Hahn [19], Willems [65]). In Lindorff [29] and Franklin et al [17], the discrete-time system is defined as a set of non-linear autonomous difference equations:

$$x[k+1] = f(x[k]) \quad (2.7)$$

where k is the discrete time index, $x \in \mathbb{R}^n$ is the vector of system states and $f(x[k]) \in \mathbb{R}^n$ is the vector of functions of x . For a given initial condition $x = x_0$ at $k = k_0$, it is assumed that a unique solution to the system (2.7) exists, and is denoted by $x[k; x_0, k_0]$. Assume that the equilibrium state of the system occurs at the origin, such that $f(0) = 0$.

Theorem 4 *The equilibrium state $x = 0$ of (2.7) is globally asymptotically stable if there exists a scalar function $V(x)$ which is continuous with respect to x and satisfies the following conditions:*

1. $V(x)$ is positive definite
2. $\Delta V(x) \equiv V(x[k+1]) - V(x[k]) = V(f(x[k])) - V(x[k]) < 0$ for all $x \neq 0$
3. $V(x) \rightarrow \infty$ as $\|x\| \rightarrow \infty$

The stability of a system defined by non-autonomous difference equations can be stated by a theorem which is entirely analogous to the continuous-time case given by Theorem 1 (Kalman and Bertram [23]).

A major difficulty with the use of Lyapunov's second method for non-linear discrete-time systems is that it can be even more difficult to find the Lyapunov function $V(x)$ for the system than for the continuous-time case (Franklin et al [17]).

2.1.5 Lyapunov's Second Method for Adaptive Systems

Lyapunov's second method is an important tool for both the analysis of adaptive systems, and for the derivation of stable adaptive laws. The method works well for many continuous-time systems, and in fact is the basis for the formulation of many robot adaptive control algorithms, including Slotine and Li's. The existence of a Lyapunov function can be used to prove global stability of an adaptive system, but as mentioned earlier, the selection of the Lyapunov function is difficult for adaptive (and hence non-linear) systems. While the description of Lyapunov stability theory given above covered both the autonomous and non-autonomous cases, it is the non-autonomous case which is of interest for adaptive control when the reference input

is time-varying. This is true for adaptive position control, where the reference input is the desired trajectory.

2.2 Craig's Adaptive Control Algorithm

Adaptive algorithms for manipulator control can be classified into two groups:

- direct methods, which use the tracking error to directly modify the controller parameters (see for example Craig et al [12], Li and Slotine [51] or Sadegh and Horowitz [43]);
- indirect methods, which use the torque prediction error to update the parameters of the system dynamic model, which is in turn used to modify the control law (see for example Middleton and Goodwin [31], Hsu et al [21] or Li and Slotine [28]).

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.

Craig et al [12] proposed a direct adaptive controller which combines the computed-torque control technique with a gradient parameter update algorithm. Craig's adaptation algorithm is derived based on the Lyapunov stability criteria, which guarantees the global convergence of the tracking error dynamics for the continuous-time case.

2.2.1 Formulation of the Manipulator Dynamics

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

$$\tau = f(\ddot{y}, \dot{y}, y) \quad (2.8)$$

where $\tau \in \mathbb{R}^n$ is the vector of joint torques and forces, $y \in \mathbb{R}^n$ is the vector of joint positions, and n is the number of joints of the manipulator.

For a rigid link manipulator with rotational joints, the equations of motion can be specified as a matrix equation according to the above formulation as follows, according to Craig et al [12]:

$$\tau = M(y)\ddot{y} + Q(\dot{y}, y) \quad (2.9)$$

where $M(y) \in \mathbb{R}^{n \times n}$ is the mass matrix, and $Q(y, \dot{y}) \in \mathbb{R}^n$ is the 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 [6] that the equations of dynamics can also be written as:

$$\tau = W(y, \dot{y}, \ddot{y})\theta. \quad (2.10)$$

The torque vector, τ , has been expressed as a linear regression in terms of the inertial parameters of the system, given by θ , and functions of the trajectory of the manipulator given by $W(y, \dot{y}, \ddot{y})$.

2.2.2 Formulation of the Adaptive Controller

Computed-Torque Control

Craig's direct adaptive robot controller is a form of model-based controller [12]. 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.

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 manipulator position. In contrast, the error dynamics for the feedforward case do change as the arm position changes (see Khosla and Kanade [25], Craig [10]).

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 (2.9), after Craig et. al [12]:

$$\tau_c = M(y)\ddot{y}^* + Q(\dot{y}, y) \quad (2.11)$$

where

τ_c = vector of joint control torques

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

$e = y_d - y$ = 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.

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 achievable². However, adaptive control can address this problem through on-line estimation of the manipulator dynamics.

Craig's Adaptive Law

Craig et al [12] proposed an adaptive controller which combines the computed-torque control technique embodied in (2.11) with a gradient update algorithm based on the direct method for on-line parameter estimation. It is assumed that the manipulator dynamic model structure is known but that the actual parameter values are unknown.

Craig's adaptation algorithm is 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. Because the direct method is used, the tracking error is directly minimized by the adaptation mechanism.

The derivation of the direct adaptation law employed by Craig is summarized below. The second method of Lyapunov is used to guarantee the global convergence of the error dynamics, for the continuous-time case.

The adaptive control law is derived from the computed-torque control law (given previously as (2.11)):

$$\tau_c = \hat{M}(y)\ddot{y}^* + \hat{Q}(\dot{y}, y), \quad (2.12)$$

where

τ_c = vector of joint control torques

$\hat{M}$ = estimate of M

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

By equating (2.9) and (2.12), and substituting in the expression for $\ddot{y}^*$, we get

²The manipulator dynamics may not be known a-priori, and may also change over time

the following error equation:

$$\hat{M}(y)\ddot{y}_d + \hat{M}(y)K_v\dot{e} + \hat{M}(y)K_pe + \hat{Q}(\dot{y}, y) = M(y)\ddot{y} + Q(\dot{y}, y). \quad (2.13)$$

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):

$$\tilde{M} = M - \hat{M} \quad (2.14)$$

$$\tilde{Q} = Q - \hat{Q}. \quad (2.15)$$

Combining (2.13) and $\tilde{e} = \ddot{y}_d - \ddot{y}$, we get:

$$\hat{M}(\tilde{e} + \ddot{y}) + \hat{M}K_v\dot{e} + \hat{M}K_pe = M\ddot{y} + Q - \hat{Q}. \quad (2.16)$$

This can be rewritten using (2.15) to get:

$$\hat{M}\tilde{e} + \hat{M}K_v\dot{e} + \hat{M}K_pe = (\tilde{M}\ddot{y} + \tilde{Q}). \quad (2.17)$$

Rewriting once again and substituting in (2.14), we get the (tracking) error equation:

$$\tilde{e} + K_v\dot{e} + K_pe = \hat{M}^{-1}(\tilde{M}\ddot{y} + \tilde{Q}). \quad (2.18)$$

If the linear regression form of the matrix equations of motion given previously by (2.10) 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:

$$\tilde{\theta} = \theta - \hat{\theta} \quad (2.19)$$

The error equation (2.18) can now be rewritten:

$$\tilde{e} + K_v\dot{e} + K_pe = \hat{M}^{-1}W\tilde{\theta}. \quad (2.20)$$

Define the filtered tracking error vector

$$e_1 = \dot{e} + \Psi e \quad (2.21)$$

where Ψ is a diagonal matrix of positive constants. The error equation (2.20) can be written in state space form as follows:

$$\dot{X} = AX + B\hat{M}^{-1}W\tilde{\theta} \quad (2.22)$$

$$e_1 = CX \quad (2.23)$$

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

$$A^T P + PA = -Q \quad (2.24)$$

$$PB = C^T \quad (2.25)$$

where 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 both X and $\tilde{\theta}$:

$$V(X, \tilde{\theta}) = X^T P X + \tilde{\theta}^T \Gamma^{-1} \tilde{\theta} \quad (2.26)$$

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 (2.26) with respect to time and making use of (2.24) and (2.25), we get

$$\dot{V}(X, \tilde{\theta}) = -X^T Q X + 2\tilde{\theta}^T (W^T \hat{M}^{-1} e_1 + \Gamma^{-1} \dot{\tilde{\theta}}). \quad (2.27)$$

By choosing

$$\dot{\tilde{\theta}} = -\Gamma W^T \hat{M}^{-1} e_1 \quad (2.28)$$

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 (2.19), a parameter adaptation law can be written from (2.28) as follows:

$$\dot{\hat{\theta}} = \Gamma W^T \hat{M}^{-1} e_1, \quad (2.29)$$

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

There are two main disadvantages of Craig's adaptive law:

- joint accelerations are employed by the adaptive law and therefore must be either directly measured or derived;
- the estimated mass matrix must be inverted.

To satisfy the richness condition, and thereby in addition to guarantee convergence of the parameter estimates, the desired trajectory must satisfy the following:

$$\alpha I_r \leq \int_{t_0}^{t_0+\rho} W_d^T W_d dt, \quad \text{for all } t_0, \quad (2.30)$$

where

$\alpha, \rho =$ positive scalars

$I_r =$ identity matrix, of dimension r

$r =$ number of parameters to be estimated

$W_d = W$ evaluated along the desired trajectory.

2.3 Slotine and Li's Adaptive Control Algorithm

Slotine and Li have proposed a direct adaptive algorithm for manipulator control [51]. This algorithm has been extensively studied, as it is relatively simple computationally, does not require measurement of accelerations, and has good theoretical stability and convergence properties (in continuous-time). Other references which cover various aspects of Slotine and Li's algorithm are Slotine and Li [52, 53], Reed and Ioannou [40], and Spong et al [55]. The implementation of Slotine and Li's adaptive algorithm is a major topic of this thesis.

2.3.1 Formulation of the Adaptive Controller

The equations of dynamics for a robot manipulator, assuming only rigid body motion, can be written as a variant of the matrix equation stated previously as (2.9):

$$\tau = M(y)\ddot{y} + C(y, \dot{y})\dot{y} + G(y) \quad (2.31)$$

where, as before, $\tau \in \mathbb{R}^n$ is the vector of joint torques and forces, $y \in \mathbb{R}^n$ is the vector of joint positions, and $M(y) \in \mathbb{R}^{n \times n}$ is the mass matrix. Now $C(y, \dot{y}) \in \mathbb{R}^{n \times n}$ is the matrix of coriolis, centrifugal and friction terms, and $G(y) \in \mathbb{R}^n$ is the vector of gravity terms. Also as before, the equations of dynamics can also be expressed in linear regression form according to (2.10).

The continuous-time control law proposed by Slotine and Li [51] can be written as,

$$\tau = \hat{M}(y)\ddot{y}_r + \hat{C}(y, \dot{y})\dot{y}_r + \hat{G}(y) - K_D s \quad (2.32)$$

where $\hat{M}$, $\hat{C}$ and $\hat{G}$ denote the estimates of M , C and G , and K_D is a positive definite matrix of feedback gains. The parameter adaptation law can be written as,

$$\dot{\hat{\theta}} = -\Gamma^{-1} W_r^T(y, \dot{y}, \dot{y}_r, \ddot{y}_r) s \quad (2.33)$$

where Γ^{-1} is a positive definite matrix of adaptation gains,

$$W_r(y, \dot{y}, \dot{y}_r, \ddot{y}_r) \hat{\theta} = \hat{M}(y)\ddot{y}_r + \hat{C}(y, \dot{y})\dot{y}_r + \hat{G}(y) \quad (2.34)$$

$$\dot{y}_r = \dot{y}_d - \Lambda e \quad (2.35)$$

$$s = \dot{e} + \Lambda e \quad (2.36)$$

and Λ is a positive definite constant matrix. Note that according to Slotine and Li's convention, the position tracking error, $e = y - y_d$, is defined as the negative of e previously defined for the formulation of Craig's adaptive controller. An important aspect of this algorithm is that the regression matrix, $W_r(y, \dot{y}, \dot{y}_r, \ddot{y}_r)$, is not a function of the manipulator acceleration, so an acceleration measurement does not have to be taken or derived.

The adaptation law is derived from the “Lyapunov-like” function³ given by,

$$V(z(t)) = \frac{1}{2}s^T M(y)s + \frac{1}{2}\tilde{\theta}^T \Gamma \tilde{\theta} \quad (2.37)$$

where $\tilde{\theta} = \hat{\theta} - \theta$ is the parameter estimation error vector, and $z(t)$ is the vector of the system states (including in this case, s and $\tilde{\theta}$). Although the adaptive system is non-autonomous, the Lyapunov function candidate as given by (2.37) is not explicitly a function of time and hence can be denoted as $V(z(t))$.

Taking the derivative of $V(z(t))$

$$\dot{V}(z(t)) = s^T \dot{M}(y)s + \frac{1}{2}s^T \dot{M}(y)s + \tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \quad (2.38)$$

and using the skew-symmetric property of the matrix $\dot{M}(y) - 2C'(y, \dot{y})$ (refer to Slotine and Li [52] or Schwartz et al [48]):

$$\dot{V}(z(t)) = s^T (\tau - G(y) - M(y)\ddot{y}_r - C(y, \dot{y})\dot{y}_r) + \tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \quad (2.39)$$

If the control law is chosen as (2.32), we can then write the derivative of the Lyapunov function as,

$$\dot{V}(z(t)) = s^T (\tilde{M}(y)\ddot{y}_r + \tilde{C}(y, \dot{y})\dot{y}_r + \tilde{G}(y) - K_D s) + \tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \quad (2.40)$$

We can then write $\tilde{M}(y)\ddot{y}_r + \tilde{C}(y, \dot{y})\dot{y}_r + \tilde{G}(y)$ as a linear regression in the inertial parameter estimation errors as,

$$\tilde{M}(y)\ddot{y}_r + \tilde{C}(y, \dot{y})\dot{y}_r + \tilde{G}(y) = W_r(y, \dot{y}, \ddot{y}_r, \ddot{y}_r)\tilde{\theta} \quad (2.41)$$

and then,

$$\dot{V}(z(t)) = s^T W_r(y, \dot{y}, \ddot{y}_r, \ddot{y}_r)\tilde{\theta} - s^T K_D s + \tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \quad (2.42)$$

To achieve a stable system in the sense of Lyapunov we must have,

$$s^T W_r(y, \dot{y}, \ddot{y}_r, \ddot{y}_r)\tilde{\theta} = -\tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \quad (2.43)$$

³The given function is not a true Lyapunov function for the system since it is not a positive definite function of the tracking error, but of the filtered error, s ; however, Lyapunov stability can nonetheless still be shown (see Slotine and Li [52], Spong et al [55]).

to give

$$\dot{V}(z(t)) = -s^T K_D s \leq 0 \quad (2.44)$$

To satisfy (2.43), the adaptation law becomes

$$\dot{\tilde{\theta}} = -\Gamma^{-1} W_r^T(y, \dot{y}, \dot{y}_r, \ddot{y}_r) s \quad (2.45)$$

This adaptation law guarantees that the tracking errors go to zero and will be driven to the sliding surface given by (2.36).

2.3.2 Theoretical Stability Properties

The derivation of the adaptive algorithm summarized above was originally outlined in Slotine and Li [51]. A more detailed derivation and further discussion of the error dynamics can be found in Slotine and Li [52] and Schwartz et al [48]. Note that because the selected positive definite function $V(z(t))$ is not truly a Lyapunov function for the system (all states are not represented), Lyapunov stability cannot strictly be proven by the above standard method, although the convergence of the tracking error does follow. However, the Lyapunov stability can indeed be shown by a more complex proof procedure (see Slotine and Li [52]), or by selection of a true Lyapunov function, as in Spong et al [55].

The Slotine and Li method of direct adaptive control for robot manipulators has advantages over a number of other adaptive methods. To implement this method one does not require an acceleration measurement. Furthermore, it can be shown that the trajectory tracking errors are asymptotically stable. The error dynamics in terms of the filtered tracking error vector s can be derived as the following:

$$M(y)\dot{s} + (C(y, \dot{y}) + K_D)s = W_r(y, \dot{y}, \dot{y}_r, \ddot{y}_r)\tilde{\theta}. \quad (2.46)$$

Thus the error dynamics are non-linear, with the parameter estimation error as the driving function. These error dynamics can be shown to be stable, and *asymptotically* stable if the desired trajectory y_d is persistently exciting (see Slotine and Li [52], Spong and Vidyasagar [56]).

To obtain persistent excitation and hence the asymptotic stability of the system, $W(y, \dot{y}, \ddot{y})$ must over some period of time span the parameter space. This condition can be written as,

$$\alpha_1 I \leq \int_{t_1}^{t_1+\rho} W^T W dt \leq \alpha_2 I \quad (2.17)$$

for some positive scalar constants α_1, α_2, ρ , and for all $t_1 \geq 0$ [52]. Hence, to ensure persistent excitation, the desired trajectory must be chosen such that (2.17) is satisfied with W replaced by W_d (W evaluated along the desired trajectory). However, the persistent excitation condition may still be satisfied even if the desired trajectory is not persistently exciting, due to the non-zero tracking error present (see Schwartz and Warshaw [47], Schwartz et al [48]).

Thus we expect that for the continuous-time case (and in the absence of disturbances and unmodeled dynamics), Slotine and Li's algorithm will give globally stable performance. In the presence of persistent excitation, the system is asymptotically stable, and the tracking error and parameter errors are expected to converge to zero. This result should apply for any choice of adaptation gain matrix Γ^{-1} (as long as Γ is a positive definite matrix as required for $V(z(t))$, given by (2.37)) based on the derivation using Lyapunov's direct method described above. However, the choice of adaptation gain affects the speed of parameter convergence and hence the tracking performance during adaptation.

2.4 Discussion of the Implementation of Robot Adaptive Control

As discussed previously in the introduction, there are some key differences between the implementations and performance of direct and indirect robot adaptive algorithms. Typically, the direct methods are more computationally simple (and hence easier to implement in real-time) than the indirect methods, and as the name implies, adapt the parameters to directly reduce the tracking error. A number of direct algorithms can claim global asymptotic convergence of the tracking error. However, in some applications the indirect methods can exhibit superior convergence proper-

ties; using various recursive least squares (RLS) algorithms, exponential convergence can be demonstrated when persistent excitation is present. An additional advantage of the indirect approach is that the adaptation can be run while the system is controlled by a standard fixed (i.e. non-adaptive) controller; this permits time for the parameters to converge before the adaptive controller is put on-line, and therefore can eliminate undesirable transient performance during the initial parameter adaptation (Warshaw [61], Schwartz et al [48]). This is not possible for the direct algorithms, since they use the tracking error for adaptation instead of the prediction error and hence must be in the loop for correct adaptation to occur.

Methods which combine the elements of direct and indirect adaptive techniques have also been proposed recently (see Duarte and Narendra [15], Slotine and Li [54]). These “composite” methods are an attempt to retain the global asymptotic convergence of the tracking error guaranteed by the direct algorithms, while improving the convergence rate. Most recently, a composite algorithm which combines the sliding mode technique of Variable Structure Control (VSC) with the direct and indirect adaptive methods has also been proposed (Narendra and Boskovic [35]). The VSC technique is used to increase the performance (improve the transient response) and to improve the robustness, in combination with the asymptotic stability provided by the direct and indirect adaptive components.

The bulk of published work on adaptive manipulator control presents analytical or simulation results, and experimental results have been somewhat more scarce. Nevertheless, some experimental results have been reported which to some extent verify the theoretical and simulation work (see for example Slotine and Li [53], Morando et al [33], Tomizuka [59], and Craig et al [12]). However, no one has done a comprehensive analysis of the effects of discretization for the algorithms. Therein lies the “problem” which is addressed by the research discussed in this document.

Chapter 3

Sampled-Data Adaptive Control for Robots

3.1 Background and Literature Review

3.1.1 The Analysis of Discrete-Time and Sampled-Data Adaptive Controllers

Modern controlled mechanical systems are generally sampled-data systems, with a continuous-time plant and a digital computer as a controller. The computer controller samples the plant outputs and provides a discrete-time control input to the plant which is held over each sampling period (generally with a Zero-Order-Hold, or Z.O.H.).

Typically, the adaptive algorithms proposed in the literature have been analyzed as continuous-time systems. The implementation then uses a simple discretization of the continuous-time algorithm (such as Euler integration). Clearly, the continuous-time analysis (especially the stability analysis) is inadequate to accurately and completely characterize the implementation without further development. This inadequacy can be a cause of the discrepancies observed between theoretical predictions of performance and stability, and experimental results (see Slotine and Li [53], Morando et al [33], Craig et al [12], and Niemeyer and Slotine [36]). It is straightforward to demonstrate this point by simulating the sampled-data implementation of Slotine

and Li's direct adaptive algorithm [51]. Even though the stability analysis of this algorithm (in continuous-time) proves stability of the algorithm for any adaptation gain (or feedback gain), the sampled-data simulation clearly shows the onset of instability when the adaptation gain is raised above a threshold. As stated by Whitcomb et al [64], the numerical integration required to discretize the continuous-time adaptation laws can destabilize the system for sufficiently high adaptation gains. This point is discussed in more depth in Section 3.4.

The analyses of discrete-time adaptive controllers for the indirect case and for linear plants has been well covered. For example, Goodwin and Sin [18] have analyzed a range of discrete adaptive controllers, but concentrated on linear plants. These analyses of discrete adaptive systems can characterize the behaviour of the sampled-data control system with a linear plant as discussed above, since a sampled continuous-time linear plant gives a linear model in discrete-time (see the discussion in Middleton [32]). However, the discrete-time model of the sampled-data system is valid as long as one is concerned with the system behaviour at the sampling instants only – otherwise the system being examined cannot be considered Linear Time-Invariant (LTI), and the analysis and design techniques established for LTI discrete systems cannot be used (see also Chen and Francis [7], Franklin et al [17]). This is because a sampled-data system is time-varying and cannot be represented by a transfer function. The continuous-time response of a sampled-data system to a signal depends on the time at which the signal is applied, relative to the sampling instant. The action of a sample-and-hold produces the time-varying nature of the system when viewed in continuous-time. For example, when a sinusoid is the input signal, the sample-and-hold outputs a staircase function, and not another sinusoid as dictated by LTI system behaviour.

Koivo and Guo [24] proposed discrete-time indirect adaptive control of a robotic manipulator using the Self-Tuning Regulator (STR) technique, and independent single joint (linear) difference equation models. This approach necessarily ignores the inherent coupled and non-linear nature of manipulator dynamics. Thus, the major obstacle for the above approaches as they apply to robotic manipulators is that linear techniques are not strictly applicable.

There have been two notable attempts to address the implementation aspects for adaptive control of robotic manipulators discussed above: Morando et al [33] proposed a direct algorithm and investigated the implementation including proof of stability of the “hybrid” (both continuous and discrete-time parts) algorithm. The difficulties with this algorithm are as follows:

1. asymptotic stability was not proven for the hybrid algorithm;
2. the boundedness of the error signals relies upon the selection of an appropriate dead zone for the parameter adaptation;
3. a number of gain terms and a dead zone for the algorithm must be picked by the user, based on the desired trajectory, without a straight-forward and explicit selection method.

Middleton [32] has investigated a hybrid indirect adaptive algorithm, and proven global boundedness for the algorithm. However, although the parameter estimation occurs in discrete-time, the non-linear control law employed is still in continuous-time. To date, no experimental validation of the theoretical results has been published for this algorithm.

As discussed previously in Section 2.1, Lyapunov’s second method is a powerful and widely- used technique for proving the global stability of non-linear and adaptive systems. This method works well for many continuous-time systems, and in fact is the basis for the formulation of many robot direct adaptive control algorithms, including Craig’s [12] and Slotine and Li’s [51].

Hsu and Sastry [22] have used Lyapunov stability theory to investigate the stability properties of both linear and non-linear closed-loop systems with discretized feedback. They proved that a sampling frequency exists such that a globally asymptotically stable continuous-time non-linear system with dynamical feedback (a general system structure which includes a wide class of adaptive systems) gives a sampled-data system with local stability. The minimum sampling frequency to ensure stability was derived only for the linear system case.

3.1.2 Robust Control and Compensation of Sampled-Data Adaptive Control

It is possible to modify the discretized adaptive control law used in a sampled-data manipulator system with the addition of a non-linear robust compensating term. This robust compensation is used in an attempt to overcome the inexact compensation of the non-linear dynamics due to both the parameter estimation error and errors due to the discretization of the control¹. Sadegh and Horowitz have made use of such a non-linear feedback term to offset the non-exact feedforward compensation in their "Desired Compensation Adaptive Law" ([43], also Morando et al [33]).

The field of robust control consists of a number of well-known and well-researched techniques for robustifying systems in the face of disturbances and uncertainties (eg. for the robot control case, unmodeled and high frequency dynamics, or torque and measurement disturbances). However, robust control methods require a trade-off between performance and robustness.

Robust control techniques are a potential means of compensating the discretized robot controller to account for the discretization effects on the sampled-data system along with other disturbances. If the adaptive control is successful in improving the controller's model of the robot dynamics, i.e. reducing the uncertainty, then robust control can be used to accommodate the remaining modelling errors, disturbances and discretization effects. In this way, it may be possible to achieve better overall performance and robustness than either robust or adaptive control alone.

A number of researchers have proposed robust adaptive control laws, and a paper published by Abdallah et al [1] contains a survey of some approaches that combine adaptive and robust control methods for robots. However, the research has been primarily aimed at either achieving robustness by preventing the drift of the parameter estimates (see for example Reed and Ioannou [40]), or adding adaptation

¹An alternative means of tackling this problem is the use of repetitive learning; Sadegh and Guglielmo [44] discuss the use of a learning controller for cancelling the unmodeled effects, including those introduced by digital implementation and sampling.

of the uncertainty parameters which are employed by a standard robust controller (see for example Singh [49]). Thus, the bulk of the previous work in the area is not strictly applicable to accommodating the discretization effects.

The use of inner/outer loop control with feedback linearization for robustifying the adaptive controller has been investigated (see for example Spong and Vidyasagar [56]). The method of Spong and Vidyasagar is based on the robust saturation control techniques of Corless and Leitmann [9] for the outer loop, employing a non-linear feedback term.

With an adaptive inverse dynamics controller, such as the one proposed by Craig et al [12], the inner loop achieves feedback linearization of the plant, and the robust outer loop design can be used to accommodate both the parameter estimation error and the discretization effects. Other types of model-based robot adaptive controllers, for example Slotine and Li's, are not strictly inverse dynamics controllers and hence do not linearize the plant (the dynamics of the error are not linear). These adaptive controllers have advantages over inverse dynamics controllers such as Craig's, especially that acceleration measurements are not required. However, since feedback linearization is not achieved by adaptive controllers such as Slotine and Li's, Spong and Vidyasagar's robust control approach is not applicable in this case.

An approach to robustify Slotine and Li's controller can be postulated by reviewing the work of Dawson and Lewis [13]. Dawson and Lewis have formulated two robot control laws which are stable in continuous time:

1. an adaptive controller which is almost entirely equivalent to Slotine and Li's;
2. a robust controller using the robust saturation control method of Corless and Leitmann [9].

The possibility of combining the elements of these adaptive and robust controllers was investigated. The robust compensation must be modified to include the discretization effects in the uncertainty terms. This approach was rejected because it is not feasible; the discrete-time parameter estimate update violates a critical as-

sumption upon which Corless and Leitmann's robust saturation control method is based. This issue is further discussed in Section 4.3.

3.2 Problem Statement

Adaptive control laws for robotic manipulator control are typically formulated in continuous-time, and then they are discretized for the implementation. The resulting manipulator system is a sampled-data one. Because the formulation of the adaptive controllers is in continuous-time, the theoretical predictions of performance and stability based on this formulation do not apply to the sampled-data implementation. Specifically, the continuous-time proof of stability is not valid for the implementation.

The aim of the research described in this thesis is to develop the means to accurately predict the stability and tracking performance of the implementation of direct adaptive control laws for robotic manipulators. Stability is not guaranteed for existing direct adaptive control laws: a discrete-time adaptive control law that does have guaranteed stability is the goal. A secondary goal is to develop the tools to predict the guaranteed bound on the tracking error.

3.3 Analysis of the State of the Art

No proof of stability for the sampled-data implementation of robot direct adaptive controllers has appeared in the published literature. This has been noted for the direct adaptive controllers proposed by Craig et al [12], Slotine and Li [51] and Sadegh and Horowitz [43].

While the analyses of discrete-time adaptive controllers for indirect adaptive control and for linear plants has been well covered (see for example, Goodwin and Sin [18]), this does not cover the case of discretized direct adaptive control of (non-linear) robotic manipulators.

Morando et al [33] and Middleton [32] have proven stability for the implementation of a direct and indirect adaptive robot controller respectively, but in both

cases the controller implementations were modelled as hybrids (both discrete- and continuous-time parts). Therefore, the stability proofs do not apply to an actual sampled-data implementation with a purely discrete-time controller.

Hsu and Sastry [22] have proven that a sampling frequency exists such that a globally asymptotically stable continuous-time non-linear system with discretized dynamical feedback gives a sampled-data system with local stability. This class of systems includes the implementation of adaptive control for robotic manipulators. However, only the *existence* of a sampling frequency which guarantees stability was proven; the means to predict the stability of the system for a given sampling frequency (and other given controller parameters), or to predict what the critical sampling frequency is, have not been demonstrated.

The use of robust compensation techniques to compensate the discretized robot controller to account for the effects of adaptive controller discretization on the sampled-data system has not been previously explicitly covered in the published literature. Sadegh and Horowitz have made use of a non-linear feedback term for their adaptive controller ([43], also Morando et al [33]), but formulated it to offset the non-exact feedforward compensation (resulting from the parameter estimation errors, and the discrepancy between the regression matrix employing the desired trajectory versus actual trajectory measurements). Spong and Vidyasagar [56] have applied robust control techniques to the adaptive robot control problem, but have not considered the discretization as a source of uncertainty to be compensated.

It can be concluded from the above analysis of the published literature that the problem being tackled in this thesis is previously unsolved.

The analysis of stability is an essential element when characterizing the behaviour of robotic manipulator controllers; practical manipulator systems must be proven to be stable. The stability analysis of robotic manipulator systems employing discrete-time adaptive controllers is a formidable analytical challenge, because the robot is a MIMO coupled non-linear sampled-data system. The existing analytical tools are not able to predict the stability of such a system, nor can they predict the tracking performance of the system.

Rohrs et al [41] proposed some guidelines for the design of discrete-time adaptive

controllers to diminish stability problems resulting from the discretization. One of their guidelines was to keep the adaptation gain small so that the adaptation is slow. However, as Whitcomb et al [64] point out, a more complete theoretical understanding of the discretized adaptive controllers is needed to permit automatic selection of appropriate adaptation gains rather than the empirical selection implied by the "small adaptation gain" guideline and used in previously reported implementations.

It may be argued that the value of stability and performance predictions for sampled-data robot adaptive controllers is purely academic, since the ever-increasing speed and decreasing cost of microprocessors will eventually permit sampling rates which adequately approximate continuous-time control. However, the predictions are also of practical value, since:

- the bandwidth of manipulators (including micro-robots now under development) and the complexity of proposed adaptive algorithms are also ever-increasing;
- the adaptive algorithms are still implemented in discrete-time, and for any choice of sampling rate may become unstable for some choices of gains (i.e. there are effectively no physical limits on the bandwidth of the adaptation).

3.4 Demonstration of the Problem by Simulation

The problem addressed in this thesis leads to the conclusion that the theoretical predictions of performance and stability based on the continuous-time formulation of Slotine and Li's direct adaptive controller do not apply to the sampled-data implementation of a manipulator. This is demonstrated by simulation of a two DOF serial link manipulator in this section.

Figure 3.1: Desired Trajectories (— joint 1, - - - joint 2)

3.4.1 Continuous-Time System

Computer simulations of the continuous-time system consisting of a two DOF serial link manipulator and Slotine and Li's adaptive algorithm were performed. The simulation software used to perform the simulations is briefly described in Section 5.1.3. The manipulator model is also described in Section 5.1.3. A 4th order Runge-Kutta algorithm was used to simulate both the continuous-time plant model and adaptive control law. The time step selected for the Runge-Kutta solver was 0.01 seconds.

A baseline adaptive case was selected using a persistently exciting desired trajectory, as discussed in Warshaw [61]. The desired trajectories (in joint space) for the two degrees of freedom are shown in Figure 3.1. The adaptation and feedback gains, and the filtered error parameter matrix were set as follows: $\Gamma^{-1} = 0.2I^{3 \times 3}$, $K_D = 8I^{2 \times 2}$, $\Lambda = 10I^{2 \times 2}$.

The norm of the filtered tracking errors, $\|s(t)\|$, and parameter estimation errors for the continuous-time adaptive system are shown in Figures 3.2 and 3.3 respectively. After the initial transient adaptation, the tracking errors converge to near zero. The parameter estimation errors also converge smoothly and quickly to zero.

If the adaptation gain is greatly increased, the system remains stable. Figures 3.4 and 3.5 show $\|s(t)\|$ and parameter estimation errors when the adaptive gains used in the baseline case are raised (to $\Gamma^{-1} = 0.38I^{3 \times 3}$). The behaviour is very similar to

Figure 3.2: Norm of Filtered Tracking Errors (— continuous-time, - - - sampled-data)

Figure 3.3: Parameter Estimation Errors, Continuous-Time System

Figure 3.4: Norm of Filtered Tracking Errors, Continuous-Time System

Figure 3.5: Parameter Estimation Errors, Continuous-Time System

the baseline case with lower adaptive gain.

The above simulation results for the continuous-time case are seen to be consistent with the theoretical results presented in Section 2.3.2.

3.4.2 Sampled-Data System

Computer simulations of the sampled-data system were also performed. As for the continuous-time simulations, a 4th order Runge-Kutta algorithm was used to simulate the manipulator continuous-time dynamics. However, in this case the adaptive controller was simulated in discrete time, resulting in an overall hybrid simulation.

Figure 3.6: Parameter Estimation Errors, Sampled-Data System

The sampled-data simulations were performed for the baseline case described in the previous sub-section, with a sampling rate of 100 Hz for the discrete-time controller.

Parameter estimation errors for the sampled-data system are shown in Figure 3.6. A comparison between $\|s(t)\|$ for the continuous-time and sampled-data systems is shown in Figure 3.2. Compared to the corresponding results for the continuous-time case, both the filtered tracking error and parameter estimation error show different behaviour: the errors now do not converge exactly to zero, but oscillate within a bound about zero.

For the case with increased adaptive gain discussed previously ($\Gamma^{-1} = 0.38 I^{3 \times 3}$), the sampled-data system appears to be unstable. Figure 3.7 shows a comparison of $\|s(t)\|$ for the continuous-time and sampled-data system. After less than one second, $\|s(t)\|$ for the sampled-data case grows quickly, and eventually the simulation ceases due to numerical overflow. Figure 3.8 shows the corresponding parameter estimation errors for the sampled-data case.

The comparison of the continuous-time and sampled-data simulation results shows that the two cases exhibit significantly different behaviour. Thus, we have a clear indication that the discretization of Slotine and Li's adaptive control algorithm changes both the tracking performance and stability from the purely continuous case.

Although it is to be expected that instability will eventually occur for the system

Figure 3.7: Norm of Filtered Tracking Errors (— continuous-time, - - - sampled-data)

Figure 3.8: Parameter Estimation Errors, Sampled-Data System

as the gain is raised (even linear systems which are stable in continuous-time will be unstable when discretized at too slow a sampling rate) this onset of instability is unfortunately not predictable using any of the analytical tools previously found in the adaptive control literature.

Chapter 4

Stability Analysis

Robotic manipulators are non-linear, coupled MIMO (multiple-input/multiple output) systems. As discussed previously, it is difficult to determine the stability and performance of the sampled-data implementation of such a system, and discrete-time adaptive control laws in the loop add a further level of difficulty to the analytical problem. As a tool for the implementation of practical adaptive controllers for robotic manipulators, it is desirable to analytically predict the stability and tracking performance of the sampled-data system as a function of the plant dynamics, sampling frequency, controller gains (feedback and adaptation), and desired trajectory. Unfortunately, explicit evaluation of the stability and tracking performance is not possible with existing standard techniques.

Consider the possibility of analyzing a discrete-time system using Lyapunov's method for continuous-time systems, by expressing the discrete system in continuous-time. This analysis method was inspired by the "discretization error" approach presented by Morando et al [33]. In this fashion, it is possible to analyze the stability of continuous, discrete and hybrid systems (such as a sampled-data system) in a unified manner. In this chapter, this Lyapunov stability approach coupled with methods of approximation are used to derive useful stability and error bound results. By formulating the sampled-data system in continuous-time and by making use of known *bounds* on the system dynamics, useful predictions of stability and tracking error bounds are possible.

4.1 Lyapunov's Second Method for Non-Linear Sampled-Data Systems

If the continuous-time component of a sampled-data system is linear, it can be easily discretized into its equivalent discrete-time form; then the stability of the overall equivalent discrete-time form of the sampled-data system can be assessed using Lyapunov's second method (see Kalman and Bertram [23]) or several other stability methods for discrete-time systems. However, if the continuous-time component of a sampled-data system is non-linear, then it is not possible in general to find its equivalent discrete-time form; the discretization of the non-linear system can be done by integration over a time step, but a closed form solution for this integration is usually unobtainable. Hence, the assumption that the closed form discretization is available is not valid, and Lyapunov's second method for discrete-time systems (e.g. as presented by Kalman and Bertram [23]) is not directly usable¹.

Because manipulator implementations are sampled-data systems with non-linear continuous-time plants, the stability cannot be analyzed using the discrete-time form of Lyapunov's second method. The use of Lyapunov's second method for continuous-time systems as suggested in the introduction to this chapter cannot be used directly either: the discrete-time parts of the sampled-data system can introduce a delta-function (impulse) in $\dot{V}(x, t)$ due to the possible instantaneous change of the value of $V(x, t)$ at the sampling instant (due to the change in the control signal produced by the discrete-time controller). The presence of the delta-function in $\dot{V}(x, t)$ can make it impossible to prove whether $\dot{V}(x, t)$ is negative (semi)definite, since $\delta(t - t_k) \rightarrow \infty$ at $t = t_k$.

To avoid the difficulty mentioned above, it is possible to consider a modified form of Lyapunov's second method tailored to sampled-data systems (or other hybrid continuous/discrete-time systems). A Lyapunov function candidate is selected and $\dot{V}(x, t)$ is calculated as for the continuous-time case, but then $\dot{V}(x, t)$ is integrated

¹It will later be shown that the use of bounds of the terms of an integration over a time step for the sampled-data system can permit the application of Lyapunov's second method to the problem, avoiding the need for an explicit solution for the discretized form.

over the sampling interval to yield ΔV :

$$\Delta V(x(t_k), t_k) = \int_{t_k^-}^{t_{k+1}^-} \dot{V}(x, t) dt = V(x(t_{k+1}^-), t_{k+1}^-) - V(x(t_k^-), t_k^-), \quad (1.1)$$

where t_k indicates the time at the sampling instant denoted by the discrete time index k , and t_k^- indicates an infinitesimally small time before the sampling instant². By doing this, the integral of the delta-function becomes a (finite) step function and so the assessment of the negative definiteness of $\Delta V(x(t_k), t_k)$ is no longer problematic. The global stability of the system can now be established according to Lyapunov's second method for discrete-time systems, as discussed by Kalman and Bertram [23]: as long as $\Delta V(x(t_k), t_k)$ (the change in $V(x(t_k), t_k)$ over the sampling time) is negative semi-definite, $V(x(t_k), t_k)$ will always be decreasing (or at least non-increasing) at the sampling instants t_k . By integrating $\dot{V}(x, t)$, the negative semidefiniteness of ΔV indicates that "on the average" (during the sampling interval), $\dot{V}(x, t)$ is negative semidefinite, and hence the system is stable.

While this formulation of the method for sampled-data systems appears to be simply a restatement of Lyapunov's second method for discrete-time systems, it is not, because the sampled-data system has both discrete and continuous parts, and the behaviour of ΔV does not strictly speaking characterize the behaviour of V between the sampling instants ($t_k < t < t_{k+1}$). In other words, if global asymptotic stability is proven according to the above theorem, this applies only to the states of the system at the sampling instants, and not the inter-sample behaviour. However, it can be argued that the change in the system states $x(t)$ during the finite sampling interval must be bounded, for any physical system with finite energy input or for any adaptation law with finite gain. Thus stability of the system based on the states at the sampling instants implies stability of the system during the inter-sampling time as well.

The approach that is described above is to apply Lyapunov's second method for continuous-time systems to the sampled-data system, and then integrate $\dot{V}(x, t)$ to check for stability. The traditional approach would be to discretize the sampled-data

²to include the possible change in discrete signals at the time t_k , but not for the following time t_{k+1} .

system first, and then use Lyapunov's second method for discrete-time systems to check for stability; as discussed previously, this approach is not possible when the closed form discretization of the non-linear sampled-data system is not available.

4.2 Analysis of the Sampled-Data Robot Adaptive System using Lyapunov's Second Method

4.2.1 Formulating the Stability Condition

The Lyapunov stability analysis of the sampled-data system consisting of the continuous-time multi-DOF robot manipulator and the discretized version of Slotine and Li's adaptive controller was described in Warshaw et al [63]. The summary of this analysis is repeated in this section. The same Lyapunov function as originally used by Slotine and Li [51] is used in the analysis presented here, because the control and adaptation laws were derived by Slotine and Li directly from this Lyapunov function (see Section 2.3). The intention is to show that using the same controller structure and Lyapunov function derived by Slotine and Li (in continuous-time) results in a sampled-data system which cannot be proven to be stable.

The plant is modeled by the continuous-time manipulator dynamics, and repeating (2.31) we get,

$$\tau = M(y)\ddot{y} + C(y, \dot{y})\dot{y} + G(y) \quad (4.2)$$

In the formulations which follow, a discrete-time signal (or sequence) is denoted with the index k in square brackets. A discrete-time signal derived by sampling a continuous-time signal is defined as follows:

$$y[k] \equiv y(t_k), \quad k \geq 0 \quad (4.3)$$

where $t_k = k\Delta t$ (time at sample k) and Δt is the sampling time.

The adaptive controller is implemented in discrete-time and consists of the con-

trol law and adaptation law, taken from (2.32) and (2.33)

$$\tau[k] = \hat{M}(y[k])\ddot{y}_r[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}(y[k]) - K_D s[k] \quad (4.4)$$

$$\hat{\theta}[k] = \hat{\theta}[k-1] - \Delta t \Gamma^{-1} W_r^T(y[k], \dot{y}[k], \dot{y}_r[k], \ddot{y}_r[k])s[k]. \quad (4.5)$$

The control and adaptation laws use the sampled versions of the continuous-time plant states (and desired path), and the adaptation law uses a simple Euler integration (recursive) form.

To complete the formulation, the hold operation must be defined. Following the development of Morando et al [33] and using the torque signal as an example, the continuous-time held signal is,

$$\tau_h(t) \equiv H\{\tau[k]\} \quad (4.6)$$

where $H\{\cdot\}$ is the ZOH operator, such that during a single sampling interval:

$$\tau_h(t) = \begin{cases} \tau[k] & \text{if } t_k \leq t < t_{k+1} \\ \tau[k+1] & \text{if } t = t_{k+1} \end{cases} \quad (4.7)$$

Using the above definition (4.6) and (4.4), the actual control torque input to the plant becomes

$$\begin{aligned} \tau_h(t) &= H\{\hat{M}(y[k])\ddot{y}_r[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] + \\ &\quad \hat{G}(y[k]) - K_D s[k]\} \end{aligned} \quad (4.8)$$

$$\begin{aligned} &= H\{\hat{M}(y[k])\ddot{y}_r[k]\} + H\{\hat{C}(y[k], \dot{y}[k])\dot{y}_r[k]\} + \\ &\quad H\{\hat{G}(y[k])\} - K_D H\{s[k]\} \end{aligned} \quad (4.9)$$

The equations (4.2), (4.5) and (4.8) represent the complete sampled-data model of the system, using $\tau(t) = \tau_h(t)$.

The Lyapunov function candidate for the general robot manipulator case is defined as (2.37):

$$V(z(t)) = \frac{1}{2} s^T M(y) s + \frac{1}{2} \tilde{\theta}^T \Gamma \tilde{\theta} \quad (4.10)$$

where $z(t)$ is the vector of system states (s and $\tilde{\theta}$). Taking the derivative of $V(z(t))$, (2.39) was derived in Section 2.3.1, and is repeated here for convenience:

$$\dot{V}(z(t)) = s^T (\tau - G(y) - M(y)\ddot{y}_r - C(y, \dot{y})\dot{y}_r) + \tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \quad (4.11)$$

By substituting the discrete-time control law with ZOH (4.8) into (4.11), the expression for $\dot{V}(z(t))$ within the single sampling interval ($t_k \leq t < t_{k+1}$) becomes:

$$\begin{aligned} \dot{V}(z(t)) = & s^T(t) \{ -K_D s[k] + (\hat{M}(y[k])\tilde{y}_r[k] - \\ & M(y(t))\tilde{y}_r(t)) + (\hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] - \\ & C(y(t), \dot{y}(t))\dot{y}_r(t)) + (\hat{G}(y[k]) - G(y(t))) \} + \tilde{\theta}^T \Gamma \dot{\tilde{\theta}} \end{aligned} \quad (4.12)$$

Within the same time interval, the parameter estimation error is

$$\tilde{\theta}(t) = \hat{\theta}[k] - \theta, \quad (4.13)$$

where the vector of true parameters, θ , is assumed to be constant. Thus, the derivative of the parameter estimation error is

$$\dot{\tilde{\theta}}(t) = \dot{\hat{\theta}}(t) = -\Delta t \Gamma^{-1} W_r^T(y[k], \dot{y}[k], \dot{y}_r[k], \tilde{y}_r[k]) s[k] \delta(t - t_k) \quad (4.14)$$

where $\delta(t - t_k)$ is the unit impulse occurring at $t = t_k$. Equation (4.14) comes from differentiating the signal $H\{\hat{\theta}[k]\}$: the derivative of a series of steps is a series of impulses.

By substituting (4.14) into (4.12), we get (shortening the arguments of the terms):

$$\begin{aligned} \dot{V}(z(t)) = & -s^T(t) K_D s[k] + s^T((\hat{M}[k]\tilde{y}_r[k] - M(t)\tilde{y}_r(t)) \\ & + (\hat{C}[k]\dot{y}_r[k] - C(t)\dot{y}_r(t)) + (\hat{G}[k] - G(t))) - \\ & \Delta t (\hat{\theta}[k] - \theta)^T W_r^T[k] s[k] \delta(t - t_k) \end{aligned} \quad (4.15)$$

As discussed previously in Section 4.1, because of the presence of the impulse function in (4.15), it is not meaningful to assess if the function $\dot{V}(z(t))$ is negative definite to satisfy the stability condition (since $\delta(t - t_k) \rightarrow \infty$ at $t = t_k$). Therefore, the continuous-time form of Lyapunov's second method cannot be applied to this case. However, Lyapunov's second method for non-linear sampled-data systems, as proposed in Section 4.1 above, can be applied to this system. To avoid the impulse, $\dot{V}(z(t))$ as expressed by (4.15) is integrated over the sampling interval (choosing the time interval to include only the impulse at $t = t_k$ by taking the integral from t_k^- to t_{k+1}^-) to yield:

$$\begin{aligned} \Delta V_k = & -\left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} K_D s[k] + \\ & \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} \hat{M}[k] \tilde{y}_r[k] - \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) M(t) \tilde{y}_r(t) dt \right\} + \\ & \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} \hat{C}[k] \dot{y}_r[k] - \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) C(t) \dot{y}_r(t) dt \right\} + \\ & \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} \hat{G}[k] - \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) G(t) dt \right\} - \\ & \Delta t (\hat{\theta}[k] - \theta)^T W_r^T[k] s[k]. \end{aligned} \quad (4.16)$$

This expression for ΔV_k gives the means to explicitly evaluate the stability of the general multi-DOF manipulator case. However, the difficulty is in evaluating the integrals appearing this expression. Accordingly, the discretization of Slotine and Li's adaptive algorithm results in a sampled-data system for which stability cannot be guaranteed based on the condition that $\Delta V_k \leq 0$, where ΔV_k is given by (4.16).

If it can be assumed that the change in the signals which compose ΔV_k , i.e. $s(t)$, $y(t)$, $\dot{y}(t)$, $\dot{y}_r(t)$, $\tilde{y}_r(t)$, change very little over a single sampling interval, then (4.16) reduces to

$$\Delta V_k \approx -\Delta t s^T[k] K_D s[k] \quad (4.17)$$

which is negative definite for K_D positive definite and $s(t) \neq 0$. This result shows that the system is stable, since it is equivalent to the continuous-time result (see (2.44)). This is not surprising, since if the sampling rate is sufficiently fast so that the

signals of interest change imperceptibly during the sampling interval (i.e. taking the limit as the sampling time goes to zero), then the discrete-time case is equivalent to the continuous-time case. Note that this limiting result is independent of the choice of adaptation gain, Γ^{-1} .

4.2.2 A Steady-State Bound on the Error

The stability condition given by (4.16) can be used to predict the observed behaviour of the sampled-data implementation of Slotine and Li's adaptive controller as presented in Section 3.4. There are two main observations (from the simulation results depicted in Figures 3.2 to 3.8) to be explained:

1. the obvious difference in performance (convergence of tracking error and parameter estimation errors) between the continuous and discrete-time cases;
2. the apparent instability of the sampled-data case for higher adaptation gains.

As will be demonstrated below, it is possible to approximate the evaluation of the stability condition (4.16) and thereby predict the steady-state bound on the norm of the filtered tracking error for the system, $\|s[k]\|$. This prediction is intended to explain the oscillations in the tracking and parameter estimation errors observed from the sampled-data simulations.

A restrictive set of assumptions will be shown below to lead to a stability condition from (4.16) which can be used to predict the bound on the norm of the filtered tracking error. The necessary assumptions are:

- the integrals can be accurately approximated using the trapezoid rule;
- the change $s[k]$, $M[k]$, $C[k]$ and $G[k]$ between sampling instants can be neglected relative to the change in $\tilde{y}_r[k]$;
- the tracking error is relatively small;
- K_D is a diagonal matrix with all non-zero elements equal.

These assumptions can be thought of as representing the steady-state case, after parameter estimate convergence.

Assume that $s[k] \approx s[k+1]$. Using this assumption and the trapezoid rule approximation of the integrals, we now get:

$$\begin{aligned} \Delta V_k \approx & \Delta t s^T[k](-K_D s[k] + \hat{M}[k]\ddot{y}_r[k] - \\ & \frac{1}{2}M[k]\ddot{y}_r[k] - \frac{1}{2}M[k+1]\ddot{y}_r[k+1] + \hat{C}[k]\dot{y}_r[k] - \\ & \frac{1}{2}C[k]\dot{y}_r[k] - \frac{1}{2}C[k+1]\dot{y}_r[k+1] + \hat{G}[k] - \\ & \frac{1}{2}G[k] - \frac{1}{2}G[k+1] - W_r[k](\hat{\theta}[k] - \theta)). \end{aligned} \quad (4.18)$$

By making use of the regression form of the dynamics (2.10),

$$W_r[k]\theta = M(y[k])\ddot{y}_r[k] + C(y[k], \dot{y}[k])\dot{y}_r[k] + G(y[k]) \quad (4.19)$$

and cancellation of terms gives

$$\Delta V_k \approx \Delta t s^T[k](-K_D s[k] + \frac{1}{2}W_r[k]\theta - \frac{1}{2}W_r[k+1]\theta) \quad (4.20)$$

Thus, to guarantee stability,

$$\Delta V_k \approx -s^T[k]K_D s[k] + \frac{s^T[k]}{2}(W_r[k] - W_r[k+1])\theta \leq 0 \quad (4.21)$$

The condition represented by inequality (4.21) is difficult to manipulate, due to the presence of $W(y[k], \dot{y}[k], \ddot{y}_r[k], \ddot{y}_r[k+1])$. To simplify the problem, assumptions can be made about the relative change in the various terms which compose the dynamics, over one sampling interval. Recall that we are trying to explain the results observed from simulation of a two DOF serial link manipulator as shown in Figures 3.2 to 3.8. The goal here is to derive a prediction of the bound on the filtered tracking errors. The first approach is to assume that

$$M[k] \approx M[k+1] \quad (4.22)$$

$$C[k]\dot{y}_r[k] \approx C[k+1]\dot{y}_r[k+1] \quad (4.23)$$

$$G[k] \approx G[k+1] \quad (4.24)$$

(this was true as a first approximation for the simulations described below), but $\tilde{y}_r$ changes rapidly such that

$$\tilde{y}_r[k] \neq \tilde{y}_r[k+1]. \quad (4.25)$$

Then (4.19) can be used to expand and then cancel terms of (4.21), resulting in the following stability condition:

$$\Delta V_k \approx -s^T[k]K_D s[k] + \frac{1}{2}M[k]\Delta'\tilde{y}_r[k] \leq 0 \quad (4.26)$$

where

$$\Delta'\tilde{y}_r[k] \equiv \tilde{y}_r[k] - \tilde{y}_r[k+1] = -\Delta\tilde{y}_r[k]. \quad (4.27)$$

Since the two terms in inequality (4.26) are scalars and K_D is assumed to be positive definite, the stability condition can be written as

$$|s^T[k]K_D s[k]| \geq \frac{1}{2}|s^T[k]M[k]\Delta'\tilde{y}_r[k]| \quad (4.28)$$

Using the Schwarz/Cauchy inequality, (4.28) becomes

$$|s^T[k]K_D s[k]| \geq \frac{1}{2}\|s[k]\| \|M[k]\Delta'\tilde{y}_r[k]\| \quad (4.29)$$

where $\|\cdot\|$ denotes the Euclidean norm of a vector. If (4.29) is satisfied, then (4.28) is satisfied since the right hand side of (4.29) is greater than or equal to the right hand side of (4.28) according to the Schwarz inequality.

By making the further simplification that K_D is a diagonal matrix with all terms equal (i.e. $K_D = k_D I^{n \times n}$), $s[k]$ and $K_D s[k]$ become linearly dependent vectors, and hence the Schwarz inequality becomes an *equality*. Thus, we can now make use of the equality

$$|s^T[k]K_D s[k]| = \|s[k]\| \|K_D s[k]\| \quad (4.30)$$

to derive

$$\|K_D s[k]\| \geq \frac{1}{2}\|M[k]\Delta'\tilde{y}_r[k]\| \quad (4.31)$$

It can also be shown that the properties of K_D assumed above can be used to derive the following inequality from (4.31):

$$\|K_D\|_2 \|s[k]\| \geq \frac{1}{2}\|M[k]\Delta'\tilde{y}_r[k]\|, \quad (4.32)$$

where $\|\cdot\|_2$ is the matrix 2-norm, defined as:

$$\|P\|_2 \equiv \sup_r \left\{ \frac{\|Pr\|}{\|r\|} \right\} \quad (4.33)$$

where P is an arbitrary matrix and r is a vector variable. The following property can be derived from the definition of the matrix 2-norm (4.33):

$$\sup_r \|Pr\| = \|P\|_2 \sup_r \|r\|. \quad (4.34)$$

This property can be used to derive the following from (4.32):

$$\|s[k]\| \geq \frac{1}{2\|K_D\|_2} \|M[k]\|_2 \sup_{\Delta' \tilde{y}_r[k]} \|\Delta' \tilde{y}_r[k]\| \quad (4.35)$$

Since $M[k]$ is positive definite and symmetric,

$$\|M[k]\|_2 = \sigma_{\max}(M[k]), \quad (4.36)$$

where $\sigma_{\max}(M[k])$ indicates the maximum eigenvalue of $M[k]$ over an allowable trajectory of the manipulator.

We also know that (see (4.27))

$$\|\Delta' \tilde{y}_r[k]\| = \|\Delta \tilde{y}_r[k]\| \quad (4.37)$$

and for small tracking errors (see (2.35)),

$$\Delta \tilde{y}_r[k] \approx \Delta \tilde{y}_d[k]. \quad (4.38)$$

The final stability condition which sets the predicted bound on the norm of the filtered error $\|s[k]\|$ can now be stated:

$$\|s[k]\| \geq \frac{\sigma_{\max}(M[k]) \sup_k \|\Delta \tilde{y}_d[k]\|}{2\|K_D\|_2} = \|s[k]\|_{\text{bound}}, \quad (4.39)$$

It must be noted that when computing the bound (4.39), it is sufficient to evaluate the maximum eigenvalue of $M[k]$ and the supremum of $\|\Delta \tilde{y}_d[k]\|$ independently over the range of $y[k]$ for the desired trajectory (assuming small tracking errors). However, this is expected to be overly conservative in general, since these two values are not necessarily independent and the interaction between the two terms may lead

to a lower predicted bound. It is important to remember that this bound is based on the assumption that the change in certain dynamic terms over a sampling interval can be neglected (see (4.22) to (4.24)), and thus refers to the steady-state condition. Also note that Γ^{-1} and Δt do not appear explicitly in the stability condition (4.39). The effect of Δt appears in the calculation $\|\Delta \tilde{y}_d\|$, which is the norm of the change in $\tilde{y}_d$ occurring during the sampling period (Δt).

Bounds on the Error from Simulations

The expected bound on $\|s[k]\|$ can be predicted by (4.39). The maximum eigenvalue of $M(y[k])$ can be evaluated from the definition of the desired path. For the sampled-data simulation of the baseline adaptive case described previously in Section 3.4, $M(y[k])$ evaluated for the *actual* path gives an essentially identical result due to the low tracking error. Once again using the desired path, the maximum value of $\|\Delta \tilde{y}_d[k]\|$ can be calculated. The calculated values are:

$$\begin{aligned}\text{maximum eigenvalue of } M[k] &= 0.3686 \\ \sup \|\Delta \tilde{y}_d[k]\| &= 1.3812 \text{ rad/sec}^2 \\ \|K_D\|_2 &= 8.0\end{aligned}$$

Thus, from (4.39), $\|s[k]\|_{\text{bound}} = 0.0318 \text{ rad/sec}$.

$\|s[k]\|$ for the simulation of the baseline sampled-data adaptive case previously discussed in Section 3.4 is shown in Figure 4.1. This baseline case is for a sampling rate of 100 Hz, and with the adaptation and feedback gains, and the filtered error parameter matrix set as follows: $\Gamma^{-1} = 0.2I^{3 \times 3}$, $K_D = 8I^{2 \times 2}$, $\Lambda = 10I^{2 \times 2}$. During the initial adaptation transient (first 1.5 seconds), $\|s[k]\|$ reaches a maximum over 0.3 rad/sec, greatly exceeding the predicted bound. As expected, the restrictive assumption used to formulate (4.39) — that the change in $s[k]$ between sampling instants must be negligible — rules out the use of the predicted bound during the initial adaptation transient.

After the initial adaptation transient, the simulation results show that $\|s[k]\|$ stays within the bound of 0.0392 rad/sec. This shows reasonable agreement between the predicted and simulated bound on the filtered error for the “steady state”

Figure 4.1: Norm of Filtered Tracking Errors, $\|s[k]\|$, for Sampled-Data System

condition, although the prediction underestimates the bound by 23%. However, the error in the prediction for the steady state condition, and the failure of the prediction to bound the error during the initial adaptation, is a result of the simplifying assumptions listed above (see also (4.22), (4.23) and (4.24)). It was determined that for a range of other simulation cases, the calculated bound was found to be much less accurate, and in some cases greatly underestimated the bound on $\|s[k]\|$ (for example, for much higher sampling rates).

It is problematic that the stability condition (4.39) was not capable of accurately predicting the bound on filtered error for the steady state condition for some simulation cases. It is also critical that the analytical tools be available to predict stability and tracking error during periods of rapid parameter adaptation. In other words, since it is the adaptive character of the sampled-data system which is of primary interest, the limited applicability of the prediction must be broadened to include the time when rapid parameter adaptation is occurring. This necessitates inclusion of the parameter adaptation gain Γ^{-1} in the stability condition; intuition, simulation and actual experimental results all indicate that Γ^{-1} is a major factor in determining the system stability.

4.2.3 Exact Evaluation of the Stability Condition for the One DOF Case

To broaden the applicability of the prediction of stability and tracking performance derived above, it is necessary to remove the simplifying assumptions that were made and to get the adaptation gain Γ^{-1} to appear in the stability condition. It has been shown in Warshaw et al [63] that it is possible to exactly evaluate the integral terms in ΔV_k for the case of a one DOF pure inertia system (representing a simplified model of one joint of a direct-drive robot) without resorting to any simplifying assumptions, by making use of the knowledge of the plant dynamics. However, the exact solution for ΔV_k at each time step comes at the expense of greatly increased complexity of the analysis.

The exact solution for ΔV_k is in terms of measurable quantities for the current time step and known quantities from the previous time step. Even for the simple one DOF case, the solution is a very complex polynomial, which is cumbersome to deal with. By making use of the symbolic math capabilities of the "Mathematica" program (a product of Wolfram Research, Inc.), an expression consisting of 103 terms was derived. This derivation is summarized below.

A simplified model of the dynamics for a joint of a manipulator (which ignores friction, and motor/amplifier dynamics) is a one DOF pure inertia. Therefore, the equations of motion for a single joint in the form of equation (2.31) degenerate to the following scalar equation:

$$\tau(t) = I\ddot{y}(t) \quad (4.40)$$

where I is the effective inertia of the motor/arm combination for the joint. The control law and adaptation law given previously as (2.32) and (2.33) respectively thus become (in this scalar case):

$$\tau(t) = \hat{I}(t)\ddot{y}_r(t) - k_D s(t) \quad (4.41)$$

$$\dot{\hat{I}}(t) = -\gamma^{-1}\ddot{y}_r(t)s(t) \quad (4.42)$$

The adaptive controller when implemented in discrete-time and consists of the

the control law and adaptation law, taken from equations (4.41) and (4.42):

$$\tau[k] = \hat{I}[k]\tilde{y}_r[k] - k_D s[k] \quad (4.43)$$

$$\hat{I}[k] = \hat{I}[k-1] - \Delta t \gamma^{-1} \tilde{y}_r[k] s[k] \quad (4.44)$$

By repeating the procedure described in Section 4.2.1, $\dot{V}(z(t))$ is derived as:

$$\dot{V}(z(t)) = -k_D s[k] s(t) + s(t) \{ \hat{I}[k] \tilde{y}_r[k] - I \tilde{y}_r(t) \} - \Delta t \tilde{y}_r[k] s[k] \{ \hat{I}[k] - I \} \delta(t - t_k). \quad (4.45)$$

The integration of $\dot{V}(z(t))$ over a single sampling interval results in the one DOF version of (4.16):

$$\begin{aligned} \Delta V_k &= \int_{t_k^-}^{t_{k+1}^-} \dot{V}(z(t)) dt \\ &= -k_D s[k] \int_{t_k^-}^{t_{k+1}^-} s(t) dt + \hat{I}[k] \tilde{y}_r[k] \int_{t_k^-}^{t_{k+1}^-} s(t) dt - \\ &\quad I \int_{t_k^-}^{t_{k+1}^-} s(t) \tilde{y}_r(t) dt - \Delta t \tilde{y}_r[k] s[k] \{ \hat{I}[k] - I \}. \end{aligned} \quad (4.46)$$

It is now possible to evaluate the integral terms in ΔV_k without resorting to any simplifying assumptions, by making use of the knowledge of the plant dynamics. Within each sampling interval ($t_k \leq t < t_{k+1}$), $s(t)$ can be expressed as a function of the manipulator state at time t_k and the control torque applied at that time according to the discretized adaptive control law (and held throughout the sampling interval). Likewise, $\tilde{y}_r(t)$ can be expressed as a function of the actual and desired state at time t_k . Using the discrete adaptation law, $\hat{I}[k]$ can be calculated from the estimate for the previous time step ($\hat{I}[k-1]$) and from $s[k]$ and $\tilde{y}_r[k]$. In this fashion, the terms of (4.46) can be exactly calculated to predict the stability at each step (assuming that the initial conditions are known). As will be demonstrated below, the exact solution for ΔV_k at each step comes at the expense of greatly increased complexity of the analysis.

Let us define $\bar{t}$ as the time since the beginning of the sampling interval (i.e. time since the last sampling instant):

$$\bar{t} \equiv t - t_k. \quad (4.47)$$

Since the control torque is held constant over the sampling interval, and the system is a pure inertia, the acceleration is also constant during the sampling interval, i.e.

$$\ddot{y}(t) = \ddot{y}[k] \text{ for } t_k \leq t < t_{k+1}. \quad (4.48)$$

The constant acceleration leads to the following dynamic behaviour during the sampling interval:

$$y(t) = y[k] + \bar{t}\dot{y}[k] + \frac{1}{2}\bar{t}^2\ddot{y}[k] \quad (4.49)$$

$$\dot{y}(t) = \dot{y}[k] + \bar{t}\ddot{y}[k]. \quad (4.50)$$

Since

$$s(t) = \dot{e}(t) + \lambda e(t) \quad (4.51)$$

and during the same sampling interval

$$e(t) = y(t) - y_d[k] \quad (4.52)$$

$$\dot{e}(t) = \dot{y}(t) - \dot{y}_d[k], \quad (4.53)$$

we get by using (4.49) and (4.50):

$$s(t) = s[k] + \lambda\bar{t}\dot{y}[k] + \bar{t}(1 + \frac{\lambda}{2}\bar{t})\ddot{y}[k]. \quad (4.54)$$

We can also state that

$$\ddot{y}_r(t) = \ddot{y}_d[k] - \lambda\dot{e}(t). \quad (4.55)$$

Equating the equation of motion and control law ((4.40) and (4.43) respectively) results in

$$\ddot{y}[k] = \frac{\hat{I}[k]\ddot{y}_r[k] - k_D s[k]}{I}. \quad (4.56)$$

The integrals (4.46) can now be rewritten in the following fashion by making use of the definition of $\bar{t}$:

$$\int_{t_k^-}^{t_{k+1}^-} s(t)dt = \int_0^{\Delta t} s(\bar{t} + t_k)d\bar{t} = \int_0^{\Delta t} s(t)d\bar{t} \quad (4.57)$$

and similarly

$$\int_{t_k^-}^{t_{k+1}^-} s(t)\ddot{y}_r(t)dt = \int_0^{\Delta t} s(t)\ddot{y}_r(t)d\bar{t}. \quad (4.58)$$

Evaluation of the two integrals above results in the following expressions:

$$\int_{t_k^-}^{t_{k+1}^-} s(t) dt = \Delta t s[k] + \frac{(\Delta t)^2 \ddot{y}[k]}{2} + \frac{(\Delta t)^3 \lambda \ddot{y}[k]}{6} + \frac{(\Delta t)^2 \lambda \dot{y}[k]}{2} \quad (4.59)$$

$$\begin{aligned} \int_{t_k^-}^{t_{k+1}^-} s(t) \ddot{y}_r(t) dt = & \quad (4.60) \\ & - \frac{(\Delta t)^4 \lambda^2 \ddot{y}^2[k]}{8} + \Delta t (s[k] \ddot{y}_d[k] - \lambda s[k] \dot{y}[k] + \lambda s[k] \dot{y}_d[k]) + \\ & (\Delta t)^3 (-\lambda \ddot{y}^2[k] + \frac{\lambda \ddot{y}[k] \ddot{y}_d[k]}{2} - \frac{3 \lambda^2 \ddot{y}[k] \dot{y}[k]}{2} + \frac{\lambda^2 \ddot{y}[k] \dot{y}_d[k]}{2})/3 + \\ & (\Delta t)^2 (-\lambda s[k] \ddot{y}[k] + \ddot{y}[k] \ddot{y}_d[k] - \lambda \ddot{y}[k] \dot{y}[k] + \lambda \ddot{y}_d[k] \dot{y}[k] - \lambda^2 \dot{y}^2[k] + \\ & \lambda \ddot{y}[k] \dot{y}_d[k] + \lambda^2 \dot{y}[k] \dot{y}_d[k])/2. \end{aligned}$$

The expressions for the integrals can now be substituted into (4.46) to yield an explicit expression for ΔV_k . Equations (4.55), (4.56) and (4.44) must also be substituted into the expression to give ΔV_k in terms of measurable quantities for the current time step and known quantities from the previous time step. When this is done, the expression becomes a very complex polynomial, which is cumbersome to deal with. Using “Mathematica”, the following expression was derived (consisting of 103 terms):

$$\begin{aligned} \Delta V_k = & \frac{(\Delta t)^2 \hat{I}[k-1]^2 \lambda^2 \dot{e}[k]^2}{2I} + \frac{(\Delta t)^3 \hat{I}[k-1]^2 \lambda^3 \dot{e}[k]^2}{2I} + \frac{(\Delta t)^4 \hat{I}[k-1]^2 \lambda^4 \dot{e}[k]^2}{8I} - \\ & (\Delta t) I \lambda \dot{e}[k] s[k] + \frac{(\Delta t)^2 \hat{I}[k-1] k_D \lambda \dot{e}[k] s[k]}{I} - \frac{(\Delta t)^2 \hat{I}[k-1] \lambda^2 \dot{e}[k] s[k]}{2} + \\ & \frac{(\Delta t)^3 \hat{I}[k-1] k_D \lambda^2 \dot{e}[k] s[k]}{I} + \frac{(\Delta t)^4 \hat{I}[k-1] k_D \lambda^3 \dot{e}[k] s[k]}{4I} + \frac{(\Delta t)^3 \gamma^{-1} \hat{I}[k-1] \lambda^3 \dot{e}[k]^3 s[k]}{I} + \\ & \frac{(\Delta t)^4 \gamma^{-1} \hat{I}[k-1] \lambda^4 \dot{e}[k]^3 s[k]}{I} + \frac{(\Delta t)^5 \gamma^{-1} \hat{I}[k-1] \lambda^5 \dot{e}[k]^3 s[k]}{4I} - (\Delta t) k_D s[k]^2 + \\ & \frac{(\Delta t)^2 k_D^2 s[k]^2}{2I} - \frac{(\Delta t)^2 k_D \lambda s[k]^2}{2} + \frac{(\Delta t)^3 k_D^2 \lambda s[k]^2}{2I} + \frac{(\Delta t)^4 k_D^2 \lambda^2 s[k]^2}{8I} + \\ & \frac{(\Delta t)^3 \gamma^{-1} k_D \lambda^2 \dot{e}[k]^2 s[k]^2}{I} - \frac{(\Delta t)^3 \gamma^{-1} \lambda^3 \dot{e}[k]^2 s[k]^2}{2} + \frac{(\Delta t)^4 \gamma^{-1} k_D \lambda^3 \dot{e}[k]^2 s[k]^2}{I} + \\ & \frac{(\Delta t)^5 \gamma^{-1} k_D \lambda^4 \dot{e}[k]^2 s[k]^2}{4I} + \frac{(\Delta t)^4 (\gamma^{-1})^2 \lambda^4 \dot{e}[k]^4 s[k]^2}{2I} + \frac{(\Delta t)^5 (\gamma^{-1})^2 \lambda^5 \dot{e}[k]^4 s[k]^2}{2I} + \\ & \frac{(\Delta t)^6 (\gamma^{-1})^2 \lambda^6 \dot{e}[k]^4 s[k]^2}{8I} + \frac{(\Delta t)^2 \hat{I}[k-1] \lambda \dot{e}[k] \ddot{y}_d[k]}{2} - \frac{(\Delta t)^2 \hat{I}[k-1] \lambda \dot{e}[k] \ddot{y}_d[k]}{I} + \\ & \frac{(\Delta t)^3 \hat{I}[k-1] \lambda^2 \dot{e}[k] \ddot{y}_d[k]}{6} - \frac{(\Delta t)^3 \hat{I}[k-1]^2 \lambda^2 \dot{e}[k] \ddot{y}_d[k]}{I} - \frac{(\Delta t)^4 \hat{I}[k-1]^2 \lambda^3 \dot{e}[k] \ddot{y}_d[k]}{4I} + \end{aligned}$$

$$\begin{aligned}
& \frac{(\Delta t)^2 k_D s[k] \ddot{y}_d[k]}{2} - \frac{(\Delta t)^2 \hat{I}[k-1] k_D s[k] \ddot{y}_d[k]}{I} + \frac{(\Delta t)^2 \hat{I}[k-1] \lambda s[k] \ddot{y}_d[k]}{2} + \\
& \frac{(\Delta t)^3 k_D \lambda s[k] \ddot{y}_d[k]}{6} - \frac{(\Delta t)^3 \hat{I}[k-1] k_D \lambda s[k] \ddot{y}_d[k]}{I} - \frac{(\Delta t)^4 \hat{I}[k-1] k_D \lambda^2 s[k] \ddot{y}_d[k]}{4I} + \\
& \frac{(\Delta t)^3 \gamma^{-1} \lambda^2 \dot{e}[k]^2 s[k] \ddot{y}_d[k]}{2} - \frac{3(\Delta t)^3 \gamma^{-1} \hat{I}[k-1] \lambda^2 \dot{e}[k]^2 s[k] \ddot{y}_d[k]}{I} + \\
& \frac{(\Delta t)^4 \gamma^{-1} \lambda^3 \dot{e}[k]^2 s[k] \ddot{y}_d[k]}{6} - \frac{3(\Delta t)^4 \gamma^{-1} \hat{I}[k-1] \lambda^3 \dot{e}[k]^2 s[k] \ddot{y}_d[k]}{I} - \\
& \frac{3(\Delta t)^5 \gamma^{-1} \hat{I}[k-1] \lambda^4 \dot{e}[k]^2 s[k] \ddot{y}_d[k]}{4I} - \frac{2(\Delta t)^3 \gamma^{-1} k_D \lambda \dot{e}[k] s[k]^2 \ddot{y}_d[k]}{I} + \\
& (\Delta t)^3 \gamma^{-1} \lambda^2 \dot{e}[k] s[k]^2 \ddot{y}_d[k] - \frac{2(\Delta t)^4 \gamma^{-1} k_D \lambda^2 \dot{e}[k] s[k]^2 \ddot{y}_d[k]}{I} - \\
& \frac{(\Delta t)^5 \gamma^{-1} k_D \lambda^3 \dot{e}[k] s[k]^2 \ddot{y}_d[k]}{2I} - \frac{2(\Delta t)^4 (\gamma^{-1})^2 \lambda^3 \dot{e}[k]^3 s[k]^2 \ddot{y}_d[k]}{I} - \\
& \frac{2(\Delta t)^5 (\gamma^{-1})^2 \lambda^4 \dot{e}[k]^3 s[k]^2 \ddot{y}_d[k]}{I} - \frac{(\Delta t)^6 (\gamma^{-1})^2 \lambda^5 \dot{e}[k]^3 s[k]^2 \ddot{y}_d[k]}{2I} - \\
& \frac{(\Delta t)^2 \hat{I}[k-1] \ddot{y}_d[k]^2}{2} + \frac{(\Delta t)^2 \hat{I}[k-1]^2 \ddot{y}_d[k]^2}{2I} - \frac{(\Delta t)^3 \hat{I}[k-1] \lambda \ddot{y}_d[k]^2}{6} + \\
& \frac{(\Delta t)^3 \hat{I}[k-1]^2 \lambda \ddot{y}_d[k]^2}{2I} + \frac{(\Delta t)^4 \hat{I}[k-1]^2 \lambda^2 \ddot{y}_d[k]^2}{8I} - (\Delta t)^3 \gamma^{-1} \lambda \dot{e}[k] s[k] \ddot{y}_d[k]^2 + \\
& \frac{3(\Delta t)^3 \gamma^{-1} \hat{I}[k-1] \lambda \dot{e}[k] s[k] \ddot{y}_d[k]^2}{I} - \frac{(\Delta t)^4 \gamma^{-1} \lambda^2 \dot{e}[k] s[k] \ddot{y}_d[k]^2}{3} + \\
& \frac{3(\Delta t)^4 \gamma^{-1} \hat{I}[k-1] \lambda^2 \dot{e}[k] s[k] \ddot{y}_d[k]^2}{I} + \frac{3(\Delta t)^5 \gamma^{-1} \hat{I}[k-1] \lambda^3 \dot{e}[k] s[k] \ddot{y}_d[k]^2}{4I} + \\
& \frac{(\Delta t)^3 \gamma^{-1} k_D s[k]^2 \ddot{y}_d[k]^2}{I} - \frac{(\Delta t)^3 \gamma^{-1} \lambda s[k]^2 \ddot{y}_d[k]^2}{2} + \frac{(\Delta t)^4 \gamma^{-1} k_D \lambda s[k]^2 \ddot{y}_d[k]^2}{I} + \\
& \frac{(\Delta t)^5 \gamma^{-1} k_D \lambda^2 s[k]^2 \ddot{y}_d[k]^2}{4I} + \frac{3(\Delta t)^4 (\gamma^{-1})^2 \lambda^2 \dot{e}[k]^2 s[k]^2 \ddot{y}_d[k]^2}{I} + \\
& \frac{3(\Delta t)^5 (\gamma^{-1})^2 \lambda^3 \dot{e}[k]^2 s[k]^2 \ddot{y}_d[k]^2}{I} + \frac{3(\Delta t)^6 (\gamma^{-1})^2 \lambda^4 \dot{e}[k]^2 s[k]^2 \ddot{y}_d[k]^2}{4I} + \\
& \frac{(\Delta t)^3 \gamma^{-1} s[k] \ddot{y}_d[k]^3}{2} - \frac{(\Delta t)^3 \gamma^{-1} \hat{I}[k-1] s[k] \ddot{y}_d[k]^3}{I} + \frac{(\Delta t)^4 \gamma^{-1} \lambda s[k] \ddot{y}_d[k]^3}{6} - \\
& \frac{(\Delta t)^4 \gamma^{-1} \hat{I}[k-1] \lambda s[k] \ddot{y}_d[k]^3}{I} - \frac{(\Delta t)^5 \gamma^{-1} \hat{I}[k-1] \lambda^2 s[k] \ddot{y}_d[k]^3}{4I} - \\
& \frac{2(\Delta t)^4 (\gamma^{-1})^2 \lambda \dot{e}[k] s[k]^2 \ddot{y}_d[k]^3}{I} - \frac{2(\Delta t)^5 (\gamma^{-1})^2 \lambda^2 \dot{e}[k] s[k]^2 \ddot{y}_d[k]^3}{I} - \\
& \frac{(\Delta t)^6 (\gamma^{-1})^2 \lambda^3 \dot{e}[k] s[k]^2 \ddot{y}_d[k]^3}{2I} + \frac{(\Delta t)^4 (\gamma^{-1})^2 s[k]^2 \ddot{y}_d[k]^4}{2I} + \\
& \frac{(\Delta t)^5 (\gamma^{-1})^2 \lambda s[k]^2 \ddot{y}_d[k]^4}{2I} + \frac{(\Delta t)^6 (\gamma^{-1})^2 \lambda^2 s[k]^2 \ddot{y}_d[k]^4}{8I} - \\
& (\Delta t)^2 \hat{I}[k-1] \lambda^2 \dot{e}[k] \dot{y}[k] - \frac{(\Delta t)^3 \hat{I}[k-1] \lambda^3 \dot{e}[k] \dot{y}[k]}{2} + (\Delta t) I \lambda s[k] \dot{y}[k] -
\end{aligned}$$

$$\begin{aligned}
& (\Delta t)^2 k_D \lambda s[k] \dot{y}[k] - \frac{(\Delta t)^3 k_D \lambda^2 s[k] \dot{y}[k]}{2} - (\Delta t)^3 \gamma^{-1} \lambda^4 \dot{e}[k]^2 s[k] \dot{y}[k] - \\
& \frac{(\Delta t)^4 \gamma^{-1} \lambda^4 \dot{e}[k]^2 s[k] \dot{y}[k]}{2} - \frac{(\Delta t)^2 I \lambda \ddot{y}_d[k] \dot{y}[k]}{2} + (\Delta t)^2 \hat{I}[k-1] \lambda \ddot{y}_d[k] \dot{y}[k] + \\
& \frac{(\Delta t)^3 \hat{I}[k-1] \lambda^2 \ddot{y}_d[k] \dot{y}[k]}{2} + 2(\Delta t)^3 \gamma^{-1} \lambda^2 \dot{e}[k] s[k] \ddot{y}_d[k] \dot{y}[k] + \\
& (\Delta t)^4 \gamma^{-1} \lambda^3 \dot{e}[k] s[k] \ddot{y}_d[k] \dot{y}[k] - (\Delta t)^3 \gamma^{-1} \lambda s[k] \ddot{y}_d[k]^2 \dot{y}[k] - \\
& \frac{(\Delta t)^4 \gamma^{-1} \lambda^2 s[k] \ddot{y}_d[k]^2 \dot{y}[k]}{2} + \frac{(\Delta t)^2 I \lambda^2 \dot{y}[k]^2}{2} + \frac{(\Delta t)^2 \hat{I}[k-1] \lambda^2 \dot{e}[k] \ddot{y}_d[k]}{2} + \\
& \frac{(\Delta t)^3 \hat{I}[k-1] \lambda^3 \dot{e}[k] \ddot{y}_d[k]}{6} - (\Delta t) I \lambda s[k] \ddot{y}_d[k] + \frac{(\Delta t)^2 k_D \lambda s[k] \dot{y}_d[k]}{2} + \\
& \frac{(\Delta t)^3 k_D \lambda^2 s[k] \dot{y}_d[k]}{6} + \frac{(\Delta t)^3 \gamma^{-1} \lambda^3 \dot{e}[k]^2 s[k] \dot{y}_d[k]}{2} + \\
& \frac{(\Delta t)^4 \gamma^{-1} \lambda^4 \dot{e}[k]^2 s[k] \dot{y}_d[k]}{6} - \frac{(\Delta t)^2 \hat{I}[k-1] \lambda \ddot{y}_d[k] \dot{y}_d[k]}{2} - \\
& \frac{(\Delta t)^3 \hat{I}[k-1] \lambda^2 \ddot{y}_d[k] \dot{y}_d[k]}{6} - (\Delta t)^3 \gamma^{-1} \lambda^2 \dot{e}[k] s[k] \ddot{y}_d[k] \dot{y}_d[k] - \\
& \frac{(\Delta t)^4 \gamma^{-1} \lambda^3 \dot{e}[k] s[k] \ddot{y}_d[k] \dot{y}_d[k]}{3} + \frac{(\Delta t)^3 \gamma^{-1} \lambda s[k] \ddot{y}_d[k]^2 \dot{y}_d[k]}{2} + \\
& \frac{(\Delta t)^4 \gamma^{-1} \lambda^2 s[k] \ddot{y}_d[k]^2 \dot{y}_d[k]}{6} - \frac{(\Delta t)^2 I \lambda^2 \dot{y}[k] \dot{y}_d[k]}{2}.
\end{aligned} \tag{4.61}$$

Inspection of the derived exact solution for ΔV_k given by (4.61) shows that it is not a negative (semi-)definite function of the filtered error. Thus, we have not succeeded in guaranteeing the stability of the discretization of Slotine and Li's adaptive algorithm for the one DOF sampled-data system. However, it is possible to use (4.61) as a means for dynamically adjusting the adaptive and feedback gains in an attempt to compensate the discretized adaptive controller and thereby guarantee stability of the sampled-data system. However, it is shown below in Section 4.2.4 that this approach is unsuccessful in obtaining uniform stability (for a sampling rate of 100 Hz).

A challenging problem is to extend the above derivation of the exact expression for ΔV_k for the sampled-data system to the general (multi-degree-of-freedom) case. The difficulty in performing this generalization is to evaluate the integrals found in the expression for ΔV_k (see (4.16)). In the multi-DOF case, the evolution of the system states over a sampling interval are not easily solved as was the case for the one DOF system described above. As an alternative to exactly evaluating ΔV_k , the terms composing it can be bounded and hence a provable stability condition can still

be derived for the multi-DOF case. This is the approach described in Section 4.4.

4.2.4 An Adaptive Controller with Variable Feedback and Adaptation Gains

It has been shown that using Slotine and Li's adaptive controller structure and the corresponding Lyapunov function does not result in a provably stable sampled-data system. However, it can be postulated that there may be a region where the adaptation gain γ^{-1} and feedback gain k_D could exist such that stability can be proven over a given desired trajectory (using the Slotine and Li Lyapunov function). It is demonstrated below that this cannot be done successfully for the discretization of Slotine and Li's adaptive algorithm.

To find values of k_D and γ^{-1} which guarantee stability, it is necessary to choose these gains such that ΔV_k is driven negative. This can be attempted by selecting k_D and γ^{-1} at each time step to minimize ΔV_k . To do this, we can make use of the exact expression for ΔV_k for the one DOF case referred to previously in Section 4.2.3. While this expression for ΔV_k is ungainly, by collecting terms it is observed to be only a second degree function of k_D and γ^{-1} , i.e. it can be expressed in the polynomial form:

$$\Delta V_k = a_1 k_D^2 + a_2 (\gamma^{-1})^2 + b_1 k_D + b_2 \gamma^{-1} + c + d k_D \gamma^{-1} \quad (4.62)$$

where the parameters a_1, a_2, b_1, b_2, c, d are complicated functions of the other quantities appearing in ΔV_k . The polynomial (4.62) can be used to pick k_D and γ^{-1} .

Modified Algorithm with Adjustment of k_D or γ^{-1}

Consider first the case where γ^{-1} is held constant, but k_D is allowed to vary to minimize ΔV_k ; (4.62) now becomes the quadratic:

$$\Delta V_k = a k_D^2 + b k_D + c \quad (4.63)$$

This is the equation of a parabola, opening upward and hence possessing a single finite minimum if $a > 0$. It can easily be shown that the value of k_D which occurs at the minimum is $k_D = -b/(2a)$.

Similarly, k_D can be held constant, with γ^{-1} varying. Once again, this gives a quadratic equation like (4.63), with the minimum for ΔV_k at $\gamma^{-1} = -b/(2a)$.

There are two caveats when contemplating the adjustment of k_D and/or γ^{-1} to get the minimum ΔV_k as described above:

- the minimum for ΔV_k may be greater than 0, i.e. the quadratic may not have real roots at times;
- the value of k_D and/or γ^{-1} for the minimum ΔV_k may be undesirable for other reasons, e.g. negative or very large.

It will be seen that these issues are indeed practical problems.

Simulations were run for the one DOF system using modified versions of Slotine and Li's algorithm, where the value of the gains were changed at each discrete time step to minimize ΔV_k for that time step. The modified algorithms are referred to as follows:

- variable k_D : γ^{-1} fixed and k_D varied;
- variable γ^{-1} : k_D fixed and γ^{-1} varied.

A sampling rate of 100 Hz was used, the system inertia was set to $I = 0.13 \text{ kg} \cdot \text{m}^2$, and the filtered error parameter was $\lambda = 10$. The desired trajectory was a sinusoid (which is persistently exciting for this one DOF system). For the baseline unmodified adaptive algorithm, the feedback and adaptive gains were fixed as $k_D = 5.7$ and $\gamma^{-1} = 0.005$.³ The position tracking error, parameter estimate and ΔV_k for this baseline case are shown in Figures 4.2 to 4.4.

The position tracking error, parameter estimate, variable gains k_D and γ^{-1} , and ΔV_k for both the modified algorithms are shown in Figures 4.5 to 4.12. The following observations can be made about the variable k_D case:

- although the tracking error is improved (lower than the nominal unmodified algorithm case) during the initial transient period, the error is worse at "steady state";

³Experience and the published literature have demonstrated that the adaptive gain must be kept small for the simulated systems to work properly and avoid numerical overflow.

Figure 4.2: Tracking Error for Unmodified Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.3: Parameter Estimate for Unmodified Adaptive Algorithm, 1 DOF Sampled-Data

Figure 4.4: ΔV_k for Unmodified Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.5: Tracking Error for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.6: Parameter Estimate for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.7: Time-Varying Gain, k_D , for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.8: ΔV_k for Variable k_D Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.9: Tracking Error for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.10: Parameter Estimate for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.11: Time-Varying Gain, γ^{-1} , for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.12: ΔV_k for Variable γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

- ΔV_k is only held negative for the first 2.5 seconds, and then undergoes increasingly large positive oscillations;
- the parameter estimate diverges from the true value;
- k_D is tending towards more negative values, but with very large amplitude spikes;

In reference to the last two comments: it is expected that the system would eventually become unstable due to numerical problems, and in a real implementation due to excessively high gains.

Observations can also be made for the variable γ^{-1} case:

1. ΔV_k is always negative;
2. the tracking error is very large;
3. the parameter estimate has huge spikes, due to large (and sometimes negative) swings in γ^{-1} .

While the observations (1) and (2) above may seem contradictory⁴, recall that

⁴ ΔV_k which is negative definite implies that the system states (s and $\tilde{I}$) must be decreasing to zero.

the definition of $V(z(t))$ was based on a constant and positive definite value for γ (see (2.37)); this is no longer the case here due to the variable value of γ^{-1} .

It must be remembered that the condition that $\Delta V_k \leq 0$ to guarantee stability says nothing directly about the tracking performance (especially during the initial period of adaptation), except that the tracking error is bounded. However, if the value of γ^{-1} is variable and at times negative (as for the case described above), the stability guarantee is no longer valid.

It is instructive to look at the quadratic equation for ΔV_k at each time step during the simulation, to see how ΔV_k changes as a function of time and k_D or γ^{-1} . This data is plotted in 3-D in Figures 4.13 and 4.14. It can be observed that at some times there is no obvious minimum within the range of k_D or γ^{-1} plotted (the parabola becomes very “shallow”); the minimum that exists may be for ΔV_k positive, and may be for k_D or γ^{-1} negative and/or very large magnitude. There is an important point that is illustrated by these figures, when using the same controller structure as Slotine and Li, but with variable feedback or adaptation gain: the attempt to guarantee stability (by showing that a value of k_D or γ^{-1} can always be chosen to ensure that ΔV_k is always negative) fails.

While the modified algorithms do not appear to offer a practical advantage, it is useful to reconsider the results of the variable k_D case: although the tracking error suffers at steady state versus the unmodified algorithm, stability is maintained. In fact, it can be demonstrated by simulation that this method is capable of stabilizing the sampled-data implementation for cases where the unmodified algorithm results in an unstable system. Examples of simulations where the unmodified algorithm is unstable but the variable k_D algorithm is stable are:

- $f_{sampling} = 100$ Hz, fixed $\gamma^{-1} = 0.2$ (high adapt. gain);
- $f_{sampling} = 25$ Hz, fixed $\gamma^{-1} = 0.005$ (undersampled, medium adapt. gain);
- $f_{sampling} = 20$ Hz, fixed $\gamma^{-1} = 0$ (undersampled non-adaptive PD).

For the latter case, the modified controller actually stabilizes the undersampled PD controller. However, it is important to reiterate that undesirable effects are present:

Figure 4.13: ΔV_k Surface for Variable k_D

Figure 4.14: ΔV_k Surface for Variable γ^{-1}

parameter estimate drift, large swings in gains, very large and/or negative gains, and large tracking errors in some cases.

Modified Algorithm with Adjustment of k_D and γ^{-1} Simultaneously

It is possible to set up a static optimization problem using (4.62) to simultaneously pick both k_D and γ^{-1} to give the minimum ΔV_k . This is solved by computing the partial derivatives of (4.62) with respect to k_D and γ^{-1} , setting these derivatives equal to zero, and then solving the resulting system of two simultaneous linear equations for k_D and γ^{-1} . However, simulation of a modified adaptive controller which implements this scheme was not successful: the simulation went unstable after very little time (7 time steps = 0.07 sec.). The apparent instability was evident before the simulation terminated by observing the very large swings in the values of k_D , γ^{-1} , ΔV_k , tracking error and the parameter estimate. It can be concluded that straightforward minimization of ΔV_k does not work, and that the behaviour of ΔV_k as a function of k_D , γ^{-1} and time step must be looked at more carefully.

At each time step, (4.62) can be represented as a 3-D surface. Unfortunately, the surface described by typical simulation data proved to be very difficult to visualize due to the very large range in magnitudes of the coefficients of the second degree equation. However, examination of the roots of the equation derived by setting $\Delta V_k = 0$ in (4.62) proved to be more tractable. These roots are the intersection of the surface for ΔV_k with the $\Delta V_k = 0$ plane, and trace out a (2-D) curve in $k_D - \gamma^{-1}$ space. A sketch of the 3-D surface and 2-D curve of the roots is shown in Figure 4.15.

From observation of typical data from simulation runs, it can be stated that the second degree curves for $\Delta V_k = 0$ are always either a pair of hyperbolas, or a highly elongated ellipse, inclined at a range of angles relative to the k_D and γ^{-1} axes. The form of these curves can be derived analytically from the coefficients of the second degree curve, by rotating the axes to normalize the equation and then observing the magnitude and sign of the normalized coefficients. A typical plot of the roots of $\Delta V_k = 0$ for a baseline adaptive simulation case with fixed k_D and γ^{-1} gives a locus of roots shaped as an inclined and highly elongated ellipse with one end near the

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

Figure 4.15: ΔV_k Surface for Variable k_D and γ^{-1}

origin.

From looking at the simulation data, it became obvious why simple minimization of ΔV_k was not successful:

- for the elongated ellipse case, minima occurred for very large values of k_D or γ^{-1} ;
- for the hyperbola case, there are no finite minima.

To derive a more successful modified algorithm, various alternative rules for picking k_D and γ^{-1} within the $\Delta V_k < 0$ region can be contemplated. An additional guideline is to pick k_D and γ^{-1} in a way which avoids exciting any high frequency unmodelled dynamics. These unmodelled dynamics may be due to the physical structural dynamics of the plant, or due to numerical issues (e.g. signals changing too quickly between sampling times to be properly represented by the discretized system). One possible set of rules is represented by the sketch in Figure 4.16. The shading on the plot indicates the region where $\Delta V_k < 0$, and thus the region where k_D and γ^{-1} should be picked to guarantee stability. The selected gains can be calculated from the coefficients of the second degree curve by finding the extrema.

The results from a simulation of the modified algorithm using the extrema rule for simultaneously picking k_D and γ^{-1} (as described in the previous paragraph)

Figure 4.16: Selection of k_D and γ^{-1} using Extrema

are plotted in Figures 4.17 to 4.21. The data shows improvement in keeping ΔV_k negative compared with the variable k_D case (see Figure 4.8), and the tracking error is also reduced compared to the variable γ^{-1} modified algorithm (Figure 4.9). However, there are still occasional large “spikes” in k_D and γ^{-1} , and ΔV_k still becomes positive at times (albeit very briefly). Thus, it can be suspected that the equation for ΔV_k is in a sense ill-behaved at some times, since the extrema rule for picking the gains is still not successful for all time steps.

As for the modified controller with variable k_D , it can be demonstrated by simulation that the modified algorithm, this time using simultaneous selection of k_D and γ^{-1} , is capable of stabilizing the sampled-data implementation for cases where the unmodified algorithm results in an unstable system. Figures 4.22 to 4.25 show the performance for the undersampled case ($f_{\text{sampling}} = 25\text{Hz}$). While the tracking performance is poor, it is bounded and the system remains stable.

It has been demonstrated by simulation that the method of variable feedback and adaptive gains is capable of stabilizing the sampled-data implementation for cases where the unmodified algorithm results in an unstable one DOF system. However, as discussed in Section 4.2.3, the method is not directly applicable to the general multi-DOF robot case. The value of this modified algorithm for a practical implementation must also be questioned in light of three other important issues:

Figure 4.17: Tracking Error for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.18: Parameter Estimate for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.19: Time-Varying Gain, k_D , for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.20: Time-Varying Gain, γ^{-1} , for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.21: ΔV_k for Variable k_D and γ^{-1} Adaptive Algorithm, 1 DOF Sampled-Data System

Figure 4.22: Tracking Error for Undersampled Case

Figure 4.23: Parameter Estimate for Undersampled Case

Figure 4.24: Time-Varying Gain, k_D , for Undersampled Case Figure 4.25: Time-Varying Gain, γ^{-1} , for Undersampled Case

1. the parameters of the polynomial (4.62) contain the quantity I , the true value of the system inertia, which is not known a-priori;
2. all the preceding analyses of the sampled-data adaptive system assume that measurements of the states, calculation of the parameter adaptation and control law, and output of the control torque, are all performed instantaneously at the beginning of each sampling interval i.e. there are no computation and associated delays;
3. while satisfying the condition on ΔV_k can be used to guarantee stability, it does not directly provide a guaranteed rate of convergence and hence cannot provide guarantees on the transient performance of the controller and the tracking error magnitudes.

An additional analytical problem occurs when modifying the gain γ^{-1} used for the modified adaptive law: the expression for V_k , and hence $\dot{V}(z(t))$, is modified. Under some conditions, this can prevent a proof of stability using Lyapunov's second method (e.g. when γ^{-1} becomes negative), since V_k is required to be a positive definite function and $\dot{V}(z(t))$ must be a negative semi-definite function (see the formulation in Section 2.3.1)⁵. Strictly speaking, if γ^{-1} is no longer a constant but a function of time (updated at discrete times, as proposed above) then the expression for $\dot{V}(z(t))$ must be modified accordingly to include the time derivative of γ^{-1} (see (4.11)), in the same way that $\dot{V}(z(t))$ includes the effect of the discrete parameter update. Due to the complex nature of the method for adjusting γ^{-1} discussed above, it was impractical to include the time derivative of γ^{-1} in $\dot{V}(z(t))$ for the purposes of the analysis.

It has been shown that even for the simplest manipulator model (the degenerate case, a one DOF system), it is difficult to prove stability of the sampled-data system. The author was not able to demonstrate uniform stability for the modified adaptive algorithm using variable k_D and γ^{-1} . An important conclusion is that the controller

⁵Recall that the objective is to determine if there is any possible way to modify the existing adaptive controller structure to guarantee stability of the sampled-data system using Lyapunov's second method.

structure and corresponding Lyapunov function originally proposed by Slotine and Li cannot be used to obtain a provably stable sampled-data system. This suggests that the alternative approach to modification of the adaptive algorithm is called for: the addition of a term to the control law to ensure $\Delta V_k < 0$.

4.3 Attempts at Robust Compensation for the Discretized Controller

As discussed previously in Section 3.1, it is possible to contemplate the use of robust compensation techniques to compensate the discretized robot controller. By adding a robust compensation term to the discretized adaptive controller, the effects of controller discretization on the sampled-data system can potentially be offset.

By using the inner/outer loop control with feedback linearization method of Spong and Vidyasagar [56], the discretization of Craig's adaptive controller can be compensated. The addition of a non-linear feedback term to the adaptive controller can be used in the outer loop design to accommodate both the parameter estimation error and the discretization effects. This is the approach briefly described in the following section.

4.3.1 Robust Compensation of Craig's Adaptive Controller

Formulation of the Robust Discretized Adaptive Law

The discretized adaptive control law with the robust compensator added is:

$$\tau[k] = \hat{M}(y[k])v[k] + \hat{Q}(y[k], \dot{y}[k]), \quad (4.64)$$

where

$$v[k] = \ddot{y}_d[k] - K\bar{e}[k] + \delta v[k], \quad (4.65)$$

$$\bar{e} = \begin{bmatrix} e \\ \dot{e} \end{bmatrix}, \quad (4.66)$$

$$e = y - y_d. \quad (4.67)$$

$$K = [K_p \ K_v], \quad (4.68)$$

and $\delta v[k]$ is the sampled version of the robust compensator $\delta v(t)$.

To formulate the robust compensation term, estimates of the uncertainties due to both the discretization errors and the parameter estimation errors must be calculated. Define Δv as the discretization error for $\delta v(t)$:

$$\Delta v = H\{\delta v[k]\} - \delta v(t). \quad (4.69)$$

Similarly,

$$\Delta Q = H\{\hat{Q}(y[k], \dot{y}[k])\} - Q(y(t), \dot{y}(t)). \quad (4.70)$$

Also, define

$$E^* = M^{-1}(y(t))H\{\hat{M}(y[k])\} - I. \quad (4.71)$$

The following assumptions about the bounds on the uncertainties can be made:

1. $\sup_{t \geq 0} \|\tilde{y}_d\| < Q_1 < \infty$, i.e., $\tilde{y}_d$ is uniformly bounded;
2. $\|E^*\| \leq \alpha < 1$, for some α and for all y ;
3. $\|\Delta Q\| \leq \phi(\bar{e}, t)$ for a known function ϕ , bounded in t ;
4. $\|M^{-1}(y(t))\|_2 \leq \bar{M}$ where $\bar{M}$ is a known positive constant (since $M(y)$ is uniformly positive definite and symmetric for all y);
5. $\|\Delta v\| \leq \mathcal{X}$, for some known function $\mathcal{X}$.

The condition

$$\|\delta v(t)\| < \rho(\bar{e}, t) \quad (4.72)$$

gives the uncertainty bound:

$$\rho(\bar{e}, t) = \frac{1}{1 - \alpha} \{ \alpha Q_1 + \alpha \|K \bar{e}\| + \bar{M} \phi(\bar{e}, t) + \mathcal{X} \}. \quad (4.73)$$

Define:

$$\bar{A} = A^* - B^* K, \quad (4.74)$$

where

$$A^* = \begin{bmatrix} 0 & I \\ 0 & 0 \end{bmatrix}, \quad (4.75)$$

and

$$B^* = \begin{bmatrix} 0 \\ I \end{bmatrix}. \quad (4.76)$$

The robust compensation term can now be defined as:

$$\delta v(t) = \begin{cases} -\rho(\bar{e}, t) \frac{B^T \mathcal{P} \bar{e}}{\|B^T \mathcal{P} \bar{e}\|} & \text{if } \|B^T \mathcal{P} \bar{e}\| \neq 0 \\ 0 & \text{if } \|B^T \mathcal{P} \bar{e}\| = 0 \end{cases}, \quad (4.77)$$

where $\mathcal{P}$ is a positive definite symmetric solution to the Lyapunov equation:

$$\bar{A}^T \mathcal{P} + \mathcal{P} \bar{A} = -Q, \quad (4.78)$$

and Q is also positive definite and symmetric.

A continuous boundary layer version of the compensator can be formulated to avoid chattering:

$$\delta v(t) = \begin{cases} -\rho(\bar{e}, t) \frac{B^T \mathcal{P} \bar{e}}{\|B^T \mathcal{P} \bar{e}\|} & \text{if } \|B^T \mathcal{P} \bar{e}\| \geq \epsilon \\ \frac{-\rho(\bar{e}, t)}{\epsilon} B^T \mathcal{P} \bar{e} & \text{if } \|B^T \mathcal{P} \bar{e}\| < \epsilon \end{cases}, \quad (4.79)$$

where ϵ is a positive scalar boundary layer thickness. This compensator gives error dynamics which are *uniformly ultimately bounded* (see Spong and Vidyasagar [56]).

Simulation Results

Computer simulations of the sampled-data system consisting of the two DOF serial link manipulator and Craig's adaptive algorithm were performed. The baseline adaptive case, originally presented in Section 3.4 for Slotine and Li's controller, was simulated ($f_{\text{sampling}} = 100$ Hz), and equivalent adaptation and feedback gains were selected to permit a valid comparison.

A comparison between the norm of the position tracking errors, $\|e\|$, for the Craig versus Slotine and Li controllers is shown in Figure 4.26. Parameter estimation errors for Craig's adaptive controller are shown in Figure 4.27. The tracking performance

Figure 4.26: Norm of Position Tracking Errors, $\|e\|$ (- Craig, - - - Slotine and Li)

Figure 4.27: Parameter Estimation Errors, Craig's Adaptive Controller

for Craig's controller is much worse than for Slotine and Li's ($\|e\|$ for Slotine and Li's controller is very close to zero on the graph compared to Craig's), due to the slower rate of parameter convergence. From a practical standpoint, the need to invert the estimated mass matrix and the need for measurements or estimates of joint accelerations are also significant disadvantages of Craig's method compared to Slotine and Li's.

Simulations of the implementation of Craig's adaptive controller with the robust compensator added were also performed. The compensator using the continuous boundary layer modification was used in an attempt to compensate for the errors

Figure 4.28: Norm of Position Tracking Errors, $\|e\|$, Craig's Controller (- - - small initial param. error, — robust compensator added, ···· large initial param. error)

introduced by the discretization of the control law. Because of the difficulty in calculating the bounds on all the uncertainties, only the uncertainties due to the effect of the ZOH were included. For simulations with a small initial parameter estimation error, the robust compensator was successful in reducing the tracking errors. Figure 4.28 is a comparison of $\|e\|$ for three different cases:

- adaptive control with small initial parameter estimation error;
- adaptive control with small initial parameter estimation error with robust compensation term added;
- adaptive control with large initial parameter estimation error (same as in Figure 4.26).

However, for large initial parameter estimation errors (e.g. initial parameter estimates set to zero) with the robust compensator added, the simulations showed apparently unstable behaviour. The failure of the robust compensator to maintain the stability may be attributed to: the omission of some of the uncertainties in the calculation of the uncertainty bound $\rho(\bar{e}, t)$; or, the violation of an assumption required for the formulation of Spong and Vidyasagar's robust compensation method due to the discrete-time parameter update (see the discussion in the next section).

4.3.2 Robust Compensation of Slotine and Li's Adaptive Controller

Slotine and Li's adaptive controller is not a strictly inverse dynamics type and does not linearize the plant (the dynamics of the error are not linear); thus, Spong and Vidyasagar's robust control approach is not applicable. While the error dynamics can be linearized around the desired trajectory, this form of approximation is still not successful in achieving the inverse dynamics control form necessary to be able to apply Spong and Vidyasagar's robust control method.

Instead of attempting to apply Spong and Vidyasagar's version of Corless and Leitmann's robust controller, Corless and Leitmann's robust saturation control method [9] can be applied directly to the discretization of Slotine and Li's adaptive algorithm. However, simulations have shown that this method of robust compensation is unsuccessful (the simulations showed apparently unstable behaviour). Unfortunately, for the sampled-data system, one of the assumptions of Corless and Leitmann's method is violated due to the discrete-time parameter estimate update: the uncertainty term is not "Caratheodory", because it is neither continuous in time, nor Lebesgue measurable.

It is possible to consider a robust adaptive law which inhibits the robust compensation when the discrete-time parameter adaptation is occurring, to avoid violating the assumptions for the uncertainty upon which Corless and Leitmann's method is based. However, it is unclear whether this approach would lead to a practical and stable controller for the sampled-data case, and has not been pursued further by the author.

An alternative approach for robust compensation of Slotine and Li's adaptive controller was proposed by Slotine and Li [51], whereby a sliding mode control term is added to the existing adaptive feedforward and feedback terms. This sliding term is intended to compensate for uncertainties on the parameter estimates as well as disturbance torques. The modification of this technique to compensate for discretization effects for the sampled-data implementation of the controller has not been explored; there has been relatively little research on discrete-time sliding mode

control. Some recent analyses of discrete-time sliding mode control have been published by Sarpturk et al [46], Drakunov and Utkin [14], and Sira-Ramirez [50], but none of these works cover the non-linear MIMO case of interest here (the first two references are for linear systems, and the last one is for non-linear but SISO systems). Another difficulty with the sliding mode technique is that even in continuous-time the error bounds can be large, especially when a saturation function is used instead of a sign function to smooth out the switching of the control law.

4.4 A Stability Condition using Bounds

The approaches used so far to evaluate the stability of the sampled-data adaptive system have all presented difficulties. In this section, a revised approach using provable bounds on the system dynamics is proposed. The derivation of the bounds makes use of the known properties of rigid-link manipulator dynamics (see for example Ortega and Spong [38], Craig [11]).

As mentioned in Section 4.2.3 above, it is not possible to exactly calculate the stability condition ΔV_k for the general multi-DOF manipulator case. What is then required is an alternative method to evaluate the expression for ΔV_k in a way which gives a useful stability condition, without resorting to the restrictive assumptions stated in Section 4.2.2 which limit the applicability of the results. It will be shown in the development which follows that making use of known bounds of the integral terms appearing in ΔV_k will permit the derivation of a sufficient condition for stability of the sampled-data multi-DOF manipulator case. This will also permit a prediction of the bound on the norm of the filtered tracking error, as discussed previously.

The integral terms of ΔV_k as given by (4.16) can be bounded by setting bounds on the signals and dynamic terms which compose them. For example, the norm of the mass matrix must be bounded by the set of joint positions and inertial constants which are physically possible for the manipulator of interest. As we will see later, this method of using the bounds to derive ΔV_k necessarily results in a very conservative prediction of the stability bounds of the system. However, with appropriate choices

for the bounds, the resulting stability condition is sufficient to guarantee stability, while it is not a necessary condition for stability.

The first integral of ΔV_k given by (4.16) is:

$$\sigma_1 = \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \quad (4.80)$$

Define a new variable, $s'[k]$, such that:

$$\sigma_1 = \Delta t(s^T[k] + s'[k]) = \Delta t s^T[k] + \Delta t s'[k]. \quad (4.81)$$

The transposed vector $s'[k]$ can be interpreted as the correction which when combined with the approximation $\Delta t s^T[k]$ results exactly in the value of the integral σ_1 .

Rearranging and substituting for σ_1 , we get:

$$s'[k] = \frac{1}{\Delta t} \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt - s^T[k] \quad (4.82)$$

where the first term on the right hand side represents the mean of $s^T(t)$ over the sampling interval from t_k to t_{k+1} .

The maximum possible value of $\|s'[k]\|$ occurs when $\|s(t)\|$ has its maximum possible slope during the entire sampling interval, denoted as $\|\dot{s}(t)\|_{max}$. For this limiting case, we get:

$$\|s'[k]\|_{max} = \frac{\|\Delta s_{max}\|}{2} \quad (4.83)$$

where $\|\Delta s_{max}\| = \|\dot{s}(t)\|_{max} \Delta t$. The same result can be derived using the Mean Value Theorem for integrals.

The first term of ΔV_k is:

$$V_1 = -\sigma_1 K_D s[k] = -\Delta t (s^T[k] + s'[k]) K_D s[k] = \Delta t \{-s^T[k] K_D s[k] - s'[k] K_D s[k]\}. \quad (4.84)$$

V_1 can be split into two terms:

$$V_1^A = -\Delta t s^T[k] K_D s[k] \quad (4.85)$$

$$V_1^B = -\Delta t s'[k] K_D s[k]. \quad (4.86)$$

Since K_D is positive definite, V_1^A is a negative definite function of $s[k]$, while V_1^B is indefinite. Thus, taken as a whole, the contribution of V_1 to ΔV_k is indefinite, i.e., it

is unknown whether V_1 will drive ΔV_k more or less negative, and hence enhance or degrade the system stability. Since V_1 is derived from the linear feedback part of the control law, it is due to the discretization effect that this term does not necessarily contribute to the stability.

Until now, the adaptation gain Γ^{-1} has not appeared in the expression for ΔV_k ; this is problematic, since we know that raising Γ^{-1} eventually results in instability. To get Γ^{-1} to appear explicitly, we can start by rewriting the term V_4 from (4.16):

$$V_4 = -\Delta t(\hat{\theta}[k] - \theta)^T W_r^T[k]s[k]. \quad (4.87)$$

By substituting in the discrete adaptation law (4.5), we get

$$V_4 = -\Delta t\{\hat{\theta}^T[k-1] - (\Delta t\Gamma^{-1}W_r^T[k]s[k])^T - \theta^T\}W_r^T[k]s[k], \quad (4.88)$$

which can be rearranged (assuming Γ is symmetrical) to yield:

$$V_4 = -\Delta t s^T[k]W_r[k]\hat{\theta}[k-1] + (\Delta t)^2(W_r^T[k]s[k])^T\Gamma^{-1}(W_r^T[k]s[k]). \quad (4.89)$$

The first term on the right hand side, V_4^A , is indefinite, whereas the second term, V_4^B , is positive definite since Γ^{-1} is positive definite. This result for V_4 is reasonable, since the larger the adaptation gain Γ^{-1} (and the larger the sampling time Δt), the more likely that the V_4^B term will drive ΔV_k positive and hence lose the condition necessary for the proof of stability.

Let us express ΔV_k as the sum of terms:

$$\Delta V_k = V_1^A + V_1^B + V_2 + V_3 + V_5 + V_6 + V_7 + V_8 + V_4^A + V_4^B \quad (4.90)$$

corresponding to the terms given in equation (4.16). To guarantee stability, it must be shown that $\Delta V_k \leq 0$. While it is not possible to evaluate ΔV_k exactly in general, it is possible to bound the magnitudes of its terms, $V_{1...8}$.

The other terms of (4.90), referring back to (4.16), are as follows:

$$V_2 = \sigma_1 \hat{M}[k]\tilde{y}_r[k] = \Delta t(s^T[k] + s'[k])\hat{M}[k]\tilde{y}_r[k] \quad (4.91)$$

$$V_3 = -\left\{\int_{t_k^-}^{t_{k+1}^-} s^T(t)M(t)\tilde{y}_r(t)dt\right\} \quad (4.92)$$

$$V_5 = \sigma_1 \hat{C}[k] \dot{y}_r[k] = \Delta t (s^T[k] + s'[k]) \hat{C}[k] \dot{y}_r[k] \quad (4.93)$$

$$V_6 = -\left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) C(t) \dot{y}_r(t) dt \right\} \quad (4.94)$$

$$V_7 = \sigma_1 \hat{G}[k] = \Delta t (s^T[k] + s'[k]) G[k] \quad (4.95)$$

$$V_8 = -\left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) G(t) dt \right\}. \quad (4.96)$$

The only term of (4.90) that is known to be negative definite is V_1^A , therefore it is sufficient (but not necessary) to satisfy the stability condition by showing that:

$$|V_1^A| \geq |V_1^B| + |V_2| + |V_3| + |V_5| + |V_6| + |V_7| + |V_8| + |V_4^A| + |V_4^B|. \quad (4.97)$$

To evaluate this inequality, the magnitudes of the terms are needed; these magnitudes cannot be calculated in general, but their maximums can be evaluated, thus in theory giving a conservative value for $|V_1^A|$ which is sufficient for stability. However, it can be easily shown that the adaptive controller cannot always satisfy this condition, no matter what choice of controller gain K_D is selected. Thus, the discretization of Slotine and Li's adaptive controller results in a sampled-data system for which stability has not been proven.

Chapter 5

Compensation of the Adaptive Controller

It is desirable to derive an adaptive law which ensures that stability is guaranteed for the sampled-data implementation of the robot control, in the present case for Slotine and Li's algorithm. Two possible methods for achieving this goal are:

1. use some rule to modify the heretofore constant feedback and adaptation gains, K_D and Γ^{-1} , over time;
2. add an additional term to the control law.

The intention of the first method is to preserve the original controller structure of Slotine and Li's adaptive algorithm, while using Lyapunov stability theory to pick appropriate gains for which stability of the sampled-data system can be proven. However, this method for modifying the adaptive control algorithm was not successful in guaranteeing the stability, as described previously in Section 4.2.4.

To achieve guaranteed stability, the addition of a new term to Slotine and Li's controller was also tried. This was first attempted using a robust compensation framework as described in Section 4.3, without success. A new approach using a discrete-time non-linear compensation term added to the existing discretized adaptive control law is described in the following. This proposed compensator is successful in obtaining guaranteed stability for the sampled-data adaptive system.

5.1 Addition of a Discrete-Time Compensator for Guaranteed Stability

Let us return to the discretized adaptive control law, and add a new compensating term, $f[k]$, which will be formulated in a way which will lead to guaranteed stability of the sampled-data system. The stability condition derived from the Lyapunov function will be used to directly formulate the compensator.

The discrete-time compensated adaptive control law is Slotine and Li's algorithm (4.4), with the compensator added:

$$\tau[k] = \hat{M}(y[k])\ddot{y}_r[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}(y[k]) - K_D s[k] + f[k]. \quad (5.1)$$

The discrete-time parameter adaptation law remains unchanged (4.5):

$$\hat{\theta}[k] = \hat{\theta}[k-1] - \Delta t \Gamma^{-1} W_r^T(y[k], \dot{y}[k], \ddot{y}_r[k]) s[k]. \quad (5.2)$$

Following the method described in Section 4.2.1, the expression for ΔV_k is the same as before (4.16), but with an additional term due to the added compensator:

$$\begin{aligned} \Delta V_k = & -\left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} K_D s[k] + \\ & \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} \hat{M}[k] \ddot{y}_r[k] - \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) M(t) \ddot{y}_r(t) dt \right\} + \\ & \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} \hat{C}[k] \dot{y}_r[k] - \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) C(t) \dot{y}_r(t) dt \right\} + \\ & \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} \hat{G}[k] - \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) G(t) dt \right\} - \\ & \Delta t (\hat{\theta}[k] - \theta)^T W_r^T[k] s[k] + \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} f[k]. \end{aligned} \quad (5.3)$$

The objective is now to select $f[k]$ so that the final term of ΔV_k as given by (5.3),

$$V_9 = \left\{ \int_{t_k^-}^{t_{k+1}^-} s^T(t) dt \right\} f[k] = \Delta t (s^T[k] + s'[k]) f[k], \quad (5.4)$$

is forced to be negative definite. Furthermore, $f[k]$ must be chosen to ensure that V_9 becomes so negative that ΔV_k as a whole becomes negative. In other words, by

Figure 5.1: Vector Representation of Filtered Error, s

choosing $f[k]$ to make V_9 as negative as possible, the other terms of ΔV_k which may be positive will be compensated, and ΔV_k will be driven negative.

Assume that $f[k]$ can be chosen as a function of $s[k]$ which will make V_9 negative definite. If $f[k]$ is in the opposite direction in n -space as the vector $s[k]$, then if either of the following conditions is satisfied, V_9 will be a negative definite function of $s[k]$:

1. the projection of $s'[k]$ on $s[k]$ is positive; or,
2. the maximum possible magnitude of $s'[k]$ is smaller than that of $s[k]$.

This concept is illustrated for the 2-dimensional case in Figure 5.1.

Thus, if $\|s[k]\| > \|s'[k]\|_{max}$, then $V_9 < 0$. Since the angle between the $s[k]$ and $s'[k]$ vectors is unknown, condition 1 above cannot be guaranteed; therefore the compensator must be designed to satisfy condition 2. This gives a limiting factor for how well the compensation can operate, since to be of benefit the compensator must only be active when $\|s[k]\| > \|s'[k]\|_{max}$.

Recall that

$$\|s'[k]\|_{max} = \frac{\|\Delta s_{max}\|}{2} \quad (5.5)$$

and $\|\Delta s_{max}\| = \|\dot{s}(t)\|_{max} \Delta t$. Thus, using the compensator we can only hope to guarantee stability when $\|s[k]\| \geq \|\dot{s}(t)\|_{max} \Delta t / 2$. Hence, the effectiveness of the

compensator is related to the maximum change in the filtered tracking error between sampling times (which is turn related to the robot dynamics, torque limits, desired trajectory, etc.). When the sampling rate is relatively high, Δt is small and the compensation will be active unless the filtered error is small i.e. the compensator will act to drive $\|s[k]\|$ to within the bound given by $\|s'[k]\|_{max}$.

A dead-zone switching structure can be defined for the compensator $f[k]$, as follows:

$$f[k] = \begin{cases} -f^* s[k] & \text{for } \|s[k]\| > \|s'[k]\|_{max} \\ 0 & \text{otherwise} \end{cases} \quad (5.6)$$

where f^* is a positive scalar, and $\|s'[k]\|_{max}$ is the (constant) bound calculated for the system. This structure defines a family of compensators which can employ various forms for f^* . This type of compensator is in some ways similar to the robust compensation term of Corless and Leitmann. The compensator will ensure that $V_9 \leq 0$, and for $\|s[k]\| > \|s'[k]\|_{max}$ and sufficiently large gain f^* , will also compensate for the other terms of ΔV_k as given by (5.3). This will drive ΔV_k negative and hence guarantee the stability of the sampled-data system, providing the following assumptions hold:

Assumption 1 *The manipulator stays within its operational limits on joint position and velocity.*

Assumption 2 *Torque saturation does not occur.*

Assumption 3 *The desired trajectory (including the first and second derivatives) is bounded.*

The stability result can now be formally stated as the following theorem:

Theorem 5 *For a robot manipulator with dynamics given by (2.31), the compensated discretized adaptive control algorithm consisting of the control law (5.1) and adaptive law (5.2) results in a sampled-data system which is globally stable, when the compensator has a dead-zone switching structure according to (5.6) and sufficiently large gain f^* , and assumptions 1 to 3 above are met.*

Proof: For sufficiently large gain f^* , ΔV_k for the system as given by (5.3) must become negative for $\|s[k]\| > \|s'[k]\|_{max}$ and hence global stability is proven by Lyapunov's second method. $\square$

An example of a compensator belonging to the family defined by (5.6) above is the non-linear compensator using

$$f^* = a(\|s[k]\| - \|s'[k]\|_{max}) \quad (5.7)$$

where a is a positive scalar constant. The definition of this compensator becomes:

$$f[k] = \begin{cases} -a(\|s[k]\| - \|s'[k]\|_{max})s[k] & \text{for } \|s[k]\| > \|s'[k]\|_{max} \\ 0 & \text{otherwise} \end{cases} \quad (5.8)$$

Outside of the dead-zone defined by $\|s'[k]\|_{max}$, the compensator contributes a positive feedback term and a negative non-linear (quadratic) feedback term to the discretized controller, as follows:

$$u_{feedback} = -K_D s[k] + f[k] = (-K_D + a\|s'[k]\|_{max}I)s[k] - a\|s[k]\|s[k]. \quad (5.9)$$

The compensator reduces the negative feedback in the first term on the right hand side, and introduces the quadratic in the second term. Thus, as the filtered error increases, the compensator's non-linear term becomes more effective. The choice of this member of the compensator family is somewhat arbitrary, but reasonable: the effective gain of the compensator, f^* , is zero at the boundary defined by $\|s'[k]\|_{max}$ and ramps up linearly from there, which avoids a step change in feedback gain which could cause chattering.

If the compensator gain f^* is chosen simply as a constant, the effect on the compensated controller is simply an increase in the effective linear feedback, outside of dead zone. The new linear feedback gain would be $K_D + f^*I$.

With the addition of the terms due to the compensator, express ΔV_k as the sum of terms:

$$\Delta V_k = V_1^A + V_1^B + V_2 + V_3 + V_5 + V_6 + V_7 + V_8 + V_4^A + V_4^B + V_9^A + V_9^B, \quad (5.10)$$

where

$$V_9^A = \begin{cases} -(\Delta t)a(\|s[k]\| - \|s'[k]\|_{max})\|s[k]\|^2 & \text{for } \|s[k]\| > \|s'[k]\|_{max} \\ 0 & \text{otherwise} \end{cases} \quad (5.11)$$

and

$$V_9^B = \begin{cases} -(\Delta t)a(\|s[k]\| - \|s'[k]\|_{\max})s'[k]s[k] & \text{for } \|s[k]\| > \|s'[k]\|_{\max} \\ 0 & \text{otherwise} \end{cases} \quad (5.12)$$

The new term V_9^A , and V_1^A , are both negative definite; V_4^B is positive definite, and the rest of the terms of ΔV_k are indefinite.

5.1.1 The Stability Condition for the Compensated System

To satisfy the stability condition that $\Delta V_k \leq 0$, it is now sufficient to show that:

$$|V_1^A| + |V_9^A| \geq |V_1^B| + |V_2| + |V_3| + |V_5| + |V_6| + |V_7| + |V_8| + |V_4^A| + |V_4^B| + |V_9^B|, \quad (5.13)$$

for the case where $\|s[k]\| > \|s'[k]\|_{\max}$ and hence the compensator is active. When the compensator is inactive, the stability condition reverts to the inequality given previously as (4.97).

It remains to evaluate the terms of (5.13) by determining their maximums, to assess if stability is guaranteed. Referring back to equation (4.85), the magnitude of V_1^A is:

$$|V_1^A| = \Delta t |s^T[k] K_D s[k]|. \quad (5.14)$$

If K_D is assumed to be a diagonal matrix with identical elements, k_D (i.e. $K_D = k_D I^{n \times n}$) then it can be shown that

$$|V_1^A| = \Delta t k_D \|s[k]\|^2. \quad (5.15)$$

By making use of the Cauchy/Schwarz inequality and the maximum possible value of $\|s'[k]\|$ given by (4.83), from (4.86) we get:

$$|V_1^B| = \Delta t |s'[k] K_D s[k]| \leq \Delta t \frac{\|\Delta s_{\max}\|}{2} \|K_D\|_2 \|s[k]\|. \quad (5.16)$$

In the present case, $\|K_D\|_2 = k_D$. In a similar fashion, and using the triangle equality as well, the following can be derived:

$$|V_2| = |\Delta t (s^T[k] + s'[k]) \hat{M}[k] \tilde{y}_r[k]| \leq \Delta t (\|s[k]\| + \frac{\|\Delta s_{\max}\|}{2}) \|\hat{M}[k]\|_2 \|\tilde{y}_r[k]\|, \quad (5.17)$$

$$|V_5| = |\Delta t(s^T[k] + s'[k])\hat{C}[k]\dot{y}_r[k]| \leq \Delta t(\|s[k]\| + \frac{\|\Delta s_{max}\|}{2})\|\hat{C}[k]\|_2\|\dot{y}_r[k]\|, \quad (5.18)$$

$$|V_7| = |\Delta t(s^T[k] + s'[k])\hat{G}[k]| \leq \Delta t(\|s[k]\| + \frac{\|\Delta s_{max}\|}{2})\|\hat{G}[k]\|. \quad (5.19)$$

To calculate the terms $|V_3|$, $|V_6|$ and $|V_8|$, it is necessary to find new bounds for the integrals composing these terms. For example:

$$|V_3| = |\int_{t_k^-}^{t_{k+1}^-} s^T(t)M(t)\tilde{y}_r(t)dt| \leq \Delta t\|s(t)\|_{max}\|M(t)\|_{max}\|\tilde{y}_r(t)\|_{max} \quad (5.20)$$

where

$$\|s(t)\|_{max} = \|s[k]\| + \|\Delta s_{max}\| \quad (5.21)$$

$$\|M(t)\|_{max} = \sup_t \|M(t)\|_2 \quad (5.22)$$

$$\|\tilde{y}_r(t)\|_{max} = \sup_t \|\tilde{y}_r(t)\| \quad (5.23)$$

The bound $\|M(t)\|_{max}$ is interpreted as the maximum value that $\|M(t)\|_2$ can attain over all t ; $\|\tilde{y}_r(t)\|_{max}$ is interpreted similarly. Thus, the bound for $|V_3|$ given by the right hand side of (5.20) is inclusive, but possibly very conservative.

Similarly,

$$|V_6| = |\int_{t_k^-}^{t_{k+1}^-} s^T(t)C(t)\dot{y}_r(t)dt| \leq \Delta t\|s(t)\|_{max}\|C(t)\|_{max}\|\dot{y}_r(t)\|_{max} \quad (5.24)$$

$$|V_8| = |\int_{t_k^-}^{t_{k+1}^-} s^T(t)G(t)dt| \leq \Delta t\|s(t)\|_{max}\|G(t)\|_{max}, \quad (5.25)$$

where

$$\|C(t)\|_{max} = \sup_t \|C(t)\|_2 \quad (5.26)$$

$$\|G(t)\|_{max} = \sup_t \|G(t)\| \quad (5.27)$$

$$\|\dot{y}_r(t)\|_{max} = \sup_t \|\dot{y}_r(t)\|. \quad (5.28)$$

From (4.89), $|V_4^A|$ can be calculated as:

$$|V_4^A| = |\Delta t s^T[k] W_r[k] \dot{\theta}[k-1]| \leq \Delta t \|s[k]\| \|W_r[k] \dot{\theta}[k-1]\| \quad (5.29)$$

$$\begin{aligned} &\leq \Delta t \|s[k]\| \{ (\|\hat{M}^*[k]\|_2 + \|M[k]\|_2) \|\ddot{y}_r[k]\| + (\|\hat{C}^*[k]\|_2 + \|C^*[k]\|_2) \|\dot{y}_r[k]\| \\ &\quad + (\|\hat{G}^*[k]\| + \|G[k]\|) \} \end{aligned} \quad (5.30)$$

where the terms $\hat{M}^*[k]$, $\hat{C}^*[k]$ and $\hat{G}^*[k]$ make use of the parameter estimates from the last time step, $\hat{\theta}[k-1]$, but the states from the current time step ($y[k]$, $\dot{y}[k]$).

It can also be shown that

$$\begin{aligned} |V_4^B| &= |(\Delta t)^2 (W_r^T[k] s[k])^T \Gamma^{-1} (W_r^T[k] s[k])| \\ &\leq (\Delta t)^2 \|W_r^T[k] s[k]\|^2 \|\Gamma^{-1}\|_2 \\ &\leq (\Delta t)^2 \gamma^{-1} (\|W_r^T[k]\|_2 \|s[k]\|)^2, \end{aligned} \quad (5.31)$$

if it is assumed that Γ^{-1} is a diagonal matrix with identical elements (i.e. $\Gamma^{-1} = \gamma^{-1} I^{p \times p}$).

The bounds for the terms of ΔV_k arising from the compensator defined by (5.8) are derived directly from (5.11) and (5.12):

$$|V_9^A| = \begin{cases} a \Delta t (\|s[k]\| - \frac{\|\Delta s_{max}\|}{2}) \|s[k]\|^2 & \text{for } \|s[k]\| > \frac{\|\Delta s_{max}\|}{2} \\ 0 & \text{otherwise} \end{cases} \quad (5.32)$$

and

$$|V_9^B| \leq \begin{cases} a \Delta t (\|s[k]\| - \frac{\|\Delta s_{max}\|}{2}) \frac{\|\Delta s_{max}\|}{2} \|s[k]\| & \text{for } \|s[k]\| > \frac{\|\Delta s_{max}\|}{2} \\ 0 & \text{otherwise} \end{cases} \quad (5.33)$$

The stability condition (5.13) can be rewritten as:

$$|V_1^A| + |V_9^A| - |V_1^B| - |V_2| - |V_3| - |V_5| - |V_6| - |V_7| - |V_8| - |V_4^A| - |V_4^B| - |V_9^B| \geq 0. \quad (5.34)$$

To be able to evaluate the global stability condition which makes use of the bounds calculated above, it is necessary to define global bounds for the norms that appear, such as $\|M(t)\|_{max}$, $\|M[k]\|_{max}$, etc. It is reasonable to assume that the maximum of the norm for the continuous-time terms (e.g. $\|M(t)\|_{max}$) is equal to the corresponding maximum for the sampled version of that term (e.g. $\|M[k]\|_{max}$).

Furthermore, the maximum for a given term employing the current parameter estimates is equal to the maximum for this term using parameter estimates from the previous time step (e.g. $\|\hat{M}[k]\|_{\max} = \|\hat{M}^*[k]\|_{\max}$), since the bounds for the parameter estimates are invariant over time. Therefore, the overall bounds for the norms can be defined as follows:

$$\|M(t)\|_{\max} = \|M[k]\|_{\max} = \|M\|_{\max} \equiv \sup_{|y| \leq y_{\lim}} \|M(y)\|_2 \quad (5.35)$$

$$\|C(t)\|_{\max} = \|C[k]\|_{\max} = \|C\|_{\max} \equiv \sup_{\substack{|y| \leq y_{\lim} \\ |\dot{y}| \leq \dot{y}_{\lim}}} \|C(y, \dot{y})\|_2 \quad (5.36)$$

$$\|G(t)\|_{\max} = \|G[k]\|_{\max} = \|G\|_{\max} \equiv \sup_{|y| \leq y_{\lim}} \|G(y)\| \quad (5.37)$$

$$\|\dot{y}_r(t)\|_{\max} = \|\dot{y}_r[k]\|_{\max} = \|\dot{y}_r\|_{\max} \equiv \|\dot{y}_d\|_{\max} + \|\Lambda\|_2(\|y_{\lim}\| + \|y_d\|_{\max}) \quad (5.38)$$

$$\|\tilde{y}_r(t)\|_{\max} = \|\tilde{y}_r[k]\|_{\max} = \|\tilde{y}_r\|_{\max} \equiv \|\tilde{y}_d\|_{\max} + \|\Lambda\|_2(\|\dot{y}_{\lim}\| + \|\dot{y}_d\|_{\max}) \quad (5.39)$$

$$\begin{aligned} \|W_r[k]\|_{\max} = \|W_r\|_{\max} \equiv & \sup_{\substack{|y| \leq y_{\lim} \\ |\dot{y}| \leq \dot{y}_{\lim} \\ \|\dot{y}_r\| \leq \|\dot{y}_r\|_{\max} \\ \|\tilde{y}_r\| \leq \|\tilde{y}_r\|_{\max}}} \|W(y, \dot{y}, \dot{y}_r, \tilde{y}_r)\|_2. \end{aligned} \quad (5.40)$$

$\|y_d\|_{\max}$, $\|\dot{y}_d\|_{\max}$ and $\|\tilde{y}_d\|_{\max}$ are easily calculated from the desired trajectory. $y_{\lim}$ is the vector of operational limits on the possible joint angular positions, based on the physical limitations of the manipulator (assumed to be symmetrical about the defined zero angular positions). Similarly, $\dot{y}_{\lim}$ is the vector of joint velocity limits, which are based on the inertial properties of the manipulator, the maximum possible joint torques, and the maximum possible travel distance as defined by $y_{\lim}$ ¹.

¹Other factors such as friction, motor back-EMF and power supply voltage capability also limit the possible joint velocities in a real manipulator implementation.

Define $\mathcal{B}$ as a positive diagonal matrix which multiplies the true parameter vector to set the known limits on the values of parameter estimates. In other words,

$$\hat{\theta}_{bound} = \mathcal{B}\theta, \quad (5.41)$$

where $\hat{\theta}_{bound}$ is the limit set on the parameter estimate vector. To simplify the analysis, assume that the uncertainty on all the parameters is equivalent (as a percentage of the true parameter values); $\mathcal{B}$ then degenerates to a diagonal matrix with identical non-zero elements. This allows us to simply use a positive scalar, β , instead of the matrix $\mathcal{B}$ to bound the parameter estimates. Since the parameters (in θ) appear linearly in the dynamic terms M, C and G , then the bound for the norms of the x terms using the estimated parameters is simply β times the bound of the norms for the true M, C and G terms. This leads to the following bounds for the norms of the estimated dynamic terms:

$$\|\hat{M}[k]\|_{max} = \|\hat{M}^*[k]\|_{max} = \|\hat{M}\|_{max} \equiv \beta\|M\|_{max} \quad (5.42)$$

$$\|\hat{C}[k]\|_{max} = \|\hat{C}^*[k]\|_{max} = \|\hat{C}\|_{max} \equiv \beta\|C\|_{max} \quad (5.43)$$

$$\|\hat{G}[k]\|_{max} = \|\hat{G}^*[k]\|_{max} = \|\hat{G}\|_{max} \equiv \beta\|G\|_{max} \quad (5.44)$$

By substituting in for the magnitudes as derived above, using the bounds given in (5.35) to (5.40), and collecting terms, the following condition is obtained from (5.34), for the case when $\|s[k]\| > \frac{\|\Delta s_{max}\|}{2}$:

$$\begin{aligned} & a\|s[k]\|^3 + \{k_D - a\|\Delta s_{max}\| - \Delta t\gamma^{-1}\|W_r\|_{max}^2\}\|s[k]\|^2 + \left\{-\frac{\|\Delta s_{max}\|}{2}k_D \right. \\ & - 2(\beta + 1)(\|M\|_{max}\|\ddot{y}_r\|_{max} + \|C\|_{max}\|\dot{y}_r\|_{max} + \|G\|_{max}) + a\left(\frac{\|\Delta s_{max}\|}{2}\right)^2\}\|s[k]\| \\ & \left. - \|\Delta s_{max}\|\left(\frac{\beta}{2} + 1\right)(\|M\|_{max}\|\ddot{y}_r\|_{max} + \|C\|_{max}\|\dot{y}_r\|_{max} + \|G\|_{max}) \geq 0. \right. \end{aligned} \quad (5.45)$$

The above inequality is the stability condition for the sampled-data system using the discretized version of Slotine and Li's adaptive algorithm and the compensator given by equation (5.8). By setting the parameter a (the compensator gain) to zero, (5.45) also applies to the uncompensated case, and for the case when $\|s[k]\| \leq \frac{\|\Delta s_{max}\|}{2}$. This stability condition can also be used for the non-adaptive, uncompensated case

Figure 5.2: Cubic Function for the Stability Condition ($\circ$ = roots, $\bullet$ = $\|s[k]\|_{\text{bound}}$)

(fixed parameter feedforward, or only PD) by dropping the terms (arising from V_2, V_5, V_7, V_4, V_9) that are not applicable.

The underlying equality represented in (5.45) is a cubic in terms of $\|s[k]\|$. By finding the roots of this equality, the stability as a function of $\|s[k]\|$ can be predicted. The largest positive root is interpreted as the stability bound, $\|s[k]\|_{\text{bound}}$, and the system error will be driven to within this bound. A typical cubic function for the stability condition is illustrated in Figure 5.2.

The stability properties for the compensated sampled-data system can be formally stated in the following theorem:

Theorem 6 *For a robot manipulator with dynamics given by (2.31), the compensated discretized adaptive control algorithm consisting of the control law (5.1), adaptive law (5.2) and compensator (5.8) results in a sampled-data system which is stable, if assumptions 1 to 3 given previously are met.*

Corollary 1 *If the conditions of Theorem 6 are satisfied, a steady-state bound on $\|s[k]\|$ for the system must exist. The greatest positive real root of the equality derived from (5.45) is this guaranteed bound on $\|s[k]\|$.*

Proof: Inspection of the cubic form of (5.45) proves that a value $\|s[k]\|_{\text{bound}}$ must exist which satisfies the inequality. Since (5.45) is derived from ΔV_k for the system

as given by (5.3), ΔV_k must become negative for $\|s[k]\| > \|s[k]\|_{\text{bound}}$ and hence stability is proven by Lyapunov's second method. $\square$

Recall that for the case without the non-linear compensator, stability cannot in general be guaranteed, and (5.45) can be used to determine for which conditions stability is predicted and what the error bound will be.

5.1.2 The Effect of Torque Saturation

It has been assumed that torque saturation limits exist for the manipulator, which enable the calculation of the bounds on the dynamic behaviour of the system and consequently permit the derivation of the stability conditions and the formulation of the compensator. However, the proven stability properties as stated in Theorems 5 and 6 assume that torque saturation does not actually occur. It can be shown that if torque saturation is occurring, the stability condition given by (5.45) is no longer sufficient to guarantee stability. Thus, the torque limits for the joints impose a restriction on when the proposed discrete-time compensator can be usefully employed.

The torque saturation issue for adaptive controllers has also been addressed by Whitcomb et al [64]. Whitcomb et al performed comparative experiments for a range of different robot adaptive controllers and stated that all of the controllers became unstable when torque saturation occurred. The stability proofs for all the adaptive controllers they tested depend upon instantaneous and unbounded torque capabilities, and hence these proofs of stability break down for practical implementations when torque saturation is present.

The discrete-time compensated adaptive control law was given as (5.1), and is repeated here for convenience:

$$\tau[k] = \hat{M}(y[k])\ddot{y}_r[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}(y[k]) - K_D s[k] + f[k]. \quad (5.46)$$

Define $\tau_{sat} \in \mathbb{R}^n$ as the vector of absolute values of the maximum joint torques (saturation), based on the motor and power supply limitations, and assuming that the torque saturation occurs symmetrically about zero torque. The necessary and

sufficient condition for avoiding torque saturation is:

$$|\tau[k]| \leq \tau_{sat} \text{ for all } k, \quad (5.47)$$

where the notation implies that the absolute value of each element of the vector $\tau[k]$ is less than or equal to the corresponding element of τ_{sat} . Substituting the control law into (5.47), the condition for no torque saturation becomes:

$$|\hat{M}(y[k])\ddot{y}_r[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}(y[k]) - K_D s[k] + f[k]| \leq \tau_{sat} \text{ for all } k. \quad (5.48)$$

A necessary (but not sufficient) condition for avoiding torque saturation arises from the feedforward component of the control law: the desired trajectory must be appropriately selected, so that it is theoretically within the dynamic capability of the manipulator (i.e. it is within the torque capability of the joint actuators). This can be stated as:

$$|M(y_d[k])\ddot{y}_d[k] + C(y_d[k], \dot{y}_d[k])\dot{y}_d[k] + G(y_d[k])| \leq \tau_{sat} \text{ for all } k. \quad (5.49)$$

Theorem 7 *For a robot manipulator with dynamics given by (2.31), and using the compensated discretized adaptive control algorithm consisting of the control law (5.1), adaptive law (5.2) and compensator structure according to (5.6), torque saturation can only be avoided for specific combinations of desired trajectory, bounds on the parameter estimates, feedback and compensator gains, and initial tracking error. The inequality (5.48) is a necessary and sufficient condition for the avoidance of torque saturation.*

Proof: In the absence of torque saturation, the system is stable according to Theorem 5. Therefore, as long as the initial conditions are such that torque saturation does not occur, inspection of the condition given as (5.48) leads to the proof of Theorem 7. $\square$

5.1.3 Two DOF Robot Manipulator Example

Simulation Software

The simulation software was written in Borland International Inc.'s Turbo Pascal (Version 5.0) programming language. Various versions of a program to simulate the

control of a two DOF robotic manipulator were written. The most general of these programs simulates a multi-rate version of Slotine and Li's adaptive controller with the non-linear discrete-time compensator added. By setting the various control and adaptive gains appropriately, as well as the initial parameter estimates, different control cases can be simulated:

- with the servo loop sampling rate and parameter adaptation rate set to be equal, single-rate control is simulated;
- with the compensator gain set to zero, Slotine and Li's adaptive controller without the compensator is simulated;
- additionally, with the adaptive gain set to zero, a fixed parameter feedforward controller is simulated;
- additionally, with the parameter estimates set to zero, a PD controller is simulated.

A 4th order Runge-Kutta algorithm is used to simulate the continuous-time plant model by numerically solving the underlying system of non-linear differential equations. The same Runge-Kutta solver can be used to simulate a continuous-time controller. For the sampled-data simulation, the adaptive controller is simulated as discrete-time law, resulting in an overall hybrid simulation.

Appendix A contains the software listing for the simulation of robot control for the manipulator model described in the following section.

Manipulator Model

The use of the stability condition given by (5.45) is illustrated for the implementation of a 2 DOF serial link robotic manipulator. This manipulator was discussed in Schwartz and Warshaw [47], and in Schwartz et al [48]. The description of the model of this manipulator follows.

The manipulator is assumed to operate in the horizontal plane (i.e. gravity terms are neglected), and its geometry is shown in Figure 5.3. The link masses are

Figure 5.3: Geometry of Serial Link Manipulator

modelled as being evenly distributed along their length, and friction is neglected. The derivation of the equations of motion using Lagrangian mechanics is detailed in Warshaw [61].

Given

$$\tau = \begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} ; \quad y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix},$$

and the parameter vector².

$$\theta = \begin{bmatrix} a_1 = 1/3m_1l_1^2 + m_2l_1^2 + 1/3m_2l_2^2 \\ a_2 = -m_2l_1l_2 \\ a_3 = 1/3m_2l_2^2 \end{bmatrix} \quad (5.50)$$

and where

$$m_1, m_2 = \text{link masses} = 2.0 \text{ kg}$$

$$l_1, l_2 = \text{link lengths} = 0.3 \text{ m},$$

²The serial link manipulator model was originally parametrized using 10 parameters (see [47], [48],[61]), but due to redundancy in the parameters, the number can be collapsed down to 3

then the scalar equations of motion are:

$$\begin{aligned} \tau_1 = & a_1 \ddot{y}_1 + a_2 \cos(y_2) \ddot{y}_1 + a_3 \ddot{y}_2 + 0.5a_2 \cos(y_2) \ddot{y}_2 - \\ & a_2 \sin(y_2) \dot{y}_1 \dot{y}_2 - 0.5a_2 \sin(y_2) \dot{y}_2^2 \end{aligned} \quad (5.51)$$

$$\tau_2 = a_3 \ddot{y}_1 + 0.5a_2 \cos(y_2) \ddot{y}_1 + a_3 \ddot{y}_2 + 0.5a_2 \sin(y_2) \dot{y}_1^2. \quad (5.52)$$

The W matrix for the linear regression form of the equations of motion is:

$$W^T(y, \dot{y}, \ddot{y}) = \begin{bmatrix} \ddot{y}_1 & 0 \\ (\ddot{y}_1 + 0.5\ddot{y}_2) \cos(y_2) - (\dot{y}_1 \dot{y}_2 + 0.5\dot{y}_2^2) \sin(y_2) & 0.5(\ddot{y}_1 \cos(y_2) + \dot{y}_1^2 \sin(y_2)) \\ \ddot{y}_2 & \ddot{y}_1 + \ddot{y}_2 \end{bmatrix}. \quad (5.53)$$

For the matrix form of the equations, the resulting M matrix and Q vector are defined as follows:

$$M(y) = \begin{bmatrix} a_1 + a_2 \cos(y_2) & a_3 + 0.5a_2 \cos(y_2) \\ a_3 + 0.5a_2 \cos(y_2) & a_3 \end{bmatrix} \quad (5.54)$$

$$Q(y, \dot{y}) = \begin{bmatrix} -a_2 \sin(y_2) \dot{y}_1 \dot{y}_2 - 0.5a_2 \sin(y_2) \dot{y}_2^2 \\ 0.5a_2 \sin(y_2) \dot{y}_1^2 \end{bmatrix}. \quad (5.55)$$

Equivalently, the C matrix is:

$$C(y, \dot{y}) = \begin{bmatrix} -0.5a_2 \sin(y_2) \dot{y}_2 & -0.5a_2 \sin(y_2) (\dot{y}_1 + \dot{y}_2) \\ 0.5a_2 \sin(y_2) \dot{y}_1 & 0 \end{bmatrix}. \quad (5.56)$$

This definition of C is consistent with skew-symmetry of the matrix $\dot{M}(y) - 2C(y, \dot{y})$, in accordance with the formulation of Slotine and Li's adaptive controller as discussed in Section 2.3.1.

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.

Evaluation of the Stability Condition

In order to evaluate the stability condition, it is necessary to calculate $\|\Delta s_{max}\|$. This requires calculation of $\|\dot{s}(t)\|_{max}$, which is defined as:

$$\|\dot{s}(t)\|_{max} = \|\ddot{y}\|_{max} + \|\ddot{y}_d\|_{max} + \|\Lambda\|_2(\|\dot{y}_{lim}\| + \|\dot{y}_d\|_{max}) \quad (5.57)$$

To evaluate (5.57), this in turn requires calculation of $\|\ddot{y}\|_{max}$, the bound on the norm of the joint acceleration vector.

By rearranging the equation of dynamics given by (2.31), we get:

$$\ddot{y}(t) = M^{-1}(y(t))(\tau(t) - C(y(t), \dot{y}(t))\dot{y}(t) + G(y(t))). \quad (5.58)$$

Thus, $\|\ddot{y}\|$ is limited as follows:

$$\|\ddot{y}(t)\| \leq \|M^{-1}(y(t))\|(\|\tau(t)\| + \|C(y(t), \dot{y}(t))\dot{y}(t)\| + \|G(y(t))\|). \quad (5.59)$$

The bound can now be defined as:

$$\|\ddot{y}\|_{max} = \sup_{|y| \leq y_{lim}} \left\{ \|M^{-1}(y)\|(\|\tau_{sat}\| + \|C(y, \dot{y}_{lim})\dot{y}_{lim}\| + \|G(y)\|) \right\}. \quad (5.60)$$

Three comments apply to this bound:

1. since M, C and G are not actually known, in practice their estimates would have to be substituted³;
2. $\dot{y}_{lim}$ can be used in (5.60) instead of finding the supremum over all possible $\dot{y}$ as well as y , since it is a known property of the matrix $C(y, \dot{y})$ that it grows linearly with $\dot{y}$ (see Tomei [58]);

³These estimates may be a-priori estimates of the dynamics, and hence are distinct from the estimates employing the on-line estimates of the parameters calculated by the adaptive law and denoted by $\hat{M}, \hat{C}, \hat{G}$.

3. the bound is expected to be very conservative, since it is not possible for $\dot{y}$ to be at $\dot{y}_{lim}$ simultaneously with $y = y_{lim}$, since the velocity must fall to zero at the travel limits.

It is possible to calculate maximum possible joint velocities $\dot{y}_{lim}$ from the equation of dynamics, based on the acceleration limits imposed by τ_{sat} and the limits of travel imposed by y_{lim} . However, since the dynamics are a non-linear and coupled differential equation, this is very difficult to do explicitly. As an approximation for the purposes of calculating $\dot{y}_{lim}$, it can be assumed that the dynamics are linear and uncoupled; thus, the dynamics degenerate to a set of one DOF pure inertias given by the diagonal elements of the mass matrix:

$$\ddot{y}_i = \frac{\tau_i}{m_{ii}} \quad \text{for } i = 1 \text{ to } n, \quad (5.61)$$

where n is the number of joints, and m_{ii} is a diagonal element of M . Define δ_i as the maximum travel distance for the i^{th} joint:

$$\delta_i = y_{i_{max}} - y_{i_{min}} = 2y_{lim,i}. \quad (5.62)$$

For the joint to be able to come to rest before hitting the travel limits, the maximum acceleration can only be applied for one half of the maximum travel distance. The maximum acceleration is greatest for the configuration of the joints which gives the lowest joint inertia:

$$m_{ii_{min}} = \inf_{|y| \leq y_{lim}} (m_{ii}). \quad (5.63)$$

Finally, a conservative estimate of the vector $\dot{y}_{lim}$ can now be calculated by calculating its elements as:

$$\dot{y}_{lim,i} = \sqrt{\frac{\tau_{sat,i} \delta_i / 2}{m_{ii_{min}}}} \quad (5.64)$$

For the two DOF manipulator described previously in this section, the following limits are selected:

$$\tau_{sat} = \begin{bmatrix} 10.0 \\ 10.0 \end{bmatrix} \text{ N-m} \quad (5.65)$$

$$y_{lim} = \begin{bmatrix} 2.8 \\ 2.0 \end{bmatrix} \text{ rad.} \quad (5.66)$$

Using these limits, the manipulator mass matrix and equation (5.64), we get:

$$\dot{y}_{lim} = \begin{bmatrix} 15.3 \\ 18.3 \end{bmatrix} \text{ rad/sec.} \quad (5.67)$$

To evaluate the stability condition (5.45) for the robot manipulator example, the maximum bounds appearing in (5.45) were calculated. This was accomplished using a combination of inspection of the equations of dynamics and numerical calculations (using MATLAB, a numerical math software package from The MathWorks, Inc.) over the range of possible system states. The bounds for the M and C matrices are:

$$\|M\|_{max} = 0.403, \quad (5.68)$$

$$\|C\|_{max} = 3.52, \quad (5.69)$$

and $G = 0$ since the manipulator operates in the horizontal plane and hence gravity terms can be neglected.

The following are calculated from the desired trajectory:

$$\|y_d\|_{max} = 3.12 \text{ rad}, \quad (5.70)$$

$$\|\dot{y}_d\|_{max} = 7.2 \text{ rad/sec}, \quad (5.71)$$

$$\|\ddot{y}_d\|_{max} = 32 \text{ rad/sec}^2. \quad (5.72)$$

The bounds for the reference velocity and acceleration, and the regression matrix are:

$$\|\dot{y}_r\|_{max} = 72.8 \text{ rad/sec}, \quad (5.73)$$

$$\|\ddot{y}_r\|_{max} = 342 \text{ rad/sec}^2, \quad (5.74)$$

$$\|W_r\|_{max} = 1440. \quad (5.75)$$

Evaluating the bound given by equation (5.60),

$$\|\ddot{y}\|_{max} = 2780 \text{ rad/sec}^2, \quad (5.76)$$

and from equation (5.57),

$$\|\dot{s}(t)\|_{max} = 3120 \text{ rad/sec}^2. \quad (5.77)$$

Table 5.1: Evaluation of Stability Condition for 2 DOF Serial Link Manipulator

Case	Description*	a	β	k_D	γ^{-1}	Δt	Real roots for $\ s\ $	Stable
1.	uncomp.	0	2	8	0.2	0.01	none	no
2.	comp.	1	2	8	0.2	0.01	4170	yes
3.	comp.	10	2	8	0.2	0.01	445	yes
4.	comp.	100	2	8	0.2	0.01	69.3	yes
5.	comp.	1000	2	8	0.2	0.01	0.104, 8.99, 26.0	yes
6.	uncomp., higher f_s	0	2	8	0.2	0.001	none	no
7.	comp., higher f_s	100	2	8	0.2	0.001	9.64	yes
8.	uncomp., lower γ^{-1}	0	2	8	0.0002	0.01	-9.67, 655	yes
9.	comp., lower γ^{-1}	100	2	8	0.0002	0.01	1.41, 8.10, 21.5	yes

* uncomp. = without non-linear compensator, comp. = with non-linear compensator

For the baseline sampling rate of 100 Hz, $\Delta t = 0.01$ seconds, and we get:

$$\|\Delta s_{max}\| = \|\dot{s}(t)\|_{max} \Delta t = 31 \text{ rad/sec.} \quad (5.78)$$

The stability for various cases can now be evaluated using the stability condition given by the inequality (5.45). Table 5.1 lists the stability predictions (if stability is guaranteed) for the system with and without the non-linear discrete-time compensator, and for various combinations of feedback, adaptation and compensator gain, bounds on the parameter estimates, and sampling rate.

The stability predictions listed in Table 5.1 can be compared to corresponding simulation results. For example, case #1 corresponds to the baseline adaptive sampled-data simulation presented earlier in Section 3.4. While the stability prediction does not predict guaranteed stability due to the absence of real roots for $\|s\|$, the simulation shows what appears to be stable and well-behaved performance.

Cases #2 to #5 are the compensated version of this baseline run with progressively higher compensator gain; stability is predicted for these cases, which cannot be confirmed by the simulations because the compensation never becomes active. This is because $\|s\|$ never exceeds the dead-zone defined for the compensator as

$\|s'[k]\|_{\max} = \frac{\|\Delta s_{\max}\|}{2} = 15.5 \text{ rad/sec}$ (see (5.6)). In fact, simulation results for the uncompensated case show that $\|s\|$ reaches a maximum of only 0.32 rad/sec (see Section 3.4 data plots). The bound for the filtered error observed from the simulations is in general much lower than the predicted bound on the filtered error, $\|s\|_{\text{bound}}$ (as given by the highest positive root in Table 5.1).

Increasing the sampling rate to 1000 Hz does not give a stability guarantee for the uncompensated case (case #6), but does reduce $\|s\|_{\text{bound}}$ for the compensated case (#7). Decreasing the adaptive gain does give guaranteed stability for the uncompensated case, but with very large $\|s\|_{\text{bound}}$ compared to the corresponding compensated case (see cases #8 and #9).

Some conclusions are as follows:

- the condition (5.45) is conservative because it is derived from a sufficient (but not necessary) condition for stability, and thus can overestimate the bound on the filtered error for the stable cases, and predict no guaranteed stability for some uncompensated cases which exhibit apparently stable performance;
- for cases without the non-linear compensator where stability cannot be guaranteed, addition of the compensator does give a prediction of guaranteed stability.

It is the nature of the (worst-case) bounds calculated for $\|M\|_{\max}$, $\|\dot{y}_r\|_{\max}$, $\|\ddot{y}\|_{\max}$, $\|\Delta s_{\max}\|$ and the other terms of (5.3) that contributes to the conservative character of the stability prediction; these overall bounds greatly exceed the bounds observed from simulations in some cases.

5.2 A Multi-Rate Discrete-Time Adaptive Controller with Compensation

While the single-rate discrete-time adaptive controller with compensation formulated previously in Section 5.1 has the needed stabilizing properties, torque saturation can be problematic in practice. This was discussed previously in Section 5.1.2.

For the two DOF manipulator example covered in Section 5.1.3, $\|\Delta s\|_{\max}$ is large due to the relatively large Δt , and $f[k]$ does not come into play until $\|s\|$ is large due to the dead-zone. This means that for good tracking and hence small $\|s\|$, the compensator never becomes active. When $\|s\|$ is large enough to be outside the compensator dead-zone, torque saturation becomes likely.

A discrete-time controller can be designed with the servo loop (feedback, feedforward, compensation) running at a high sampling rate, and the parameter adaptation running at a low sampling rate. This “multi-rate” controller permits the use of the proposed discrete-time compensator while avoiding the problem discussed above. The multi-rate controller structure improves the performance of the compensator by reducing $\|\Delta s\|_{\max}$ due to a smaller sampling time Δt , and hence allowing the compensator to become active at a lower level of $\|s\|$ due to the smaller dead-zone. In addition, the multi-rate controller provides benefits for implementation in a practical system:

- running the adaptation at a lower sampling rate reduces the computational load;
- the adaptation using regression terms summed over a number of time steps of the servo loop has a filtering effect and can improve the robustness of the controller.

One of the guidelines proposed by Rohrs et al [41] for the design of discrete-time adaptive controllers (to diminish stability problems resulting from the discretization) was to sample the system slowly enough to reduce the effects of unmodeled dynamics on the adaptive law; the multi-rate controller is a means of following this guideline while maintaining a high sampling rate for the control law.

The multi-rate adaptive controller structure is related to the topic of hybrid control (mixed continuous- and discrete-time parts) as discussed in Morando et al [33] and Narendra and Annaswamy [34]. Narendra and Annaswamy discuss two different hybrid control approaches for integrating the continuous parameter adaptation law (given by a derivative of the parameter estimate vector, $\dot{\hat{\theta}}$) over the discrete sampling time:

1. integrate the derivative as a whole;
2. integrate the constituents of the derivative separately, and then combine them;

These hybrid concepts are applicable to the case of interest — the discrete and multi-rate adaptive controller.

Borrowing some notation from Morando et al [33], the multi-rate controller can be formulated by first defining the two sampling times:

$$\Delta t = t_{k+1} - t_k = \text{sampling time for fast (servo) rate,} \quad (5.79)$$

$$\Delta \bar{t} = m\Delta t = \text{sampling time for slow (adaptation) rate,} \quad (5.80)$$

$$m = \text{positive integer.} \quad (5.81)$$

Set

$$\bar{k} = \text{Integer} \left\{ \frac{k}{m} \right\}, \quad (5.82)$$

where the operator $\text{Integer}\{\cdot\}$ rounds down to the nearest integer, so that

$$\bar{k} = \text{index for discrete adaptation,} \quad (5.83)$$

$$k = \text{index for discrete servo loop.} \quad (5.84)$$

Using Narendra and Annaswamy's first hybrid approach, the discrete parameter update becomes:

$$\hat{\theta}[\bar{k}] = \hat{\theta}[\bar{k} - 1] - \int_{t_{\bar{k}-1}}^{t_{\bar{k}}} \dot{\hat{\theta}}(t) dt, \quad (5.85)$$

where

$$t_{\bar{k}} = t \text{ at } k = \bar{k}m, \quad (5.86)$$

$$t_{\bar{k}-1} = t \text{ at } k = (\bar{k} - 1)m, \quad (5.87)$$

and where $\dot{\hat{\theta}}(t)$ was given as equation (2.33):

$$\dot{\hat{\theta}}(t) = -\Gamma^{-1} W_r^T(y(t), \dot{y}(t), \dot{y}_r(t), \tilde{y}_r(t)) s(t). \quad (5.88)$$

Figure 5.4: Multi-Rate Adaptive Control

By using an Euler approximation for the integral in (5.85), the discrete parameter update becomes:

$$\hat{\theta}[\bar{k}] = \hat{\theta}[\bar{k} - 1] - \Delta t \Gamma^{-1} \sum_{i=k-m+1}^k \{W_r^T(y[i], \dot{y}[i], \ddot{y}_r[i], \tilde{y}_r[i])s[i]\}. \quad (5.89)$$

Compare this to the (single-rate) adaptive law given previously as equation (4.5).

The discretized, multi-rate, compensated version of Slotine and Li's algorithm is now:

$$\tau[k] = \hat{M}_k(y[k])\ddot{y}_r[k] + \hat{C}_k(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}_k(y[k]) - K_D s[k] + f[k], \quad (5.90)$$

where the $\bar{k}$ subscripts indicate parameter estimates updated at the lower rate according to (5.89). Figure 5.4 conceptually shows how the control torque and parameter estimates are updated at different rates for the multi-rate controller.

5.2.1 The Stability Condition

The stability condition can now be evaluated for the multi-rate case. For the discrete time steps when $\bar{k} = k/m$, i.e. time steps k for which parameter adaptation takes place, we have as before (5.10):

$$\Delta V_k = V_1^A + V_1^B + V_2 + V_3 + V_5 + V_6 + V_7 + V_8 + V_4^A + V_4^B + V_9^A + V_9^B. \quad (5.91)$$

Defining the vector

$$S_k = \sum_{i=k-m+1}^k \{W_r^T[i]s[i]\}, \quad (5.92)$$

and by using the parameter update law 5.89 (and following the same method as in Section 4.4, but substituting S_k for $W_r^T[k]s[k]$), we get:

$$V_4 = V_4^A + V_4^B = -\Delta t \tilde{\theta}^T[\bar{k}] S_k, \quad (5.93)$$

where

$$V_4^A = -\Delta t S_k^T \tilde{\theta}[\bar{k} - 1], \quad (5.94)$$

$$V_4^B = (\Delta t)^2 S_k^T \Gamma^{-1} S_k. \quad (5.95)$$

As for the single-rate case, V_4^A is indefinite and V_4^B is positive definite.

ΔV_k for the discrete time steps when $\bar{k} \neq k/m$, i.e. time steps k for which parameter adaptation does *not* take place, is the same as (5.91) but with the V_4^A and V_4^B terms omitted. This means that the stability condition for the time steps for which parameter adaptation takes place is harder to satisfy (as expected), and hence the analysis below will only consider this case.

The maximum magnitudes for V_4^A and V_4^B can be derived as follows:

$$|V_4^A| = \Delta t \left| \sum_{i=k-m+1}^k \{s^T[i] W_r[i]\} \tilde{\theta}[\bar{k} - 1] \right| \quad (5.96)$$

$$\begin{aligned} &\leq \Delta t \sum_{i=k-m+1}^k \{\|s[i]\| \|W_r[i] \tilde{\theta}[\bar{k} - 1]\|\} \\ &\leq \Delta t \sum_{i=k-m+1}^k \{\|s[i]\| ((\|\hat{M}_{k-1}[i]\|_2 + \|M[i]\|_2) \|\dot{y}_r[i]\| + \\ &\quad (\|\hat{C}_{k-1}[i]\|_2 + \|C[i]\|_2) \|\dot{y}_r[i]\| + (\|\hat{G}_{k-1}[i]\| + \|G[i]\|))\}, \end{aligned}$$

$$\begin{aligned} |V_4^B| &= (\Delta t)^2 |S_k^T \Gamma^{-1} S_k| \\ &\leq (\Delta t)^2 \|S_k\|^2 \|\Gamma^{-1}\|_2 \\ &\leq (\Delta t)^2 \|\Gamma^{-1}\|_2 \left(\sum_{i=k-m+1}^k \{\|W_r[i]\|_2 \|s[i]\|\} \right)^2. \end{aligned} \quad (5.97)$$

Using the maximum bounds defined in Section 5.1.1,

$$|V_4^A| \leq \Delta t (J+1) (\|M\|_{max} \|\dot{y}_r\|_{max} + \|C\|_{max} \|\dot{y}_r\|_{max} + \|G\|_{max}) \sum_{i=k-m+1}^k \|s[i]\|, \quad (5.98)$$

and

$$\sum_{i=k-m+1}^k \|s[i]\| = \sum_{i=k-m+1}^k \|s[k] + \Delta s[i, k]\|, \quad (5.99)$$

where

$$\Delta s[i, k] = s[i] - s[k]. \quad (5.100)$$

Thus, the sum can be bounded as

$$\sum_{i=k-m+1}^k \|s[i]\| \leq \sum_{i=k-m+1}^k \{\|s[k]\| + \|\Delta s[i, k]\|\}, \quad (5.101)$$

where

$$\|\Delta s[i, k]\| \leq (k - i) \|\Delta s_{\max}\|. \quad (5.102)$$

Using the above results, it can be shown that

$$\sum_{i=k-m+1}^k \|s[i]\| \leq m \|s[k]\| + m^* \|\Delta s_{\max}\|, \quad (5.103)$$

where

$$m^* = \sum_{j=1}^{m-1} j \quad (\text{for } m \geq 2), \quad (5.104)$$

and $m^* = 0$ for $m = 1$ (the single-rate case). This gives:

$$|V_4^A| \leq \Delta t (\beta + 1) (\|M\|_{\max} \|\tilde{y}_r\|_{\max} + \|C\|_{\max} \|\dot{y}_r\|_{\max} + \|G\|_{\max}) (m \|s[k]\| + m^* \|\Delta s_{\max}\|). \quad (5.105)$$

Using the same method and notation, the bound for $|V_4^B|$ can be derived as:

$$|V_4^B| \leq (\Delta t)^2 \|\Gamma^{-1}\|_2 \|W_r\|_{\max}^2 (m^2 \|s[k]\|^2 + 2m \|\Delta s_{\max}\| m^* \|s[k]\| + (m^*)^2 \|\Delta s_{\max}\|^2). \quad (5.106)$$

Comparing (5.105) and (5.106) with (5.29) and (5.31), it can be seen that the bounds on $|V_4^A|$ and $|V_4^B|$ for the multi-rate case are the same as the bounds for the single-rate case, but with the substitution of $(m \|s[k]\| + m^* \|\Delta s_{\max}\|)$ for $\|s[k]\|$.

As for the single-rate case (5.13), the stability condition for the multi-rate case is:

$$|V_1^A| + |V_9^A| \geq |V_1^B| + |V_2| + |V_3| + |V_5| + |V_6| + |V_7| + |V_8| + |V_4^A| + |V_4^B| + |V_9^B|. \quad (5.107)$$

Substituting in for the magnitudes, the stability condition becomes:

$$\begin{aligned}
 & a\|s[k]\|^3 + \{k_D - a\|\Delta s_{max}\| - \Delta t\gamma^{-1}m^2\|W_r\|_{max}^2\}\|s[k]\|^2 \quad (5.108) \\
 & + \left\{-\frac{\|\Delta s_{max}\|}{2}k_D - (m+1)(\beta+1)(\|M\|_{max}\|\ddot{y}_r\|_{max} + \|C\|_{max}\|\dot{y}_r\|_{max} + \|G\|_{max})\right. \\
 & \quad \left.- 2\Delta t\gamma^{-1}mm^*\|W_r\|_{max}^2\|\Delta s_{max}\| + a\left(\frac{\|\Delta s_{max}\|}{2}\right)^2\right\}\|s[k]\| \\
 & - \|\Delta s_{max}\|\left(\frac{\rho}{2} + m^*(\beta+1) + 1\right)(\|M\|_{max}\|\ddot{y}_r\|_{max} + \|C\|_{max}\|\dot{y}_r\|_{max} + \|G\|_{max}) \\
 & \quad - \Delta t\gamma^{-1}(m^*)^2\|W_r\|_{max}^2\|\Delta s_{max}\|^2 \geq 0.
 \end{aligned}$$

This stability condition can be used to predict how large the filtered error can become before it is guaranteed that ΔV_k will become negative and hence ultimately $\|s[k]\|$ will decrease, defining a steady state bound for $\|s[k]\|$. The stability condition degenerates to the single-rate case if we set $m = 1$. By setting the parameter a (the compensator gain) to zero, (5.108) also applies to the uncompensated case, and for the case when $\|s[k]\| \leq \frac{\|\Delta s_{max}\|}{2}$.

The underlying equality represented in (5.108) is a cubic in terms of $\|s[k]\|$. By finding the roots of this equality, the bound on the filtered error as a function of $\|s[k]\|$ can be predicted. The largest positive root is interpreted as the stability bound, $\|s[k]\|_{bound}$. If $\|s[k]\| > \|s[k]\|_{bound}$, then the error will be ultimately driven within $\|s[k]\|_{bound}$.

The stability properties for the sampled-data system with a multi-rate compensated adaptive controller can be formally stated in the following theorem:

Theorem 8 *For a robot manipulator with dynamics given by (2.31), the compensated discretized multi-rate adaptive control algorithm consisting of the control law (5.90), adaptive law (5.89) and compensator (5.8) results in a sampled-data system which is stable, if assumptions 1 to 3 given previously are met.*

Corollary 2 *If the conditions of Theorem 8 are satisfied, a steady state bound on $\|s[k]\|$ for the system must exist. The greatest positive real root of the equality derived from (5.108) is the steady state error bound.*

Proof: Inspection of the cubic form of (5.108) proves that a value $\|s[k]\|_{bound}$ must exist which satisfies the inequality. Since (5.108) is derived from ΔV_k for the system

Table 5.2: Stability Condition for 2 DOF Serial Link Manipulator with Multi-Rate Controller

Case	Description	a	β	m	k_D	γ^{-1}	Δt	roots for $\ s\ $	Stable
1.	uncomp.	0	2	5	8	0.2	0.002	none	no
2.	comp.	100	2	5	8	0.2	0.002	236	yes
3.	uncomp., high γ^{-1}	0	2	5	8	5.0	0.002	none	no
4.	comp., high γ^{-1}	100	2	5	8	5.0	0.002	5200	yes
5.	comp., high γ^{-1}	1000	2	5	8	5.0	0.002	548	yes
6.	uncomp., high f_s	0	2	20	8	0.2	0.0005	none	no
7.	comp., high f_s	100	2	20	8	0.2	0.0005	860	yes
8.	uncomp., high f_s, γ^{-1}	0	2	20	8	5.0	0.0005	none	no
9.	comp., high f_s, γ^{-1}	100	2	20	8	5.0	0.0005	21000	yes
10.	comp., high f_s, γ^{-1}	1000	2	20	8	5.0	0.0005	2100	yes
11.	uncomp., high f_s, k_D	0	2	20	14	5.0	0.0005	none	no

as given by (5.3), ΔV_k must become negative for $\|s[k]\| > \|s[k]\|_{bound}$ and hence global stability is proven by Lyapunov's second method. $\square$

Recall that for the case without the non-linear compensator, stability cannot in general be guaranteed, and (5.108) can be used to determine for which conditions stability is predicted and what the steady state error bound will be.

5.2.2 Two DOF Robot Manipulator Example

Returning to the two DOF example of Section 5.1.3, the stability condition for the multi-rate controller can be evaluated using (5.108). The stability predictions are listed in Table 5.2, and can be compared to corresponding simulation results.

Cases #1 to #5 are for $m = 5$, i.e. the servo loop operates at 5 times the rate of the parameter update ($f_{sampling} = 500$ Hz, $f_{adaptation} = 100$ Hz). As for the single rate case, stability is not predicted for the uncompensated case with medium adaptation g in γ^{-1} (case #1), while the simulation shows stable perfor-

mance. The performance is very similar to the corresponding single-rate simulation with $f_{sampling} = 100$ Hz, but the multi-rate simulation shows better and smoother performance after the initial adaptation transient (presumably due to the filtering effect of the multi-rate adaptation algorithm). Stability is predicted when compensation is added (case #2).

When the adaptation gain is raised to 5.0, simulation of the uncompensated controller shows apparently unstable behaviour, with the parameter estimates hitting their bounds and the control torque saturating at times — this instability is predicted from the stability condition (see case #3). While stability is predicted when the compensator is added (cases #4 and #5), this cannot be demonstrated by simulation since $\|s\|$ does not exceed $\|s'[k]\|_{max}$ and hence the compensator does not become active (see discussion of results for single-rate simulations in Section 5.1.3).

Cases #6 to #10 are for $m = 20$, with $f_{sampling} = 2000$ Hz and $f_{adaptation} = 100$ Hz. The above comments for $m = 5$ (cases #1 to #5) apply equally for $m = 20$. Figures 5.5, 5.6 and 5.7 show the parameter estimates, filtered error and control torques respectively for the uncompensated case with $\gamma^{-1} = 5.0$ (case #8). The instability is indicated by the parameters hitting their bounds and the control torques saturating at 10 N-m. The filtered error, $\|s\|$, reaches relatively high values at times. Compensation is not possible for this case using the a-priori calculated value $\|\Delta s_{max}\| = 1.56$, since $\|s\|$ for the simulation never reaches $\frac{\|\Delta s_{max}\|}{2}$ for the uncompensated case (see Fig. 5.6). Examination of the simulation data shows that the actual value of $\|\Delta s_{max}\|$ is 0.1⁴. Using this value in the compensator, simulation shows stable performance, with reduced filtered error as shown in Figure 5.6 and control torques as shown in Figure 5.8. The compensator becomes active any time $\|s\| > \frac{\|\Delta s_{max}\|}{2}$, and the effective compensator gain, $g = a(\|s\| - \frac{\|\Delta s_{max}\|}{2})$ is shown in Fig. 5.9 (refer to the definition of the compensator given by (5.8)).

Case #11 of Table 5.2 is the uncompensated, $m = 20$ case but with higher

⁴This value of $\|\Delta s_{max}\|$ can be used in the compensator to define the dead zone and still guarantee stability according to Theorem 8, since it is the actual value of $\|\Delta s_{max}\|$ which actually limits the performance of the compensator in driving ΔV negative according to the formulation of the compensator

Figure 5.5: Parameter Estimates, Uncompensated Controller

Figure 5.6: Norm of Filtered Tracking Errors (— compensated, - - - uncompensated)

feedback gain k_D . While stability is not predicted, the simulation does show stable performance.

It has been demonstrated by prediction and simulation of the two DOF serial link manipulator that the compensator can stabilize and improve the tracking performance of the multi-rate sampled-data adaptive controller. As for the single-rate case discussed previously in Section 5.1.3, for cases without the non-linear compensator where stability cannot be guaranteed, addition of the compensator does give a prediction of guaranteed stability. However, the sufficient condition predicted for stability given by Theorem 8 and its corollary are relatively conservative. The ap-

Figure 5.7: Control Torques, Uncompensated Controller (— joint 1, - - - joint 2)

Figure 5.8: Control Torques, Compensated Controller (— joint 1, - - - joint 2)

Figure 5.9: Effective Compensator Gain

proach for improving the prediction of the stability and bounds on the filtered error is described in the following section.

5.3 Derivation of an Error Bound using a Local Approach

An obstacle to the practical use of the predicted filtered error bounds presented so far is their conservative nature. This is due to the fact that the bounds on the dynamics used to calculate the error bounds make use of overall, global bounds for the system which apply to any state of the system within the physical limits. The bounds on the filtered error so derived are in general too conservative to be of practical value; the error bound condition predicts a large bound on $\|s\|$ for many cases where the actual bound on $\|s\|$ from simulations is very small. To improve the prediction of the error bounds, a modified approach which examines the error bound condition locally along the nominal desired trajectory is of interest and is the subject of this chapter.

A stability condition based on the *local* as opposed to global approach is derived below, to tighten the prediction of error bounds. This local condition is based on the assumption that the manipulator is always near to the desired path:

Assumption 4 *The manipulator state remains “close” to the desired trajectory at all times, hence $s[k]$ is sufficiently small to be considered zero for the purposes of calculating the error bound.*

In this way, the local error bounds about the desired path are predicted.

5.3.1 The Error Bound Condition

As a first step towards deriving the local error bounds, $\|\tilde{y}_r\|_{max}$, $\|\dot{y}_r\|_{max}$ and $\|W_r\|_{max}$ can be replaced by the corresponding bounds evaluated for the desired trajectory ($\|\dot{y}_d\|_{max}$, $\|\ddot{y}_d\|_{max}$ and $\|W_d\|_{max}$). The bound on the norm of the acceleration, $\|\ddot{y}\|_{max}$, is replaced with the bound evaluated over the desired path.

Table 5.3: Error Bounds for 2 DOF Serial Link Manipulator

<i>Case</i>	<i>a</i>	β	<i>m</i>	k_D	γ^{-1}	Δt	<i>Global</i> $\ s\ _{bound}$	<i>Local</i> $\ s\ _{bound}$
1.	0	2	1	8	0.2	0.01	none	29.4
2.	0	2	5	8	0.2	0.002	none	none
3.	0	2	5	8	5.0	0.002	none	none
4.	0	2	20	8	0.2	0.0005	none	none
5.	0	2	20	8	5.0	0.0005	none	none
6.	100	2	20	8	5.0	0.0005	21000	22.6
7.	1000	2	20	8	5.0	0.0005	2100	4.67

Using this approach, the predicted bounds for $\|s\|$ are much smaller in general, and hence closer to the simulation results, but are still conservative. Table 5.3 compares the predicted error bounds for the previous (global) approach (see Sections 5.1 and 5.2) and the local approach. The error bounds are shown for various single- and multi-rate adaptive control cases, with and without compensation (indicated by the value of a).

As a second step towards deriving the local error bounds, it is possible to evaluate ΔV and hence the bound on $\|s\|$ at each discrete time step along the desired trajectory. This is in contrast to the previous approach which evaluated ΔV once for the entire trajectory and therefore resulted in an overall, global, worst-case prediction of the error bound.

The procedure is now to successively look at each discrete time step k , evaluate the bounds on the various terms composing ΔV for that time step only, and find the bound on $\|s\|$ as a function of discrete time, $\|s\|_{bound}[k]$. To easily perform these calculations it is necessary to make the approximating assumption that the values of the various terms in ΔV (e.g. $\ddot{y}_d(t)$, $\dot{y}_r(t)$, $W_r(t)$, etc.) do not change significantly enough during the single time step to noticeably affect the error bound. This makes it possible to use the values of these terms at the beginning of the time step and to

apply them over the entire time step.

Instead of using the saturation torque, τ_{sat} , to calculate the bound $\|\tilde{y}\|_{max}$, it is also possible to use the form of the control law to bound the input torque for each time step and hence more accurately calculate the bound on the acceleration. Recall that the discrete adaptive control law with compensator is:

$$\tau_c[k] = \hat{M}(y[k])\tilde{y}_r[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}(y[k]) - K_D s[k] + f[k]. \quad (5.109)$$

Since good tracking of the desired trajectory is assumed, the feedback parts of the control law drop out leaving the feedforward torque:

$$\tau_c[k] = \hat{M}(y[k])\tilde{y}_d[k] + \hat{C}(y[k], \dot{y}[k])\dot{y}_d[k] + \hat{G}(y[k]). \quad (5.110)$$

Thus, the bound on the control torque is:

$$\|\tau_c\|_{max}[k] = \|\hat{M}[k]\|_{max}\|\tilde{y}_d[k]\| + \|\hat{C}[k]\|_{max}\|\dot{y}_d[k]\| + \|\hat{G}[k]\|_{max}, \quad (5.111)$$

and the bound on the torque input to the manipulator is:

$$\|\tau\|_{bound}[k] = \min(\|\tau_c\|_{max}[k], \|\tau_{sat}\|). \quad (5.112)$$

The bounds on the norms of the various terms to be used in the stability condition (5.108) are now redefined as norms which are dependent on the time step k , as opposed to single global values used previously. These bounds are defined as follows:

$$\|M\|_{max} = \|M\|_{max}[k] \equiv \|M(y_d[k])\|_2 \quad (5.113)$$

$$\|C\|_{max} = \|C\|_{max}[k] \equiv \|C(y_d[k], \dot{y}_d[k])\|_2 \quad (5.114)$$

$$\|G\|_{max} = \|G\|_{max}[k] \equiv \|G(y_d[k])\| \quad (5.115)$$

$$\|\dot{y}_r\|_{max} \equiv \|\dot{y}_d[k]\| \quad (5.116)$$

$$\|\tilde{y}_r\|_{max} \equiv \|\tilde{y}_d[k]\| \quad (5.117)$$

$$\|W_r\|_{max} = \|W_d\|_{max}[k] \equiv \|W(y_d[k], \dot{y}_d[k], \tilde{y}_d[k])\|_2 \quad (5.118)$$

$$\|\dot{y}_d\|_{max} \equiv \|\dot{y}_d[k]\| \quad (5.119)$$

$$\|\tilde{y}_d\|_{max} \equiv \|\tilde{y}_d[k]\| \quad (5.120)$$

$$\|\tilde{y}\|_{max} = \|\tilde{y}\|_{max}[k] \equiv \|M^{-1}(y_d[k])\|_2(\|\tau\|_{bound}[k] + \|C(y_d[k], \dot{y}_d[k])\dot{y}_d[k]\| + \|G(y_d[k])\|) \quad (5.121)$$

$$\|\dot{s}[k]\|_{max} \equiv \|\tilde{y}\|_{max}[k] + \|\tilde{y}_d[k]\| \quad (5.122)$$

$$\|\Delta s_{max}\| = \|\Delta s_{max}[k]\| \equiv \Delta t \|\dot{s}[k]\|_{max}. \quad (5.123)$$

Substituting these new bounds into the previous stability condition (5.108), the condition for each time step becomes:

$$\begin{aligned} a\|s[k]\|^3 + \{k_D - a\|\Delta s_{max}[k]\| - \Delta t\gamma^{-1}m^2\|W_d\|_{max}^2[k]\}\|s[k]\|^2 + \{-\frac{\|\Delta s_{max}[k]\|}{2}k_D \\ - (m+1)(\beta+1)(\|M\|_{max}[k]\|\tilde{y}_d[k]\| + \|C\|_{max}[k]\|\dot{y}_d[k]\| + \|G\|_{max}[k]) \\ - 2\Delta t\gamma^{-1}nm^*\|W_d\|_{max}^2[k]\|\Delta s_{max}[k]\| + a(\frac{\|\Delta s_{max}[k]\|}{2})^2\}\|s[k]\| - \|\Delta s_{max}[k]\|(\frac{\beta}{2} \\ + m^*(\beta+1)+1)(\|M\|_{max}[k]\|\tilde{y}_d[k]\| + \|C\|_{max}[k]\|\dot{y}_d[k]\| + \|G\|_{max}[k]) \\ - \Delta t\gamma^{-1}(m^*)^2\|W_d\|_{max}^2[k]\|\Delta s_{max}[k]\|^2 \geq 0. \end{aligned} \quad (5.124)$$

The inequality (5.124) must be evaluated at each time step along the desired path to check the overall error bound based on the local error bounds at each point. The overall bound on $\|s[k]\|$ is given by evaluating (5.124) for each step k :

$$\{\|s\|_{bound}\}_{overall} = \sup_k \{\|s\|_{bound}[k]\}. \quad (5.125)$$

The local error bound properties for the sampled-data system with a multi-rate compensated adaptive controller can be formally stated in the following theorem:

Theorem 9 *If the conditions of Theorem 8 are satisfied and if in addition assumption 4 given previously is met, then a bound on $\|s[k]\|$ must exist for the system for each k . The greatest positive real root of the equality derived from (5.124) for a given time step k is the steady state bound on $\|s[k]\|$, $\|s\|_{bound}[k]$.*

Corollary 3 *If the conditions of Theorem 9 are satisfied for all time steps k , a steady state bound on $\|s[k]\|$ must exist for the system, given as $\{\|s\|_{bound}\}_{overall}$ by (5.125).*

Proof: Inspection of the cubic form of (5.124) proves that a value $\|s[k]\|_{bound}[k]$ must exist which satisfies the inequality for each k . Since (5.124) is derived from ΔV_k

Figure 5.10: Norm of Desired Torque, $\|\tau_d\|$

for the system as given by (5.3), ΔV_k must become negative for $\|s[k]\| > \|s\|_{bound}[k]$ and hence stability is proven by Lyapunov's second method. Thus, the error $\|s[k]\|$ must be driven back to within $\|s\|_{bound}[k]$. $\square$

5.3.2 Two DOF Manipulator Example

The error bound derived from (5.124) has been evaluated for various sampled-data simulation cases, for the two DOF serial link manipulator discussed previously in Sections 5.1.3 and 5.2.2.

Once again, the baseline adaptive case was selected using a persistently exciting desired trajectory. The adaptation and feedback gains, and the filtered error parameter matrix were set as: $\Gamma^{-1} = 0.2I^{3 \times 3}$, $K_D = 8I^{2 \times 2}$, $\Lambda = 10I^{2 \times 2}$.

Figure 5.10 shows the norm of the desired torque, which is the torque required to exactly follow the desired path based on the true dynamics of the manipulator. It can be seen that the norm of the desired torque, $\|\tau_d\|$, reaches a maximum of 10 N-m, whereas $\tau_{sat}^T = [10 \ 10]$ and $\|\tau_{sat}\| = 14$ N-m. Thus, torque saturation should not occur for perfect tracking.

The error bound condition (5.124) can easily be modified to apply to the non-adaptive (PD only) control case, by dropping out the non-applicable terms. Figure 5.11 is a comparison of the predicted $\|s\|_{bound}[k]$ with the actual values of $\|s[k]\|$

Figure 5.11: Norm of Filtered Tracking Errors, $\|s\|$ (— actual (simulation), - - - predicted bound)

from the simulation of the PD control case. The predicted bound now envelopes the simulation results nicely.

Figure 5.12 shows $\|\tau\|_{bound}[k]$ for the baseline single-rate adaptive case (without the added compensator). The norm of the maximum possible torque, due to only the feedforward control, reaches the saturation limit for part of the path. The resulting maximum norm of the acceleration is shown in Figure 5.13. The predicted bound $\|s\|_{bound}[k]$ is shown in Figure 5.14. This bound is still very conservative, since the maximum value of $\|s[k]\|$ obtained from the simulation is 0.284. However, the prediction of the error bound is still an improvement over the previous global bound, which did not guarantee stability (regardless of the magnitude of $\|s[k]\|$).

Figure 5.15 shows the roots for $\|s[k]\|$ given by the stability condition (5.124) for the multi-rate adaptive case without the non-linear compensator and with moderate adaptation gain ($m = 20$, $\gamma^{-1} = 0.2$). $\|\tau\|_{bound}[k]$ and the resulting maximum norm of the acceleration for this case are the same as for the baseline adaptive case (Figures 5.12 and 5.13). Uniform stability is not predicted since a positive root for $\|s\|_{bound}[k]$ does not always exist, whereas the simulation appears to be stable, with $\|s[k]\|$ as shown in Figure 5.16. The previous global stability condition did not guarantee stability.

The predicted bound $\|s\|_{bound}[k]$ for the multi-rate adaptive case with high adap-

Figure 5.12: Norm of Maximum Input Torque, $\|\tau\|_{bound}[k]$ Figure 5.13: Norm of Maximum Acceleration, $\|\ddot{y}\|_{max}[k]$ Figure 5.14: Bound on Filtered Tracking Errors, $\|s\|_{bound}[k]$, baseline adaptive case

Figure 5.15: Roots for Filtered Tracking Error Bound, $\|s\|_{bound}[k]$, for multi-rate adaptive control

Figure 5.16: Norm of Filtered Tracking Errors, $\|s\|$, for Simulation without Compensator

Figure 5.17: Bound on Filtered Tracking Errors, $\|s\|_{bound}[k]$, for high adaptation gain and compensation

tation gain and compensation ($m = 20$) is shown in Figure 5.17. Refer to Figures 5.12 and 5.13 for $\|\tau\|_{bound}[k]$ and $\|\tilde{y}\|_{max}[k]$. This bound $\|s\|_{bound}[k]$ is also still very conservative, since the maximum value of $\|s[k]\|$ obtained from the simulation is approximately 0.15. However, the prediction of the error bound of between 1 and 20 rad/sec is an improvement compared to the previous global bound, which predicted $\|s\|_{bound} = 21000$. Figure 5.18 is a comparison showing that the predicted bound on the norm of the torque, $\|\tau\|_{bound}[k]$, envelopes the actual norm of the torque from the simulation.

The above results demonstrate that the predictions of $\|s\|_{bound}$ based on the local approach are less conservative than the previous global bounds approach. In particular, the predicted bound $\|s\|_{bound}[k]$ for the PD control case envelopes the simulation data nicely. This approach also produces a prediction of whether or not torque saturation is possible due to the feedforward component of the control law based on the maximum possible parameter error. The adaptive cases examined above show that according to the prediction of torque (given by $\|\tau\|_{bound}[k]$), the torque can indeed saturate for some portions of the trajectory.

Figure 5.18: Norm of Torque (— predicted bound $\|\tau\|_{bound}[k]$, - - - actual (simulation))

Chapter 6

Two DOF Manipulator Implementation

6.1 Description of the Manipulator

The discrete-time adaptive controller with the non-linear compensator added has been implemented for the Carleton Direct Drive Manipulator, to experimentally validate the analytical and simulation results. This manipulator is a two DOF, *Selective Compliance Assembly Robot Arm* (SCARA) type which operates in the horizontal plane, and has a 4-link parallelogram geometry. This manipulator has a design which enables inertial decoupling of the two joints when the link inertias are appropriately “tuned”. Figure 6.1 shows the geometry of the manipulator, and Plate 6.1 is a photograph of the manipulator.

The robot 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 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.

A 486 PC computer running at 33 MHz is used for control and data acquisition. The computer is interfaced to the robot motors and servo amplifiers via two interface cards manufactured by Data Translation, Inc. An analog-to-digital (A/D) and digital-to-analog (D/A) conversion card acquires the analog tachometer signals and outputs the analog control voltages to the servo amplifiers. A digital-to-digital

Figure 6.1: Geometry of Carleton Direct Drive Manipulator (top view)

(D/D) parallel interface card acquires the digital position signals and various status flags from the servo amplifiers. A wiring panel with terminal blocks is used to interconnect the amplifiers, motors and computer interfaces. A block diagram of the manipulator system is shown in Figure 6.2, and Plate 6.2 shows the system as set up in the lab.

The mechanical design and construction of the Carleton Direct Drive Manipulator was described by Rajguru [39]. A description of the integration and initial testing of the manipulator system can be found in Warshaw [61]. The specifications for the servo amplifiers, motors, resolvers, tachometers and interface cards can also be found in Appendix E of Warshaw [61].

As discussed in Warshaw [61], measurement noise can be a problem for the manipulator system. Specifically, the noise present in the velocity signals measured from the analog tachometer voltages can be significant. To reduce the noise levels on the velocity signals, passive RC filters were inserted between each tachometer and the A/D inputs. These filters were designed with a bandwidth of 48 Hz, to attenuate the high frequency noise without adding excessive attenuation and phase lag to the actual joint velocity signals. For example, at a frequency of 1 Hz, the RC filter attenuates the velocity signal by only 0.3 %, and introduces a phase lag of 1.2 degrees. The RC filters also serve as rudimentary anti-aliasing filters.

Figure 6.2: System Block Diagram of Carleton Direct Drive Manipulator

Plate 6.1: Carleton Direct Drive Manipulator

Plate 6.2: Robot System Laboratory Setup

Figure 6.3: Coordinates and Link Lengths for the Manipulator (top view)

6.2 Kinematics and Dynamics of the Manipulator

The forward kinematic equations are the equations relating the end effector (end point) position of the robot in cartesian coordinates to the joint (motor) angular positions. These equations were given in Rajguru [39], and are as follows:

$$p_x = l_1 \cos y_1 - (l_5 - l_4) \cos y_2, \quad (6.1)$$

$$p_y = l_1 \sin y_1 - (l_5 - l_4) \sin y_2, \quad (6.2)$$

where p_x and p_y are the cartesian coordinates of the end point, y_1 and y_2 are the angular positions of the joints, and l_1 , l_4 , and l_5 are link lengths. The coordinate systems and link lengths for the manipulator are shown in Figure 6.3.

Using the matrix form of the equations of dynamics for a manipulator given previously as (2.31), the dynamics for the Carleton Direct Drive Manipulator can be expressed as:

$$\tau = M(y)\ddot{y} + C(y, \dot{y})\dot{y}, \quad (6.3)$$

where

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

$$M(y) = \begin{bmatrix} m_{11} & m_{12} \\ m_{12} & m_{22} \end{bmatrix} \quad (6.4)$$

$$C(\dot{y}, y) = \begin{bmatrix} 0 & c_{12} \\ c_{21} & 0 \end{bmatrix}. \quad (6.5)$$

The vector of gravity terms, $G(y)$, has been omitted from the dynamics since motion is only permitted in the horizontal plane. The effective inertias for the two joints are represented by the terms m_{11} and m_{22} , and the coupling inertia is m_{12} . It is a known property of manipulator dynamics (see, for example, Ortega and Spong [38]) that $M(y)$ must be a symmetrical and positive definite matrix; thus the coupling inertia (off-diagonal) terms of $M(y)$ are identical. The centrifugal "force" terms are represented by c_{12} and c_{21} . Due to the direct drive design and use of low friction ball bearings, friction can be neglected for the manipulator model, and hence friction terms do not appear in $C(\dot{y}, y)$.

The elements of the M and C matrices are:

$$m_{11} = a_1, \quad (6.6)$$

$$m_{22} = a_2, \quad (6.7)$$

$$m_{12} = a_3 \cos(y_1 - y_2), \quad (6.8)$$

$$c_{12} = a_3 \sin(y_1 - y_2) \dot{y}_2, \quad (6.9)$$

$$c_{21} = -a_3 \sin(y_1 - y_2) \dot{y}_1. \quad (6.10)$$

The parameter vector, θ , consists of the three inertial parameters appearing in M and C , which are defined as follows:

$$a_1 = m_1 l_{1c}^2 + i_1 + m_3 l_{3c}^2 + i_3 + m_4 l_1^2 + m_0 l_1^2, \quad (6.11)$$

$$a_2 = m_2 l_{2c}^2 + i_2 + m_3 l_2^2 + m_4 l_{4c}^2 + i_4 + m_0 (l_5 - l_4)^2 + i_0, \quad (6.12)$$

$$a_3 = m_3 l_2 l_{3c} - m_4 l_1 l_{4c} - m_0 l_1 (l_5 - l_4). \quad (6.13)$$

The m , l , and i terms which compose the inertial parameters are the link masses, link lengths and link center-of-gravity (C-of-G) distances, and link inertias (about their C-of-G's) respectively. These terms, as calculated from the geometrical information provided in Rajguru [39], are shown in Table 6.1.

Table 6.1: Link Parameters for the Carleton Direct Drive Manipulator

lengths (meters)	C-of-G's (meters)	link masses (kg)	link inertias (kg-m ²)
$l_1 = 0.30$	$l_{1c} = 0.150$	$m_1 = 2.3$	$i_1 = 0.033$
$l_2 = 0.25$	$l_{2c} = 0.125$	$m_2 = 1.3$	$i_2 = 0.014$
$l_3 = 0.30$	$l_{3c} = 0.150$	$m_3 = 1.4$	$i_3 = 0.020$
$l_4 = 0.25$	$l_{4c} = 0.018$	$m_4 = 4.9$	$i_4 = 0.213$
$l_5 = 0.61$	(link 4 + boom)		
		$m_0 = \text{end load}$	$i_0 = \text{end load inertia}$

Using the regression form of the dynamics as defined previously by (2.10), the manipulator dynamics are:

$$\tau = W(y, \dot{y}, \ddot{y})\theta, \quad (6.14)$$

where

$$\theta = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}, \quad (6.15)$$

$$W(y, \dot{y}, \ddot{y}) = \begin{bmatrix} \ddot{y}_1 & 0 & \cos(y_1 - y_2)\ddot{y}_2 + \sin(y_1 - y_2)\dot{y}_2^2 \\ 0 & \ddot{y}_2 & \cos(y_1 - y_2)\ddot{y}_1 - \sin(y_1 - y_2)\dot{y}_1^2 \end{bmatrix}. \quad (6.16)$$

During the course of experimentation using the Carleton Direct Drive Manipulator, it was discovered that the servo amplifiers produced a noticeable and time-varying torque bias for each of the two motors. In other words, for a zero control torque command to the servo amplifiers, a non-zero torque is produced by the motors. The manipulator dynamic model can be augmented to account for this by adding two torque bias parameters to the parameter vector θ . We can now define the joint torque vector as:

$$\tau = \tau_c + \tau_{bias}, \quad (6.17)$$

where τ_c is the (requested) control torque vector and τ_{bias} is the torque bias vector.

Expressing the dynamics in terms of the control torque and combining (6.3) and (6.17), the augmented manipulator dynamics become:

$$\tau_c = M(y)\ddot{y} + C(y, \dot{y})\dot{y} - \tau_{bias}. \quad (6.18)$$

The parameter vector and regression matrix for the augmented dynamic model are:

$$\theta = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}, \quad (6.19)$$

$$W(y, \dot{y}, \ddot{y}) = \begin{bmatrix} \ddot{y}_1 & 0 & \cos(y_1 - y_2)\ddot{y}_2 + \sin(y_1 - y_2)\dot{y}_2^2 & -1 & 0 \\ 0 & \ddot{y}_2 & \cos(y_1 - y_2)\ddot{y}_1 - \sin(y_1 - y_2)\dot{y}_1^2 & 0 & -1 \end{bmatrix}, \quad (6.20)$$

with

$$\tau_{bias} = \begin{bmatrix} a_4 \\ a_5 \end{bmatrix}. \quad (6.21)$$

6.3 The Control Software

All control and data acquisition software was written in Borland International Inc.'s Turbo Pascal (Version 5.5) programming language. The control software uses an interrupt driven structure, employing the A/D card's built-in timer to interrupt the PC's CPU, thereby synchronizing the control loop timing to the desired sampling frequency.

Various versions of a program to control the two DOF's of the robot using an adaptive controller were written and tested. The most general of these control programs implements a multi-rate version of Slotine and Li's adaptive controller with the non-linear discrete-time compensator added. By setting the various control and adaptive gains appropriately, as well as the initial parameter estimates, different control cases can be tested with this control program:

- with the servo-loop sampling rate set equal to the adaptation rate, the multi-rate implementation degenerates to the single-rate case;

- with the compensator gain set to zero, Slotine and Li's adaptive controller without the compensator is implemented;
- additionally, with the adaptive gain set to zero, a fixed parameter feedforward controller is implemented;
- additionally, with the parameter estimates set to zero, a PD controller is implemented.

The desired trajectory is defined by a data file containing the desired joint positions for a set sampling frequency. This data file can be generated by various means, including a form of "teaching", whereby the joint positions of the manipulator as it is moved by hand can be stored to a file (by another program written in Turbo Pascal). The control program loads the position data and filters it using the discrete-time implementation of a 3rd order filter. This pre-filtering provides four important features:

1. the desired velocities and accelerations are synthesized from only the position data;
2. the sampling frequency of the pre-defined desired joint trajectory data can be different from the controller sampling time;
3. the pre-filter avoids an extreme start-up transient by providing a smooth path to follow between the actual initial positions of the manipulator joints and the desired trajectory;
4. the pre-filter limits the bandwidth of the desired trajectory to keep it within the dynamic capability of the closed-loop system.

The pre-filter typically used for the experiments was the discretization of a continuous-time filter with a triple pole at -10.0 , giving a bandwidth of 0.8 Hz.

The sampled joint velocity data can optionally be filtered before use by a discrete-time implementation of a 2nd order Butterworth filter, to reduce the level of measurement noise. The Butterworth filter typically used for the experiments was the

discretization of a continuous-time filter with a bandwidth of 25 Hz. A numerical analysis of the performance of the discretized filter using MATLAB showed that the magnitude and phase response are essentially identical to the continuous-time filter for frequencies less than 80 Hz for a sampling rate of 750 Hz. However, while the filtering was beneficial in reducing the effects of measurement noise for the adaptive case without the non-linear compensator added, the filtering does introduce sufficient attenuation and phase lag to interfere with the correct operation of the non-linear compensation. Therefore, the filtering was disabled for certain runs as specified in the results section which follows.

A steady-state Kalman filter has previously been used to provide smoothed estimates of the joint velocities (see Warshaw [61]), but this was rejected for the current implementation for two reasons:

1. the Kalman filter introduces additional dynamics which are not accounted for in the system model and hence are not included in the stability and performance analyses;
2. the Kalman filtered velocity estimates are sensitive to unknown torque disturbances.

The control software design is modular, making use of separate units to perform the following tasks:

- A/D and D/D card interfacing;
- calculation of the discretized filter coefficients;
- matrix math used by the filter unit;

The control program consists of eight main components:

1. program control, initialization and input/output;
2. desired trajectory pre-filtering;
3. acquisition of joint measurements (position, velocity);

4. velocity measurement filtering;
5. parameter adaptation;
6. control torque calculation and output;
7. an interrupt service routine which controls the components 2. to 6. above;
8. display of the joint states on the computer screen.

6.3.1 The Implementation of Slotine and Li's Adaptive Controller

The multi-rate discretized version of Slotine and Li's Adaptive Controller was formulated in Section 5.2 (equations (5.90) and (5.89)). These equations describing the discretized controller are repeated below for convenience:

$$\tau[k] = \hat{M}_{\bar{k}}(y[k])\ddot{y}_r[k] + \hat{C}_{\bar{k}}(y[k], \dot{y}[k])\dot{y}_r[k] + \hat{G}_{\bar{k}}(y[k]) - K_D s[k] + f[k], \quad (6.22)$$

$$\hat{\theta}[\bar{k}] = \hat{\theta}[\bar{k} - 1] - \Delta t \Gamma^{-1} \sum_{i=k-m+1}^k \{W_r^T(y[i], \dot{y}[i], \dot{y}_r[i], \ddot{y}_r[i])s[i]\}. \quad (6.23)$$

For the Carleton Direct Drive Manipulator:

$$\hat{\theta}[k] = \begin{bmatrix} \hat{a}_1[k] \\ \hat{a}_2[k] \\ \hat{a}_3[k] \\ \hat{a}_4[k] \\ \hat{a}_5[k] \end{bmatrix}, \quad (6.24)$$

$$W_r(y[k], \dot{y}[k], \dot{y}_r[k], \ddot{y}_r[k]) = \begin{bmatrix} \ddot{y}_{r1}[k] & 0 & \cos(y_1[k] - y_2[k])\ddot{y}_{r2}[k] + \sin(y_1[k] - y_2[k])\dot{y}_2[k]\dot{y}_{r2}[k] & -1 & 0 \\ 0 & \ddot{y}_{r2}[k] & \cos(y_1[k] - y_2[k])\ddot{y}_{r1}[k] - \sin(y_1[k] - y_2[k])\dot{y}_1[k]\dot{y}_{r1}[k] & 0 & -1 \end{bmatrix}. \quad (6.25)$$

Appendix B contains the software listing for the multi-rate adaptive control with non-linear compensation version of the robot control program.

Figure 6.4: Desired Position (— joint 1, - - - joint 2)

6.4 Experimental Results

A persistently exciting trajectory consisting of a combination of sinusoids (in joint space) was selected. One cycle of the desired trajectory for the joints is shown in Figure 6.4. The manipulator model was parameterized using 5 parameters: 3 inertial parameters and 2 parameters representing the torque bias of the motor/servo-amplifier pairs. Unless otherwise specified, the gains were set as follows: $\Gamma^{-1} = 0.16 I^{5 \times 5}$, $K_D = 9.74 I^{2 \times 2}$, $\Lambda = 10 I^{2 \times 2}$. The torque saturation limit for each joint was 8.5 N-m (based on a maximum possible torque control signal of ± 5 V, and a measured torque constant of 1.7 N-m/V), giving $\|\tau_{sat}\| = 12$ N-m.

6.4.1 Single-Rate Experiments

A sampling rate of 750 Hz was used for the single-rate experiments. This is the maximum sampling rate possible with the current hardware and software configuration.

Figure 6.5 is a comparison of the norm of the filtered tracking error, $\|s\|$, for three different experiment runs:

1. FD control only;
2. adaptive control;

Figure 6.5: Norm of Filtered Tracking Errors, $\|s\|$ ($\cdots$ PD, $- - -$ adaptive, $—$ compensated adaptive)

3. adaptive control with compensation;

As expected, the adaptive controller has lower tracking error than the PD controller after the initial parameter adaptation (approximately the first two seconds)¹. The compensated adaptive controller has lower tracking error than the PD controller at all times and significantly lower tracking error than the uncompensated adaptive controller during the initial parameter adaptation.

Figure 6.6 shows the norm of the desired torque (see Section 5.3.2 for explanation of this term). $\|\tau_d\|$ does not reach the value of $\|\tau_{sat}\| = 12$ N-m, and $\tau_{sat}^T = [8.5 \ 8.5]$ N-m, so torque saturation should not occur for perfect tracking.

Figure 6.7 is a comparison of the predicted $\|s\|_{bound}[k]$ with the actual values of $\|s[k]\|$ from an experimental run using PD control only. The predicted bound follows the actual experimental results closely.

Figure 6.8 shows the position tracking errors for an adaptive control experiment with $\beta = 2$ and no non-linear compensation. The corresponding parameter estimates are shown in Figure 6.9 (recall that a_1 to a_3 are the inertial parameters; a_4 and a_5 are torque bias parameters). Figure 6.10 shows $\|\tau\|_{bound}[k]$ for this same adaptive case. This norm of the maximum possible torque due to the feedforward control

¹This should be true as long as the adaptation gains are selected appropriately to avoid instability.

Figure 6.6: Norm of Desired Torque, $\|\tau_d\|$

Figure 6.7: Norm of Filtered Tracking Errors, $\|s\|$, PD Control (— actual, - - - predicted bound)

Figure 6.8: Position Tracking Errors, Adaptive Control (— joint 1, - - - joint 2)

Figure 6.9: Parameter Estimates, Adaptive Control (a_4 , a_5 values divided by 10)

predicts that saturation is possible at times. The resulting maximum norm of the acceleration is shown in Figure 6.11². The predicted bound $\|s\|_{bound}[k]$ and the experiment results are compared in Figure 6.12, showing that the predicted bound envelopes the test results. The desired torque $\|\tau_d\|$, predicted bound on the torque $\|\tau\|_{bound}[k]$, and actual torque from the experiment are compared in Figure 6.13.

Figure 6.14 compares the predicted $\|s\|_{bound}[k]$ with the actual values of $\|s[k]\|$ from the experimental run of the adaptive case with non-linear compensation added.

² $\|\ddot{y}\|_{max}[k]$ appears to be simply a scaled version of $\|\tau\|_{bound}[k]$; this is true as an approximation due to the weak coupling between the two joints. As seen in Figure 6.9, the parameter representing the degree of coupling between the two joints, a_3 , is relatively small.

Figure 6.10: Norm of Maximum Input Torque, $\|\tau\|_{bound}[k]$, Adaptive Control

Figure 6.11: Norm of Maximum Acceleration, $\|\ddot{y}\|_{max}[k]$, Adaptive Control

Figure 6.12: Norm of Filtered Tracking Errors, $\|s\|$, Adaptive Control (— actual, - - - predicted bound)

Figure 6.13: Norm of Input Torque, Adaptive Control (— actual, - - - predicted bound, . . . desired)

Figure 6.14: Norm of Filtered Tracking Errors, $\|s\|$, Adaptive Control with Compensator (— actual, - - - predicted bound)

$\|\tau\|_{bound}[k]$ and $\|\tilde{y}\|_{max}[k]$ for this case are the same as for the adaptive control case without compensation (Figures 6.10 and 6.11). The predicted bound $\|s\|_{bound}[k]$ once again envelopes the test results. The effective compensator gain, g , is shown in Figure 6.15.

6.4.2 Multi-Rate Experiments

To demonstrate the stabilizing effect of the non-linear compensator, adaptive control experiments were run using the multi-rate control software. The desired trajectory, controller feedback gains and torque saturation limits were the same as those described previously for the single-rate controller experiments. The sampling rate for the servo loop was kept at 750 Hz, but the parameter update rate was lowered to 75 Hz, giving multi-rate control with $m = 10$. The adaptation gain was raised to $\Gamma^{-1} = 2.2I^{5 \times 5}$. To avoid interference with the non-linear compensator, the discrete-time butterworth velocity filtering was removed.

With the high adaptation gain, the adaptive controller without the added compensator shows apparently unstable behaviour. The inertial parameter estimates often hit their limits, as shown in Figure 6.16. The torque also saturates heavily, as shown in Figure 6.18. When the non-linear compensator is added, the system

Figure 6.15: Effective Compensator Gain

appears to be stable. The torque does not saturate after an initial transient period, and the parameter estimates remain within their limits. The parameter estimates and torque signals for the compensated case are shown in Figures 6.17 and 6.19 respectively. Figure 6.20 shows the effective compensator gain for this case. The compensated controller also has much lower filtered tracking errors, as shown by the comparison in Figure 6.21. Figure 6.22 compares the predicted $\|s\|_{bound}[k]$ with the actual values of $\|s[k]\|$ from the experimental run of the multi-rate adaptive case with non-linear compensation added. The predicted bound $\|s\|_{bound}[k]$ envelopes the test results.

The experimental results described above have demonstrated that the local approach for deriving the error bounds is valid. The error bound predictions conservatively envelope the experimental results for the adaptive control case, with and without the non-linear compensation. For the PD control case, the bound $\|s\|_{bound}[k]$ accurately predicts the behaviour of the implementation (as well as the behaviour of the simulation as seen previously).

It has also been demonstrated that the proposed non-linear discrete-time compensator is capable of stabilizing the system for the case where high adaptation gains otherwise result in apparently unstable behaviour of the adaptive system. Thus the multi-rate experiments also confirm the theoretical and simulation results in this respect.

Figure 6.16: Parameter Estimates, Multi-Rate Adaptive Control without Compensator

Figure 6.17: Parameter Estimates, Multi-Rate Adaptive with Compensator

Figure 6.18: Input Torque, Multi-Rate Adaptive Control

Figure 6.19: Input Torque, Multi-Rate Adaptive with Compensator

Figure 6.20: Effective Compensator Gain, Multi-Rate Controller

Figure 6.21: Norm of Filtered Tracking Errors, $\|s\|$, Multi-Rate (- - - adaptive, compensated adaptive)

Figure 6.22: Norm of Filtered Tracking Errors, $\|s\|$, Multi-Rate Adaptive Control with Compensator (— actual, - - - predicted bound)

Chapter 7

Conclusions

7.1 Discussion and Conclusions

The sampled-data robot adaptive control problem stated in Chapter 3 has been investigated. Methods to predict the stability and tracking performance of the implementation of direct adaptive control laws for robotic manipulators were formulated and described in Chapters 4 and 5. It was demonstrated in Chapter 4 that stability is not guaranteed for Slotine and Li's direct adaptive control law. A discrete-time adaptive control law that does have guaranteed stability was formulated in Chapter 5, and a means of predicting the guaranteed bound on the tracking error was also presented.

Simulation of Slotine and Li's direct adaptive control algorithm has been used to demonstrate the effect on performance and stability when a continuous-time adaptive algorithm is converted to discrete-time to be implemented in a sampled-data system. The discretization of Slotine and Li's algorithm results in a sampled-data system for which stability has not been guaranteed. Furthermore, simulation results indicate that under certain conditions, the system appears to be unstable.

By formulating the entire sampled-data manipulator system in continuous-time, Lyapunov's second method has been used to examine the stability and error bounds. The derived stability condition has been used to formulate a discrete-time non-linear compensator to guarantee the stability of the sampled-data implementation, and improve the tracking performance. This stability condition also gives a prediction

of the bound on the filtered tracking error.

A multi-rate discrete-time controller implementation of the compensated adaptive controller has also been investigated. The stabilizing effect of the compensator was demonstrated through simulation of a two degree-of-freedom serial link manipulator.

Experimental results from PD and adaptive control of a two DOF direct drive manipulator implementation have been presented. The beneficial effects of the proposed non-linear discrete-time compensator for both reducing the filtered tracking error of the discretizations of Slotine and Li's direct adaptive controller, and for stabilizing the multi-rate discretization, have been demonstrated.

7.2 Future Research

A number of directions are recommended for continuation of the research described in this thesis:

1. implementation of the multi-rate form of the adaptive controller with the non-linear compensator added using a multi-processor Digital Signal Processor (DSP) hardware architecture to achieve higher sampling rates;
2. implementation of alternative forms of the discrete-time non-linear compensator belonging to the derived compensator structure family, to compare the performance benefits;
3. further analysis of the necessary conditions for avoiding torque saturation when employing the adaptive controllers, and the relationship to the parameter estimate convergence rate;
4. application of the stability analysis and compensation methodology to other contemporary adaptive control laws, and to robotic manipulators with three or more degrees-of-freedom.

These suggested areas of future research would be useful in further determining the practical value of both the analysis techniques presented in this thesis, and

the proposed discrete-time non-linear compensator for discretized adaptive control algorithms.

7.3 Summary of Contributions

The following is a summary of the major original contributions to the field of adaptive control of robotic manipulators which are embodied in this thesis research:

1. derivation of an explicit condition for stability of a sampled-data manipulator system using continuous-time Lyapunov stability theory;
2. demonstration that stability is not guaranteed for the discretization of Slotine and Li's algorithm;
3. formulation of a discrete-time non-linear compensator giving guaranteed stability and reduced tracking error;
4. derivation of a predicted bound on the filtered tracking error;
5. validation of the theoretical results by both simulation and experiment.

Bibliography

- [1] Abdallah, C., Dawson, D., Dorato, P., and Jamshidi, M., "Survey of Robust Control for Rigid Robots", *IEEE Control Systems Magazine*, February 1991, pp.24-30.
- [2] An, C.H., Atkeson, C.G., and Hollerbach, J.M., *Model-Based Control of a Robot Manipulator*. MIT Press, Cambridge, Massachusetts, 1988.
- [3] 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.
- [4] Astrom, K.J., and Hagglund, T., "Practical Experiences of Adaptive Techniques", *Proceedings of the American Control Conference*, San Diego, California, May 1990.
- [5] Astrom, K.J., and Wittenmark, B., *Adaptive Control*, Addison-Wesley, Reading, Massachusetts, 1989.
- [6] 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.
- [7] Chen, T., and Francis, B.A., "Input-Output Stability of Sampled-Data Systems". *IEEE Transactions on Automatic Control*, Vol.36, No.1, January 1991.
- [8] Cook, P.A., *Nonlinear Dynamical Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.

- [9] Corless, M.J., and Leitmann, G., "Continuous State Feedback Guaranteeing Uniform Ultimate Boundedness for Uncertain Dynamic Systems", *IEEE Transactions on Automatic Control*, Vol.AC-26, No.5, October 1981, pp.1139-1144.
- [10] Craig, J.J., *Introduction to Robotics — Mechanics and Control*, Addison-Wesley, Reading, Massachusetts, 1986.
- [11] Craig, J.J., *Adaptive Control of Mechanical Manipulators*, Addison-Wesley, Reading, Massachusetts, 1988.
- [12] 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.
- [13] Dawson, D.M., and Lewis, F.L., "Adaptive and Robust Control of a Robot Manipulator without Acceleration Measurements", *Proceedings of the 28th Conference on Decision and Control*, Tampa, Florida, December 1989, pp.2629-2631.
- [14] Drakunov, S.V., and Utkin, V.I., "On Discrete-Time Sliding Modes", *Proceedings of the IFAC Symposium on Nonlinear Control Systems (NOLCOS)*, Capri, Italy, 1989, pp.273-278.
- [15] Duarte, M.A., and Narendra, K.S., "Combined Direct and Indirect Approach to Adaptive Control", *IEEE Transactions on Automatic Control*, Vol.34, No.10, October 1989, pp.1071-1075.
- [16] 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.
- [17] Franklin, G.G., Powell, J.D., and Workman, M.L., *Digital Control of Dynamic Systems*, 2nd ed., Addison-Wesley, Reading, Massachusetts, 1990.
- [18] Goodwin, G.C., and Sin, K.S., *Adaptive Filtering, Prediction and Control*, Prentice-Hall, Englewood Cliffs, New Jersey, 1984.
- [19] Hahn, W., *Theory and Application of Liapunov's Direct Method*, Prentice-Hall, Englewood Cliffs, New Jersey, 1963.

- [20] 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.
- [21] 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.
- [22] Hsu, P., and Sastry, S., "The Effect of Discretized Feedback in a Closed Loop System", *Proceedings of the 26th Conference on Decision and Control*, Los Angeles, California, December 1987.
- [23] Kalman, R.E., and Bertram, J.E., "Control System Analysis and Design Via the Second Method of Lyapunov: II. Discrete-Time Systems", *Transactions of the ASME, Journal of Basic Engineering*, Vol.82, Series D, No.2, June 1960, pp.394-400.
- [24] 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.
- [25] 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.
- [26] Leitmann, G., "On the Efficacy of Nonlinear Control in Uncertain Linear Systems", *Journal of Dynamic Systems, Measurement and Control*, Vol.102, June 1981, pp.95-102.
- [27] Li, W., and Slotine, J.-J.E., "Parameter Estimation Strategies for Robotic Applications", *Proceedings of the ASME Winter Annual Meeting*, 1987.
- [28] Li, W., and Slotine, J.-J.E., "Indirect Adaptive Robot Control", *Proceedings of the IEEE Conference on Robotics and Automation*, 1988.

- [29] Lindorff, D.P., *Theory of Sampled-Data Control Systems*, John Wiley and Sons, New York, 1965.
- [30] Ljung, L., *System Identification: Theory for the User*, Prentice-Hall, Englewood Cliffs, New Jersey, 1987.
- [31] 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.
- [32] Middleton, R.H., "Adaptive Control for Robot Manipulators Using Discrete Time Identification", *IEEE Transactions on Automatic Control*, Vol.AC-35, No.5, May 1990, pp.633-637.
- [33] Morando, A., Horowitz, R., and Sadegh, N., "Digital Implementation of Adaptive Control Algorithms for Robot Manipulators", *Proceedings of the IEEE Conference on Robotics and Automation*, May 1989, pp.1656-1662.
- [34] Narendra, K.S. and Annaswamy, A.M., *Stable Adaptive Systems*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989.
- [35] Narendra, K.S., and Boskovic, J.D., "A Combined Direct, Indirect and Variable Structure Method for Robust Adaptive Control", *Proceedings of the American Control Conference*, San Diego, California, May 1990.
- [36] Niemeyer, G., and Slotine, J-J.E., "Performance in Adaptive Manipulator Control", *International Journal of Robotics Research*, Vol.10, No.2, April 1991, pp.149-161.
- [37] Ogata, K., *Modern Control Engineering*, 2nd ed., Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
- [38] Ortega, R., and Spong, M.W., "Adaptive Motion Control of Rigid Robots: A Tutorial", *Proceedings of the 27th Conference on Decision and Control*, Austin, Texas, December 1988.

- [39] Rajguru, S., *Analysis and Design of a Direct Drive Robot*, Master's Thesis, Carleton University, Ottawa, Ontario, 1989.
- [40] Reed, J.S., and Ioannou, P.A., "Instability Analysis and Robust Adaptive Control of Robotic Manipulators", *Proceedings of the 27th Conference on Decision and Control*, Austin, Texas, December 1988.
- [41] Rohrs, C.E., Athans, M., Valavani, L., and Stein, G., "Some Design Guidelines for Discrete-time Adaptive Controllers", *Automatica*, Vol.20, No.5, 1984, pp.653-660.
- [42] Rohrs, C.E., Valavani, L., Athans, M., and Stein, G., "Robustness of Continuous-Time Adaptive Control Algorithms in the Presence of Unmodeled Dynamics", *IEEE Transactions on Automatic Control*, Vol.AC-30, No.9, September 1985.
- [43] Sadegh, N., and Horowitz, R., "Stability and Robustness Analysis of a Class of Adaptive Controllers for Robotic Manipulators", *The International Journal of Robotics Research*, Vol.9, No.3, June 1990.
- [44] Sadegh, N., and Guglielmo, K., "Design and Implementation of Adaptive and Repetitive Controllers for Mechanical Manipulators", *IEEE Transactions on Robotics and Automation*, Vol.8, No.3, June 1992, pp.395-400.
- [45] Sadegh, N., Horowitz, R., Kao, W-W., and Tomizuka, M., "A Unified Approach to the Design of Adaptive and Repetitive Controllers for Robotic Manipulators", *Journal of Dynamic Systems, Measurement and Control*, Vol.112, December 1990.
- [46] Sarpturk, S.Z., Istefanopulos, Y., and Kaynak, O., "On the Stability of Discrete-Time Sliding Mode Control Systems", *IEEE Transactions on Automatic Control*, Vol. AC-32, No.10, October 1987, pp.930-932.
- [47] 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.

- [48] Schwartz, H.M., Warshaw, G., and Janabi, T., "Issues in Robot Adaptive Control", *Proceedings of the American Control Conference*, San Diego, California, May 1990, pp.2797-2805.
- [49] Singh, S.N., "Adaptive Model Following Control of Nonlinear Robotic Systems", *IEEE Transactions on Automatic Control*, Vol. AC-30, No.11, November 1985, pp.1099-1100.
- [50] Sira-Ramirez, H., "Non-Linear Discrete Variable Structure Systems in Quasi-Sliding Mode", *International Journal of Control*, Vol.54, No.5, 1991, pp.1171-1187.
- [51] 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.
- [52] Slotine, J-J.E., and Li, W., "Theoretical Issues in Adaptive Manipulator Control", *Proceedings of the 5th Yale Workshop on Applications of Adaptive Systems Theory*, May 1987.
- [53] 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.
- [54] Slotine, J-J.E., and Li, W., "Composite Adaptive Control of Robot Manipulators", *Automatica*, Vol.25, No.4, 1989, pp.509-519.
- [55] Spong, M.W., Ortega, R., and Kelly, R., Comments on "Adaptive Manipulator Control: A Case Study", *IEEE Transactions on Automatic Control*, Vol.35, No.6, June 1990, pp.761-762.
- [56] Spong, M.W., and Vidyasagar, M., *Robot Dynamics and Control*, John Wiley and Sons, New York, 1989.
- [57] Stewart, G.W., *Introduction to Matrix Computations*, Academic Press, New York, 1973.

- [58] Tomei, P., "Adaptive PD Controller for Robot Manipulators", *IEEE Transactions on Robotics and Automation*, Vol.7, No.4, August 1991, pp.565-570.
- [59] Tomizuka, M., "Adaptive Control for Robot Manipulators", *Proceedings of the 5th Yale Workshop on Applications of Adaptive Systems Theory*, May 1987.
- [60] Vidyasagar, M., *Non-Linear Systems Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1978.
- [61] Warshaw, G.D., *Investigations of Adaptive Control for a Direct Drive Robotic Manipulator*, Master's Thesis, Carleton University, Ottawa, Ontario, 1990.
- [62] Warshaw, G.D. and Schwartz, H.M., "Compensation of Sampled-Data Robot Adaptive Controllers for Improved Stability", *Proceedings of the 1993 Canadian Conference on Electrical and Computer Engineering*, Vancouver, British Columbia, September 1993, pp.845-850.
- [63] Warshaw, G.D., Schwartz, H.M., and Aşner, H., "Stability and Performance of Sampled-Data Robot Adaptive Controllers", *Proceedings of the 1992 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Raleigh, North Carolina, July 1992, pp.95-104.
- [64] Whitcomb, L.L., Rizzi, A.A., and Koditschek, D.E., "Comparative Experiments with a New Adaptive Controller for Robot Arms", *IEEE Transactions on Robotics and Automation*, Vol.9, No.1, February 1993, pp.59-70.
- [65] Willems, J.L., *Stability Theory of Dynamical Systems*, John Wiley and Sons, New York, 1970.

Appendix A

Simulation Software Listing

A.1 Main Program Listing

```
program robots17;
```

```
{-----}
{Written by: Gabriel Warshaw}
{Revised: February 18, 1993}
{
{ This program simulates the motion of two joints of 2 DOF
{ serial link robotic manipulator. The controller uses
{ Slotine & Li's adaptive control algorithm (adaptive
{ feedforward method plus PD feedback) with a non-linear compensator
{ added.
{-----}

{ A fourth order Runge-Kutta method is used to solve the
  nonlinear differential equations which model the
  manipulator's behavior.

  Calculation of the tracking error is of the form:
      e(t)=yd(t)-y(t)

  The system parameters are estimated
  using the Slotine & Li's Direct method of adaptation. A
  discrete-time update of the parameters (and related measurements),
  is used.
  [This program also includes some code used to implement a
   simulation of a continuous-time implementation of Slotine & Li's
   adaptive controller, but this option has been disabled since the
   non-linear compensator does not apply to the continuous-time case.]

   A compensation term is added to the control law to improve
   the stability of the sampled-data system.
   A multi-rate form of the controller is implemented, where
   the parameter update can occur at a slower rate than the servo-loop
   (PD + feedforward + compensator).
   Hard bounds on the allowable range of parameter estimates, and
   torque limiting to simulate saturation, are both implemented.
   Gaussian noise can be added to the measurements of position and
   velocity.
   A disturbance torque can be added to the joint torque input.

  The structure of the dynamics is formulated to ensure the
  skew-symmetry of the matrix (M_dot - 2C).
  Taking advantage of the symmetry of the dynamics, only 3
  parameters need to be estimated.

  Two possible desired trajectories are selectable:
  (a) a Persistently Exciting (PE) path, which gives a 'loopy'
  path in cartesian space;
  (b) a non-PE path, which is simply regulation to a fixed
  position from another initial position.}

{-----}

uses simcode;

const maxpoints=400; {maximum number of data points to be stored}
      nstate=11; {number of robot states=4 (for runge-kutta solver),
                  includes parameter estimates=3 and desired
                  trajectory states=4 for continuous simulation}
```



```

nparam=3; {number of model parameters}
ndata=12; {number of data items for output to file}
lambda1:double = 10.; {used for calculating filtered errors}
                        {lambda = 10 gives effective Kp = 10*Kv}
lambda2:double = 10.;
a:double =100.0; {feedback compensation term gain}
norm_sprime:double = 0.05; {compensation term parameter}
                        {bounds on allowable range of parameter estimates}
para_min: array[1..nparam] of double = (0.0,2*(-0.18),0.0);
para_max: array[1..nparam] of double = (2*0.3,0.0,2*0.06);
sigmap = 0.0; {noise on position, rad}
sigmav = 0.2; {noise on velocity, rad/sec}
torque_sat:double=10.0; {value for motor torque
saturation limits (N-m)}

type
  state_vector = array[1..nstate] of double;
  param_vector = array[1..nparam] of double;

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 *)
           state_vector;

  phi1,phi2, (* Linear regression vectors for joint 1 & 2 *)
  para, (* Parameter vector for both joints *)
  suma: (* vector of sums used for discrete adaptation of params.*)
         param_vector;

  params: (* Matrix of parameter estimates to be saved *)
           array[1..maxpoints,1..nparam] of double;

  data : (* Matrix which contains the state data to be saved *)
           array[1..maxpoints,1..ndata] of double;

  t,      (* Time in simulation *)
  freqs, (* Sampling frequency for servo loop & runge-kutta simulation*)
  freqa, (* Sampling frequency for adaptive loop *)
  dt,     (* Time interval for control loop and simulation *)
  dta,    (* Time interval for adaptation loop *)
  ftime,  (* Final time of the simulation *)
  w,      (* fundamental frequency for desired trajectory *)
  ampl,   (* amplitude scaling for desired trajectory *)
  theta1,theta2, {intermediate values used for computing efficiency}
  err1,err2, (* robot position error motor 1 & 2 *)
  derr1,derr2, (* error in the velocity of motor 1 & 2 *)
  kv,     (* Velocity feedback gain *)
  u1nom,u2nom, (* Feedforward component of control inputs, motor 1 & 2 *)
  control1,control2, (* control torques for motor 1 & 2 *)
  u1in,u2in, (* torque input for joint 1 & 2 *)
  tau1,tau2, (*disturbance torque for joint 1 & 2 *)
  a1,a2,a3,a4,a5,a6,a7,a8,a9,a10, {manipulator inertial constants}
  yn1,yn2,yn3,yn4, {measurements with added noise}
  cosyn3,sinyn3, {cos and sin functions of yn3}
  yda1,yda2, {desired angular velocities for motor 1 & 2}

```

```

yd0a1,yd0a2, { " " " ,last time step}
yrv1,yrv2, {reference velocity for motor 1 & 2 }
yra1,yra2, { " acceleration " " " }
s1,s2, {reference trajectory errors for motor 1 & 2}
silast,s2last, {s1,s2 from previous time step}
npoints, {number of output data points desired}
max_terr1,max_terr2, {max.abs.value of tracking errors}
max_normds, {max. value of the norm of the change in filtered error}
gamma, {gain for adaptation law (all gamma vector elements)}
v3,v4,v5,v6, {random number values}
g, {gain for control compensation term}
norms {norm of filtered error vector}
      :double;

{***special note: when using 8087 compile option, use of "real"
type gives run-time error 207 --> use of "double" real solves
the problem (something to do with stack overflow & type
conversion?)}

rerun,mode,npar,counter,i,j,jj,select,
skip,skipcount, {number of time steps to skip for
                  data output,current skip count}
skipa,skipcounta: {as above, but for adaptation loop}
integer;

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

noise,firststep,pe_path,input_disturb,saturation,param_bound,
continuous_update: boolean;

choice,yes: char;

exitsave: pointer; {pointer saved for exit pointer used by system}

(*****)
function inverse_cos(value: double): double;

(* 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
  l1,l2,m1,m2 : double;
  i,j : integer;

begin
  writeln(' Do you wish to accept default values for');
  writeln(' sampling freq. and velocity feedback gain: 100, 8 (y/n)?');
  readln(yes);
  if (yes='y') or (yes='Y') then begin

```

```

    freqs:=100.;kv:=8.0;
    end
  else begin
    writeln(' Enter the sampling frequency (for servo loop)');
    readln(freqs);
    write(' Enter velocity feedback gain');
    writeln(' (effective position gain calculated by default)');
    readln(kv);
  end; {if}
  dt:=1./freqs;

(* compute the inertial constants *)
  l1:=0.3;l2:=0.3;m1:=2.0;m2:=2.0;

  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;      {0.5*a2}
  a5 := l1*l2*m2;          {a2}
  a6 := 0.5*l1*l2*m2;      {0.5*a2}
  a7 := (1.0/3.0)*l2*l2*m2; {a3}
  a8 := 0.5*l1*l2*m2;      {0.5*a2}
  a9 := (1.0/3.0)*l2*l2*m2; {a3}
  a10:= a6;                {0.5*a2}

  yes := 'y';

(* Read in the initial parameter vector (for discrete update
   case only *)

  for i := 1 to nparam do
    read(indata, para[i]);

  {reset all regression terms}
  for i:=1 to nparam do begin
    phi1[i]:=0.;phi2[i]:=0.;
  end;

end;

(*****
procedure initialize;

begin
  continuous_update:=false;
  writeln(' Enter value for adaptation gain (gamma_inverse):');
  readln(gamma);
  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 if disturbance to be added to torque inputs (y/n)');
  readln(choice);
  if (choice='y')or (choice='Y') then input_disturb:=true
    else input_disturb:=false;
  write(' Enter type of desired trajectory -- ');
  writeln(' l = loopy (P.E.), f = fixed (not P.E.)');
  readln(choice);
  if choice='f' then pe_path:=false else pe_path:=true;
  if pe_path=true then begin
    ftime:=5.; w:=1.257; ampl:=1.0;
    writeln(' -- Loopy (P.E.) path selected');
  end

```

```

    else begin
        ftime:=5.; w:=0.;
        writeln(' -- Fixed position selected (not P.E.)');
    end;
    writeln(' -- Default values: final time per run = ',ftime:4:2,' seconds,');
    writeln('                        path frequency = ',w:5:3,' rad/sec. ');
    writeln('                        amplitude scaling = ',ampl:4:2);
    writeln(' Do you wish to change final time per run,');
    writeln('      or amplitude scaling for trajectory (y/n)?');
    readln(choice);
    if choice='y' then begin
        writeln(' Enter new value for final time: ');
        readln(ftime);
        writeln(' Enter new amplitude scaling factor: ');
        readln(ampl);
    end;

    writeln(' Enter if motor torque saturation is desired (y/n)');
    readln(choice);
    if (choice='y') or (choice='Y') then saturation:=true
    else saturation:=false;

    writeln(' Enter the adaptation loop frequency: ');
    readln(freqa);
    skipa:=trunc(freqs/freqa);
    writeln(' --> Adaptation will be calculated every <',skipa,
        '> cycles of servo loop');

    dta:=1./freqa;

    writeln(' Enter number of data points to be output');
    readln(npoints);
    skip:=round((ftime/dt)/npoints);
    writeln(' --> Data will be output every <',skip,
        '> time steps');

    writeln(' Enter number of times to do simulation (full cycle of path)');
    readln(rerun);
    firststep:=true;

end;
(*****)

procedure initializec;
var
    i: integer;
begin
    (* initialize the inputs *)
    u1in := 0.0; u2in:=0.0;
    for i:=1 to nstate do f[i] := 0.0;

    (* initialize the manipulator state, i.e. position, vel. and accel. *)

    if pe_path=true then begin
    {for "loopy path"}
        y[1] := 1.5;
        y[2] := 0.0;
        y[3] := pi/2.0 - 1.5;
        y[4] := ampl*3.0*w;
        yda1:=ampl*12.0*w*w;

```

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

```

yda2:=0.0;
if continuous_update then begin {i.c's for oscillators used to
    produce sine desired trajectories -- gives sine
    for joint 2 with lead of 90 deg.}
    y[1]:= 0.;
    y[2]:= (2.*w)/2.;
    y[3]:= 0.5;
    y[4]:= 0.;
    yda1:= 0.; yda2:= -sqr(3.8*w);
    y[8]:=y[1]; y[9]:=y[2]; y[10]:=y[3]; y[11]:=y[4];
end
end

else begin
{for fixed position}
y[1] := 0.1;
y[2] := 0.0;
y[3] := 0.1;
y[4] := 0.0;
yda1:=0.;
yda2:=0.;
end;

{initialize states for continuous update of param. estimates}
for i:=1 to nparam do y[i+4]:=para[i];

t := 0.0; (* initial time is zero *)
counter := 0; (* initialize data counter to zero *)
skipcount:=skip; (* initialize output skip counter *)
skipcounta:=0; (* initialize adaptation skip counter *)

(* initialize the desired state to the initial state *)
for i := 1 to nstate do
begin
    yd0[i] := y[i];
    yd[i] := y[i];
end;

{initialize data at t=0 }
taud1:=0.; taud2:=0.;
err1:=0.; err2:=0.;
derr1:=0.; derr2:=0.;
s1:=0.; s2:=0.;
yra1:=yda1; yra2:=yda2;
(* sigma_s:=0.;*)
control1:=0.; control2:=0.;

max_terr1:=0.0;max_terr2:=0.0; {initialize maximums}
max_normds:=0.0;

for i:=1 to nparam do suma[i]:=0.0;
end;

(*****)
procedure compute_desired_state;

(* This procedure computes the desired position
and velocity of the manipulator. *)
(* It also computes the disturbance torques to be added to the
joint inputs *)

var
i: integer;

```

```

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

  {Compute desired state for current time step}

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

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

  else begin
    (*fixed position at 0.0*)
    yd[1] :=0.;
    yd[2] :=0.;
    yd[3] :=0.;
    yd[4] :=0.;

    yda1:=0.;
    yda2:=0.;
    end;

  {compute the disturbance torques}
  if input_disturb=true then begin
    taud1:=9.;taud2:=9.;
    end
    {constants}
  else begin
    taud1:=0.;taud2:=0.;
    end;
  end;

  (*****)
  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 one degree of freedom of a
  manipulator, which is operating in the horizontal plane. *)

  var
    x1,x2,x3,x4,sny3,csy3 : double;
    i,k: integer;

  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;

if continuous_update then begin
    if noise then begin
        nrand(v3,v4);
        nrand(v5,v6);
        v3:=sigmav*v3;
        v4:=sigmap*v4;
        v5:=sigmav*v5;
        v6:=sigmap*v6;
    end
    else begin
        v3 := 0.;
        v4 := 0.;
        v5 := 0.;
        v6 := 0.;
    end;{if}

    {calculate states with noise added}
    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);

    {Compute desired state for current time}
    if pe_path=true then begin
        {This trajectory follows a sine paths in joint space, using
        2nd order dynamics as oscillators}
        f[8] := y[9]; {yd1''}
        f[9] := -sqr(2.0*w)*y[8]; {yd1''}
        f[10] := y[11]; {yd2''}
        f[11] := -sqr(3.8*w)*y[10]; {yd2''}
    end

    else begin
        (*fixed position at 0.0*)
        yd[1] :=0.;
        yd[2] :=0.;
        yd[3] :=0.;
        yd[4] :=0.;
        yda1:=0.;
        yda2:=0.;
    end;

    {compute the disturbance torques}
    if input_disturb=true then begin
        taud1:=9.;taud2:=9.; {constants}
    end
    else begin
        taud1:=0.;taud2:=0.;
    end;

    yd[1]:=y[8]; yd[3]:=y[10];
    {compute the tracking errors}
    err1 := yd[1] - yn1; {errors including measurement noise}

```



```

    derr1 := f[8] - yn2;
    err2 := yd[3] - yn3;
    derr2 := f[10] - yn4;

    {compute the reference trajectories}
    yrv1:=f[8]+lambda1*err1; {velocities}
    yrv2:=f[10]+lambda2*err2;
    yra1:=f[9]+lambda1*derr1; {accelerations}
    yra2:=f[11]+lambda2*derr2;

    {compute the reference trajectory, errors = filtered tracking errors}
    s1:=yn2-yrv1;
    s2:=yn4-yrv2;

    {calculate regression vectors}
    phi1[1] := yra1;
    phi1[2] := (yra1+0.5*yra2)*cosyn3-0.5*(yn4*yrv1+(yn2+yn4)*yrv2)*
               sinyn3;
    phi1[3] := yra2;

    phi2[2] := 0.5*(yra1*cosyn3+yn2*yrv1*sinyn3);
    phi2[3] := yra1+yra2;

    {calculate control torque inputs for both motors, including
    disturbances}
    control1:=0.; control2:=0.;
    for i:=1 to nparam do control1:=control1+phi1[i]*y[i+4];
    control1:=control1-kv*s1;
    for i:=1 to nparam do control2:=control2+phi2[i]*y[i+4];
    control2:=control2-kv*s2;

    if saturation then begin
        control1:=sat(control1,torque_sat);
        control2:=sat(control2,torque_sat);
    end;
    u1in:=control1+taud1;
    u2in:=control2+taud2;

    {compute derivatives for parameter estimates}
    for i:=1 to nparam do
        f[i+4]:=-gamma*(phi1[i]*s1+phi2[i]*s2 (* +sigma_s*y[i+4]*));
    end;{if}

    (* compute the derivative vector f(t,y(t)) for robot states *)
    f[1] := y[2]; {y1'}
    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]; {y2'}
    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;

(*****
{I rungekut.inc}      (*include runge-kutta routine*)
(*****
procedure compute_input;

(* This procedure computes the actual input to the
robot's motors, using the feedforward (computed torque)
technique. *)

```

```

var i:integer; {loop index}
begin
  {calculate norm of filtered error vector}
  norms:=sqrt(sqr(s1)+sqr(s2));
  if norms>norm_sprime then g:=a*(norms-norm_sprime)
    else g:=0.0;

  {calculate control torque inputs for both motors, including
  disturbances}
  control1:=0.; control2:=0.;
  for i:=1 to nparam do control1:=control1+phi1[i]*para[i];
    control1:=control1-(kv+g)*s1;
  for i:=1 to nparam do control2:=control2+phi2[i]*para[i];
    control2:=control2-(kv+g)*s2;

  if saturation then begin
    control1:=sat(control1,torque_sat);
    control2:=sat(control2,torque_sat);
  end;
  u1in:=control1+taud1;
  u2in:=control2+taud2;

end;
(*****
procedure ident_direct;

(* This procedure identifies the parameters which describe
the dynamics of the manipulator. An adaptation law based
on Lyapunov stability and the Direct method is used, according
to Slotine and Li's method. *)

var
  k:integer; (*loop index*)

begin
  if continuous_update=false then
    (* discrete update of parameter vector with new estimates,
    every "skipa" number of timesteps, with bounds on estimates*)

    {update sums for discrete update of parameters at rate "freqa"}
    for k:=1 to nparam do suma[k]:=suma[k]+(phi1[k]*s1+phi2[k]*s2);

    if skipcounta = skipa then begin
      param_bound:=false;
      for k:=1 to nparam do begin
        para[k]:=para[k]-dt*gamma*suma[k] (* +sigma_s*para[k] *) ;
        if para[k]<para_min[k] then begin
          para[k]:=para_min[k]; param_bound:=true
        end
        else if para[k]>para_max[k] then begin
          para[k]:=para_max[k]; param_bound:=true
        end
      end;{for}
      skipcounta:=0;
      for k:=1 to nparam do suma[k]:=0.0;
    end;{if}
    skipcounta:=skipcounta+1;
  end;

```

```

(*****)
procedure regression;

(* This procedure defines the regression vector to be used
   by the system identification algorithm, and also calculates
   various error signals *)

begin
  compute_f;

  if noise then begin
    nrand(v3,v4);
    nrand(v5,v6);
    v3:=sigmav*v3;
    v4:=sigmap*v4;
    v5:=sigmav*v5;
    v6:=sigmap*v6;
  end
  else begin
    v3 := 0.;
    v4 := 0.;
    v5 := 0.;
    v6 := 0.;
  end;{if}

  {calculate states with noise added}
  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);

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

  {compute the reference trajectories}
  yrv1:=yd[2]+lambda1*err1; {velocities}
  yrv2:=yd[4]+lambda2*err2;
  yra1:=yd[1]+lambda1*derr1; {accelerations}
  yra2:=yd[3]+lambda2*derr2;

  {compute the reference trajectory errors = filtered tracking errors}
  s1last:=s1;s2last:=s2;
  s1:=yn2-yrv1;
  s2:=yn4-yrv2;

  {calculate regression vectors}
  phi1[1] := yra1;
  phi1[2] := (yra1+0.5*yra2)*cosyn3-0.5*(yn4*yrv1+(yn2+yn4)*yrv2)*
             sinyn3;
  phi1[3] := yra2;

  phi2[2] := 0.5*(yra1*cosyn3+yn2*yrv1*sinyn3);
  phi2[3] := yra1+yra2;

end;

```

```

(*****)
procedure collect_data;

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

var i: integer; {loop index}

begin
  counter := counter + 1;
  if counter <= maxpoints
  then
    begin
      data[counter,1] := yd[1]; (*desired pos.1*)
      data[counter,2] := yra1;  (*ref. acceleration 1*)
      data[counter,3] := err1;  (*tracking error1*)
      data[counter,4] := s1;    (* filtered error 1*)
      data[counter,5] := yd[3]; (*desired pos.2*)
      data[counter,6] := yra2;  (*ref. acceleration 2*)
      data[counter,7] := err2;  (*tracking error2*)
      data[counter,8] := s2;    (* filtered error 2*)
      data[counter,9] := t;     (* time*)
      data[counter,10] := g;    (* gain of control compensation *)
      data[counter,11] := control1; (* control torque for motor 1 *)
      data[counter,12] := control2; (* control torque for motor 2 *)

      (*parameter values*)
      if continuous_update then
        for i:=1 to nparam do params[counter,i] :=y[i+4]
      else {discrete update}
        for i:=1 to nparam do params[counter,i] :=para[i];

      writeln(' t= ',t:5:2,' sec, norm_of_s = ',sqrt(s1*s1+s2*s2):7:3);

      if g>0.0 then writeln(' t= ',t:5:2,
        ' sec, Compensation active; Gain = ',g:6:3);
      if saturation then
        if (abs(control1)>torque_sat) or (abs(control2)>torque_sat) then
          writeln(' >> t= ',t:5:2,' sec, Torque Saturating');

      if param_bound then writeln(' ----> t= ',t:5:2,
        ' sec, Parameter(s) at bound');

    end; {if counter...}
end;

(*****)
procedure find_max_track_errors;

{finds maximum absolute values of tracking
 errors and the norm of the change in the filtered error,
 for the current run}

var abs_val,norm_delta_s: double;

begin
  abs_val:=abs(err1);
  if abs_val > max_terr1 then max_terr1:=abs_val;
  abs_val:=abs(err2);
  if abs_val > max_terr2 then max_terr2:=abs_val;
  norm_delta_s:=sqrt(sqr(s1-s1last)+sqr(s2-s2last));

```

```

{norm of the change in 's' between time steps}
  if norm_delta_s > max_normds then max_normds:=norm_delta_s;
end;

(*****)
procedure simulate;
begin
  (* The simulation loop starts here. *)
  while t <= ftime do
    begin
      if continuous_update=false then begin
        (* Compute the desired state one time step into the future *)
        compute_desired_state;

        (* Define the regression vectors *)
        regression;

        (* Compute the identification algorithm *)
        ident_direct;

        (* Compute the input into the robot motors *)
        compute_input;
        end; {if}

        (* 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;

        (* find maximum tracking errors *)
        find_max_track_errors;

        t := t + dt;
      end;
    end;
  end;

  (*****)
  procedure save_data_to_files;
  begin
    {output values of parameters to a file}
    writeln('--> Saving data to files....');
    writeln(outpara);

    if continuous_update then
      for i := 1 to nparam do write(outpara,y[i+4])
    else
      for i := 1 to nparam do write(outpara,para[i]);
    writeln(outpara);

    (* write out the data needed for plotting the results *)
    if counter>maxpoints then counter:=maxpoints;
  end;
end;

```

```

for i := 1 to counter do
  begin
    for j := 1 to ndata do write(outdata,data[i,j]:15);
    for j := 1 to nparam do write(xdata,params[i,j]:15);
    writeln(outdata);
    writeln(xdata);
  end;

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

end;

(*****)
{$F+}
procedure exit_handler;{$F-}

(*this procedure handles normal and abnormal termination --
  it saves all data and closes files*)

begin
  exitproc:=exitsave; {restore original exit pointer}

  if exitcode<>0 then begin
    writeln('****Run-Time Error:');
    writeln('          Error code is ',exitcode);
  end;

  collect_data;

  save_data_to_files;

  writeln '** SIMULATION COMPLETED **');
end;
(*****)
(*****)
begin {main}

{set up pointer to exit handler procedure}
  exitsave:=exitproc;
  exitproc:=@exit_handler;

{start-up message to user}
  writeln;
  writeln('++++++ ROBOT ADAPTIVE CONTROL SIMULATION (2 DOF) +++++');
  writeln;

  assign(outdata,'data\outdat.dat');
  rewrite(outdata);
  assign(xdata,'data\outdatp.dat');
  rewrite(xdata);
  assign(outpara,'data\params.dat');
  rewrite(outpara);
  assign(partrend,'data\partrend.dat');
  rewrite(partrend);
  assign(indata,'data\ins.dat');
  reset(indata);

(* Initialize all the data in the system *)
  initializea;

```

```

(* Simulate the robot as many times as you like *)
(* (successive runs use parameter estimates from previous runs) *)
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
      begin
        initializec;
        writeln('--> Running simulation #',i);
        simulate;
        {write values of parameter vector for this run}
        for jj:=1 to nparam do begin
          if continuous_update then para[jj]:=y[jj+4];
          write(partrend,para[jj]:15)
        end;
        write(partrend,max_terr1:10:3,max_terr2:10:3,
          max_normds:10:3);
        writeln(partrend);
      end;
    end;
    (* Do you want another simulation ? *)
    writeln('Current parameter estimates are:');
    if continuous_update then
      for i := 1 to nparam do write(y[i+4]:13)
    else
      for i := 1 to nparam do write(para[i]:13);
    writeln;
    writeln('--> Do you want to rerun, losing the current data (y/n)?');
    readln(yes)
  end;
  {"exit_handler" procedure called by default}
end.

```

A.2 Program Unit for General Purpose Simulation Routines

```

-----
{ unit SIMCODE.PAS
{   written by:
{       Gabriel D. Warshaw
{   Revised: August 15, 1990
{
{   This file contain a collection of functions and procedures to be
{   used for simulation of the control of the 2 DOF direct drive robot.
{
{   The interface section shows the prototypes of the functions and the

```

```
{ procedures while the implementation section contain the actual code }
{ If you find a bug in this implementation, DO NOT MODIFY THIS FILE }
{ Ask the writer to fix the bug. }
{-----}
```

```
unit simcode;
```

```
interface
```

```
(*****)
procedure nrand(var x1,x2:double);
```

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

```
(*****)
function sat(x, limit: double):double;
```

```
{this code implements a saturation function}
```

```
(*****)
implementation
(*****)
```

```
procedure nrand(var x1,x2:double);
```

```
var v1,v2,u1,u2,s,sroot:double;{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;
```

```
(*****)
function sat(x, limit: double):double;
```

```
var xabs: double;
```

```
begin
```

```
xabs:=abs(x);
```

```
if xabs>limit then sat:=limit*x/xabs
```

```
else sat:=x;
```

```
end;
```

```
end.
```


A.3 Program "Include File" for Runge-Kutta Routine

```
{Include file: RUNGEKUT.INC}

{-----}
{Written by: Gabriel D. Warshaw, Howard Schwartz    }
{Revised: August 15, 1990 (GDW)                    }
{ }
{ This file contains the code for the 4th order runge-kutta routine}
{ for numerical solution of systems of first order nonlinear      }
{ differential equations.    }
{   The routine is intended to solve the manipulator dynamics    }
{   equations.    }
{ This routine in turn calls a procedure which computes the values}
{ of the derivatives representing the nonlinear system of          }
{ equations.    }
{-----}

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 4th order
runge-kutta recursion;


$$y(t + dt) = y(t) + 1/6(k_0 + 2k_1 + 2k_2 + k_3)$$


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
calculated four times. *)

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

    if i = 2
    then
        begin
            for j := 1 to nstate 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;
        (* end of if statement *)

    if i = 3
    then
        begin
            for j := 1 to nstate 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;
        (* end of if statement *)

    if i = 4
    then
        begin
            for j := 1 to nstate 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[1,j] + 2.0*k[2,j] +
                        2.0*k[3,j] + k[4,j]);

                    (* The robot state has been updated *)
                end;
            end;
        end;
        (* end of loop for j=1,nstate *)
    end;
    (* end of loop for the if i=4 decision *)
end;
(* end of the for loop i=1,4 *)
end;
(* end of the procedure *)

```

Appendix B

Experimental Control Software Listing

B.1 Main Program Listing

```
program adaptive_robot_control;
```

```
{-----}
{Written by: Gabriel D. Warshaw}
{Revised: August 6,1993}
{
{This program performs adaptive control of 2 degrees-of-freedom of the}
{Carleton Direct Drive Manipulator, using Slotine and Li's Direct }
{algorithm, with a non-linear compensation term added.}
{-----}
```

The coupled non-linear dynamic model is used, including torque bias terms for the (time-varying) torque bias of the servo amplifiers. Five parameters are estimated.

A multi-rate form of the controller is implemented, where the parameter update can occur at a slower rate than the servo loop.

The manipulator is made to follow a pre-defined path in joint space. This path (position) is loaded from a file, and is pre-filtered to limit the bandwidth of the desired trajectory, and to generate desired velocity and acceleration.

A butterworth low-pass filter is used for velocity smoothing.

The program is interrupt-driven (instead of polling) for the A/D conversion.}

```
{-----}
```

```
uses dos,crt,datacard,filter,m_arith;
```

```
const
```

```
{i/f card register addresses}
```

```
ad_status_addr = $218;
```

```
ad_chan_addr = $219;
```

```
da0_loaddr = $21A; {same as a/d data low byte }
```

```
da0_hiaddr = $21B; { " " " hi " }
```

```
dd_cntrlreg_addr = $228;
```

```
dd_port0_addr = $229; {motor 1 position low byte}
```

```
dd_port1_addr = $22A; { " 1 " high " }
```

```
dd_port2_addr = $22B; { " 2 " low " }
```

```
dd_port3_addr = $22C; { " 2 " high " }
```

```
maxpoints = 750; {maximum number of data points to be saved}
```

```
maxvia = 1000; { " " " via " " " " }
```

```
nparam = 5; {number of model parameters to be estimated}
```

```
torque_limit=4.9; {control volt. amplitude limit i.e. s/w defined}
{ saturation point.}
```

```
njoint=3; {code for number of joints under control:
```

```
1 = joint 1
```

```
2 = joint 2
```

```
3 = joint 1 and 2 }
```

```
name_of_code='ctrl129';
```

```
max_vel=250.; {maximum possible velocity used to detect faulty
position data, in degrees/second}
```

```

lambda=10.0; {constant for calculation of filtered error}
gamma_scaling:array[1..nparam] of single=(1.,1.,0.01745,3283.0,
                                           3283.0);

freqs=750; {effective sampling frequency, Hz (for 2 a/d's),
           a/d interrupts set for 2*freqs}
freqa=50; {frequency of parameter update (must divide evenly
           into "freqs")}

data_items=18; {number of different data items for file output}

pole=-10.0; {location of (triple) pole for pre-filter}
           {-10.0 gives BW = 5.0 rad/sec = 0.80 Hz}

bandwidth=25.0; {bandwidth (Hz) for butterworth low-pass filter}

a=0.0179; {compensation term gain parameter}
norm_sprime=3.76/2.0; {compensation term "dead zone" parameter}
           {bounds on allowable range of parameter estimates}
para_min: array[1..nparam] of single = (0.0,0.0,-0.002,-1.0,-1.0);
para_max: array[1..nparam] of single = (0.02,0.01,0.002,1.0,1.0);

{interrupt constants}
PIC_00 = $20; {Programmable Interrupt Controller control register}
PIC_01 = $21; { " " " " " mask register}
EOI = $20; {end of interrupt command to PIC}

var
dummy_input, {dummy value of a/d input (binary)}
step, {current number of time step for desired trajectory}
nsteps, {number of steps for data collection during run}
numb_via, {number of trajectory via points}
divisor, {number used to index the via points}
counter, {current step used for data storage}
skip, {number of time steps to skip for data output}
skipa, {number of time steps to skip for para. update}
skipcount,skipcounta, {current skip count for data and para.update}
icount, {counter used in ISR for sequencing}
glitch_count1,glitch_count2
           {counters used for faulty position measure. detection}
:integer;
bkgnnd_count {count of no. of complete cycles of "background" tasks}
:longint;

hr,min,sec,frac {arguments from procedure which gets time}
:word;

pos1,pos2, {joint angular positions in degrees}
pos1last,pos2last, {last values of motor 1 and 2 positions}
pos1_init,pos2_init, {actual values of motor positions at start of run}
t, (* Time from start of run *)
dt,dt2, (* Time interval for a step, twice the time interval *)
dtdtover2, {time interval squared over 2 (speeds calculation)}
reject, {max. reasonable change in position over 1 sampling interval}
ftime, (* Final time of the run *)
time_via, {length of time for run of loaded trajectory}
freq_via, {sampling frequency for loaded via points}
gamma, (* gain for parameter adaptation law *)
g, {gain for compensation term}
norms, {norm of "s" vector}

```

```

max_normds, {max. value of the norm of the change in "s"}
tmax {time at which "max_normds" occurred}
      :single;

kv, {velocity feedback gain}
control, (* Control input for motors, in volts *)
posd, {array of desired positions for motor 1 and 2}
accd, { " " " " " accelerations " " " " " }
err, (* robot position error *)
derr, (* error in the velocity *)
velr,accr, {reference velocities and accelerations}
s, {filtered tracking error}
slast, {"s" from previous time step}
posdlast,veldlast, {desired pos.& vel. for previous step}
posdf,posdfm1, {arrays of pre-filtered desired ...}
veldf,veldfm1, {...positions, velocities and accelerations...}
accdi,accdfm1 {...for motor 1 and 2}
      : array[1..2] of single;

vel, {joint angular velocities in deg/sec}
velf,velfm1, {low-pass filtered velocity, and previous values}
velfdot,velfdotm1 {derivatives of low-pass filtered velocity}
      : state;

phi {regression matrix}
      : array[1..2,1..nparam] of single;

para, {parameter vector}
suma {vector of sums used for discrete adaptation of param's}
      :array[1..nparam] of single;

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

trajectory: array[1..2,1..maxvia] of single; {array of trajectory (via)
      points input from a file}

outdata,outinfo,indata,ininfo: text; {i/o file assignment names}

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

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

yes_comment, {variable for user to select input of comments}
key_press, {dummy variable for user key press action}
choice: char; {variable for input of user selection character}

fb:state3; {vector of input term coefficients for pre-filter}
fa:matrix3; {matrix of AR " " " " " " }

bfb:state; {vector of input term coeff's for butterworth filter}
bfa:matrix2; {matrix of AR " " " " " " }

old_level2_intr, {old level 2 interrupt vector to be saved}
exitsave:pointer; {pointer saved for exit pointer used by system}

(*****)
procedure check_status(first_addr,second_addr:integer;

```

```

                                var enable_status:boolean);

const bit6 = $40;

begin
    if(port[first_addr] AND port[second_addr] AND bit6) = bit6 then
        enable_status:=true
    else enable_status:=false;
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_loadaddr]+(port[da0_hiaddr])shl 8;
    port[ad_status_addr]:=$15; {set a/d to mode 1, clear error bit,
                                enable interrupt}
    if(timer_freq(2*freqs)) then {"timer_freq" sets timer for chosen
                                sample rate and detects error}
    begin
        writeln('Invalid sampling frequency !!');
        writeln('Program aborted');
        halt;
    end;

    {set d/a outputs to 0 Volts}
    send_volt_da(1,0);
    send_volt_da(2,0);

    {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)');
        key_press:=readkey;
    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.*)

var year,month,day,dayofweek: word;
    i:integer;

begin
    getdate(year,month,day,dayofweek);

    rewrite(outinfo);
    writeln(outinfo,'filename: ',filename,'          Date: ',month,'/',
              day,'/',year);

```

```

writeln(outinfo,'code used to produce data: "',name_of_code,'"');
writeln(outinfo,' data: sampling freq(Hz),runtime(sec)',
        ',skip interval,# data pts, joint #');
writeln(outinfo,' successive lines list: Kp,Kv');
writeln(outinfo);
writeln(outinfo,freqs,' ',ftime:6:2,' ',skip,' ',counter,' ',njoint);
writeln(outinfo,lambda*kv[1]:10:5,kv[1]:10:5);
writeln(outinfo,lambda*kv[2]:10:5,kv[2]:10:5);
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;
writeln(outinfo,'Gamma= ',gamma:12);
writeln(outinfo,'Gamma scaling vector= ');
for i:=1 to nparam do write(outinfo, gamma_scaling[i]:11);
writeln(outinfo);
writeln(outinfo,'Compensator parameters: a= ',a:10:3,
        ', norm_sprime= ',norm_sprime:10:3);
writeln(outinfo,'Max. norm of delta_s = ',max_normds:11,
        ' @ t= ',tmax:7:2,' sec');
writeln(outinfo,'param. update freq.= ',freqa,
        ' Hz, updates every ',skipa,' servo cycles');

close(outinfo);
end;

(*****)
procedure load_trajectory_file;

var i: integer; {loop index}

begin
{prompt user and load input files}
    writeln('Enter path and name for trajectory input file ');
    writeln(' (extensions ".via" for the data file and ".vin" for the');
    writeln(' matching information file will be appended automatically):');
    readln(filename);
    assign(indata,filename+'.via');
    reset(indata);
    assign(ininfo,filename+'.vin');
    reset(ininfo);

    writeln('--> data being loaded from file.....');
    readln(ininfo);
    readln(ininfo);
    readln(ininfo);
    readln(ininfo,freq_via,time_via,numb_via);
    for i:=1 to numb_via do readln(indata,trajectory[1,i],trajectory[2,i]);
    close(ininfo);
    close(indata);

    divisor:=round(freqs/freq_via);

end;

```



```

(*****)
procedure initialize_parameters;
(* This procedure initializes all the parameters in
   the system, including those input by the user. *)

   var i,j:integer; {loop indices}

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');
  assign(outinfo,filename+'.inf');

  write('Enter Motor 1 velocity feedback gain: ');
  readln(kv[1]);
  write('Enter Motor 2 velocity feedback gain: ');
  readln(kv[2]);
  write('Enter adaptation gain, Gamma: ');
  readln(gamma);
  writeln('Run will take ',time_via:6:2,' seconds. ');
  ftime:=time_via;
  write('Do you wish to specify a different run time (y/n)? ');
  readln(choice);
  if (choice='Y') or (choice='y') then begin
    write('Enter run time (seconds): ');
    readln(ftime); end{if};
  write('Enter number of data points to be stored during the run: ');
  readln(nsteps);
  skip:=round((ftime*freqs)/nsteps);
  writeln('--> At ',freqs,' Hz sampling, data will be stored'
    , ' every <',skip,'> time steps');
  skipa:=round(freqs/freqa);
  writeln('--> At ',freqs,' Hz sampling (servo loop), and ',freqa,
    ' Hz parameter update');
  writeln('   parameters are updated every <',skipa,'> time steps');

{initialize variables}
  skipcount:=skip;
  skipcounta:=1;
  t:=0.0;
  dt:=1.0/freqs; dt2:=dt*2.0; dtdtover2:=dt*dt/2.0;
  counter:=0;
  icount:=1;
  bkgnd_count:=0;
  glitch_count1:=1; glitch_count2:=1;
  done_flag:=false;
  step:=0;
  max_normds:=0.0;

  reject:=max_vel*dt; {calculate rejection value for faulty position
                       measurement}

  for i:=1 to nparam do begin
    para[i]:=0.0; {initialize parameter estimates}
    suma[i]:=0.0; {initialize para. adapt. sums}
  end; {for}
  for i:=1 to 2 do
    for j:=1 to nparam do
      {zero the regression matrix}

```

```

    phi[i,j]:=0.0;

    {set regression terms for torque bias parameters to constant values}
    phi[1,4]:=-1.0; phi[1,5]:=0.0;
    phi[2,4]:=0.0; phi[2,5]:=-1.0;

    for i:=1 to 2 do begin
        posd[i]:=0.0;
        posdf[i]:=0.0; veldf[i]:=0.0; accdf[i]:=0.0;
        velf[i]:=0.0; velfdot[i]:=0.0; s[i]:=0.0;
    end;{for}

    writeln('...computing pre-filter and butterworth filter parameters...');
    pre_filter3(dt,pole,fa,fb); {compute filter coefficients for desired path
                                3rd order pre-filter}

    butterworth2(dt,bandwidth*2.0*pi,bfa,bfb); {compute filter coeff's
                                                for butterworth low-pass filtering of velocity measurements}

end;

(*****
procedure compute_desired_state;

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

const twopi=6.283185;
    f=1.0;    {freq.for sine input (Hz)}
    w=twopi*f;
    amplitude=10.0; {amplitude of sine input (deg)}
    amplitude_2=1.5;
    w_2=2.7*w;

var index, i: integer;

begin
    {select the appropriate trajectory via point}
    index:=(step div divisor)+1;
    if index > numb_via then begin {return to begin. of trajectory}
        step:=0; index:=1; end; {if}
    for i:=1 to 2 do posd[i]:=trajectory[i,index];
    step:=step+1;

    {store desired states from last step}
    posdlast[1]:=posdf[1]; veldlast[1]:=veldf[1];
    posdlast[2]:=posdf[2]; veldlast[2]:=veldf[2];

    {pre-filter the desired positions, to ensure that
     the requested trajectory stays within the system bandwidth --
     generate pre-filtered position for one time step into the future}
    for i:=1 to 2 do begin
        posdfm1[i]:=posdf[i];
        veldfm1[i]:=veldf[i];
        accdfm1[i]:=accdf[i];
        posdf[i]:=fa[1,1]*posdfm1[i]+fa[1,2]*veldfm1[i]
            +fa[1,3]*accdfm1[i]+fb[1]*posd[i];
        veldf[i]:=fa[2,1]*posdfm1[i]+fa[2,2]*veldfm1[i]
            +fa[2,3]*accdfm1[i]+fb[2]*posd[i];
        accdf[i]:=fa[3,1]*posdfm1[i]+fa[3,2]*veldfm1[i]
            +fa[3,3]*accdfm1[i]+fb[3]*posd[i];
    end;

```

```

    end; {for}

    (*
    {sinusoidal input of position for motor 2}
    posdf[2]:=pos2_init+amplitude - (amplitude*cos(w*t)
        +amplitude_2*cos(w_2*t) );
    veldf[2]:=amplitude*w*sin(w*t) +amplitude_2*w_2*sin(w_2*t) ;
    accd[2]:=amplitude*w*w*cos(w*t) +amplitude_2*w_2*w_2*cos(w_2*t) ;
    *)
    end;

    (*****)
    procedure get_initial_positions;
    {gets initial value of motor positions}
    begin
        pos1_init:= get_position_d(1);
        pos1:=pos1_init;
        pos1last:=pos1;
        posdf[1]:=pos1;

        pos2_init:= get_position_d(2);
        pos2:=pos2_init;
        pos2last:=pos2;
        posdf[2]:=pos2;

    end;

    (*****)
    procedure save_data_to_file;
    var i,j:integer; {loop indices}
    begin
        writeln('--> data being saved to file.....');
        rewrite(outdata);
        if counter > maxpoints then counter:=maxpoints;
        for i := 1 to counter do
            begin
                for j := 1 to 6 do write(outdata,data[j,i]:10:2);
                for j := 7 to data_items do write(outdata,data[j,i]:12);
                writeln(outdata);
            end;
        close(outdata);
    end;

    (*****)
    procedure get_measurements_1;
    begin
        {get velocity of motor 1}
        vel[1] := get_vel_d_i(1,ad_error);
        if ad_error then begin
            writeln('A/D conversion error has occurred -- PROGRAM ABORTED');
            halt;
        end;

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

        {get position of motor 1

```

```

    including check and correct for spurious position data}
    pos1:= get_position_d(1);
    if (abs(pos1-pos1last) > glitch_count1*reject) then begin
        pos1:=pos1last;
        glitch_count1:=glitch_count1+1;
    end
    else glitch_count1:=1;
    pos1last:=pos1;
end;

(*****)
procedure get_measurements_2;

begin
{get velocity of motor 2}
    vel[2] := get_vel_d_i(2,ad_error);
    if ad_error then begin
        writeln('A/D conversion error has occurred -- PROGRAM ABORTED');
        halt;
    end;

    {start a/d conversion for channel 0 (motor 1), gain=1}
    port[ad_chan_addr]:=0;

    {get position of motor 2
    including check and correct for spurious position data}
    pos2:= get_position_d(2);
    if (abs(pos2-pos2last) > glitch_count2*reject) then begin
        pos2:=pos2last;
        glitch_count2:=glitch_count2+1;
    end
    else glitch_count2:=1;
    pos2last:=pos2;
end;

(*****)
procedure filter_velocities;
    {low-pass filter the measured velocities, to reduce the noise}

var i: integer;

begin
    for i:=1 to 2 do begin
        velfm1[i]:=velf[i];
        velfdotm1[i]:=velfdot[i];
        velf[i]:=bfa[1,1]*velfm1[i]+bfa[1,2]*velfdotm1[i]+bfb[1]*vel[i];
        velfdot[i]:=bfa[2,1]*velfm1[i]+bfa[2,2]*velfdotm1[i]+bfb[2]*vel[i];
    end; {for}
end;

(*****)
procedure identify_parameters;

{This procedure identifies the parameters which describe the
dynamics of the manipulator. An adaptation law based on
Lyapunov stability, passivity and the direct method is used,
according to Slotine and Li's method.}

const deg_to_rad:single = 3.1415927/180.0;

var k: integer; {loop indices}
    x12,c12,s12, {intermediate calculated values}
    vel1,vel2:single;

```

```

begin
{select use of filtered or unfiltered velocity measurements (comment
  out unwanted option)}
(* {using measured velocity}
  vel1:=vel[1]; vel2:=vel[2]; *)
{using low-pass filtered velocity}
  vel1:=velf[1]; vel2:=velf[2];

{calculate tracking errors}
  err[1] := posdf[1] - pos1;
  derr[1] := veldf[1] - vel1;
  err[2] := posdf[2] - pos2;
  derr[2] := veldf[2] - vel2;

{calculate sine and cosine values}
  x12:= deg_to_rad*(pos1-pos2);
  c12:= cos(x12);
  s12:= sin(x12);

{compute the reference trajectories}
  for k:=1 to 2 do begin
    velr[k]:=veldf[k]+lambda*err[k];
    accr[k]:=accdf[k]+lambda*derr[k];
  end; {for}

{compute reference traj. errors = filtered tracking errors}
  for k:=1 to 2 do slast[k]:=s[k];
  s[1]:=vel1-velr[1];
  s[2]:=vel2-velr[2];

{compute regression matrix}
  phi[1,1]:=accr[1];
  phi[1,3]:=c12*accr[2]+s12*vel2*velr[2];
  phi[2,2]:=accr[2];
  phi[2,3]:=c12*accr[1]-s12*vel1*velr[1];
  {regression terms for torque bias param's (4 and 5) are constants
    (initialized previously)}

{update sums for discrete update of para's}
  for k:=1 to nparam do
    suma[k]:=suma[k]+(phi[1,k]*s[1]+phi[2,k]*s[2]);

  if skipcounta=skipa then begin
{discrete update of parameter vector with new estimates,
  at rate "freqa"}
    for k:=1 to nparam do begin
      para[k]:=para[k]-dt*gamma*gamma_scaling[k]*suma[k];

      {limit parameter estimates to pre-defined bounds}
      if para[k]<para_min[k] then para[k]:=para_min[k]
      else if para[k]>para_max[k] then para[k]:=para_max[k];
    end; {for}
    skipcounta:=0;
    for k:=1 to nparam do suma[k]:=0.0;
  end;{if}
  skipcounta:=skipcounta+1;
end;

(*****
procedure compute_control_input;

```

```

var i,j: integer;
    abs_val:single;      {absolute value of control input voltage}

begin
{calculate norm of filtered error and compensation term gain}
    norms:=sqrt(sqr(s[1])+sqr(s[2]));
    if norms>norm_sprime then g:=a*(norms-norm_sprime)
{   if norms>norm_sprime then g:=a*norms           }
        else g:=0.0;

{calculate control input values (voltages),
    corresponding to a torque command for each motor}
    for i:=1 to 2 do begin
        control[i]:=0.0;
        for j:=1 to nparam do
            control[i]:=control[i]+phi[i,j]*para[j];
        control[i]:=control[i]-(kv[i]+g)*s[i];
        {check for saturation}
        abs_val:=abs(control[i]);
        if abs_val > torque_limit then begin
            (*   writeln('**Caution: torque command ',i,' has saturated at',
                        torque_limit,' V');   *)
            control[i]:=(abs_val/control[i])*torque_limit;
        end; {if}
    end; {for}

{convert control voltages to 12 bit binary, and send to d/a}
    send_volt_da(1,control[1]);
    send_volt_da(2,control[2]);
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] := pos1;
            data[2,counter] := vel[1];
            data[3,counter] := posdf[1]; {pre-filtered desired path}
            data[4,counter] := control[1];
            data[5,counter] := s[1]; {filtered tracking error 1}
            data[6,counter] := velf[1]; {filtered velocity 1}
            data[7,counter] := pos2;
            data[8,counter] := vel[2];
            data[9,counter] := posdf[2]; {pre-filtered desired path}
            data[10,counter] := control[2];
            data[11,counter] := s[2]; {filtered tracking error 2}
            data[12,counter] := velf[2]; {filtered velocity 2}
            data[13,counter] := para[1]; {parameter estimates}
            data[14,counter] := para[2];
            data[15,counter] := para[3];
            data[16,counter] := para[4];
            data[17,counter] := para[5];
            data[18,counter] := g; {compensator gain}

```

```

end;
end;

(*****)
procedure display_state;
{this procedure displays the state of the robot on the screen}
begin
  gotoXY(1,2);
  writeln('posdf[1]:',posdf[1]:6:2,' pos1:',pos1:6:2,' err[1]:',
                                     err[1]:6:2);
  writeln('          vel1:',vel[1]:6:2,' control[1]:',
                                     control[1]:6:2);
  writeln('posdf[2]:',posdf[2]:6:2,' pos2:',pos2:6:2,' err[2]:',
                                     err[2]:6:2);
  writeln('          vel2:',vel[2]:6:2,' control[2]:',
                                     control[2]:6:2);
end;

(*****)
procedure find_max_norm_delta_s;
{finds maximum of the norm of the change in the filtered error,
"s", between time steps}
var norm_delta_s:single;
begin
  norm_delta_s:=sqrt(sqr(s[1]-slast[1])+sqr(s[2]-slast[2]));
  if norm_delta_s > max_normds then begin
    max_normds:=norm_delta_s;
    tmax:=t
  end {if};
end;

(*****)
procedure enableinterrupts;
inline($FB/$90); {sti/nop}

procedure disableinterrupts;
inline($FA/$90); {cli/nop}

(*****)
procedure interrupt_service; interrupt;
{this procedure (ISR) performs the measurement and control tasks which
are required at the synchronized "sampling" frequency, driven by
the A/D card interrupts}
begin
  disableinterrupts;
  case icount of
    1: begin
        t:=t+dt;
        if t>=ftime then done_flag:=true;
        get_measurements_1;
      end; {case 1}
    2: begin
        compute_desired_state;
        get_measurements_2;

```

```

    filter_velocities;
    identify_parameters;
    compute_control_input;
    find_max_norm_delta_s;
    if skipcount = skip then begin
        counter:=counter+1;
        store_data;
        skipcount:=0;
    end;
    skipcount:=skipcount+1;

    icount:=0;
    end; {case 2}

end; {case}

icount:=icount+1;
port[PIC_00]:= EOI; {clears the PIC}
enableinterrupts;
end;

(*****
{$F+}
procedure exit_handler;{$F-}

(*this procedure handles normal and abnormal termination --
it saves all data and closes files*)

begin
    exitproc:=exitsave; {restore original exit pointer}

    disableinterrupts;
    setintvec(10,old_level2_intr); {restore old level 2 interrupt vector}
    enableinterrupts;

    if exitcode<>0 then begin
        writeln('***Run-Time Error:');
        writeln('          Error code is ',exitcode);
    end;

    {set d/a back to 0 volts}
    send_volt_da(1,0);
    send_volt_da(2,0);

    save_data_to_file;

    create_info_file;

    writeln('** Please Disable Motor Servos Now **');
end;
(*****
(*****
begin {main}

{set up pointer to exit handler procedure}
    exitsave:=exitproc;
    exitproc:=@exit_handler;

{save old level 2 interrupt vector }
    getintvec(10,old_level2_intr); {interrupt type 10 = interrupt line IR2}

{set timer interrupt vector to new interrupt service routine (ISR)}

```



```

disableinterrupts;
setintvec(10,@interrupt_service);
enableinterrupts;

{***special note: interrupts are automatically enabled when
  the "writeln" command calls DOS}

{start-up message to user}
writeln;
writeln('++++++ ROBOT ADAPTIVE CONTROL PROGRAM (Motor 1 & 2) ++++++');
writeln;

{enter loop to do repeated runs}
repeat
  initialize_interfaces;

  load_trajectory_file;

  initialize_parameters;

  get_initial_positions;
  writeln;
  writeln('Current position of Motor 1 is ',pos1:6:2,' degrees. ');
  writeln('Current position of Motor 2 is ',pos2:6:2,' degrees. ');
  writeln;

  writeln('Hit any key to start operation (after 5 second delay). ');
  key_press:=readkey;
  clrscr;

{5 second delay to give operator time}
  delay(10000);

{get the time}
  gotoXY(1,1);
  gettime(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);

  get_initial_positions; {in case motors were repositioned }

{start a/d conversion for channel 0 (motor 1), gain=1}
  port[ad_chan_addr]:=0;

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

{enter loop for background tasks, while waiting for interrupts}
repeat
  display_state;

(*      check_status(dd_port1_addr,dd_port3_addr,enabled);
*)
  bkgnd_count:=bkgnd_count+1;

  until (enabled=false) OR done_flag;

  port[ad_status_addr]:=$10; {set a/d to mode 0, clear error bit, disable
    any further interrupts from A/D}

{stop operation if motor(s) disabled}

```

```

    if enabled=false then
        writeln('***Operation halted:  motor is disabled.');
```

{set d/a back to 0 volts}

```

        send_volt_da(1,0);
        send_volt_da(2,0);
```

{get the time}

```

        gettime(hr,min,sec,frac);
        gotoXY(1,10);
        if frac<10 then
            writeln('--> Operation complete, at time ',hr,':',min,':',
                sec,':',frac)
        else writeln('--> Operation complete, at time ',hr,':',min,':',
                sec,':',frac);
        writeln('  --> ',bkgnd_count,' cycles of background tasks completed');
```

{prompt user for another run}

```

        write('Do you wish to quit -- saving data to file (y/n)? ');
        readln(choice);
        until ((choice='Y') OR (choice='y'));

        {"exit_handler" procedure called by default}
    end.
```

B.2 Program Unit for Interface Card Routines

```

-----
unit DATACARD.PAS
written by:
    Gabriel D. Warshaw
    and
    Richard Gourdeau

Revised: December 16, 1992 (G.D.W.) -- reversed polarity for
                                         DA #2 output

This file contain a collection of functions and procedures to be
used with the DATA TRANSLATION CARD to control the 2 DOF robot

The interface section shows the prototypes of the functions and the
procedures while the implementation section contain the actual code

If you find a bug in this implementation, DO NOT MODIFY THIS FILE
Ask one of the writers to fix the bug.
-----

unit datacard;

interface

function timer_freq(freq : integer) : boolean;
    {return True if an error occurred}

function get_dd_word(motor : byte) : integer;
    {get 12 bits value from digital input composed of two ports}

function get_dd_byte(motor : byte) : byte;
```



```

begin
  get_dd_word := port[dd_port_addr[motor,1]]
               + (port[dd_port_addr[motor,2]] and 16#0000) shl 8;
end;

{*****}
function get_dd_byte(motor : byte) : byte;
const
  {digital inputs port addresses for values}
  dd_port_addr : array[1..4] of integer = ($229,$22A,$22B,$22C);
begin
  get_dd_byte := port[dd_port_addr[motor]];
end;

{*****}
procedure send_volt_da(da_number: byte; value : single);
const
  {port addresses of low and high byte of d/a}
  da_addr : array[1..2,1..2] of integer = (($21A,$21B),
                                             ($21C,$21D));

  {12 bit binary value for 0 volts, based on -5 to +5 range}
  zerovolts_bin : integer = 800;

  da_cvrt : single = 4096.0/10.0;
var
  v_bin : integer;
begin
  if da_number=2 then value:=-value; {reverse direction for motor #2}
  v_bin := zerovolts_bin + round(value*da_cvrt);
  port[da_addr[da_number,1]] := lo(v_bin);
  port[da_addr[da_number,2]] := hi(v_bin);
end;

{*****}
function get_ad_word(motor : byte; var error:boolean): integer;
const
  bit6 = 16#40;
  bit7 = 16#80;
  ad_loadaddr = $21A;
  ad_hiaddr = $21B;
  ad_status_addr = $218;
  ad_chan_addr = $219;
var
  status:byte;
begin
  port[ad_chan_addr] := motor-1; {switch a/d to channel motor-1, gain=1}
  repeat status := port[ad_status_addr]
  until (status AND bit7) = bit7;
  if(status and bit6) = bit6 then error := true;
  get_ad_word := port[ad_loadaddr] + port[ad_hiaddr] shl 8;

```

```
end;
```

```
{*****}
function get_ad_word_i(var error:boolean): integer;
```

```
const
  bit6 = $40;
  bit7 = $80;
  ad_loadaddr = $21A;
  ad_hiaddr = $21B;
  ad_status_addr = $218;
  ad_chan_addr = $219;
```

```
var   status:byte;
```

```
begin
  status:=port[ad_status_addr];
  if(status and bit6) = bit6 then error := true;
  get_ad_word_i := port[ad_loadaddr] + port[ad_hiaddr] shl 8;
end;
```

```
{*****}
function get_vel_d(motor : byte; var error:boolean): single;
```

```
const
  resol = 10.0/4096.0;
  ktach : array[1..2] of single = (328.0,244.0); {deg/s*volt}
```

```
begin
  get_vel_d := (get_ad_word(motor,error)*resol-5.0)*ktach[motor];
end;
```

```
{*****}
function get_vel_d_i(motor : byte; var error:boolean): single;
```

```
const
  resol = 10.0/4096.0;
  ktach : array[1..2] of single = (328.0,244.0); {deg/s*volt}
```

```
begin
  get_vel_d_i := (get_ad_word_i(error)*resol-5.0)*ktach[motor];
end;
```

```
{*****}
function get_vel_r(motor : byte; var error:boolean): single;
```

```
const
  resol = 10.0/4096.0;
  ktach : array[1..2] of single = (5.7247,4.2586); {rad/s*volt}
```

```
begin
  get_vel_r := (get_ad_word(motor,error)*resol-5.0)*ktach[motor];
end;
```

```
end.
```

B.3 Program Unit for Filtering Routines

```

-----
unit FILTER.PAS
written by:
    Gabriel D. Warshaw
    and
    Richard Gourdeau

Revised: November 17, 1992 (GDW)

This file contain a collection of functions and procedures to be
used for filtering of data during control of the 2 DOF direct drive
robot.

The interface section shows the prototypes of the functions and the
procedures while the implementation section contain the actual code

If you find a bug in this implementation, DO NOT MODIFY THIS FILE
Ask one of the writers to fix the bug.
-----

unit filter;

interface

uses m_arith;

type
    matrix = array[1..2] of state;

(*****)
procedure kalman_gains(a,b,dt:single; var kal:state);
    {calculate kalman gains to be used for kalman filtering of state
    measurements, based on first order system parameters and sampling
    interval}

procedure filter_states(t,dt2,pos:single; zk,kal:state; var x:state);
    {calculate kalman filtered values of measured states}

procedure predict_next_states(x:state; a,b,control,dt,dttover2:single;
                                var zk:state);
    {calculate predicted values of next states, for use in kalman
    filtering of next set of state measurements}

procedure pre_filter(delta_t,pole:single; var ad:matrix2; var bd:state);
    {calculate the coefficients for a discrete 2nd order filter, to
    be used for pre-filtering the desired trajectory (position) of
    the manipulator joints. The filter is critically damped, with
    two (identical) real poles placed at the value "pole" in the
    arguments.}

procedure pre_filter3(delta_t,pole:single; var ad:matrix3; var bd:state3);
    {calculate the coefficients for a discrete 3rd order filter, to
    be used for pre-filtering the desired trajectory (position) of
    the manipulator joints. The filter has 3 (identical) real
    poles placed at the value "pole" in the arguments.}

procedure butterworth2(delta_t,w0:single; var ad:matrix2; var bd:state);
    {calculate the coefficients for a discrete 2nd order butterworth
    filter, to be used for low-pass filtering the velocity measurements
  
```

```

    for the manipulator joints, to reduce the noise. The bandwidth of
    the filter is set at "w0" rad/sec.}

procedure butterworth3(delta_t,w0:single; var ad:matrix3; var bd:state3);
    {calculate the coefficients for a discrete 3rd order butterworth
    filter, to be used for low-pass filtering the velocity measurements
    for the manipulator joints, to reduce the noise. The bandwidth of
    the filter is set at "w0" rad/sec.}

    (*****
    implementation
    *****)

procedure kalman_gains(a,b,dt:single; var kal:state);
    var
        i: integer;
        hphv,vv,vw:single;
        pk,p:matrix;
    begin
        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 5000 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*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(t,dt2,pos:single; xk,kal:state; var x:state);
    *****)

    var
        perr:single; {prediction error}
    begin
        if t < dt2 then xk[1] := pos;
        perr := pos-xk[1];
        x[1] := xk[1]+kal[1]*perr;
        x[2] := xk[2]+kal[2]*perr;
    end;

    (*****
    procedure predict_next_states(x:state; a,b,control,dt,dtdtover2:single;
    var xk:state);
    *****)

    begin
        xk[1] := x[1]+(dt-dtdtover2*a)*x[2]+dtdtover2*b*control;
        xk[2] := (1.0-dt*a)*x[2]+b*dt*control;
    end;

```



```

(*****)
procedure pre_filter(delta_t,pole:single; var ad:matrix2; var bd:state);
const iterations=20;
var i,j: integer;
    factorial,alpha,beta,omega,tpower: single;
    amatrix,atemp,atemp2,i2x2,btemp:matrix2;
    bvector:state;
begin
    beta:= -2.0*pole;
    omega:=-pole;
    alpha:=omega*omega;

    i2x2[1,1]:=1.0;
    i2x2[1,2]:=0.0;
    i2x2[2,1]:=0.0;
    i2x2[2,2]:=1.0;

    amatrix[1,1]:=0.0;
    amatrix[1,2]:=1.0;
    amatrix[2,1]:=-alpha;
    amatrix[2,2]:=-beta;

    bvector[1]:=0.0;
    bvector[2]:=alpha;

    matrix_const(1.0,i2x2,ad);      {set ad = I matrix}
    matrix_const(1.0,i2x2,atemp);   {set atemp = I matrix}
    factorial:=1.0;
    tpower:=1.0;

    {calculate Ad matrix}
    for i:=1 to iterations do begin
        matrix_mult(atemp,amatrix,atemp);
        factorial:=i*factorial;
        tpower:=tpower*delta_t;
        matrix_const(tpower/factorial,atemp,atemp2);
        matrix_add(ad,atemp2,ad);
    end; {for}

    matrix_const(delta_t,i2x2,btemp); {set btemp= I*delta_t matrix}
    matrix_const(1.0,i2x2,atemp); {set atemp = I matrix}
    factorial:=1.0;
    tpower:=delta_t;

    {calculate Bd matrix}
    for i:=2 to iterations+1 do begin
        matrix_mult(atemp,amatrix,atemp);
        factorial:=i*factorial;
        tpower:=tpower*delta_t;
        matrix_const(tpower/factorial,atemp,atemp2);
        matrix_add(btemp,atemp2,btemp);
    end; {for}
    vector_mult(btemp,bvector,bd);
end;

(*****)
procedure pre_filter3(delta_t,pole:single; var ad:matrix3; var bd:state3);
const iterations=20;
var i,j: integer;

```

```

factorial,a,b,c,omega,tpower: single;
amatrix,atemp,atemp2,i3x3,btemp:matrix3;
bvector:state3;

begin
  a:=-3.0*pole;
  b:=3.0*pole*pole;
  c:=-pole*pole*pole;

  for i:=1 to 3 do for j:=1 to 3 do i3x3[i,j]:=0.0;
  for i:=1 to 3 do i3x3[i,i]:=1.0;

  amatrix[1,1]:=0.0; amatrix[1,2]:=1.0; amatrix[1,3]:=0.0;
  amatrix[2,1]:=0.0; amatrix[2,2]:=0.0; amatrix[2,3]:=1.0;
  amatrix[3,1]:=-c; amatrix[3,2]:=-b; amatrix[3,3]:=-a;

  bvector[1]:=0.0;
  bvector[2]:=0.0;
  bvector[3]:=c;

  matrix_const3(1.0,i3x3,ad);      {set ad = I matrix}
  matrix_const3(1.0,i3x3,atemp);   {set atemp = I matrix}
  factorial:=1.0;
  tpower:=1.0;

  {calculate Ad matrix}
  for i:=1 to iterations do begin
    matrix_mult3(atemp,amatrix,atemp);
    factorial:=i*factorial;
    tpower:=tpower*delta_t;
    matrix_const3(tpower/factorial,atemp,atemp2);
    matrix_add3(ad,atemp2,ad);
  end; {for}

  matrix_const3(delta_t,i3x3,btemp); {set btemp= I*delta_t matrix}
  matrix_const3(1.0,i3x3,atemp);   {set atemp = I matrix}
  factorial:=1.0;
  tpower:=delta_t;

  {calculate Bd matrix}
  for i:=2 to iterations+1 do begin
    matrix_mult3(atemp,amatrix,atemp);
    factorial:=i*factorial;
    tpower:=tpower*delta_t;
    matrix_const3(tpower/factorial,atemp,atemp2);
    matrix_add3(btemp,atemp2,btemp);
  end; {for}
  vector_mult3(btemp,bvector,bd);
end;

(*****)
procedure butterworth2(delta_t,w0:single; var ad:matrix2; var bd:state);
const iterations=6;

var i,j: integer;
    factorial,alpha,beta,tpower: single;
    amatrix,atemp,atemp2,i2x2,btemp:matrix2;
    bvector:state;

begin
  beta:=sqrt(2.0)*w0;
  alpha:=w0*w0;

```

```

i2x2[1,1]:=1.0;
i2x2[1,2]:=0.0;
i2x2[2,1]:=0.0;
i2x2[2,2]:=1.0;

amatrix[1,1]:=0.0;
amatrix[1,2]:=1.0;
amatrix[2,1]:=-alpha;
amatrix[2,2]:=-beta;

bvector[1]:=0.0;
bvector[2]:=alpha;

matrix_const(1.0,i2x2,ad);      {set ad = I matrix}
matrix_const(1.0,i2x2,atemp);   {set atemp = I matrix}
factorial:=1.0;
tpower:=1.0;

{calculate Ad matrix}
for i:=1 to iterations do begin
    matrix_mult(atemp,amatrix,atemp);
    factorial:=i*factorial;
    tpower:=tpower*delta_t;
    matrix_const(tpower/factorial,atemp,atemp2);
    matrix_add(ad,atemp2,ad);
end; {for}

matrix_const(delta_t,i2x2,btemp); {set btemp= I*delta_t matrix}
matrix_const(1.0,i2x2,atemp); {set atemp = I matrix}
factorial:=1.0;
tpower:=delta_t;

{calculate Bd matrix}
for i:=2 to iterations+1 do begin
    matrix_mult(atemp,amatrix,atemp);
    factorial:=i*factorial;
    tpower:=tpower*delta_t;
    matrix_const(tpower/factorial,atemp,atemp2);
    matrix_add(btemp,atemp2,btemp);
end; {for}
vector_mult(btemp,bvector,bd);
end;

(*****)
procedure butterworth3(delta_t,w0:single; var ad:matrix3; var bd:state3);

const iterations=20;

var i,j: integer;
    factorial,a,b,c,tpower: single;
    amatrix,atemp,atemp2,i3x3,btemp:matrix3;
    bvector:state3;

begin
    a:=2.0*w0;
    b:=2.0*w0*w0;
    c:=w0*w0*w0;

    for i:=1 to 3 do for j:=1 to 3 do i3x3[i,j]:=0.0;
    for i:=1 to 3 do i3x3[i,i]:=1.0;

    amatrix[1,1]:=0.0; amatrix[1,2]:=1.0; amatrix[1,3]:=0.0;
    amatrix[2,1]:=0.0; amatrix[2,2]:=0.0; amatrix[2,3]:=1.0;

```

```

amatrix[3,1]:=-c; amatrix[3,2]:=-b; amatrix[3,3]:=-a;
bvector[1]:=0.0;
bvector[2]:=0.0;
bvector[3]:=c;

matrix_const3(1.0,i3x3,ad);      {set ad = I matrix}
matrix_const3(1.0,i3x3,atemp);   {set atemp = I matrix}
factorial:=1.0;
tpower:=1.0;

{calculate Ad matrix}
for i:=1 to iterations do begin
    matrix_mult3(atemp,amatrix,atemp);
    factorial:=i*factorial;
    tpower:=tpower*delta_t;
    matrix_const3(tpower/factorial,atemp,atemp2);
    matrix_add3(ad,atemp2,ad);
end; {for}

matrix_const3(delta_t,i3x3,btemp); {set btemp= I*delta_t matrix}
matrix_const3(1.0,i3x3,atemp);   {set atemp = I matrix}
factorial:=1.0;
tpower:=delta_t;

{calculate Bd matrix}
for i:=2 to iterations+1 do begin
    matrix_mult3(atemp,amatrix,atemp);
    factorial:=i*factorial;
    tpower:=tpower*delta_t;
    matrix_const3(tpower/factorial,atemp,atemp2);
    matrix_add3(btemp,atemp2,btemp);
end; {for}
vector_mult3(btemp,bvector,bd);
end;
end.

```

B.4 Program Unit for Matrix Math Routines

```

{-----}
{ unit M_ARITH.PAS
{ written by:
{           Gabriel D. Warshaw
{
{ Revised: Oct. 5, 1992
{
{ This file contain a collection of functions and procedures to be
{ used for performing a range of simple matrix computations.
{
{ The interface section shows the prototypes of the functions and the
{ procedures while the implementation section contains the actual code.
{
{ If you find a bug in this implementation, DO NOT MODIFY THIS FILE
{ Ask one of the writers to fix the bug.
{-----}

unit m_arith;

```

```

interface
  type
    state = array[1..2] of single;
    matrix2 = array[1..2,1..2] of single;
    state3 = array[1..3] of single;
    matrix3 = array[1..3,1..3] of single;

  (*****
  procedure matrix_add(a,b:matrix2; var c:matrix2);
    {add two 2x2 matrices}

  procedure matrix_mult(a,b:matrix2; var c:matrix2);
    {multiply two 2x2 matrices}

  procedure vector_mult(a:matrix2; b:state; var c:state);
    {multiply a 2x2 matrix by a 2x1 vector}

  procedure matrix_const(a:single; b:matrix2; var c:matrix2);
    {multiply a 2x2 matrix by a scalar constant}

  procedure matrix_add3(a,b:matrix3; var c:matrix3);
    {add two 3x3 matrices}

  procedure matrix_mult3(a,b:matrix3; var c:matrix3);
    {multiply two 3x3 matrices}

  procedure vector_mult3(a:matrix3; b:state3; var c:state3);
    {multiply a 3x3 matrix by a 3x1 vector}

  procedure matrix_const3(a:single; b:matrix3; var c:matrix3);
    {multiply a 3x3 matrix by a scalar constant}

  (*****
  implementation
  (*****
  procedure matrix_add(a,b:matrix2; var c:matrix2);
    {add two 2x2 matrices}

  var i,j: integer;
  begin
    for i:=1 to 2 do
      for j:=1 to 2 do
        c[i,j]:=a[i,j]+b[i,j];
  end;

  (*****
  procedure matrix_mult(a,b:matrix2; var c:matrix2);
    {multiply two 2x2 matrices}

  var i,j,k: integer;
    sum:single;
  begin
    for i:=1 to 2 do
      for j:=1 to 2 do begin
        sum:=0.0;
        for k:=1 to 2 do
          sum:=sum+a[i,k]*b[k,j];
        c[i,j]:=sum;
      end;
    end;
  end;

```

```

(*****)
procedure vector_mult(a:matrix2; b:state; var c:state);
    {multiply a 2x2 matrix by a 2x1 vector}

var i,k: integer;
    sum:single;
begin
    for i:=1 to 2 do begin
        sum:=0.0;
        for k:=1 to 2 do
            sum:=sum+a[i,k]*b[k];
        c[i]:=sum;
    end;
end;

(*****)
procedure matrix_const(a:single; b:matrix2; var c:matrix2);
    {multiply a 2x2 matrix by a scalar constant}

var i,j: integer;
begin
    for i:=1 to 2 do
        for j:=1 to 2 do
            c[i,j]:=a*b[i,j];
        end;
    end;

(*****)
procedure matrix_add3(a,b:matrix3; var c:matrix3);
    {add two 3x3 matrices}

var i,j: integer;
begin
    for i:=1 to 3 do
        for j:=1 to 3 do
            c[i,j]:=a[i,j]+b[i,j];
        end;
    end;

(*****)
procedure matrix_mult3(a,b:matrix3; var c:matrix3);
    {multiply two 3x3 matrices}

var i,j,k: integer;
    sum:single;
begin
    for i:=1 to 3 do
        for j:=1 to 3 do begin
            sum:=0.0;
            for k:=1 to 3 do
                sum:=sum+a[i,k]*b[k,j];
            c[i,j]:=sum;
        end;
    end;

(*****)
procedure vector_mult3(a:matrix3; b:state3; var c:state3);
    {multiply a 3x3 matrix by a 3x1 vector}

var i,k: integer;
    sum:single;
begin
    for i:=1 to 3 do begin
        sum:=0.0;
        for k:=1 to 3 do

```

```
        sum:=sum+a[i,k]*b[k];
      c[i]:=sum;
    end;
  end;

  (*****)
  procedure matrix_const3(a:single; b:matrix3; var c:matrix3);
    {multiply a 3x3 matrix by a scalar constant}

  var i,j: integer;

  begin
    for i:=1 to 3 do
      for j:=1 to 3 do
        c[i,j]:=a*b[i,j];
      end;
    end;
  end.
```

END

0 8 - 1 1 - 9 4

FIN